**The Distributed Oceanographic Data System (DODS): An Extensible Approach to Distributing Scientific Data**

**Tutorial Paper**

Number: 12

Name: Patrick Kellogg

**The Distributed Oceanographic Data System (DODS): An Extensible Approach to Distributing Scientific Data**

**Tutorial Paper**

# Abstract

Oceanographic science creates several unique and interesting problems of data storage and distribution. Not only are there a large variety of parameters (and instruments to record those parameters), but also the very nature of the data set is often at odds with the desired types of analysis. With the invention of new types of oceanographic instruments, and the importance of studying the world’s oceans, data is arriving to oceanographic databases faster than it can be easily organized with useful metadata. The Distributed Oceanographic Data System (DODS) is an attempt to create an easy to use and easily extensible method of distributing scientific data. Our system not only works for several different storage definitions, but can also produce output in several different formats. More importantly, the solution is scalable and open, and will be used by many scientific institutions in the next several years. The CEDARweb Perl script was written as a user-friendly front-end to the CEDAR dataset. The metadata, stored in MySQL, is easy to navigate and use, and the DODS server delivers exactly the data to the user needs and no more, due to its unique object-oriented return structure.

# I. Introduction

There are many agencies supporting or sponsoring oceanographic sciences, including the National Oceanographic and Atmospheric Administration (NOAA), the Intergovernmental Oceanographic Commission (IOC), and the Scientific Committee for Ocean Research (SCOR), just to name a few. However, since all the world’s oceans are connected, inter-agency communication is extremely important for this field. Unfortunately, communicating effectively is a lot harder than merely gathering data (Muntz). With the invention of the Internet, a lot of data is “out there”, and available to download and use. But without a good understanding of what the data set means, how it was collected, and any problems or pitfalls of using the information, the data is just another large set of meaningless numbers. So, a good data delivery system needs to convey both the data itself as well as the “metadata”, which is defined as “data about data” (Strebel).

The DODS project was conceived as a way to allow users to only retrieve the data that they need (Gallagher, Cornillon). Often, a researcher must download several megabytes of data, only to retrieve a single column or set of rows of the data. DODS tries to prevent this with several methods. First, we allow the user to see what data is available by letting them examine the metadata. Next, we allow the user to choose only the fields and rows that they desire. Finally, we give the user feedback about the size and nature of the data that will be returned. There might be a possibility that the original query was wrong, and the user should be able to fix it before proceeding. By offering the user “data about data”, the user can create better queries, hopefully saving time both during the http transfer, and when analyzing the data (Olson, Treinish).

# II. Oceanography

**i. History**

The oceanographic community has long tried to combine multiple data sets at a centralized location. In 1957, the World Data Center-A was set up as a clearinghouse for ocean data, and proved to be quite successful (Report of the Inter-American Conference on Marine Studies). However, in the eighties, the trend reversed, due to the popularity of the personal computer and decentralized architectures. Many smaller, localized libraries were set up, particularly to study local phenomenon seen in lakes, bays, and inland seas. As a result, we now have a large mixture of local and global data centers all over the world.

Unfortunately, many of these data centers collect information and store it in incompatible formats. The field of computer science has produced many interesting inventions, such as relational databases, object-oriented databases, and other encapsulated formats. However, due to the incompatibility of import functions and analysis programs, most scientific data is still stored as plain ASCII data in marginally structured text files. Usually, this data is just a set of numbers with a comma- or tab-delimited layout. Any metadata about the dataset or variable names is often placed at the top of end of the data in a text paragraph. Creating programs to automatically read and processed metadata in this format can be very hard.

Meanwhile, trying to import a lot of scientific data into a relational database management system (RDBMS) might also be inconvenient. First of all, the data might not be organized in a relational manner, and the process of importing the data might take a lot of time. There are often justifiable reasons why the scientific institution might want to keep their data in a flat file or another format. For example, other programs might depend on the data arriving in the original format, and the data needs to remain compatible.

**ii. Data Centers**

The World Data Center is actually a combination of almost fifty data centers, combined logistically instead of geographically. The World Data Center for oceanography is located in Silver Spring, Maryland. The Center describes itself as “one component of a global network of discipline subcenters that facilitate international exchange of scientific data” (quoted from their web page). There are also “sister” Oceanographic Data Centers in Obninsk, Russia, and Tianjion, China, a World Data Center for Marine Geology and Geophysics located in Boulder, Colorado (as well as a sister site in Moscow, Russia), as well as a World Data Center for Tides in Brussels, Belgium.

The National Climatic Data Center (formerly the Joint Weather Records Center), is the largest of America's thirteen data centers, and is the world's largest archive of weather data, due to the large amount of satellite, NEXRAD, and ASOS data that it receives. In fact, the Center has more than 150 years of data on hand with 55 gigabytes of new information added each day. However, only part of the data is available electronically, and an even smaller percentage is available online. The rest is stored on paper records, microfiche, and unmounted backup tapes. The backlog of unprocessed information, as well as the large amount of new data, is a huge challenge for the Data Centers.

# iii. Issues

The Data Centers have some data problems that ordinary scientific research centers don’t have. First of all, the average amount of data stored at a Data Center is several terabytes (240 bytes) of uncompressed data. Second, the *rate* of data acquisition is large, and increasing every year. For example, by 2005 the CERN large hadron collector will produce several petabytes (250 bytes) of raw physics data every year (Arderiu-Ribera). Finally, requests for data are also increasing, possibly because of the greater availability of data on the Internet. All of these details add up to a complex problem.

Part of the solution is solved with sophisticated compression techniques.

However, each dataset needs an individual data compression algorithm in order “tune” it effectively. To try and use one single scheme for compressing all of the data will lead to suboptimal results. Faster and larger hardware and disk arrays will help, but the rate of data acquisition is quickly outpacing the rate at which hardware is improving (Thompson). There is no “silver bullet” for storing large amounts of scientific data.

**iv. Combined Efforts**

The first large-scale international scientific enterprises were the International Polar Years of 1882-1883, which eventually led to the International Geophysical Year of 1957-1958 (IGY). The IGY was the first major attempt at a unified census of global data. Many different countries and research centers were involved to try and gain a complete “snapshot” of scientific data for that period. Since then, there has been the International Quiet Sun Year (1964-65) and the Active Sun Year (1968-69), among many other combined global efforts in different scientific fields.

The High Altitude Observatory at the National Center for Atmospheric research has been involved with the several joint projects, including: Coupling, Energetics and Dynamics of Atmospheric Regions (CEDAR), Geospace Environment Modeling (GEM), the Joint Global Ocean Flux Study (JGOFS), and Radiative Inputs of the Sun to Earth (SunRISE). The DODS project at NCAR was an attempt to store data for these and other joint projects for the scientific community. Even though the DODS project was created to serve oceanographic data, it can deliver any kind of formatted data, scientific or otherwise. However, oceanographic data is interesting because it combines the features of many other areas of science.

# III. Oceanographic Data

# i. Parameters of Study

For example, oceanographers study sea surface wind data, much like meteorologists. The field of acoustics and signal processing is used for analyzing water density, as well as for sonar and other underwater applications. “Biological oceanography” is the study of marine life, and has often been overlooked by data collections (Paterson). Naturally, chemists and seismologists are also interested in collecting and analyzing oceanographic data. Some researchers concentrate on geological aspects, and map out different aspects of the ocean, using Global Information Systems (GIS) (Lingsch). Recently, using satellites has been an extremely productive way to remotely sense ocean data (Cho, Lingsch, Gower).

Some interesting parameters of study include sea surface temperature and sea surface salinity, two important and complex variables connected with fresh water fluxes into and out of the ocean. Also, the sea level, both “relative” between coast and ocean, and “absolute” as referenced to a geographic center can be recorded (also called ocean topography), either directly using wind instruments or remotely with radar. Surface ocean carbon fluxes (both the total and percentage of CO2 concentration) and the sea ice coverage and extent have been popular to study recently, since they both relate to theories of global warming (Idso, Chierici). Most of the measurements can also be repeated for subsurface temperatures, salinity, current/velocity, and depth profiles, and can be used to create models of ocean activity. There are more parameters to study, such as surface heat and fresh water flux, surface wind and wind stress, water mass renewal, and interior ocean circulation, but it is enough to say that the ocean is rich for large-scale data collection.

In addition, the data can be collected in a variety of different ways. Data may be obtained from ships, buoys (either moored or drifting) and satellites, among many other possible instruments. Data from moving ships can be difficult to analyze because the data is not arranged in a convenient latitude-and-longitude grid for easy plotting. Instead, the circuit that the boat travels forms a path in time as well as space that need to be “assimilated” before the data can be used (Lorenc, Malanotte-Rizzoli). One interesting note is ocean-going vessels are the oldest method of oceanographic data collection, and data can be found going back as far as 200 years ago (Elms, MEDEA).

# IV. DODS

# i. Input and Output Formats

The DODS server can use different output formats as long as they are written as correct “output filters” for the DODS core. With the new version of DODS, as many (or as few) output filters can be installed as the user chooses. This allows one data center to serve NetCDF and CEDAR data, for example, while next site only serves FITS data. The user doesn’t need to care about the underlying data storage, since all the data can be returned in a standard DODS object, or any number of other data formats. Currently, DODS can serve data that is stored in the CEDAR format, the Madrigal format (an early variant of CEDAR), FITS, FreeForm, JGOFS, and NetCDF. NetCDF in particular has been a very productive way to store scientific data, and is robustly supported at Unidata in Boulder, Colorado (Rew). The output formats currently supported include flat ASCII format, binary format, and a tab-delimited output. In addition, DODS can serve several different types of DODS “objects” that encapsulate the data. See Appendix A for several examples of DODS output using this class structure.

The DODS format encapsulates the data inside object-oriented sections that resemble a class descriptor. In fact, the DODS format can be used inside C++ code to create an object-oriented data class. The class structure can be placed directly inside existing code, without compilation errors. This allows the returned output to be used as a C++ structure, where the data can be retrieved by standard object-oriented calls.

First, the "DODS Descriptor Service" (DDS) is a structure that describes the type of the data that will be returned in the DODS object. For example, the data might be an array of 200 floats. The DDS can be placed into a header file, so other C++ routines can identify the variable types. Or, the DDS could be used by a routine to allocate memory for the entire object before the data is actually retrieved. This gives client programs an easy way find out how large the returned data object will be before requesting all the data. If the object is too large, the operation can abort before any http request is made.

Next, the "DODS Attribute Service" (DAS) contains text information that explains what the returned variables represent. This data can be used as headers for a table, for example. Again, our philosophy is to give users as much information and metadata as possible before they have to commit to a large data query. The larger DODS object (DODS) encapsulates the DAS and DDS into a larger, complete object, and also includes the data. The data can be returned at the end of the structure as a compressed binary file to save space and transmission time, or can be returned in an encapsulated ASCII format for greater readability and more flexible use by a C++ program.

# V. Metadata

# i. The DODS Approach

A large part of our DODS effort is concerned with storing and retrieving information about the CEDAR data sets. CEDAR stands for “Coupling, Energetics and Dynamics of Atmospheric Regions”, and is a large set of thermosphere, ionosphere, and exosphere data used to understand coupling, energetics, chemistry, and dynamics on regional and global scales. The DODS website states that their two main assumptions are, “data are often most appropriately distributed by the individual or group that has developed them; and the user will in general like to access data from the application software with which s/he is most familiar” (quoted from the DODS website).

# ii. MySQL

For the CEDAR metadata, we decided to use MySQL as the underlying database. MySQL has its faults and detractors (Adida), but it was used because it is free, readily available, and integrates well with Perl and the Apache web server. Since we are using the MySQL data as a “static” metadata dataset (that is, information is added infrequently and the database can be recreated at will), we didn’t care about rollback or atomic operations. Also, all of our queries are very simple, and do not require subqueries, stored procedures, or database triggers. We will think about using a more robust database in the future, but for now, MySQL works just fine.

The data can be imported overnight in batch mode, using the same code to parse through the CEDAR database that the DODS core uses. The code parses through each file in the database, and extracts the relevant header information. Then, it walks through each record in the file and extracts several tables of information. See Appendix B: Metadata Layout for MySQL for a complete description of the database tables. In the future, we would like to be able to add or subtract CEDAR files at will, and MySQL should be able to update or delete existing table information. However, since we only update the CEDAR database twice a year, and the creation of the metadata can be performed overnight, we haven’t added this functionality. Perhaps, due to future combined efforts, such as the TIMED project, this might be added soon.

# VI. CEDAR and CEDARweb

# i. Heliospatial Data

The CEDAR dataset contains approximately 14 gigabytes of data from almost fifty different instruments, including incoherent scatter radars, HF radar, digisondes, Fabry-Perot interferometers, infrared Michelson interferometers, lidars, and other middle atmosphere instruments. In addition, the database contains data from over twenty numerical models that can be compared to actual data. The database group holds an annual conference and tutorial each June in Boulder, Colorado, to discuss data access and coverage.

Internally, the CEDAR data is stored in a slightly cumbersome format based on the Cray Operating System (COS). The data is uniquely identified by a three-part key comprised of the KINST, KINDAT, and the time. The KINST is the instrument code, while the KINDAT is the “kind of data” that the instrument is delivering. Together, the KINST and the KINDAT form a unique key for that type of data and instrument in the database. The idea of “time” is encapsulated by the idea of a data record made up of a manageable number of bytes (usually under 200k, but occasionally larger). This creates a slight problem, since the data records are indivisible. That is, the user can’t request part of a data record, but must retrieve the entire range.

In addition, each data record contains a JPAR and MPAR region. The JPAR is used to store single values, while the MPAR is made up of “multiple parameter” array values. This way, we don’t have to waste space by storing a column of data whose value doesn’t change throughout the record. Using the DODS interface, the user can subselect by JPAR values, but not by MPAR values. Because of the way the MPAR is structured, it can’t be used as a key. Again, the user has to retrieve all the data in the array of an MPAR parameter. Though this interface may send a little too much data, some flexibility had to be given up to accommodate the CEDAR data format.

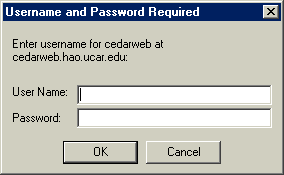
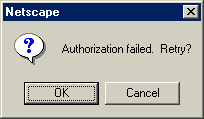
# ii. The Interface

Version 1.0 of the CEDAR interface was called “cmenu”, and was originally written in Fortran 90, then later ported to native Perl code by Roy Barnes. Files were available by an ftp interface, hosted by a local CEDAR server. The interface was given a web interface by Patrick Kellogg, but the architecture remained the same. The interface merely called the Fortran code, and then created web pages and hypertext links dynamically. Neither of these versions used the DODS core, and the system proved to be unstable and unreliable.

The interface took a giant step forward with the creation of version 2.1 of the CEDARweb interface. See Appendix C for a copy of the Perl source code. For the first time, the code was using the DODS core, and we were able to use complex data query methods. The code calls the Apache server directly, through the “mod\_perl” module. This allowed us to write Apache modules entirely in Perl. In addition, every time Perl is called, we don’t need to start an external interpreter and wait for Perl to execute. Since the module is now “persistent”, we see faster code and MySQL requests.

# iii. Walkthrough

We are using Apache authentication in order to control access to the CEDARweb data. When the user tries to connect to the cedarweb.pl page, Apache brings up a screen for the user to enter a username and password. If the authentication succeeds, the main CEDARweb page is displayed. If the user name and password fails, the user receives a “failure” message and must try again.

### Figures 1 and 2: CEDARweb Authorization

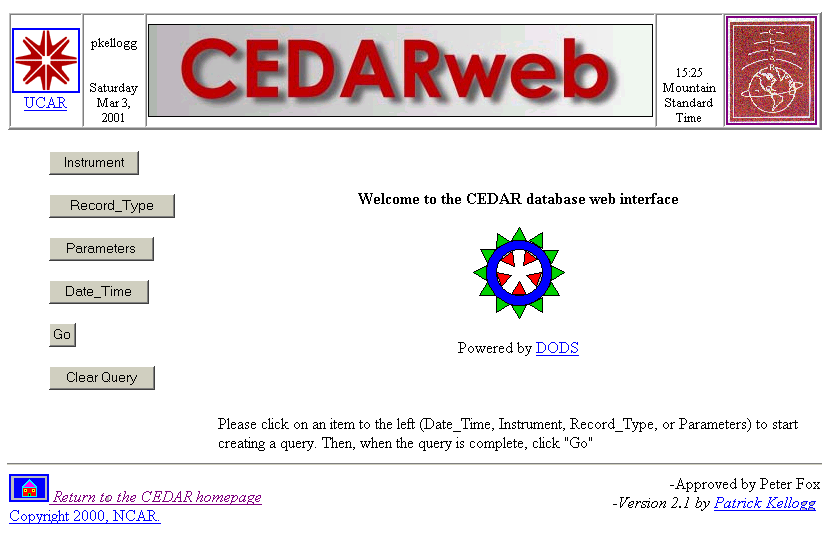


Figure 3: CEDARweb Starting Page

The main page credits the University Corporation for Atmospheric Research (UCAR), and the CEDAR community. Also, we are trying to show support for the DODS project development, both in-house and at Unidata.

The main page uses a standard Internet format, with a display banner across the top of the page, and navigation buttons at the left edge. The top banner displays the current local time in Boulder, Colorado, as well as the user's login name. The bottom of the page adds a link back to the main page, as well as contact information for our group.

This page was designed to be extremely modular. The code to display the main banner is encapsulated in a Perl subroutine, so it can be reused for every page. This eliminates overhead, as well as helps testing and verification, since the same code is used over and over again. Likewise, the left navigation buttons and the bottom footer are the same for every page (with some modifications). Only the information in the main window will change as the user walks through building a database query.

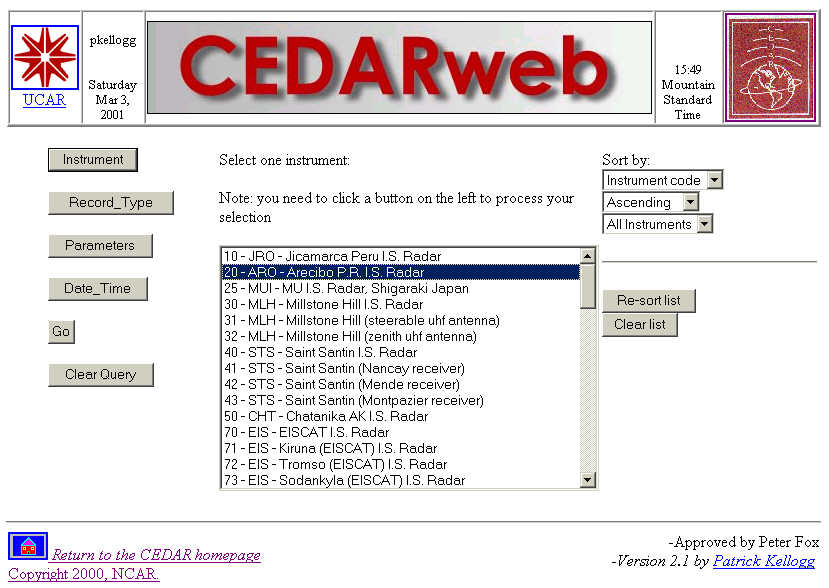


Figure 4: The Instrument Page

This is the page the user will see when they click on the “Instrument” navigation button on the left side of the window. Note that the user doesn’t have to use all of the navigation buttons, nor do they have to click on them in any certain order. In fact, for many queries, it might be useful to choose the instrument last. However, for this walkthrough, we will assume that the user wants to first select the location where the data was collected.

As of March 2001, the CEDAR database contains data from seventy different instruments from all over the world. Note, however, that many instruments are from the same general location or scientific institution. If the user wants to get data from more than one instrument at a single location, they must read the instrument’s description and decipher the location themselves. The list box describes both the unique numerical CEDAR code for the instrument, as well as an alphanumeric identifier for the institution. However, many new users aren’t familiar with that notation, so we also provide a text description of the type of instrument. Currently, the user can only query one instrument at a time. We have looked into creating “append queries” that will combine data from more than one institution in a single database query, but we decided that the concatenation issues and problems were not worth the added flexibility. Instead, the user must choose a single instrument from the list, and then proceed with another navigation button.

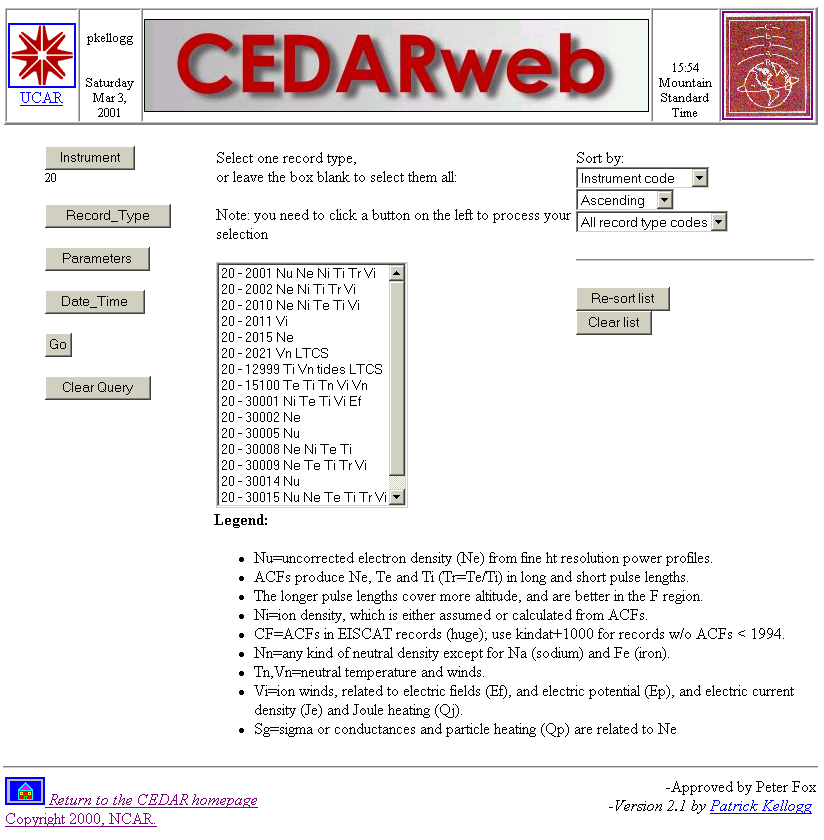


Figure 5: The Record\_Type Page

The CEDAR database has a slightly confusing concept of a “record type”. This is a collection of one or more parameters that are grouped together in the database. For example, an instrument may record electron temperature by itself. Or, since analyzing the electron temperature depends heavily on the ion temperature, those two parameters could be combined together into a two-parameter record type. This idea is often useful when we want to ensure that any error measurement is delivered along with the original parameter. However, users that are new to the CEDAR community have been confused by the idea of a record type, so we also provide the functionality to choose individual parameters, if the user wishes.

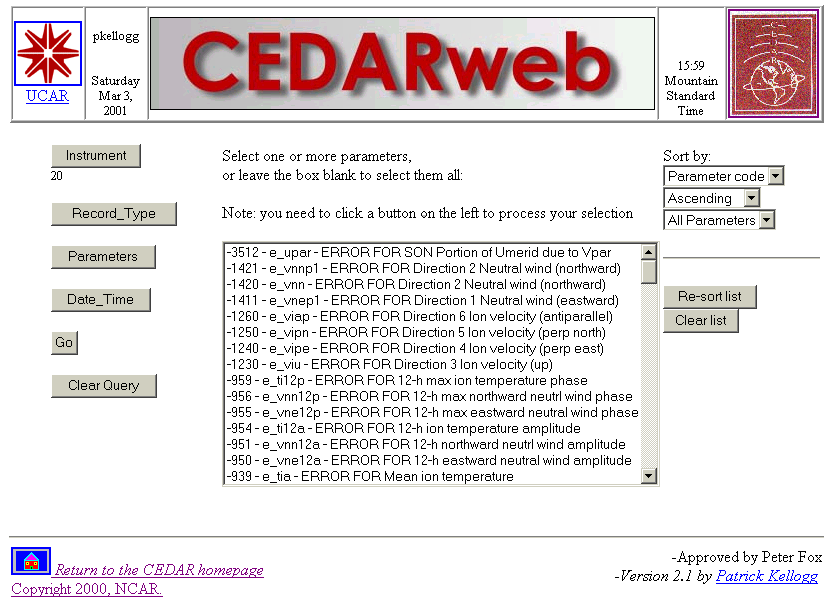


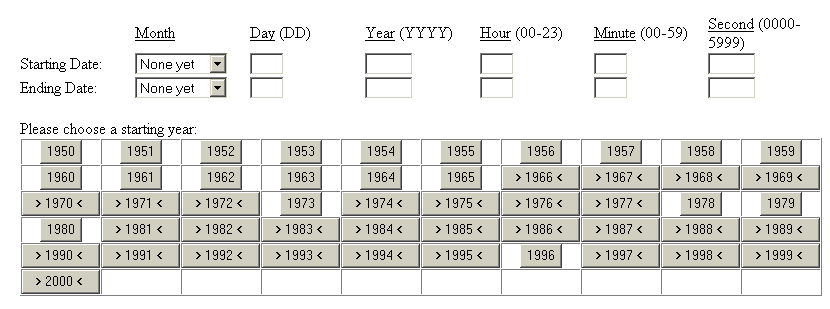
Figure 6: The Parameter Page

Note that as the user builds the database query, the values chosen so far appear as information in the left-most navigation column. In this example, since the user has chosen “10 – ARO – Arecibo P.R. I.S. Radar”, the number 10 is placed below the “Instrument” button as an easy to remember shorthand.

Also, since the user has already chosen something, all future screens will show a subset of the metadata. This page, for example, only shows parameters that are valid for the chosen Arecibo radar. This will reduce the amount of information that the user has to examine in order to build a query. However, note that if the user chooses a parameter, and then modifies which instrument they would like to examine, the resulting query may return no data. In other words, the changed “instrument-and-parameter” combination might not exist in the database.

We looked into enforcing that all queries are valid at all stages of the operation, but decided against adding that functionality. First of all, the query might be in an invalid state only temporarily, while the user intends on fixing the query in a later step. Also, users didn’t like the delay time needed to constantly perform a database call make sure the query is consistent. Finally, users were confused by the error message that their query was invalid. Since we show them the URL to the data before performing any data extraction, the user will find out that the set of chosen options returns no data before finishing the query.

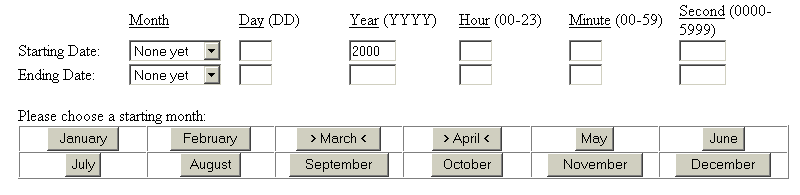
Again, we only allow the user to choose one parameter at a time. If they want more than one variable, they could leave the list box blank (and retrieve all the available parameters). Or, they could choose a record type that includes all of the parameters that they wish to study.



*Figure 7: The Interactive Calendar (Year)*

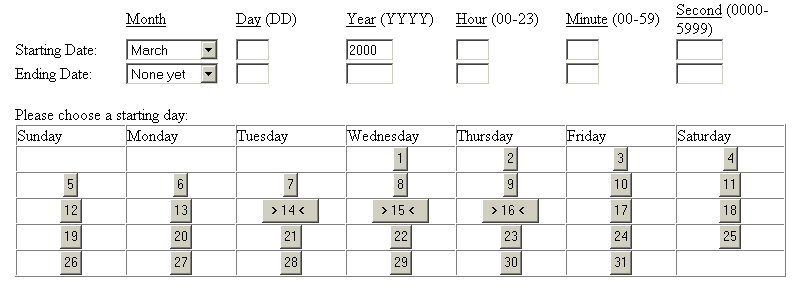
The interface contains a clever “interactive calendar” where the user can examine the data coverage as they build the database query. Of course, the user can always type in a data range in the text boxes at the top of the screen. However, the interactive calendar is a quick way to prevent the user from choosing invalid data ranges where the CEDAR database has no data.

Buttons that are highlighted with a pair of brackets (i.e. “> <”) contain data for the chosen parameters. They are the only buttons that can be clicked, since the other non-highlighted buttons are disabled. When a user chooses a valid year, a list of months for that year is displayed.



*Figure 8: The Interactive Calendar (Month)*

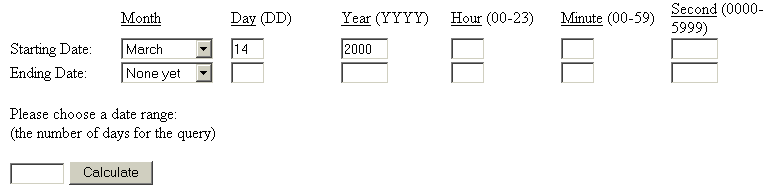
Again, the interface gives a quick way to examine the data coverage within a given timeframe. Choosing a valid highlighted month shows the user a calendar for that month.



*Figure 9: The Interactive Calendar (Day)*

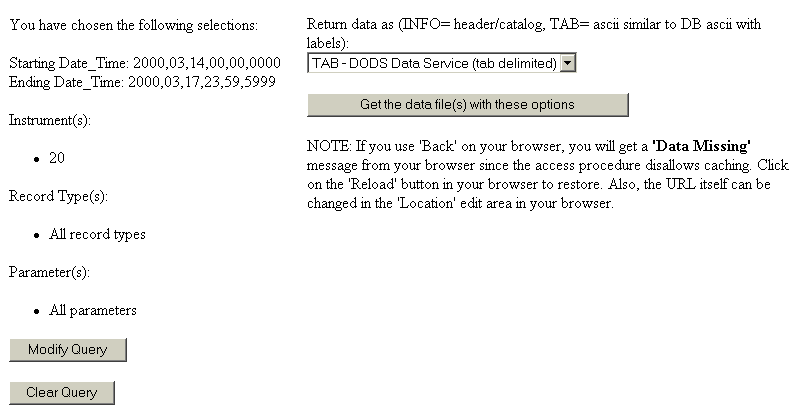
The final step to choosing a starting date is to click on a highlighted day within the chosen month. The selection process stops here, instead of going on to show the user coverage within a given day. We decided that the user can choose the hour, minute, or second easier using a text box at the top of the screen, than by trying to add any “stopwatch” functionality. Also, the CEDAR database often forces users to select data that fits entirely within a single data record, so attempts to be extremely precise will be ignored by the DODS server.

Note that the calendar is correct for the chosen year and month. The number of days is correct, and the first of the month falls on the correct day of the week. This is important when trying to analyze combined initiative projects that might have taken place over a weekend or particular day. The calendar gives quick feedback that the starting date has been chosen correctly.



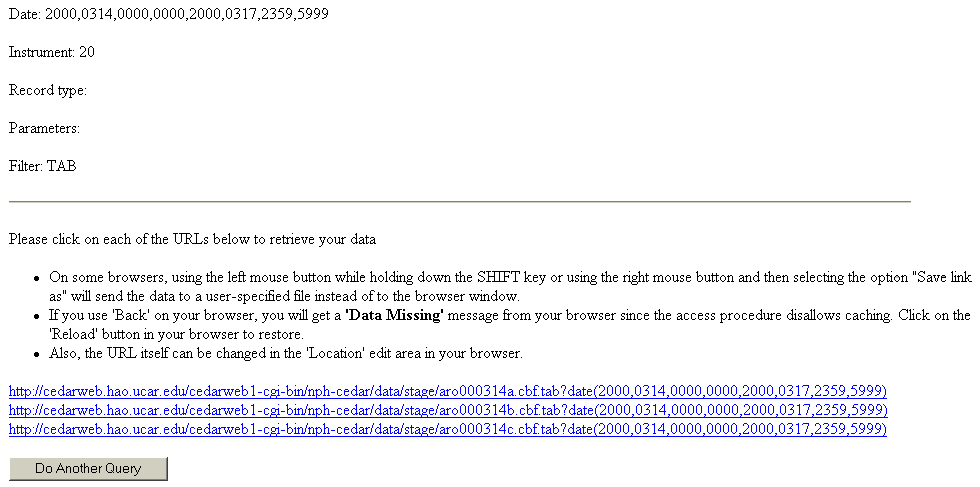
*Figure 10: The Interactive Calendar (Range)*

Instead of providing the same functionality to choose an ending date, this screen will let the user choose a date range they would like to study. Since the previous month calendar displayed the coverage inside the chosen month, hopefully the user will know how many days they would like to add to their date range to contain all the required data. While not the best user interface, this screen is a quick solution that prevents the user form having to choose a year/month/day all over again.



*Figure 11: Finishing the Query with the “Go” Button*

Now that the query is complete, the user should click the “Go” button to finalize their selection. The most important item left to choose is the format and type that the data will be returned. DODS supports many different styles of output formats, and many others can be added as filters in the future. In addition, DODS can return other information about the file, such as an “INFO” file with statistics about the data set. Or, the interface can return internal DODS objects. Finally, the user can always query the DODS server for the “HELP” help file or the current “VER” version of the server.



*Figure 12: Getting the Data Files*

The final screen shows the user a fully qualified URL that will call the DODS server. Please note that unlike the CEDAR version 1.0, no data has actually been created at this point. Instead of performing an ftp retrieval, this URL will send the chosen parameters to the DODS server, which will then create the data files and send them as an http request.

Users can right-click on the URL to save the data set as a file. Since the data files can be quite large, this prevents the data from being displayed in the browser window, saving processor speed and memory. Since the parameters are displayed again at the top of the screen, the user can be sure that they will get the data that they want.

# iii. Data Output

As described previously, DODS supports many different types of output data formats: DODS binary, DODS ASCII, and a tab-delimited flat file. The tab-delimited format is popular in the scientific community, since many mathematical analysis packages can import data in that format. However, a flat file disconnects the data from the metadata, so it is often awkward to use.

# VII. Future Directions

# i. Metadata

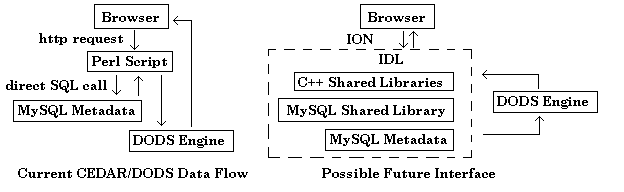
There exists a standard for storing metadata: the ANSI/NISO standard Z39.50, which describes a communication protocol for information retrieval (Mateescu). However, the standard, developed in the 1980’s, only specifies how the data should be transferred in a “stateful connection”, and doesn’t enforce how the metadata is stored. Newer W3C standards, such Resource Description Framework (RDF), and Extensible Markup Language (XML), are promising, and the DODS project hopes to incorporate some of these ideas in the future. The object-oriented and encapsulated nature of the DODS output makes it seem ideal to be translated into XML. Similar work with meteorological data (Neith) and aerospace (Crichton) have been very successful.

One current problem is that the DODS server is heavily dependent on the MySQL metadata. In the future, we would like to offer a service where the metadata itself could be queried by DODS. For example, if the metadata was in NetCDF format, the DODS server could ask questions about what files are in the database or what parameters can be used for subqueries. That way, all the data doesn’t have to located in the same data site. Once the user builds a data query, the URL could be pointing to a DODS server at another site. We still need to decide details on how to support distributed metadata, but this idea would be extremely useful for the DODS project.

# iii. Supported Data Input and Output Formats

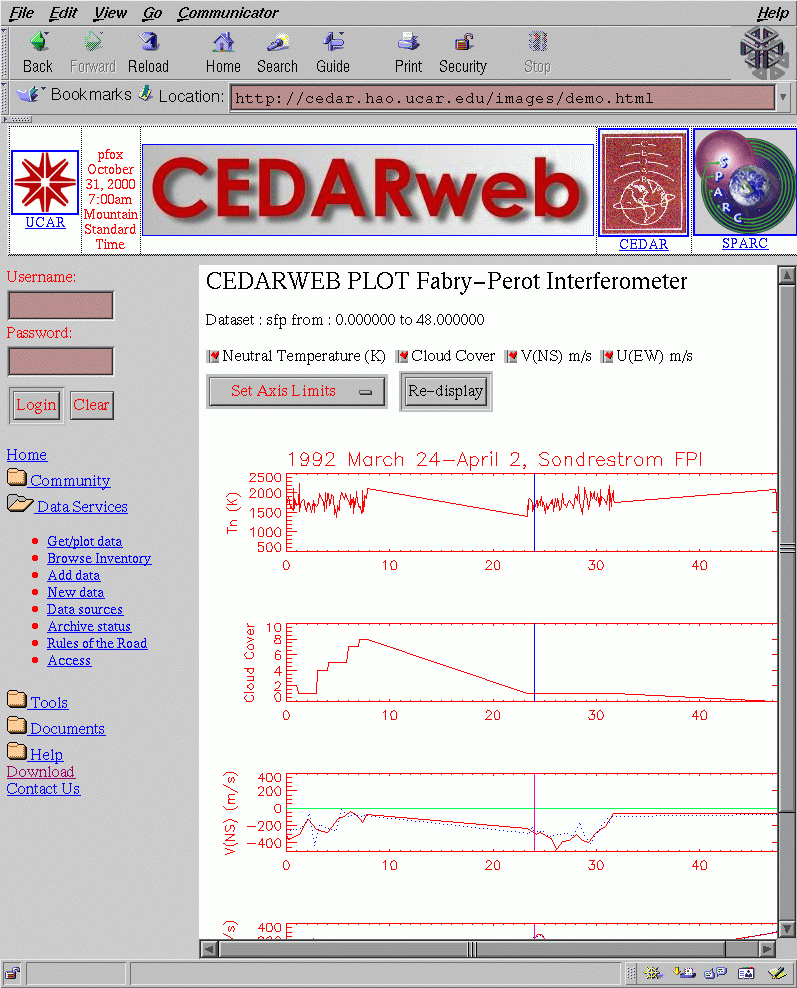
Some work has been completed on serving additional data formats, such as GRIB, CDF (“common data format” often used in space sciences), TIF/GIF, PNG, and general SQL. The nice thing about the structure of the DODS server is that the work needed to create the output filter doesn’t need to be accomplished at HAO/NCAR. Instead, each data site that wants to use DODS can install it on their own servers, download the documentation, and write the filter themselves. Hopefully, this approach will expand the amount and different types of data that will be available from DODS.

Also, we would like users to be able to use DODS data from inside any desired interface. Since the metadata interface uses standard URL statements to pass to the DODS server, there is no reason that other programs couldn’t encapsulate those URL strings. For example, we have recently created a demo of a CEDARweb 3.0 interface that uses IDL and ION from Research Systems to interactively process and plot data that is returned from a DODS server in real-time, with color charts and graphs.



*Figure 13: Possible Future Metadata Catalog Interaction*

IDL is a mathematical language similar to MATLAB or Mathmatica, and is quickly growing to become a popular tool for scientific analysis. In the new data flow shown above, the user will be able to submit data queries as usual, but the data will be returned to IDL instead of the browser window. This allows IDL to process and manipulate the data before it is dumped to a data file or directly to the browser. This direction is supported by Jose Garcia’s recent advances with “distributed DODS”. He has created a unified C++ library to submit DODS calls for a future IDL interface. This library integrates well with the existing C++ shared library that handles the MySQL queries; in fact, the two libraries share a lot of the same code. At the other end, David Helwig is writing IDL code to display the data. This allows us to create some interesting demos for the CEDAR community.



*Figure 14: Screen Snapshot of CEDARweb 3.0 Demo*

This demo screen shows some of the power of the IDL language. Here, the DODS URL was built using an interface that is very similar to the existing Perl script. However, IDL, not Perl, submitted the query and used the returned DDS results to build a structure to hold the DODS ASCII data. A second query filled the structure, and then other routines processed and sampled the data. The snapshot above uses part of the data to create a line plot that can be modified and expanded in “real time”.

# iv. POGO

The Partnership for Observation of the Global Oceans (POGO) is a newly formed major combined effort of oceanographic institutions from fourteen countries. They hope to use DODS to help collect and distribute data among the member organizations. At the second POGO meeting November 28 in Sao Paulo, permission was granted by the steering committee to investigate DODS as a feasible data delivery platform. Hopefully, this pilot project will create interest in DODS for the oceanographic community.

# VIII. Criticisms of the DODS Project

# i. Alternate Architectures and Relational Databases

We have been contacted by several other institutions that would like to share or “mirror” the CEDAR database at their site. However, most of these proposals involve importing our data into their relational database, or some other method of modifying the underlying data format. Since the CEDAR data is stored in an archaic CRAY-blocked format that relies on the legth of the following data segments, importing the data into a relational database would be a lot of work. In addition, there are certain handling and security concerns that make it difficult for another institution to serve CEDAR data.

At the same time, there might be strong advantages to porting the data into a more reliable and structured data format. Relational databases have been studied and profiled for years and are well understood. For example, backing up and mirroring relational datasets is a lot easier than moving entire CEDAR datasets. Creating a relational database for CEDAR data would probably lead to a substantial speed increase, since DODS needs to parse through an entire dataset to get to the final record. Finally, many third-party tools available that integrate automatically with relational databases, and our users easily use other programs they already own that support SQL queries to external databases. For example, many users in the CEDAR community have asked how to use CEDAR datasets inside of Microsoft Office products such as Excel.

However, porting the CEDAR database over to a relational (or object-oriented) database format would be a difficult decision that would be made unilaterally. There is a CEDAR steering committee currently made up of seventeen members from scientific institutions throughout the world. They meet twice a year to discuss the direction and development of the data. Before a large change in the data format could be performed, over one hudred data providers would have to noftified that they might need to change their output format. Then, every CEDAR database user would need to be notified, and a large amount of infrastructural code would need to be changed at NCAR and other data serving locations. Such a data translation *could* happen, but it would take a lot of foresight and planning to occur.

# ii. The Open Madrigal Initiative

CEDAR initially started out at the MIT Haystack Observatory, home of the Millstone Hill Incoherent Scatter Radar. As the Millstone Hill dataset grew, they combined their data holdings with other institutions including the ESICAT radars in Norway, and served the unified data as “Madrigal”, which was the start of the CEDAR project.

The Madrigal database is still very active today, and can be reached at

<http://www.haystack.edu/madrigal/>. Recently, they have implemented data access through the web in a method that is very similar to the CEDAR project. Users can browse the existing data holdings on-line, and dynamically create a query that is processed by a data server. Other institutions in the CEDAR community also have similar methods to extract scientific data, but it is the fact that there are some many different ways to query scientific data sets that is the problem. Only when several datasets can be combined regardless of where they are located geographically will thy truly be useful. Users shouldn’t have to learn a unique interface every time they want to retrieve data.

# iii. The DODS Portal

Of course, merely building a system to deliver scientific data doesn’t mean that institutions will use it. Popularizing and promoting the DODS architecture has proven to be more difficult that initially thought. To use a DODS engine at other university, the exact URL must be known. This has lead to many researchers trading long complex URLs with each other in order to notify other people that their DODS server is up and running. Instead, we would like to create a DODS “portal” similar to the popular portals and search engines on the Internet. This portal would list all the different DODS servers that exist and what kind of data they serve. Ideally, this list would be maintained automatically without the need for a human being to enter new data. A portal should make it easier to find data sources, and not add an additional level of abstraction. As of early 2001, design is just starting on a design for the portal, and will reflect the recent ideas for a distributed data catalog and dynamically combined data queries.

# IX. Conclusion

DODS is a very promising solution to the problem of delivering data to users, even though the information may be in several different formats. The way DODS uses encapsulation and flexibility of output formats means that the users don’t need to know about the state of the original database. This will make it easier to share data, even for obscure or seldom-used scientific databases (Domenico). The storage format of data is often specific to a particular system, making it difficult to view or combine several datasets even though, as Pursch points out, combining data sets is often a key requirement of global-scale earth science (Pursch). The major success of the DODS project will come when researchers can query several different databases that are scattered geographically, and still be able to retrieve the data they want in the correct usable format they desire. This “cross-pollination” of data sets will hopefully allow many new types of oceanographic research that would be difficult or impossible to perform today.

# VIII. Acknowledgements

This paper reflects work performed by the author at the University Corporation for Atmospheric Research in Boulder, Colorado from summer 1998 to spring 2000. The position was made possible by a grant from the National Science Foundation. Thanks goes to Peter Fox, Chief Computational Scientist of the High Altititude Observatory at NCAR, and Jose Garcia, lead programmer for the HAO DODS effort.

# IX. Statistics

Word Count: 6549

# References

## Papers and Talks

Adida, Ben "Why Not MySQL?", OpenACS Project white paper, Washington D.C., 2000

Arderiu-Ribera, Eva, et. al. "GIOD: Globally Interconnected Object Databases", Caltech, 2000

Chierici, Melissa "On the fluxes of dissolved inorganic carbon in the Arctic Mediterranean Sea", Analytical and Marine Chemistry, Göteborg University, Sweden 1998

Cho, Koheia and Toshibumi Sakata "Satellite Data Fusion for Oceanography" Research & Information Center, Tokai University, Tokyo, Japan, 1993

Clark, David M., National Geophysical Data Center, Scientific Assistant to the Director (Personal interview)

Cornillon, P., G. Flierl, J. Gallagher and G. Milkowski "Report on the First Workshop for the Distributed Oceanographic Data System", The University of Rhode Island, Graduate school of Oceanography, 1993

Crichton, Daniel, J. Steven Hughes, Jason Hyon, and Sean Kelly, "A Framework for Science Data Access Using XML", Jet Propulsion Laboratory, Pasadena California, 2000

Domenico, Ben, Sally Bates, and Dave Fulker "Internet Data Distribution (IDD)", Unidata Program Center 1, Boulder, Colorado, 1993

Elms, Joe, Scott Woodruff, and Steve Worley "Preparing Historical Marine Data for COADS: Recently Digitized Ship Logbook Records and Global Telecommunication System (GTS) Archives"

Fox, P. A., Jose Garcia, Patrick Kellogg, 2000 “The HAO Data Service: Experience in Interdisciplinary Data Delivery” (poster), CODATA 2000 Workshop, US National Academy, 2000

Gallagher, James, and George Milkowski "Data Transport Within The Distributed Oceanographic Data System", University of Rhode Island, Fourth International World Wide Web Conference, 1995

Gower, J.F.R., and J.R. Apel (editors) “Opportunities and Problems in Satellite Measurements of the Sea”, Unecso Technical Papers in Marine Science, Unesco, 1986

Idso, C. D. and K. E. Idso “Carbon Dioxide and Global Warming: Where We Stand on the Issue”, Center for the Study of Carbon Dioxide and Global Change, 2000

Lane, Dr. Neal “'Tis Better to Give and Receive: Benefits of Data Sharing for Research and Education”, Plenary Address, Conference on Scientific and Technical Data Exchange and Integration, National Institutes of Health, 1997

Lingsch, Stephen, and C. Robinson "Processing, Presentation, and data basing of Acoustic Imagery", oceans '95 MTS/IEEE Conference proceedings, 1995

Lingsch, Stephen, Kenneth Grossman, William John, and Sharon Mesick "The Integration of Tools and Data Bases for Geophysical and Oceanographic Applications in a GIS Environment" Naval Oceanographic Office, Stennis Space Center, Mississippi, 1996

Lorenc, A.C. "Data Analysis & Assimilation and Observation Processing", Second WMO Symposium on Assimilation of Observations in Meteorology and Oceanography, Tokyo, Japan, 1995

Mateescu, Ion, Lucy Nowell, and Leigh Williams “Advanced Visualization of Scientific Metadata”, Center for International Earth Science Information Network, Columbia University,

Moen, William E. "The Development of ANSI/NISO Z39.50: A Case Study in Standards Evolution", Syracuse University, School of Information Studies, 1998

Muntz, R., E. Mesrobian, and C. R. Mechoso, "Integrating Data Analysis, Visualization, and Data Management in a Heterogeneous Distributed Environment", Information Systems Newsletter vol. 20, No. 2, 1995

Neith, M. and O. E. Kiselyov "Disseminating and Storing of Meteorological Data with Extensible Markup Language (XML)" 15th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, American Meteorological Society, 1999.

NISO, “ANSI/NISO Z39.50-1995, Information Retrieval (Z39.50): Application Service Definition and Protocol Specification”, National Information Standards Organization, 1995

Olson, R.J. and R.A. McCord "Integrating Data from the Internet: Are Metadata All We Need?", Second National Conference on Scientific and Technical Data, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 2000

Paterson, Gordon, et.al. “Where are all the data?” Oceanography, Vol. 13, No. 3, 2000

Rew, R. K., D. P. and Davis "NetCDF: An Interface for Scientific Data Access", IEEE Computer Graphics and Applications, Vol.10, No.4, 1990

Rew, R. K. and G. P. Davis "The Unidata NetCDF: Software for Scientific Data Access", Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, American Meteorology Society, 1990

Rosenblum, Lawrence J. "Visualizing Oceanographic Data" IEEE Computer Graphics and Applications”, Vol. 9, No.3, 1989

Springmeyer, Rebecca, Nancy Werner, and Jeffery Long "Mining Scientific Data Archives through Metadata Generation", Lawrence Livermore National Laboratory, First IEEE Metadata Conference, Silver Spring, Maryland, 1996

Strebel, D., B. Meeson, and J. Frithesen “Metadata Standards and Concepts for Interdisciplinary Scientific Systems “, Position papers from IEEE Metadata Workshop, Washington D.C., 1994,

## Thompson, D.A. and J. S. Best "The Future of Magnetic Data Storage Technology", IBM, 1999

Treinish, Lloyd A. "Interactive Archives for Scientific Data", IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1993

Voorhees, L. D., P. Kanciruk, B. T. Rhyne, and S. E. Attenberger "Mercury: Managing Distributed Multidisciplinary Scientific Data", Second National Conference on Scientific and Technical Data, Oak Ridge National Laboratory, Oak Ridge, Tennesee, 2000

## Collected Journals, Books, and Conference Proceedings

Acoustical Society of America, et. al. "Electronic Engineering in Oceanography", Churchill College, Cambridge, UK, 1994.

Barnes, H. "Apparatus and Methods of Oceanography" Interscience Publishers, New York, 1959.

Capurro, Luis R. A. "Oceanography for Practicing Engineers", Barnes & Noble, Inc., New York, 1970.

Electronics Division of the Institution of Electrical Engineers "Seventh International Conference on Electronic Engineering in Oceanography", Southampton Oceanography Centre, UK, 1997.

Emery, William J., and Richard E. Thomson "Data Analysis Methods in Physical Oceanography", Pergamon, UK, 1998

Gaul, Roy D., et. al. (editors) "Marine Sciences Instrumentation: Volume 1" Plenum Press, New York, 1962.

GOFS, "Global Ocean Flux Study: Proceedings of a workshop", Woods Hole Study Center, National Academy Press, Washington, D.C., 1984

IEEE "Ocean Electronics Symposium", Western Periodicals Co., Honolulu, 1966.

Institution of Electronic and Radio Engineers "Electronic Engineering in Oceanography", University of Southampton, UK, 1966.

Institution of Electronic and Radio Engineers "Instrumentation in Oceanography", University College of North Wales, Bangor, 1975.

Lawrence, L. George "Electronics in Oceanography" Howard W. Sams & Co., Inc., Indianapolis, 1967.

Malanotte-Rizzoli, P. (editor) "Modern Approaches to Data Assimilation in Ocean Modeling", Elsevier Oceanography Series, UK, 1996

Marine Technology Society, Oceanic Engineering Society, IEEE "Proceedings OCEANS '83", IEEE, San Francisco, 1983.

MEDEA "Scientific Utility of Naval Environmental Data" Special Task Force Report, 1995

National Research Council "Energy Systems of Extended Endurance in the 1-100 Kilowatt Range for Undersea Applications", National Academy of Sciences, Washington, D.C., 1968.

Naval Ocean System Center "Oceanic Data Base Information Exchange Workshop", IEEE Computer Society, San Diego, 1977.

Office of Naval Research "Report of the Inter-American Conference on Marine Sciences", National Academy of Sciences, Washington, D.C., 1963.

## Webpages

CEDARweb Page

<http://cedarweb.hao.ucar.edu>

DODS Unidata Page

<http://www.unidata.ucar.edu/packages/dods/>

DODS HAO page

<http://dods.hao.ucar.edu>

**Appendix A: Sample DODS Output**

## DAS “DODS Attribute Service”

Attributes {

Data\_Descriptor\_for\_KINDAT\_2010\_KINST\_20 {

KINST {

String INST\_20 "Arecibo P.R. I.S. Radar 18.345 293.247 0. ARO";

}

JPAR\_0 {

String PARAMETER\_CODE\_60 "Integration time for these data 1. s";

}

JPAR\_1 {

String PARAMETER\_CODE\_115 "Altitude averaging interval 1. km";

}

JPAR\_2 {

String PARAMETER\_CODE\_116 "Additional increment to ht avgng intrvl 1.E-01 m";

}

JPAR\_3 {

String PARAMETER\_CODE\_130 "Mean azimuth angle (0=geog N,90=east) 1.E-02 deg";

}

JPAR\_4 {

String PARAMETER\_CODE\_140 "Elevation angle (0=horiz,90=vert) 1.E-02 deg";

}

JPAR\_5 {

String PARAMETER\_CODE\_535 "log10 (max Ne in m-3) 1.E-03 lg(m-3)";

}

JPAR\_6 {

String PARAMETER\_CODE\_540 "Height of maximum electron density 1. km";

}

MPAR\_0 {

String PARAMETER\_CODE\_110 "Altitude (height) 1. km";

}

MPAR\_1 {

String PARAMETER\_CODE\_111 "Additional increment to altitude 1.E-01 m";

}

MPAR\_2 {

String PARAMETER\_CODE\_520 "log10 (Ne in m-3) 1.E-03 lg(m-3)";

}

MPAR\_3 {

String ERROR\_IN\_PARAMETER\_CODE\_520 "Error log10 (Ne in m-3) 1.E-03 lg(m-3)";

}

MPAR\_4 {

String PARAMETER\_CODE\_550 "Ion temperature 1. K";

}

MPAR\_5 {

String ERROR\_IN\_PARAMETER\_CODE\_550 "Error in Ion temperature 1. K";

}

MPAR\_6 {

String PARAMETER\_CODE\_560 "Electron temperature 1. K";

}

MPAR\_7 {

String ERROR\_IN\_PARAMETER\_CODE\_560 "Error in Electron temperature 1. K";

}

MPAR\_8 {

String PARAMETER\_CODE\_580 "Line of sight ion velocity (pos = away) 1. m/s";

}

MPAR\_9 {

String ERROR\_IN\_PARAMETER\_CODE\_580 "Error in Line (pos = away) 1. m/s";

}

MPAR\_10 {

String PARAMETER\_CODE\_650 "Ion Composition - [HE+]/Ne 1.E-03 ";

}

MPAR\_11 {

String ERROR\_IN\_PARAMETER\_CODE\_650 "Error in Composition - [HE+]/Ne 1.E-03 ";

}

MPAR\_12 {

String PARAMETER\_CODE\_660 "Ion Composition - [H+]/Ne 1.E-03 ";

}

MPAR\_13 {

String ERROR\_IN\_PARAMETER\_CODE\_660 "Error in Composition - [H+]/Ne 1.E-03 ";

}

MPAR\_14 {

String PARAMETER\_CODE\_420 "Reduced-chi square of fit 1.E-03 ";

}

MPAR\_15 {

String PARAMETER\_CODE\_921 "Number of coefficients in analysis 1. ";

}

}

}

## DDS “DODS Descriptor Service”

Dataset {

Structure {

Structure {

Int16 KINST;

Int16 KINDAT;

UInt16 IBYRT;

UInt16 IBDTT;

UInt16 IBHMT;

UInt16 IBCST;

UInt16 IEYRT;

UInt16 IEDTT;

UInt16 IEHMT;

UInt16 IECST;

} prologue;

Structure {

Int16 inttms;

Int16 altav;

Int16 altavi;

Int16 azm;

Int16 elm;

Int16 nemaxl;

Int16 hmax;

} JPAR;

Structure {

Int16 gdalt[15];

Int16 gdalti[15];

Int16 nel[15];

Int16 error\_nel[15];

Int16 ti[15];

Int16 error\_ti[15];

Int16 te[15];

Int16 error\_te[15];

Int16 vo[15];

Int16 error\_vo[15];

Int16 phep[15];

Int16 error\_phep[15];

Int16 php[15];

Int16 error\_php[15];

Int16 chisq[15];

Int16 nc[15];

} MPAR;

} data\_record\_1;

} aro000314a;

## DODS Binary Output

Dataset {

Structure {

Structure {

Int16 KINST;

Int16 KINDAT;

UInt16 IBYRT;

UInt16 IBDTT;

UInt16 IBHMT;

UInt16 IBCST;

UInt16 IEYRT;

UInt16 IEDTT;

UInt16 IEHMT;

UInt16 IECST;

} prologue;

Structure {

Int16 inttms;

Int16 altav;

Int16 altavi;

Int16 azm;

Int16 elm;

Int16 nemaxl;

Int16 hmax;

} JPAR;

Structure {

Int16 gdalt[15];

Int16 gdalti[15];

Int16 nel[15];

Int16 error\_nel[15];

Int16 ti[15];

Int16 error\_ti[15];

Int16 te[15];

Int16 error\_te[15];

Int16 vo[15];

Int16 error\_vo[15];

Int16 phep[15];

Int16 error\_phep[15];

Int16 php[15];

Int16 error\_php[15];

Int16 chisq[15];

Int16 nc[15];

} MPAR;

} data\_record\_1;

} aro000314a;

Data:

u8®Ê”®Êê&uqÿÿ€s‘¯¾ÍÜëú6Tˆˆˆˆ... (binary data follows)

## DODS ASCII Output

Dataset {

Structure {

Structure {

Int16 KINST = 20;

Int16 KINDAT = 2010;

UInt16 IBYRT = 2000;

UInt16 IBDTT = 314;

UInt16 IBHMT = 1547;

UInt16 IBCST = 2700;

UInt16 IEYRT = 2000;

UInt16 IEDTT = 314;

UInt16 IEHMT = 1547;

UInt16 IECST = 2700;

} prologue

Structure {

Int16 inttms = 12;

Int16 altav = 37;

Int16 altavi = 915;

Int16 azm = -17999;

Int16 elm = 7499;

Int16 nemaxl = 12152;

Int16 hmax = 331;

} JPAR

Structure {

Int16 gdalt[15] = {144, 181, 219, 256, 293, 330, 367, 404, 441, 478, 515, 552, 589, 627, 664};

Int16 gdalti[15] = {8888, 9803, 718, 1634, 2549, 3464, 4380, 5295, 6210, 7126, 8041, 8956, 9873, 787, 1703};

Int16 nel[15] = {11294, 11410, 11715, 11937, 12092, 12152, 12112, 12027, 11931, 11808, 11688, 11559, 11451, 11351, 11251};

Int16 error\_nel[15] = {10518, 10326, 10528, 10438, 10611, 10587, 10591, 10511, 10424, 10320, 10236, 10145, 10083, 10025, 9972};

Int16 ti[15] = {821, 1093, 1096, 1210, 1242, 1252, 1287, 1327, 1308, 1353, 1392, 1469, 1488, 1522, 1564};

Int16 error\_ti[15] = {107, 57, 44, 28, 29, 23, 28, 39, 39, 44, 52, 64, 78, 95, 113};

Int16 te[15] = {934, 1398, 2074, 2001, 1579, 1457, 1431, 1406, 1458, 1448, 1544, 1658, 1809, 1964, 2082};

Int16 error\_te[15] = {140, 118, 90, 40, 43, 36, 39, 40, 41, 43, 46, 52, 59, 71, 89};

Int16 vo[15] = {-21, -9, 7, -4, -5, 0, -3, 0, 0, 4, -7, 6, 0, 5, 19};

Int16 error\_vo[15] = {7, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9};

Int16 phep[15] = {0, 0, 0, 0, 0, 0, 0, -5, -6, -7, -3, -11, 1, 9, 15};

Int16 error\_phep[15] = {-32766, -32766, -32766, -32766, -32766, -32766, -32766, 12, 13, 14, 15, 18, 21, 26, 32};

Int16 php[15] = {0, 0, 0, 0, 0, 0, 0, 3, 3, 1, 0, 1, -2, -3, -3};

Int16 error\_php[15] = {-32766, -32766, -32766, -32766, -32766, -32766, 2, 4, 4, 5, 6, 7, 8, 10, 12};

Int16 chisq[15] = {0, 0, 4, 463, 418, 604, 338, 826, 852, 720, 799, 510, 397, 292, 470};

Int16 nc[15] = {3, 3, 3, 3, 3, 3, 4, 5, 5, 5, 5, 5, 5, 5, 5};

} MPAR

} data\_record\_1

} aro000314a;

DODS INFO file

CATALOG AND HEADER RECORD CORRESPONDING TO: aro000314a

LTOT KREC KINST KINDAT IBYRT IBDTT IBHMT IBCST IEYRT IEDTT IEHMT IECST LPROL JPAR MPAR NROW

6680 3002 20 2010 2000 314 1547 2700 2000 316 1348 0 16 7 16 N/A

COLS:1-8 9-16 17-24 25-64 65-72 73-80

CKEYWORD DRWDNO VALUE DESCRIPTION UNITS

C DRWDNO is data record word number

KRECH 3002 header record, version 2

KINST 3 20 Arecibo P.R. I.S. Radar

KINDAT 4 2010 KINDAT definition follows:

C This describes data analyzed using the ASP code wdfit(95 version)

C developed by Mike Sulzer, and reformulated to the CEDAR Data Base

C format by Qihou Zhou using ao2cedarF (June 95 version).

C

C The analysis solves between 3 and 5 parameters simultaneously.

C Code 921 indicates the number of free parameters assumed. If

C it is 5, then Ne, Ti, Te, H+/Ne and He+/Ne are solved simultaneously.

C (See next paragraph concerning limitations.) If there are 4 free

C parameters, then He+/Ne is assumed to be zero, and the others are

C fit. If there are 3 free parameters, then both He+/Ne and H+/Ne are

C assumed to be zero, and Ne, Ti and Te are solved. The line-of-sight

C velocity is found in a separate fit to the slope of the auto-correlation

C function (ACF).

C

C The seven frequency radar technique (ref. below) was intended for use in

C the F region with O+ and at most small amounts of H+. Large fractions of

C light ions are present at times however, and special techniques were

C developed for extracting the ion fractions. There are limitations:

C 1. Temperatures cannot be measured when [H+] is large. Extracting a

C fraction requires that a temperatures be supplied from lower

C altitudes using a constraint technique. This situation can be recognized

C looking at [H+] and the temperature errors, which normally increase

C with altitude, but stop increasing when constraints cut in.

C 2. [He+] is extremely noisy when [H+] is large and much averaging is

C required. Also the [He+] is biased when the values are small; bias is

C in the range -.02 to -.04. Small [He+] can be used only to determine

C and upper limit on a density. Large [He+] can give a density; check

C early solar max conditions, winter half of the year. Plot all data;

C use common sense and proper procedures! Stated fraction errors are

C statistical only.

C

C The goodness of fit (code 420) is a function of the chi square

C of the fit, and the number of free parameters, It is a probability

C (between 0 and 1) and is defined in Chapter 15, section 2 of Numerical

C Recipes (ref. below). In this file, it is multiplied by 1000.

C Very small values indicate that the fit is not good.

C Anything above 0.1 (100 in code 420) is almost certainly good.

C This analysis uses the actual sigmas of the data points, which have been

C determined as accurately as possible. Note that if actual sigmas are not

C used, but mere weighting factors, goodness of fit is assumed, and the

C accuracy of the errors depends on this assumption.

C

C Typical goodness of fits are normal above the F peak, but below normal

C below the peak because of the distortion introduced by the length of the

C pulse required to get sufficient frequency resolution. The reduced parameters

C at the lowest altitude (144 km) may not be reliable except the line of

C sight velocity. The data at other altitudes below the F-region peak are

C usually useful, although the errors are larger than expected, and so have

C been increased accordingly in this altitude range. The increased error bars

C may not be accurate for all conditions.

C

C In the lower altitudes, a molecular ion model is assumed. The model

C was determined by looking for consistency in the other parameters.

C The assumed molecular ion composition is as

C Alt(km) 145 182 219 256 > 293.

C [M. ion]/[Ne] 1. 0.3 0.2 0.07 0.

C The molecular ion mass is assumed to be 30 amu.

C

C There are several corrections that need to be applied to the data.

C One of these is the chirp correction for the velocities. This is

C done by measuring the transmitted spectrum and has a typical value

C of 10 m/s and varies according to many factors, including the current

C size of the capacitor bank, the transmitter high voltage level and

C tuning, and temperature.

C

C Satellites give rise to spikes in the data, but are relatively easy

C to remove because the Arecibo signal is strong and 7 frequencies

C are transmitted at once so large deviations from a repeating pattern

C can be detected. There will be some contamination of the data from

C satellites, but typically ground based radio communication is a greater

C problem. Bad records are eliminated when possible but some outliners

C remain, and so data time series should always be plotted and examined

C carefully.

C

C The electron densities are calibrated with good ionosonde data in post

C processing. The ionosonde values for the run are typed into the computer

C (will be automatic in the future) and are compared to the radar values

C determined by a cubic spline interpolation to the peak of each profile.

C A scale factor is derived by trial and error which gives the correct Ne

C near the beginning of the run. All the data from the run are fit using

C this scale factor. Comparisons are made between the peak plasma frequency

C derived from the fitted Ne and the ionosonde readings throughout the run.

C Corrections are made to the fitted Ne. If these corrections are large

C enough and Ne low enough so that there is a significant Debye length

C correction, scale factors are derived for the entire run and the data are

C refit. The peak electron densities (code 535) and hmax (code 540) in

C the 1D parameter field are determined from cubic spline interpolations

C to the calibrated [Ne] profiles.

C

C Additional information on the Arecibo IS radar and on the technique

C can be found in the following papers and references therein:

C

C Sulzer, M. P., A phase modulation technique for a sevenfold statistical

C improvement in incoherent scatter data-taking, Radio Sci., vol 21,

C No. 4, pp 737-744, 1986

C

C Press, William, et al. Numerical Recipes in C (or FORTRAN),

C first or second edition, Cambridge University Press, 1989.

C

C Erickson, P. and W. Swartz, Mid-latitude incoherent scatter observations

C of helium and hydrogen atoms, GRL Vol 21, No 24, pp 2,745-48 Dec. 1, 1994,

C (constraints)

C

C Sulzer, M. P., A document describing fitting this data in detail is in

C preparation at this time. Contact below.

C

IBYRT 2000 Beginning year for this data set

IBDTT 314 Beginning month and day

IBHMT 1547 Beginning UT hour and minute

IBCST 2700 Beginning centisecond

IEYRT 2000 Ending year for this data set

IEDTT 316 Ending month and day

IEHMT 1348 Ending UT hour and minute

IECST 0 Ending centisecond

LPROL 13 16 Data record prologue length

JPAR 14 7 Number of single-valued parameters

MPAR 15 16 Number of multiple-valued parameters

NROW 16 15 Number of rows of multiple-valued parameters

C NROW may vary from the above value in data records

KODS( 1) 17 60 Integration time for these data 1.E+00 s

KODS( 2) 18 115 Altitude averaging interval 1.E+00 km

KODS( 3) 19 116 Additional increment to ht avgng intrvl 1.E-01 m

KODS( 4) 20 130 Mean azimuth angle (0=geog N,90=east) 1.E-02 deg

KODS( 5) 21 140 Elevation angle (0=horizontal,90=vert) 1.E-02 deg

KODS( 6) 22 535 log10 (max Ne in m-3) 1.E-03 lg(m-3)

KODS( 7) 23 540 Height of maximum electron density 1.E+00 km

KODM( 1) 31 110 Altitude (height) 1.E+00 km

KODM( 2) 32 111 Additional increment to altitude 1.E-01 m

KODM( 3) 33 520 log10 (Ne in m-3) 1.E-03 lg(m-3)

KODM( 4) 34 -520 log10 (Ne in m-3) error 1.E-03 lg(m-3)

KODM( 5) 35 550 Ion temperature 1.E+00 K

KODM( 6) 36 -550 Ion temperature error 1.E+00 K

KODM( 7) 37 560 Electron temperature 1.E+00 K

KODM( 8) 38 -560 Electron temperature error 1.E+00 K

KODM( 9) 39 580 Line of sight ion velocity (pos = away) 1.E+00 m/s

KODM(10) 40 -580 Error in parameter 580 1.E+00 m/s

KODM(11) 41 650 Ion Composition - [HE+]/Ne 1.E-03

KODM(12) 42 -650 Ion Composition - [HE+]/Ne error 1.E-03

KODM(13) 43 660 Ion Composition - [H+]/Ne 1.E-03

KODM(14) 44 -660 Ion Composition - [H+]/Ne error 1.E-03

KODM(15) 45 420 Reduced-chi square of fit 1.E-03

KODM(16) 46 921 Number of coefficients in analysis 1.E+00

C Missing parameters are assigned a value of -32767

C Errors of assumed parameters are assigned a value of -32766

C...............................................................................

C

C Further questions, concerns, and suggestions can be addressed to:

C

C Dr. Michael P. Sulzer or Dr. Qihou Zhou

C Arecibo Observatory

C HC3 Box 53995

C Arecibo, PR 00612

C

C Phone: (787) 878-2612

C FAX: (787) 878-1861

C email: sulzer@naic.edu zhou@naic.edu

C

CANALYST Mike Sulzer and Qihou Zhou

CARDATE May 2000.

C...............................................................................

## DODS HELP file

You are reading the help page for this server.

A URL is handled thus:

http://machine/cgi-bin/nph-cedar/file.cbf.dods?val

^^^^^^^^^ ^^^^^^^^ ^^^^ ^^^

| | | \

| | \ - The constraint expression.

| | - action: What kind of data stream is requested.

| \

\ - File to be open by the server.

- Server program

This server recognizes the following actions:

1.- dds: request the data descriptor structure.

2.- das: request the data attributes.

3.- dods: request for the data stream, this output is an octec binary stream which requires the analysis by the client dods library.

4.- asc: request for the data stream, this output is an ascii representation of the data.

5.- info: extra documentation related to the data set. info is equivalent to the header records and catalog records.

6.- help: prints this help.

7.- tab: request for the data stream as a tab base file

8.- ver: prints the version for this server.

9.- html: generates an HTML page that represents the dataset.

When the access to data is restricted, users must first get authenticated into the web server. In some cases

this access is performed through a proxy which is supplying the login for the authentication. For this cases,

it is possible for the proxy to sign on behalf of whom is requesting the data.

This is done by using the server side function "signature (string s)"

where the variable 's' is the login for the person using the proxy. The variable s can not contain spaces

and in general it must follow of the rules for valid logins in a computer system.

Constraint expressions:

To create a constraint expression this server offers the following server side functions:

a.- date(BEGIN\_year,BEGIN\_date,BEGIN\_hour\_min,BEGIN\_centisecond,END\_year,END\_date,END\_hour\_min,END\_centisecond)

The 8 parameters expose above are REQUIRED. This function is used to let the server know that any data

record whose date is in the range [BEGIN, END] must be returned.

b.- record\_type(kinst(0)/kindat(0),kinst(1)/kindat(1),...,kinst(n)/kindat(n)

This function can take n parameters.

The parameters are a kinst-kindat combination which defines a record type.

Repeated record types are not accepted.

A kinst value is a unique positive integer that defines an instrument, a kindat values is a positive integer

defined by the institution that manages the instrument.

c.- parameters(p1,p2,p3,...,pn)

This function can take n parameters where each parameter is a "CEDAR define parameter" (i.e 54 Magnetic local time)

A parameter is an integer number.

Please notice that is possible to request this function like this: parameters(). In this case no data is

returned, just the prologues for the data records.

Constraint expressions are only analized when you request either dds,asc,dods,info or tab.

For other actions the constraint expression is ignored, this allow you, for example, to modified the "ce"

while reading the das which contains all the information you need to fill record\_type(...) or parameters(...)

Server side functions are "independent" among then, this means that you may request all of then or none of

then or only one or any couple combination. In case that more than one server side functions are present each

data record should satisfy ALL of the constraints in order to be returned.

For more information please contact Jose Garcia, jgarcia@ucar.edu

**Appendix B: Metadata Layout for MySQL**

tbl\_date

+---------+---------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------+------+-----+---------+----------------+

--------| DATE\_ID | int(10) | | UNI | 0 | auto\_increment |

| | YEAR | int(5) | | PRI | 0 | |

| | MONTH | int(5) | | PRI | 0 | |

| | DAY | int(5) | | PRI | 0 | |

| +---------+---------+------+-----+---------+----------------+

| tbl\_date\_in\_file

| +---------+---------+------+-----+---------+-------+

| | Field | Type | Null | Key | Default | Extra |

| +---------+---------+------+-----+---------+-------+

--------| DATE\_ID | int(10) | | PRI | 0 | |

--------| FILE\_ID | int(10) | | PRI | 0 | |

| +---------+---------+------+-----+---------+-------+

| tbl\_cedar\_file

| +-----------+-----------+------+-----+---------+----------------+

| | Field | Type | Null | Key | Default | Extra |

| +-----------+-----------+------+-----+---------+----------------+

--------| FILE\_ID | int(10) | | UNI | 0 | auto\_increment |

| | FILE\_NAME | char(255) | | PRI | | |

| | FILE\_SIZE | int(10) | | | 0 | |

| | FILE\_MARK | char(50) | | | | |

| | NRECORDS | int(10) | | | 0 | |

| +-----------+-----------+------+-----+---------+----------------+

| tbl\_file\_info

| +----------------+---------+------+-----+---------+-------+

| | Field | Type | Null | Key | Default | Extra |

| +----------------+---------+------+-----+---------+-------+

--------| FILE\_ID | int(10) | | PRI | 0 | |

--------| RECORD\_TYPE\_ID | int(10) | | PRI | 0 | |

| +----------------+---------+------+-----+---------+-------+

| tbl\_record\_type

| +----------------+---------+------+-----+---------+----------------+

| | Field | Type | Null | Key | Default | Extra |

| +----------------+---------+------+-----+---------+----------------+

--------| RECORD\_TYPE\_ID | int(10) | | UNI | 0 | auto\_increment |

| | KINDAT | int(10) | | PRI | 0 | |

| ---| KINST | int(10) | | PRI | 0 | |

| | +----------------+---------+------+-----+---------+----------------+

| | tbl\_record\_info

| | +----------------+---------+------+-----+---------+-------+

| | | Field | Type | Null | Key | Default | Extra |

| | +----------------+---------+------+-----+---------+-------+

--------| RECORD\_TYPE\_ID | int(10) | | PRI | 0 | |

--------| PARAMETER\_ID | int(10) | | PRI | 0 | |

| | +----------------+---------+------+-----+---------+-------+

| | tbl\_parameter\_code

| | +---------------+----------+------+-----+--------------+-------+

| | | Field | Type | Null | Key | Default | Extra |

| | +---------------+----------+------+-----+--------------+-------+

--------| PARAMETER\_ID | int(10) | | PRI | 0 | |

| | LONG\_NAME | char(50) | YES | | UNDEFINED | |

| | SHORT\_NAME | char(50) | YES | | UNDEFINED | |

| | MADRIGAL\_NAME | char(50) | YES | | UNDEFINED | |

| | UNITS | char(50) | YES | | UNDEFINED | |

| | SCALE | char(50) | YES | | UNDEFINED | |

| +---------------+----------+------+-----+--------------+-------+

|

| tbl\_instrument

| +-----------+-------------+------+-----+-------------------+--------+

| | Field | Type | Null | Key | Default | Extra |

| +-----------+-------------+------+-----+-------------------+--------+

---| KINST | int(10) | | PRI | 0 | |

| INST\_NAME | char(40) | YES | | UNDEFINED | |

| LATITUDE | float(10,8) | YES | | -500.00000000 | |

| LONGITUDE | float(10,8) | YES | | -500.00000000 | |

| ALT | float(10,8) | YES | | -500.00000000 | |

| PREFIX | char(3) | | | UND | |

+-----------+-------------+------+-----+-------------------+--------+

**Appendix C: CEDARweb code**

Cedarweb.pl

#!/opt/local/bin/perl

#This is a perl script for

#the Cedarweb interface

#Patrick Kellogg pkellogg@hao.ucar.edu x1544

#Version 2.0

#Oct 19, 1999

#Essential lines for security

use CGI;

$ENV{PATH} = '/bin:/usr/bin:/cedar/e/mysql/bin';

$ENV{IFS} = "" if $ENV{IFS} ne "";

#Include the data routines

use Time::ParseDate;

use Time::CTime;

#Global variables

#

#main\_site: cedarweb.hao.ucar.edu, the main html location

$main\_site = "http://cedarweb.hao.ucar.edu";

#

#cgi\_site: http://www.hao.ucar.edu/internal-cgi-bin/ where all cgi-bin files are located

$cgi\_site = "http://cedarweb.hao.ucar.edu/cedarweb-cgi-bin";

#

#hao\_site: http://www.hao.ucar.edu the main HAO web site

$hao\_site = "http://www.hao.ucar.edu";

#

#image\_site: http://cedarweb.hao.ucar.edu/images where the images are

$image\_site = "http://cedarweb.hao.ucar.edu/images";

#

#mysql\_home: home directory for MySQL binary files.

$mysql\_home="/cedar/e/mysql/bin/";

#

#mysql\_client: name of the MySQL client program.

$mysql\_client=$mysql\_home."mysql";

#

#user: Defines the user ID to connect to mysqld.

$user="madrigal";

#

# pass: The password for the user which is getting connected to mysqld.

$pass="madrigal";

#

# host: Defines the host where mysqld is running.

$host="cedar";

#

# CEDARDB: Defines the MySQL databases to get connected.

$CEDARDB = "CEDAR\_CATALOG";

#

# beginning\_year" Sets the beginning year of data in the CEDAR database

$beginning\_year = 1950;

#

# final\_year: Sets the final year of data in the CEDAR database

$final\_year = 2000;

# Sey this variable to TRUE if you want to debug...

$DebugSQLClause = "FALSE";

#Find out who is logged in

$RemoteUser = $ENV{"REMOTE\_USER"};

#Create a new query object (see CGI.pl)

$query = new CGI;

#See if the user clicked "Clear"

$Clear = $query->param('Clear');

if ($Clear eq "") {

#Parse the fields

$Stage = $query->param('Stage');

$NewStage = $query->param('NewStage');

$StartYear = $query->param('StartYear');

$StartMonth = $query->param('StartMonth');

$StartDay = $query->param('StartDay');

$StartHour = $query->param('StartHour');

$StartMinute = $query->param('StartMinute');

$StartSecond = $query->param('StartSecond');

$EndYear = $query->param('EndYear');

$EndMonth = $query->param('EndMonth');

$EndDay = $query->param('EndDay');

$EndHour = $query->param('EndHour');

$EndMinute = $query->param('EndMinute');

$EndSecond = $query->param('EndSecond');

$DateRange = $query->param('DateRange');

$NewMonth = $query->param('NewMonth');

$NewDay = $query->param('NewDay');

$NewYear = $query->param('NewYear');

$Kinst = $query->param('Kinst');

$KinstSortBy = $query->param('KinstSortBy');

$KinstAscDesc = $query->param('KinstAscDesc');

$KinstShow = $query->param('KinstShow');

$Kindat = $query->param('Kindat');

$KindatSortBy = $query->param('KindatSortBy');

$KindatAscDesc = $query->param('KindatAscDesc');

$KindatShow = $query->param('KindatShow');

@Parameter = $query->param('Parameter');

$ParameterSortBy = $query->param('ParameterSortBy');

$ParameterAscDesc = $query->param('ParameterAscDesc');

$ParameterShow = $query->param('ParameterShow');

$ParameterSearch = $query->param('ParameterSearch');

$ClearInstruments = $query->param('ClearInstruments');

$ClearKindats = $query->param('ClearKindats');

$ClearParameters = $query->param('ClearParameters');

$Filter = $query->param('filter');

} else {

#Remember the filter anyway

$Filter = $query->param('filter');

}

#Create a hash to translate the list of months

%MonthHash = (

"" => "00",

"January" => "01",

"February" => "02",

"March" => "03",

"April" => "04",

"May" => "05",

"June" => "06",

"July" => "07",

"August" => "08",

"September" => "09",

"October" => "10",

"November" => "11",

"December" => "12",

);

%HashMonthPart = (

"00" => "",

"01" => "Jan",

"02" => "Feb",

"03" => "Mar",

"04" => "Apr",

"05" => "May",

"06" => "Jun",

"07" => "Jul",

"08" => "Aug",

"09" => "Sep",

"10" => "Oct",

"11" => "Nov",

"12" => "Dec",

);

%HashMonthFull = (

"00" => "",

"01" => "January",

"02" => "February",

"03" => "March",

"04" => "April",

"05" => "May",

"06" => "June",

"07" => "July",

"08" => "August",

"09" => "September",

"10" => "October",

"11" => "November",

"12" => "December",

);

#Create the page

print "Content-type: text/html", "\n\n";

print "<HTML>", "\n\n";

print "<HEAD>", "\n";

#Don't let the server cache this page

print "<META HTTP-EQUIV='Pragma' CONTENT='no-cache'>", "\n";

#print "<META HTTP-EQUIV='Expires' CONTENT='Mon, 01 Jan 1990 00:00:01 GMT'>", "\n";

#Give the title

print "<TITLE>CEDARweb Data Query Page for the CEDAR Database</TITLE>", "\n";

print "</HEAD>", "\n\n";

#Start the body section

print "<BODY BGCOLOR='#C0C0C0'>", "\n";

# Jose Garcia / Debug

# print "<P>@Parameter</P>\n";

#See if we have moved to another page

if ($NewStage ne "") {

$Stage = $NewStage;

$NewStage = "";

}

#See if we have just chosen a new day, month, or year

if ($NewDay ne "") {

#Remove the brackets

if ((substr($NewDay, 0, 2)) eq "> ") {

#Chop off two characters from the front \*and\* the back

$TempDay = substr($NewDay,2,99);

$StartDay = substr($TempDay,0,-2);

} else {

$StartDay = $NewDay;

}

}

if ($NewMonth ne "") {

#Remove the brackets

if ((substr($NewMonth, 0, 2)) eq "> ") {

#Chop off two characters from the front \*and\* the back

$TempMonth = substr($NewMonth,2,99);

$StartMonth = substr($TempMonth,0,-2);

} else {

$StartMonth = $NewMonth;

}

}

if ($NewYear ne "") {

#Remove the brackets

if ((substr($NewYear, 0, 2)) eq "> ") {

$StartYear = substr($NewYear,2,4);

} else {

$StartYear = $NewYear;

}

}

#Check the starting year

if (!($StartYear =~ /^(\d\*)$/)) {

$StartYearError = "error";

$Stage = "Date\_Time";

} elsif ($StartYear ne "") {

if ($StartYear < $beginning\_year) {

$StartYearError = "toolow";

$Stage = "Date\_Time";

} elsif ($StartYear > $final\_year) {

$StartYearError = "toohigh";

$Stage = "Date\_Time";

} else {

#Set the effective starting year

$EffectiveStartYear = $StartYear;

}

} else {

#Clear the effective starting year

$EffectiveStartYear = $beginning\_year;

}

if ($MonthHash{$StartMonth} ne "00") {

#See if they haven't chosen a year

if ($StartYear eq "") {

$StartMonthError = "noyear";

$Stage = "Date\_Time";

} else {

#Set the effective starting month

$EffectiveStartMonth = $MonthHash{$StartMonth};

}

} else {

#Clear the effective starting month

$EffectiveStartMonth = "01";

}

if (!($StartDay =~ /^(\d\*)$/)) {

$StartDayError = "error";

$Stage = "Date\_Time";

} elsif ($StartDay ne "") {

#See if they have entered a month or year

if (($StartMonth eq "") || ($StartYear eq "")) {

if ($StartYear eq "") {

$StartDayError = "noyear";

$Stage = "Date\_Time";

}

if ($StartMont eq "") {

$StartDatError = "nomonth";

$Stage = "Date\_Time";

}

} else {

#Find the last day of the month

$LastStartDay = &FindLastDayOfMonth($EffectiveStartMonth, $EffectiveStartYear);

#See if the day is later than the last day of the month

if ($StartDay < 1) {

$StartDayError = "tooearly";

$Stage = "Date\_Time";

} elsif ($StartDay > $LastStartDay) {

$StartDayError = "toolate";

$Stage = "Date\_Time";

} else {

#Set the effective starting day

if ($StartDay < 10) {

$EffectiveStartDay = "0" . (0 + $StartDay);

} else {

$EffectiveStartDay = "" . (0 + $StartDay);

}

}

}

} else {

#Clear the effective starting day

$EffectiveStartDay = "01";

}

if (!($StartHour =~ /^(\d\*)$/)) {

$StartHourError = "error";

$Stage = "Date\_Time";

} elsif ($StartHour ne "") {

if (($StartHour < 0) || ($StartHour > 23)) {

$StartHourError = "invalid";

$Stage = "Date\_Time";

} else {

#Set the effective starting hour

if ($StartHour < 10) {

$EffectiveStartHour = "0" . (0 + $StartHour);

} else {

$EffectiveStartHour = "" . (0 + $StartHour);

}

}

} else {

#Clear the effective starting hour

$EffectiveStartHour = "00";

}

if (!($StartMinute =~ /^(\d\*)$/)) {

$StartMinuteError = "error";

$Stage = "Date\_Time";

} elsif ($StartMinute ne "") {

if (($StartMinute < 0) || ($StartMinute > 59)) {

$StartMinuteError = "invalid";

$Stage = "Date\_Time";

} else {

#Set the effective starting minute

if ($StartMinute < 10) {

$EffectiveStartMinute = "0" . (0 + $StartMinute);

} else {

$EffectiveStartMinute = "" . (0 + $StartMinute);

}

}

} else {

#Clear the effective starting minute

$EffectiveStartMinute = "00";

}

if (!($StartSecond =~ /^(\d\*)$/)) {

$StartSecondError = "error";

$Stage = "Date\_Time";

} elsif ($StartSecond ne "") {

if (($StartSecond < 0) || ($StartSecond > 5999)) {

$StartSecondError = "invalid";

$Stage = "Date\_Time";

} else {

#Set the effective starting second

if ($StartSecond < 10) {

$EffectiveStartSecond = "000" . (0 + $StartSecond);

} elsif ($StartSecond < 100) {

$EffectiveStartSecond = "00" . (0 + $StartSecond);

} elsif ($StartSecond < 1000) {

$EffectiveStartSecond = "0" . (0 + $StartSecond);

} else {

$EffectiveStartSecond = "" . (0 + $StartSecond);

}

}

} else {

#Clear the effective starting second

$EffectiveStartSecond = "0000";

}

#See if the user has picked a date range

if ($DateRange ne "") {

#Make sure the date range is numeric

$DateRange = 0 + $DateRange;

#Figure out the starting date and the time to add

$TempTime = parsedate($EffectiveStartMonth."/".$EffectiveStartDay."/".$EffectiveStartYear);

$AddTime = 60 \* 60 \* 24 \* $DateRange;

$FinalTime = ($TempTime + $AddTime);

#Now it's in an array (see Perl localtime docs for help)

@MyTime = localtime($FinalTime);

#Note: this is Y2K complient... it will @MyTime[5] will be 101 at 2001,

#so 1900 + @MyTime[5] will still work

$NewYear = 1900 + @MyTime[5];

$NewMonth = 1 + @MyTime[4];

if ($NewMonth < 10) {

$NewMonth = "0".$NewMonth;

}

$NewDay = @MyTime[3];

if ($NewDay < 10) {

$NewDay = "0".$NewDay;

}

#Set the new values

$EndYear = $NewYear;

$EndMonth = $HashMonthFull{$NewMonth};

$EndDay = $NewDay;

}

if (!($EndYear =~ /^(\d\*)$/)) {

$EndYearError = "error";

$Stage = "Date\_Time";

} elsif ($EndYear ne "") {

if ($EndYear < $beginning\_year) {

$EndYearError = "toolow";

$Stage = "Date\_Time";

} elsif ($EndYear > $final\_year) {

$EndYearError = "toohigh";

$Stage = "Date\_Time";

} else {

#Set the effective ending year

$EffectiveEndYear = $EndYear;

}

} else {

#Clear the effective ending year

$EffectiveEndYear = $final\_year;

}

if ($MonthHash{$EndMonth} ne "00") {

#See if they haven't chosen a year

if ($EndYear eq "") {

$EndMonthError = "noyear";

$Stage = "Date\_Time";

} else {

#Set the effective ending month

$EffectiveEndMonth = $MonthHash{$EndMonth};

}

} else {

#Clear the effective ending month

$EffectiveEndMonth = "12";

}

if (!($EndDay =~ /^(\d\*)$/)) {

$EndDayError = "error";

$Stage = "Date\_Time";

} elsif ($EndDay ne "") {

#See if they have entered a month or year

if (($EndMonth eq "") || ($EndYear eq "")) {

if ($EndYear eq "") {

$EndDayError = "noyear";

$Stage = "Date\_Time";

}

if ($EndMont eq "") {

$EndDatError = "nomonth";

$Stage = "Date\_Time";

}

} else {

#Find the last day of the month

$LastEndDay = &FindLastDayOfMonth($EffectiveEndMonth, $EffectiveEndYear);

#See if the day is later than the last day of the month

if ($EndDay < 1) {

$EndDayError = "tooearly";

$Stage = "Date\_Time";

} elsif ($EndDay > $LastEndDay) {

$EndDayError = "toolate";

$Stage = "Date\_Time";

} else {

#Set the effective ending day

if ($EndDay < 10) {

$EffectiveEndDay = "0" . (0 + $EndDay);

} else {

$EffectiveEndDay = "" . (0 + $EndDay);

}

}

}

} else {

#Clear the effective ending day

$EffectiveEndDay = &FindLastDayOfMonth($EffectiveEndMonth, $EffectiveEndYear);

}

if (!($EndHour =~ /^(\d\*)$/)) {

$EndHourError = "error";

$Stage = "Date\_Time";

} elsif ($EndHour ne "") {

if (($EndHour < 0) || ($EndHour > 23)) {

$EndHourError = "invalid";

$Stage = "Date\_Time";

} else {

#Set the effective ending hour

if ($EndHour < 10) {

$EffectiveEndHour = "0" . (0 + $EndHour);

} else {

$EffectiveEndHour = "" . (0 + $EndHour);

}

}

} else {

#Clear the effective ending hour

$EffectiveEndHour = "23";

}

if (!($EndMinute =~ /^(\d\*)$/)) {

$EndMinuteError = "error";

$Stage = "Date\_Time";

} elsif ($EndMinute ne "") {

if (($EndMinute < 0) || ($EndMinute > 59)) {

$EndMinuteError = "invalid";

$Stage = "Date\_Time";

} else {

#Set the effective ending minute

if ($EndMinute < 10) {

$EffectiveEndMinute = "0" . (0 + $EndMinute);

} else {

$EffectiveEndMinute = "" . (0 + $EndMinute);

}

}

} else {

#Clear the effective ending minute

$EffectiveEndMinute = "59";

}

if (!($EndSecond =~ /^(\d\*)$/)) {

$EndSecondError = "error";

$Stage = "Date\_Time";

} elsif ($EndSecond ne "") {

if (($EndSecond < 0) || ($EndSecond > 5999)) {

$EndSecondError = "invalid";

$Stage = "Date\_Time";

} else {

#Set the effective ending second

if ($EndSecond < 10) {

$EffectiveEndSecond = "000" . (0 + $EndSecond);

} elsif ($EndSecond < 100) {

$EffectiveEndSecond = "00" . (0 + $EndSecond);

} elsif ($EndSecond < 1000) {

$EffectiveEndSecond = "0" . (0 + $EndSecond);

} else {

$EffectiveEndSecond = "" . (0 + $EndSecond);

}

}

} else {

#Clear the effective ending second

$EffectiveEndSecond = "5999";

}

#See if the user is trying to go on without entering a KINST or KINDAT

if (($Stage eq "Go") && ($Kinst eq "") && ($Kindat eq "")) {

#Don't let the user go on

$KinstError = "cantgo";

$Stage = "Instrument";

}

#Create the banner

&CreateBanner;

#See if MySQL is running

my $IsRunning="";

$IsRunning=`$mysql\_client -u$user -p$pass -e"show tables;" $CEDARDB`;

if ($IsRunning eq "") {

#Give the user a warning message

print "<P ALIGN='center'>Sorry</P>", "\n";

print "<P ALIGN='center'>The CEDAR database is currently unavailable. Please try back again later.</P>", "\n";

print "<P ALIGN='center'>For news about maintenance schedules or planned down time, please read ", "\n";

print "<A HREF='" . $main\_site . "/Downtime.html'>" . $main\_site . "/Downtime.html</A></P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

} elsif ($Stage eq "Go") {

#Show the final page

&DoGo;

} else {

#Query the mysql database to find the starting and ending date indicies

$StartClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DATE\_ID from tbl\_date WHERE (YEAR = " . $EffectiveStartYear . ") AND (MONTH = " . $EffectiveStartMonth . ") AND (DAY = " . $EffectiveStartDay . ");' $CEDARDB";

#Open the SQL

open (STARTINDEX, "$StartClause|") || DoExit ("MySQL could not open MySQL for StartIndex");

$StartIndex = 1;

while (<STARTINDEX>) {

$StartIndex = $\_;

chop($StartIndex);

}

$EndClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DATE\_ID from tbl\_date WHERE (YEAR = " . $EffectiveEndYear . ") AND (MONTH = " . $EffectiveEndMonth . ") AND (DAY = " . $EffectiveEndDay . ");' $CEDARDB";

#Open the SQL

open (ENDINDEX, "$EndClause|") || DoExit ("MySQL could not open MySQL for EndIndex");

$EndIndex = 1;

while (<ENDINDEX>) {

$EndIndex = $\_;

chop($EndIndex);

}

#Find the index of the very last record in the database

$LastClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DATE\_ID from tbl\_date WHERE (YEAR = " . $final\_year . ") AND (MONTH = 12) AND (DAY = 31);' $CEDARDB";

#Open the SQL

open (LASTINDEX, "$LastClause|") || DoExit ("MySQL could not open MySQL for EndIndex");

$LastIndex = 1;

while (<LASTINDEX>) {

$LastIndex = $\_;

chop($LastIndex);

}

#Put everything into one form

print "<FORM ACTION='" . $cgi\_site . "/cedarweb.pl' METHOD='POST'>", "\n";

#Do the left and right sections

print "<TABLE BORDER='0' CELLPADDING='1' CELLSPACING='1' WIDTH='100%'>", "\n";

print "<TR>", "\n";

#Add the left contents bar

print "<TD WIDTH='20%' VALIGN='top' ALIGN='left'>", "\n";

&CreateNav;

print "</TD>", "\n";

#Add the right main page

print "<TD WIDTH='80%' VALIGN='top' ALIGN='left'>", "\n";

&CreateMain;

print "</TD>", "\n";

#Close up the table

print "</TABLE>", "\n";

#Create hidden fields

print "<INPUT TYPE=HIDDEN NAME='Stage' VALUE=", $Stage, ">", "\n";

#Close the form

print "</FORM>", "\n";

}

#Do the footer

print "<HR>", "\n";

#Create a table

print "<TABLE BORDER='0' CELLPADDING='1' CELLSPACING='1' WIDTH='100%'>", "\n";

print "<TR>", "\n";

print "<TD WIDTH='50%' VALIGN='top' ALIGN='left'>", "\n";

#Do the left side (return and copyright)

print "<P><A HREF='http://" . $hao\_site . "'><IMG SRC='/icons/home03.gif' WIDTH=36 HEIGHT=24 ALT='CEDAR homepage'></A><A HREF='" . $main\_site . "'> <I>Return to the CEDAR homepage</I></A><BR>", "\n";

print "<A HREF='" . $hao\_site . "/public/home/copyright.html'>Copyright 2000, NCAR. </A></P>", "\n";

print "</TD>", "\n";

#Do the right side (approval and mailto)

print "<TD WIDTH='50%' VALIGN='top' ALIGN='right'>", "\n";

print "<P>-Approved by Peter Fox<BR>", "\n";

print "<I>-Version 2.1 by </I><A HREF='mailto:jgarcia\@hao.ucar.edu'><I>Patrick Kellogg</I></A></P>", "\n";

print "</TD>", "\n";

print "</TABLE>", "\n";

print "</BODY>", "\n\n";

print "</HTML>", "\n";

exit(0);

sub CreateBanner {

# Create the banner at the top of the page

#

# &CreateBanner;

#

# INPUTS:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)(

#Create the header in a three-column table

print "<P>", "\n";

print "<TABLE BORDER='2' WIDTH='100%' CELLPADDING='1' CELLSPACING='1'>", "\n";

print "<TR>", "\n";

print "<TD WIDTH='10%' BGCOLOR='#FFFFFF'><A HREF='http://www.ucar.edu'><P ALIGN='center'><IMG SRC='" . $image\_site . "/CedarwebUCAR.gif' ALT='UCAR'><BR>UCAR</A></P></TD>", "\n";

#Print the user name and the date

print "<TD WIDTH='10%' BGCOLOR='#FFFFFF' VALIGN='bottom' ALIGN='center'><FONT SIZE='2'>";

#Print the user name

print $RemoteUser . "<BR>&nbsp;<BR>&nbsp;<BR>", "\n";

#Tricky logic: it's using the localtime function to put array values into separate variables (see Programming Perl p. 185)

($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdat) = localtime(time);

#Print the day of the week. Note the 6th array element is the mday, just like a "tm" stucture

$thisday = (Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday)[(localtime)[6]];

print $thisday, "<BR>";

#This function turns a number from 0-11 into a month name (for internationalization)

$thismonth = (Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec)[(localtime)[4]];

#localtime is Y2K complient. The "+1900" will work until UNIX dates have trouble

$year = $year+1900;

#Print the result, and close the cell

print $thismonth, " ", $mday, ", ", $year, "\n";

print "</FONT></TD>", "\n";

#Print "CEDARweb"

print "<TD WIDTH='60%' BGCOLOR='#FFFFFF'><P ALIGN='center'><IMG BORDER='1' SRC='" . $image\_site . "/Cedarweb.jpg' ALT='CEDARweb'></P></TD>", "\n";

#Print the time

print "<TD WIDTH='10%' BGCOLOR='#FFFFFF' VALIGN='bottom' ALIGN='center'><FONT SIZE='2'>";

#Turn 4:9 into 4:09

if ($min<10) {

$min = "0" . $min;

}

#Print the rest

print $hour, ":", $min, "<BR>Mountain Standard Time", "\n";

print "</TD>", "\n";

#Finish the banner

print "<TD WIDTH='10%' BGCOLOR='#FFFFFF'><P ALIGN='right'><A HREF='" . $main\_site . "'><IMG SRC='" . $image\_site . "/CedarwebCedar.gif' ALT='CEDAR'></A></P></TD>", "\n";

print "</TR>", "\n";

print "</TABLE>", "\n";

print "</P>", "\n";

}

sub CreateNav {

# Create the navigation buttons at the left of the page

#

# &CreateNav;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Show the Instruments section

print "<P>", "\n";

#print "<IMG SRC='" . $image\_site . "/NavInstrument.gif' ALT='Instrument'>", "\n";

print "<BLOCKQUOTE>", "\n";

#Show the Instruments button

print "<INPUT NAME='NewStage' VALUE='Instrument' TYPE='submit'>", "\n";

#See if the user has chosen an instrument

if ($Kinst ne "") {

print "<FONT SIZE='2'>", "\n";

print "<BR>", "\n";

print $Kinst, "\n";

print "</FONT>", "\n";

}

print "</BLOCKQUOTE>", "\n";

print "</P>", "\n";

#Show the Kindats section

print "<P>", "\n";

#print "<IMG SRC='" . $image\_site . "/NavRecordType.gif' ALT='Record Type'>", "\n";

print "<BLOCKQUOTE>", "\n";

#Show the Kindats button

print "<INPUT NAME='NewStage' VALUE='Record\_Type' TYPE='submit'>", "\n";

#See if the user has chosen a Kindat

if ($Kindat ne "") {

print "<FONT SIZE='2'>", "\n";

print "<BR>", "\n";

print $Kindat . "\n";

print "</FONT>", "\n";

}

print "</BLOCKQUOTE>", "\n";

print "</P>", "\n";

#Show the Parameters section

print "<P>", "\n";

#print "<IMG SRC='" . $image\_site . "/NavParameters.gif' ALT='Parameters'>", "\n";

print "<BLOCKQUOTE>", "\n";

#Show the Parameters button

print "<INPUT NAME='NewStage' VALUE='Parameters' TYPE='submit'>", "\n";

#See if the user has chosen any parameters

$number\_of\_parameters\_in\_list=@Parameter;

if ($number\_of\_parameters\_in\_list>0) {

print "<FONT SIZE='2'>", "\n";

print "<BR>", "\n";

foreach $par(@Parameter)

{

print $par . "\n";

}

print "</FONT>", "\n";

}

print "</BLOCKQUOTE>", "\n";

print "</P>", "\n";

#Show the Date/ Time section

print "<P>", "\n";

#print "<IMG SRC='" . $image\_site . "/NavDateTime.gif' ALT='Date\_Time'>", "\n";

print "<BLOCKQUOTE>", "\n";

#Show the Date button

print "<INPUT NAME='NewStage' VALUE='Date\_Time' TYPE='submit'>", "\n";

#See if the user has chosen a starting date

if ($StartMonth ne "") {

$FullStartDate = $HashMonthPart{$EffectiveStartMonth};

}

if ($StartDay ne "") {

#Add a space

if ($FullStartDate ne "") {

$FullStartDate = $FullStartDate . " ";

}

$FullStartDate = $FullStartDate . $EffectiveStartDay;

}

if ($StartYear ne "") {

#Add a comma

if ($FullStartDate ne "") {

$FullStartDate = $FullStartDate . ", ";

}

$FullStartDate = $FullStartDate . $EffectiveStartYear;

}

if ($FullStartDate ne "") {

print "<FONT SIZE='2'>", "\n";

print "<BR>", "\n";

print "Start: " . $FullStartDate, "\n";

print "</FONT>", "\n";

}

#See if the user has chosen an ending date

if ($EndMonth ne "") {

$FullEndDate = $HashMonthPart{$EffectiveEndMonth};

}

if ($EndDay ne "") {

#Add a space

if ($FullEndDate ne "") {

$FullEndDate = $FullEndDate . " ";

}

$FullEndDate = $FullEndDate . $EffectiveEndDay;

}

if ($EndYear ne "") {

#Add a comma

if ($FullEndDate ne "") {

$FullEndDate = $FullEndDate . ", ";

}

$FullEndDate = $FullEndDate . $EffectiveEndYear;

}

if ($FullEndDate ne "") {

print "<FONT SIZE='2'>", "\n";

print "<BR>", "\n";

print "End: " . $FullEndDate, "\n";

print "</FONT>", "\n";

}

print "</BLOCKQUOTE>", "\n";

print "</P>", "\n";

#Show the "Go" section

print "<P>", "\n";

#print "<IMG SRC='" . $image\_site . "/NavGo.gif'> ALT='Go'>", "\n";

print "<BLOCKQUOTE>", "\n";

#Show the Go button

print "<INPUT NAME='NewStage' VALUE='Go' TYPE='submit'>", "\n";

print "</BLOCKQUOTE>", "\n";

print "</P>", "\n";

#Show the Clear All section

print "<P>", "\n";

#print "<IMG SRC='" . $image\_site . "/NavClear.gif' ALT = 'Clear'>", "\n";

print "<BLOCKQUOTE>", "\n";

#Show the Clear All button

print "<INPUT NAME='Clear' VALUE='Clear Query' TYPE='submit'>", "\n";

print "</BLOCKQUOTE>", "\n";

print "</P>", "\n";

}

sub CreateMain {

# Create the information on the main page

#

# &CreateMain;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#See which page we are supposed to be on

if ($Stage eq "Date\_Time") {

#Create the Date query page

&DoDateTime;

} elsif ($Stage eq "Instrument") {

#Create the Instruments page

&DoInstruments;

} elsif ($Stage eq "Record\_Type") {

#Create the Kindats page

&DoKindats;

} elsif ($Stage eq "Parameters") {

#Create the Parameters page

&DoParameters;

} else {

#Create a welcome page

print "<P>&nbsp;</P>", "\n";

print "<P ALIGN='center'><B>Welcome to the CEDAR database web interface</B></P>", "\n";

print "<P ALIGN='center'><A HREF='http://www.unidata.ucar.edu/packages/dods/index.html'><IMG SRC='" . $image\_site . "/dods-logo.gif' ALT='DODS' BORDER='0'></A>";

print "<P ALIGN='center'>Powered by <A HREF='http://www.unidata.ucar.edu/packages/dods/index.html'>DODS</A></P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P ALIGN='left'>Please click on an item to the left (Date\_Time, Instrument, Record\_Type, or Parameters) ", "\n";

print "to start creating a query. Then, when the query is complete, click &quot;Go&quot;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

print "<P>&nbsp;</P>", "\n";

#Save the hidden variables, just in case

print "<INPUT TYPE=HIDDEN NAME='StartYear' VALUE=",$StartYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMonth' VALUE=",$StartMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartDay' VALUE=",$StartDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartHour' VALUE=",$StartHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMinute' VALUE=",$StartMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartSecond' VALUE=",$StartSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndYear' VALUE=",$EndYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMonth' VALUE=",$EndMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndDay' VALUE=",$EndDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndHour' VALUE=",$EndHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMinute' VALUE=",$EndMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndSecond' VALUE=",$EndSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kinst' VALUE=",$Kinst,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstSortBy' VALUE=",$KinstSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstAscDesc' VALUE=",$KinstAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstShow' VALUE=",$KinstShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kindat' VALUE=",$Kindat,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatSortBy' VALUE=",$KindatSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatAscDesc' VALUE=",$KindatAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatShow' VALUE=",$KindatShow,">", "\n";

foreach $par(@Parameter)

{

print "<INPUT TYPE=HIDDEN NAME='Parameter' VALUE=$par>\n",

}

print "<INPUT TYPE=HIDDEN NAME='ParameterSortBy' VALUE=",$ParameterSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterAscDesc' VALUE=",$ParameterAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterShow' VALUE=",$ParameterShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterSearch' VALUE=",$ParameterSearch,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='filter' VALUE=",$Filter,">", "\n";

}

}

sub DoDateTime {

# Create the page for the date and time selection

#

# &DoDateTime;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Tell the user if they have made an error

print "<P>", "\n";

if ($StartYearError eq "error") {

print "<FONT COLOR=red>Starting year must be numeric</FONT><BR>";

} elsif ($StartYearError eq "toolow") {

print "<FONT COLOR=red>Starting year must be greater than or equal to ". $beginning\_year . "</FONT><BR>";

} elsif ($StartYearError eq "toohigh") {

print "<FONT COLOR=red>Starting year must be less than or equal to " . $final\_year . "</FONT><BR>";

}

if ($StartMonthError eq "noyear") {

print "<FONT COLOR=red>You have chosen a starting month, so you must enter a starting year</FONT><BR>";

}

if ($StartDayError eq "error") {

print "<FONT COLOR=red>Starting day must be numeric</FONT><BR>";

} elsif ($StartDayError eq "noyear") {

print "<FONT COLOR=red>You have chosen a starting day, so you must enter a starting year</FONT><BR>";

} elsif ($StartDayError eq "nomonth") {

print "<FONT COLOR=red>You have chosen a starting day, so you must enter a starting month</FONT><BR>";

} elsif ($StartDayError eq "tooearly") {

print "<FONT COLOR=red>Starting day must be greater than 0</FONT><BR>";

} elsif ($StartDayError eq "toolate") {

print "<FONT COLOR=red>There are only " . $LastStartDay . " days in the chosen starting month</FONT><BR>";

}

if ($StartHourError eq "error") {

print "<FONT COLOR=red>Starting hour must be numeric</FONT><BR>";

} elsif ($StartHourError eq "invalid") {

print "<FONT COLOR=red>Starting hour must be between 0 and 23</FONT><BR>";

}

if ($StartMinuteError eq "error") {

print "<FONT COLOR=red>Starting minute must be numeric</FONT><BR>";

} elsif ($StartMinuteError eq "invalid") {

print "<FONT COLOR=red>Starting minute must be between 00 and 59</FONT><BR>";

}

if ($StartSecondError eq "error") {

print "<FONT COLOR=red>Starting centisecond must be numeric</FONT><BR>";

} elsif ($StartSecondError eq "invalid") {

print "<FONT COLOR=red>Starting centisecond must be between 0000 and 5999</FONT><BR>";

}

if ($EndYearError eq "error") {

print "<FONT COLOR=red>Ending year must be numeric</FONT><BR>";

} elsif ($EndYearError eq "toolow") {

print "<FONT COLOR=red>Ending year must be greater than or equal to ". $beginning\_year . "</FONT><BR>";

} elsif ($EndYearError eq "toohigh") {

print "<FONT COLOR=red>Ending year must be less than or equal to " . $final\_year . "</FONT><BR>";

}

if ($EndMonthError eq "noyear") {

print "<FONT COLOR=red>You have chosen an ending month, so you must enter a ending year</FONT><BR>";

}

if ($EndDayError eq "error") {

print "<FONT COLOR=red>Ending day must be numeric</FONT><BR>";

} elsif ($EndDayError eq "noyear") {

print "<FONT COLOR=red>You have chosen an ending day, so you must enter a ending year</FONT><BR>";

} elsif ($EndDayError eq "nomonth") {

print "<FONT COLOR=red>You have chosen an ending day, so you must enter a ending month</FONT><BR>";

} elsif ($EndDayError eq "tooearly") {

print "<FONT COLOR=red>Ending day must be greater than 0</FONT><BR>";

} elsif ($EndDayError eq "toolate") {

print "<FONT COLOR=red>There are only " . $LastEndDay . " days in the chosen ending month</FONT><BR>";

}

if ($EndHourError eq "error") {

print "<FONT COLOR=red>Ending hour must be numeric</FONT><BR>";

} elsif ($EndHourError eq "invalid") {

print "<FONT COLOR=red>Ending hour must be between 0 and 23</FONT><BR>";

}

if ($EndMinuteError eq "error") {

print "<FONT COLOR=red>Ending minute must be numeric</FONT><BR>";

} elsif ($EndMinuteError eq "invalid") {

print "<FONT COLOR=red>Ending minute must be between 00 and 59</FONT><BR>";

}

if ($EndSecondError eq "error") {

print "<FONT COLOR=red>Ending second must be numeric</FONT><BR>";

} elsif ($EndSecondError eq "invalid") {

print "<FONT COLOR=red>Ending centisecond must be between 0000 and 5999</FONT><BR>";

}

#Create the table

print "<TABLE BORDER='0' CELLPADDING='0' CELLSPACING='0' WIDTH='100%'>", "\n";

#Print a neat "header row" with the days of the week

print "<TR>", "\n";

print "<TD WIDTH='14%'> </TD>", "\n";

print "<TD WIDTH='14%'><U>Month</U></TD>", "\n";

print "<TD WIDTH='14%'><U>Day</U> (DD)</TD>", "\n";

print "<TD WIDTH='14%'><U>Year</U> (YYYY)</TD>", "\n";

print "<TD WIDTH='14%'><U>Hour</U> (00-23)</TD>", "\n";

print "<TD WIDTH='14%'><U>Minute</U> (00-59)</TD>", "\n";

print "<TD WIDTH='14%'><U>Second</U> (0000-5999)</TD>", "\n";

print "</TR>", "\n";

#Print the Starting information

print "<TR>", "\n";

#Print a label

print "<TD WIDTH='14%'>", "\n";

print "Starting Date:", "\n";

print "</TD>", "\n";

#Create a Starting Month

print "<TD WIDTH='14%'>", "\n";

print "<SELECT NAME='StartMonth' SIZE='1'>", "\n";

&CreateStartingMonthCombo;

print "</SELECT>", "\n";

print "</TD>", "\n";

#Create a Starting Day

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='StartDay' SIZE='3'";

if ($StartDay ne "") {

print " VALUE='" . $StartDay . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Starting Year

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='StartYear' SIZE='5'";

if ($StartYear ne "") {

print " VALUE='" . $StartYear . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Starting Hour

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='StartHour' SIZE='3'", "\n";

if ($StartHour ne "") {

print " VALUE='" . $StartHour . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Starting Minute

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='StartMinute' SIZE='3'", "\n";

if ($StartMinute ne "") {

print " VALUE='" . $StartMinute . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Starting Second

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='StartSecond' SIZE='5'", "\n";

if ($StartSecond ne "") {

print " VALUE='" . $StartSecond . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Close the row and start a new one

print "</TR>", "\n";

print "<TR>", "\n";

#Let the user select an ending date

print "<TD WIDTH='14%'>", "\n";

print "Ending Date: ", "\n";

print "</TD>", "\n";

#Do the Ending Month

print "<TD WIDTH='14%'>", "\n";

print "<SELECT NAME='EndMonth' SIZE='1'>", "\n";

&CreateEndingMonthCombo;

print "</SELECT>", "\n";

print "</TD>", "\n";

#Create a Ending Day

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='EndDay' SIZE='3'";

if ($EndDay ne "") {

print " VALUE='" . $EndDay . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Ending Year

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='EndYear' SIZE='5'";

if ($EndYear ne "") {

print " VALUE='" . $EndYear . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Ending Hour

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='EndHour' SIZE='3'", "\n";

if ($EndHour ne "") {

print " VALUE='" . $EndHour . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Ending Minute

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='EndMinute' SIZE='3'", "\n";

if ($EndMinute ne "") {

print " VALUE='" . $EndMinute . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Create a Ending Second

print "<TD WIDTH='14%'>", "\n";

print "<INPUT TYPE='text' NAME='EndSecond' SIZE='5'", "\n";

if ($EndSecond ne "") {

print " VALUE='" . $EndSecond . "'";

}

print ">", "\n";

print "</TD>", "\n";

#Close the row and the table

print "</TR>", "\n";

print "</TABLE>", "\n";

#See which calendar we should bring up

if ((($StartYear ne "") && ($StartMonth ne "") && ($StartDay ne "") && ($StartHour eq "")) || ($NewDay ne "")) {

#Either we've chosen a year, month, and day, or we just chose a day

&CreateDateRange;

} elsif ((($StartYear ne "") && ($StartMonth ne "") && ($StartDay eq "")) || ($NewMonth ne "")) {

#Either we've chosen a year and month, or we just chose a month

&CreateDayCalendar;

} elsif ((($StartYear ne "") && ($StartMonth eq "")) || ($NewYear ne "")) {

#Either we've chosen a year but no month, or we've just chosen a month

&CreateMonthCalendar;

} else {

#Bring up the year calendar

&CreateYearCalendar;

}

#Create some hidden variables

print "<INPUT TYPE=HIDDEN NAME='Kinst' VALUE=",$Kinst,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstSortBy' VALUE=",$KinstSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstAscDesc' VALUE=",$KinstAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstShow' VALUE=",$KinstShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kindat' VALUE=",$Kindat,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatSortBy' VALUE=",$KindatSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatAscDesc' VALUE=",$KindatAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatShow' VALUE=",$KindatShow,">", "\n";

foreach $par(@Parameter)

{

print "<INPUT TYPE=HIDDEN NAME='Parameter' VALUE=$par>\n",

}

print "<INPUT TYPE=HIDDEN NAME='ParameterSortBy' VALUE=",$ParameterSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterAscDesc' VALUE=",$ParameterAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterShow' VALUE=",$ParameterShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterSearch' VALUE=",$ParameterSearch,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='filter' VALUE=",$Filter,">", "\n";

}

sub CreateStartingMonthCombo {

# Create a combo box with starting month information

#

# &CreateStartingMonthCombo

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Create a "month" combo box

print "<OPTION VALUE=''";

if ($StartMonth eq '') {

print " SELECTED";

}

print ">None yet</OPTION>", "\n";

print "<OPTION VALUE='January'";

if ($StartMonth eq 'January') {

print " SELECTED";

}

print ">January</OPTION>", "\n";

print "<OPTION VALUE='February'";

if ($StartMonth eq 'February') {

print " SELECTED";

}

print ">February</OPTION>", "\n";

print "<OPTION VALUE='March'";

if ($StartMonth eq 'March') {

print " SELECTED";

}

print ">March</OPTION>", "\n";

print "<OPTION VALUE='April'";

if ($StartMonth eq 'April') {

print " SELECTED";

}

print ">April</OPTION>", "\n";

print "<OPTION VALUE='May'";

if ($StartMonth eq 'May') {

print " SELECTED";

}

print ">May</OPTION>", "\n";

print "<OPTION VALUE='June'";

if ($StartMonth eq 'June') {

print " SELECTED";

}

print ">June</OPTION>", "\n";

print "<OPTION VALUE='July'";

if ($StartMonth eq 'July') {

print " SELECTED";

}

print ">July</OPTION>", "\n";

print "<OPTION VALUE='August'";

if ($StartMonth eq 'August') {

print " SELECTED";

}

print ">August</OPTION>", "\n";

print "<OPTION VALUE='September'";

if ($StartMonth eq 'September') {

print " SELECTED";

}

print ">September</OPTION>", "\n";

print "<OPTION VALUE='October'";

if ($StartMonth eq 'October') {

print " SELECTED";

}

print ">October</OPTION>", "\n";

print "<OPTION VALUE='November'";

if ($StartMonth eq 'November') {

print " SELECTED";

}

print ">November</OPTION>", "\n";

print "<OPTION VALUE='December'";

if ($StartMonth eq 'December') {

print " SELECTED";

}

print ">December</OPTION>", "\n";

}

sub CreateEndingMonthCombo {

# Create a combo box with ending month information

#

# &CreateEndingMonthCombo

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Create a "month" combo box

print "<OPTION VALUE=''";

if ($EndMonth eq '') {

print " SELECTED";

}

print ">None yet</OPTION>", "\n";

print "<OPTION VALUE='January'";

if ($EndMonth eq 'January') {

print " SELECTED";

}

print ">January</OPTION>", "\n";

print "<OPTION VALUE='February'";

if ($EndMonth eq 'February') {

print " SELECTED";

}

print ">February</OPTION>", "\n";

print "<OPTION VALUE='March'";

if ($EndMonth eq 'March') {

print " SELECTED";

}

print ">March</OPTION>", "\n";

print "<OPTION VALUE='April'";

if ($EndMonth eq 'April') {

print " SELECTED";

}

print ">April</OPTION>", "\n";

print "<OPTION VALUE='May'";

if ($EndMonth eq 'May') {

print " SELECTED";

}

print ">May</OPTION>", "\n";

print "<OPTION VALUE='June'";

if ($EndMonth eq 'June') {

print " SELECTED";

}

print ">June</OPTION>", "\n";

print "<OPTION VALUE='July'";

if ($EndMonth eq 'July') {

print " SELECTED";

}

print ">July</OPTION>", "\n";

print "<OPTION VALUE='August'";

if ($EndMonth eq 'August') {

print " SELECTED";

}

print ">August</OPTION>", "\n";

print "<OPTION VALUE='September'";

if ($EndMonth eq 'September') {

print " SELECTED";

}

print ">September</OPTION>", "\n";

print "<OPTION VALUE='October'";

if ($EndMonth eq 'October') {

print " SELECTED";

}

print ">October</OPTION>", "\n";

print "<OPTION VALUE='November'";

if ($EndMonth eq 'November') {

print " SELECTED";

}

print ">November</OPTION>", "\n";

print "<OPTION VALUE='December'";

if ($EndMonth eq 'December') {

print " SELECTED";

}

print ">December</OPTION>", "\n";

}

sub CreateDateRange {

# Create a simple way to select a date range

#

# &CreateDateRange;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Print a header

print "<P>", "\n";

print "Please choose a date range:<BR>", "\n";

print "(the number of days for the query)</P>", "\n";

#Create a simple text box

print "<INPUT TYPE='text' NAME='DateRange' SIZE='6'>", "\n";

#Create a "Calculate" button

print "<INPUT TYPE='submit' NAME='Calculate' VALUE='Calculate'";

print ">", "\n";

print "</P>", "\n";

}

sub CreateDayCalendar {

# Create the calendar that shows starting days

#

# &CreateDayCalendar;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Print a header

print "<P>", "\n";

print "Please choose a starting day:<BR>", "\n";

#Figure out what day of the month the 1st falls on (from 0 to 6)

$FirstDayIndex = &FindDay($EffectiveStartMonth, $EffectiveStartYear);

#Figure out the last day of the month

$LastDayIndex = &FindLastDayOfMonth($EffectiveStartMonth, $EffectiveStartYear);

#Create the mysql statement

$MainClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT tbl\_date.DAY";

$FromClause = " FROM tbl\_date,tbl\_date\_in\_file";

$WhereClause = " WHERE tbl\_date.DATE\_ID=tbl\_date\_in\_file.DATE\_ID";

$JoinClause = " AND tbl\_date.YEAR=" . $EffectiveStartYear . " AND tbl\_date.MONTH =" . $EffectiveStartMonth;

#See if the user has chosen anything yet

$tt=@Parameter;

if (($Kinst ne "") || ($Kindat ne "") || ($tt>0)) {

#Add to the clauses

$FromClause = $FromClause . ",tbl\_cedar\_file,tbl\_file\_info,tbl\_record\_type,tbl\_record\_info";

$WhereClause = $WhereClause . " AND tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID=tbl\_file\_info.RECORD\_IN\_FILE\_ID AND tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_cedar\_file.FILE\_ID=tbl\_file\_info.FILE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_type.RECORD\_TYPE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

}

#If the user has chosen a parameter, we need a little bit more information

if ($NumParameterKeys > 0) {

#Add to clauses

$FromClause = $FromClause . ",tbl\_record\_info";

$WhereClause = $WhereClause . " AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

}

#See if the user has specified any instruments

if ($Kinst ne "") {

$JoinClause = $JoinClause . " AND (tbl\_record\_type.KINST=" . $Kinst . ")";

}

#See if the user has specified any record\_type

if ($Kindat ne "") {

#Do the split

($KinstPart,$KindatPart) = split(/\//,$Kindat);

$JoinClause = $JoinClause . " AND ((tbl\_record\_type.KINST=" . $KinstPart . ") AND (tbl\_record\_type.KINDAT=" . $KindatPart . "))";

}

#See if the user has specified any parameters

$tt1=@Parameter;

if ($tt1>0) {

$JoinClause = $JoinClause . " AND (";

$index=1;

foreach $par(@Parameter)

{

$JoinClause = $JoinClause . "(tbl\_record\_info.PARAMETER\_ID=" . $par . ")";

if($index<$tt1)

{

$JoinClause = $JoinClause . " OR ";

$index=$index+1;

}

}

$JoinClause = $JoinClause .") "

}

#Do the query to find the years that have data

$SqlClause = $MainClause . $FromClause . $WhereClause . $JoinClause . " ORDER BY tbl\_date.DAY ASC;' $CEDARDB";

# Jose Garcia / Debug

# print "<p>$SqlClause</p>\n";

open (FINDDATES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for CreateDay");

#Create a hash of the values

%DateHash = ();

while(<FINDDATES>) {

$NextDate = $\_;

chop($NextDate);

#Insert it into the hash

$DateHash{$NextDate} = 'Chosen';

}

#For testing

if ($DebugSQLClause eq "TRUE") {

print "<P>" . $SqlClause . "</P>", "\n";

}

#Create the table

print "<TABLE BORDER='1' CELLPADDING='0' CELLSPACING='0' WIDTH='100%'>", "\n";

#Print a neat "header row" with the days of the week

print "<TR>", "\n";

print "<TD WIDTH='14%'>Sunday</TD>", "\n";

print "<TD WIDTH='14%'>Monday</TD>", "\n";

print "<TD WIDTH='14%'>Tuesday</TD>", "\n";

print "<TD WIDTH='14%'>Wednesday</TD>", "\n";

print "<TD WIDTH='14%'>Thursday</TD>", "\n";

print "<TD WIDTH='14%'>Friday</TD>", "\n";

print "<TD WIDTH='14%'>Saturday</TD>", "\n";

print "</TR>", "\n";

#Loop for each week (six weeks maximum)

for ($TempDay = 1; $TempDay < 42; ($TempDay = $TempDay + 7)) {

#See if we need to print another row

#i.e. the actual sunday is still less or equal than the last day index

if (($TempDay-$FirstDayIndex) <= $LastDayIndex ) {

#Print the row

print "<TR>", "\n";

#Loop seven times for each row

for ($TempLooper = 0; $TempLooper < 7; $TempLooper++) {

#Find the actual day value

$ActualDay = ($TempDay + $TempLooper - $FirstDayIndex);

#Print the cell

print "<TD WIDTH='14%'>";

#Make sure the day is between 1 and the last day

if (($ActualDay >= 1) && ($ActualDay <= $LastDayIndex )) {

#Show the data

print "<CENTER>";

if ($DateHash{$ActualDay} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='";

print "> " . $ActualDay . " <", "\n";

print "' NAME='NewDay'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='";

print $ActualDay;

print "' NAME='NewDay'>";

}

print "</CENTER>", "\n";

} else {

#Leave the cell blank

print "&nbsp;";

}

print "</TD>", "\n";

}

#Close the row

print "</TR>", "\n";

}

}

#Close the table

print "</TABLE>", "\n";

print "</P>", "\n";

#Close the handle

close(FINDDATES);

}

sub CreateMonthCalendar {

# Create the calendar that shows starting months

#

# &CreateMonthCalendar;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Print a header

print "<P>", "\n";

print "Please choose a starting month:<BR>", "\n";

#Create the mysql statement

$MainClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT tbl\_date.MONTH";

$FromClause = " FROM tbl\_date,tbl\_date\_in\_file";

$WhereClause = " WHERE tbl\_date.DATE\_ID=tbl\_date\_in\_file.DATE\_ID";

$JoinClause = " AND tbl\_date.YEAR=" . $EffectiveStartYear;

#See if the user has chosen anything yet

$tt3=@Parameter;

if (($Kinst ne "") || ($Kindat ne "") || ($tt3>0)) {

#Add to the clauses

$FromClause = $FromClause . ",tbl\_cedar\_file,tbl\_file\_info,tbl\_record\_type,tbl\_record\_info";

$WhereClause = $WhereClause . " AND tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID=tbl\_file\_info.RECORD\_IN\_FILE\_ID AND tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_cedar\_file.FILE\_ID=tbl\_file\_info.FILE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_type.RECORD\_TYPE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

}

#If the user has chosen a parameter, we need a little bit more information

if ($NumParameterKeys > 0) {

#Add to clauses

$FromClause = $FromClause . ",tbl\_record\_info";

$WhereClause = $WhereClause . " AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

}

#See if the user has specified any instruments

if ($Kinst ne "") {

$JoinClause = $JoinClause . " AND (tbl\_record\_type.KINST=" . $Kinst . ")";

}

#See if the user has specified any record\_type

if ($Kindat ne "") {

#Do the split

($KinstPart,$KindatPart) = split(/\//,$Kindat);

$JoinClause = $JoinClause . " AND ((tbl\_record\_type.KINST=" . $KinstPart . ") AND (tbl\_record\_type.KINDAT=" . $KindatPart . "))";

}

#See if the user has specified any parameters

$tt2=@Parameter;

if ($tt2>0) {

$JoinClause = $JoinClause . " AND (";

$index=1;

foreach $par(@Parameter)

{

$JoinClause = $JoinClause . "(tbl\_record\_info.PARAMETER\_ID=" . $par . ")";

if($index<$tt2)

{

$JoinClause = $JoinClause . " OR ";

$index=$index+1;

}

}

$JoinClause = $JoinClause .") "

}

#Do the query to find the years that have data

$SqlClause = $MainClause . $FromClause . $WhereClause . $JoinClause . " ORDER BY tbl\_date.MONTH ASC;' $CEDARDB";

open (FINDDATES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for CreateMonth");

#Create a hash of the values

%DateHash = ();

while(<FINDDATES>) {

$NextDate = $\_;

chop($NextDate);

#Insert it into the hash

$DateHash{$NextDate} = 'Chosen';

}

#For testing

if ($DebugSQLClause eq "TRUE") {

print "<P>" . $SqlClause . "</P>", "\n";

}

#Create the table

print "<TABLE BORDER='1' CELLPADDING='0' CELLSPACING='0' WIDTH='100%'>", "\n";

#Print the row

print "<TR>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'1'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> January <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='January' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'2'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> February <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='February' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'3'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> March <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='March' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'4'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> April <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='April' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'5'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> May <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='May' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'6'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> June <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='June' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Close the row

print "</TR>", "\n";

#Print the row

print "<TR>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'7'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> July <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='July' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'8'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> August <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='August' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'9'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> September <' NAME='NewMonth'>";

print "</FONT>", "\n";

} else {

#Load the next date

$NextDate = $\_;

chop($NextDate);

print "<INPUT TYPE='submit' VALUE='September' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'10'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> October <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='October' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'11'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> November <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='November' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Print the cell

print "<TD WIDTH='16%'>";

print "<CENTER>";

if ($DateHash{'12'} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='> December <' NAME='NewMonth'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='December' NAME='NewMonth'>";

}

print "</CENTER>", "\n";

print "</TD>", "\n";

#Close the row

print "</TR>", "\n";

#Close the table

print "</TABLE>", "\n";

print "</P>", "\n";

}

sub CreateYearCalendar {

# Create the calendar that shows starting years

#

# &CreateYearCalendar;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Print a header

print "<P>", "\n";

print "Please choose a starting year:<BR>", "\n";

#Create the mysql statement

$MainClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT tbl\_date.YEAR";

$FromClause = " FROM tbl\_date,tbl\_date\_in\_file";

$WhereClause = " WHERE tbl\_date.DATE\_ID=tbl\_date\_in\_file.DATE\_ID";

$JoinClause = "";

#See if the user has chosen anything yet

@tt4=@Parameter;

if (($Kinst ne "") || ($Kindat ne "") || ($tt4>0)) {

#Add to the clauses

$FromClause = $FromClause . ",tbl\_cedar\_file,tbl\_file\_info,tbl\_record\_type,tbl\_record\_info";

$WhereClause = $WhereClause . " AND tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID=tbl\_file\_info.RECORD\_IN\_FILE\_ID AND tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_cedar\_file.FILE\_ID=tbl\_file\_info.FILE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_type.RECORD\_TYPE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

}

#If the user has chosen a parameter, we need a little bit more information

if ($NumParameterKeys > 0) {

#Add to clauses

$FromClause = $FromClause . ",tbl\_record\_info";

$WhereClause = $WhereClause . " AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

}

#See if the user has specified any instruments

if ($Kinst ne "") {

$JoinClause = $JoinClause . " AND (tbl\_record\_type.KINST=" . $Kinst . ")";

}

#See if the user has specified any record\_type

if ($Kindat ne "") {

#Do the split

($KinstPart,$KindatPart) = split(/\//,$Kindat);

$JoinClause = $JoinClause . " AND ((tbl\_record\_type.KINST=" . $KinstPart . ") AND (tbl\_record\_type.KINDAT=" . $KindatPart . "))";

}

#See if the user has specified any parameters

$tt5=@Parameter;

if ($tt5>0) {

$JoinClause = $JoinClause . " AND (";

$index=1;

foreach $par(@Parameter)

{

$JoinClause = $JoinClause . "(tbl\_record\_info.PARAMETER\_ID=" . $par . ")";

if($index<$tt5)

{

$JoinClause = $JoinClause . " OR ";

$index=$index+1;

}

}

$JoinClause = $JoinClause .") "

}

#Do the query to find the years that have data

$SqlClause = $MainClause . $FromClause . $WhereClause . $JoinClause . " ORDER BY tbl\_date.YEAR ASC;' $CEDARDB";

open (FINDDATES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for CreateYear");

#Create a hash of the values

%DateHash = ();

while(<FINDDATES>) {

$NextDate = $\_;

chop($NextDate);

#Insert it into the hash

$DateHash{$NextDate} = 'Chosen';

}

#For testing

if ($DebugSQLClause eq "TRUE") {

print "<P>" . $SqlClause . "</P>", "\n";

}

#Create the table

print "<TABLE BORDER='1' CELLPADDING='0' CELLSPACING='0' WIDTH='100%'>", "\n";

#Loop as many times as needed, in groups of ten

for ($TempYear = $beginning\_year; $TempYear <= $final\_year; ($TempYear = $TempYear + 10)) {

#Print the row

print "<TR>", "\n";

#Loop nine times for each row

for ($TempLooper = 0; $TempLooper < 10; $TempLooper++) {

#Find the actual value

$ActualYear = ($TempYear + $TempLooper);

#Print the cell

print "<TD WIDTH='10%'>";

#See if we've gone too far

if ($ActualYear <= $final\_year) {

#Show the data

print "<CENTER>";

if ($DateHash{$ActualYear} ne "") {

#Show name in red with >brackets<

print "<FONT COLOR=RED>", "\n";

print "<INPUT TYPE='submit' VALUE='";

print "> " . $ActualYear . " <", "\n";

print "' NAME='NewYear'>";

print "</FONT>", "\n";

#Load the next date

$NextDate = $\_;

chop($NextDate);

} else {

print "<INPUT TYPE='submit' VALUE='";

print $ActualYear;

print "' NAME='NewYear'>";

}

print "</CENTER>", "\n";

} else {

#Leave the cell blank

print "&nbsp;";

}

print "</TD>", "\n";

}

#Close the row

print "</TR>", "\n";

}

#Close the table

print "</TABLE>", "\n";

print "</P>", "\n";

#Close the handle

close(FINDDATES);

}

sub DoInstruments {

# Create the page for the instrument selection

#

# &DoInstruments;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Create a table

print "<TABLE BORDER='0' WIDTH='100%' CELLPADDING='1' CELLSPACING='1'>", "\n";

print "<TR>", "\n";

#For the left cell, put a header, the instrument table, and a "more info" button

print "<TD WIDTH='60%' VALIGN='top' ALIGN='left'>", "\n";

#See if there is an error

print "<P>", "\n";

if ($KinstError eq "cantgo") {

print "<FONT COLOR=red>Please choose at least one instrument or record type before proceeding</FONT></P><P>";

}

#Print the KinstList

print "Select one instrument:</P>", "\n";

print "<P>Note: you need to click a button on the left to process your selection</P>", "\n";

print "<P>", "\n";

&CreateKinstList;

print "</P>", "\n";

print "</TD>", "\n";

#Add the right main page with "sort by", "Ascending/Descending", and "show" combo boxes,

#buttons to re-sort and clear the list, and a button to go to the world map

print "<TD WIDTH='40%' VALIGN='top' ALIGN='left'>", "\n";

#Create a "sort by" combo box

print "<P>", "\n";

print "Sort by:<BR>", "\n";

print "<SELECT NAME='KinstSortBy' SIZE='1'>", "\n";

print "<OPTION VALUE='Code'";

if ($KinstSortBy eq 'Code') {

print " SELECTED";

}

print ">Instrument code</OPTION>", "\n";

print "<OPTION VALUE='Prefix'";

if ($KinstSortBy eq 'Prefix') {

print " SELECTED";

}

print ">Prefix</OPTION>", "\n";

print "<OPTION VALUE='Name'";

if ($KinstSortBy eq 'Name') {

print " SELECTED";

}

print ">Instrument name</OPTION>", "\n";

print "</SELECT>", "\n";

print "<BR>", "\n";

#Create an "Ascending/Descending" combo box

print "<SELECT NAME='KinstAscDesc' SIZE='1'>", "\n";

print "<OPTION VALUE='Ascending'";

if ($KinstAscDesc eq 'Ascending') {

print " SELECTED";

}

print ">Ascending</OPTION>", "\n";

print "<OPTION VALUE='Descending'";

if ($KinstAscDesc eq 'Descending') {

print " SELECTED";

}

print ">Descending</OPTION>", "\n";

print "</SELECT>", "\n";

print "<BR>", "\n";

#Create a "show" combo box

print "<SELECT NAME='KinstShow' SIZE='1'>", "\n";

print "<OPTION VALUE='All'";

if ($KinstShow eq 'All') {

print " SELECTED";

}

print ">All Instruments</OPTION>", "\n";

print "<OPTION VALUE='FabryPerot'";

if ($KinstShow eq 'FabryPerot') {

print " SELECTED";

}

print ">Fabry-Perot</OPTION>", "\n";

print "<OPTION VALUE='ISRadar'";

if ($KinstShow eq 'ISRadar') {

print " SELECTED";

}

print ">I.S. Radar</OPTION>", "\n";

print "<OPTION VALUE='Models'";

if ($KinstShow eq 'Models') {

print " SELECTED";

}

print ">Models</OPTION>", "\n";

print "<OPTION VALUE='Miscellaneous'";

if ($KinstShow eq 'Miscellaneous') {

print " SELECTED";

}

print ">Miscellaneous</OPTION>", "\n";

print "</SELECT>", "\n";

print "</P>", "\n";

#Give a hard break

print "<HR>", "\n";

#Let the user resort or clear the list

print "<P>", "\n";

print "<INPUT NAME='ResortInstruments' VALUE='Re-sort list' TYPE='submit'>", "\n";

print "<BR>", "\n";

print "<INPUT NAME='ClearInstruments' VALUE='Clear list' TYPE='submit'>", "\n";

print "</P>", "\n";

#Finish the right cell

print "</TD>", "\n";

#Close up the table

print "</TR>", "\n";

print "</TABLE>", "\n";

#Create some hidden variables

print "<INPUT TYPE=HIDDEN NAME='StartYear' VALUE=",$StartYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMonth' VALUE=",$StartMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartDay' VALUE=",$StartDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartHour' VALUE=",$StartHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMinute' VALUE=",$StartMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartSecond' VALUE=",$StartSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndYear' VALUE=",$EndYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMonth' VALUE=",$EndMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndDay' VALUE=",$EndDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndHour' VALUE=",$EndHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMinute' VALUE=",$EndMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndSecond' VALUE=",$EndSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kindat' VALUE=",$Kindat,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatSortBy' VALUE=",$KindatSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatAscDesc' VALUE=",$KindatAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatShow' VALUE=",$KindatShow,">", "\n";

foreach $par(@Parameter)

{

print "<INPUT TYPE=HIDDEN NAME='Parameter' VALUE=$par>\n",

}

print "<INPUT TYPE=HIDDEN NAME='ParameterSortBy' VALUE=",$ParameterSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterAscDesc' VALUE=",$ParameterAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterShow' VALUE=",$ParameterShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterSearch' VALUE=",$ParameterSearch,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='filter' VALUE=",$Filter,">", "\n";

}

sub CreateKinstList {

# Uses the MySQL meta-database to create a list of instruments

# at runtime.

#

# &CreateKinstList;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# all page variables

#Set up the Kinst list

print "<SELECT NAME='Kinst' SIZE='";

print "15";

print "'>", "\n";

#Create the mysql statement

$MainClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT concat(tbl\_instrument.INST\_NAME,\"%\",tbl\_instrument.PREFIX,\"%\",tbl\_instrument.KINST)";

$FromClause = " FROM tbl\_instrument,tbl\_record\_info,tbl\_record\_type";

$WhereClause = " WHERE tbl\_instrument.KINST=tbl\_record\_type.KINST AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

$JoinClause = "";

#See if the user has selected a date

if (($StartIndex != 1) || ($EndIndex != $LastIndex)) {

#Add to the FromClause

$FromClause = $FromClause . ",tbl\_file\_info,tbl\_cedar\_file,tbl\_date\_in\_file";

#Add to the WhereClause

$WhereClause = $WhereClause . " AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_file\_info.RECORD\_TYPE\_ID AND tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_file\_info.RECORD\_IN\_FILE\_ID=tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID";

#Add to the JoinClause

$JoinClause = " AND (tbl\_date\_in\_file.DATE\_ID >= " . $StartIndex. ") AND (tbl\_date\_in\_file.DATE\_ID <= " . $EndIndex .")";

}

#See if the user has specified any record\_type

if ($Kindat ne "") {

#Do the split

($KinstPart,$KindatPart) = split(/\//,$Kindat);

$JoinClause = $JoinClause . " AND ((tbl\_record\_type.KINST=" . $KinstPart . ") AND (tbl\_record\_type.KINDAT=" . $KindatPart . "))";

}

#See if the user has specified any parameters

$tt6=@Parameter;

if ($tt6>0) {

$JoinClause = $JoinClause . " AND (";

$index=1;

foreach $par(@Parameter)

{

$JoinClause = $JoinClause . "(tbl\_record\_info.PARAMETER\_ID=" . $par . ")";

if($index<$tt6)

{

$JoinClause = $JoinClause . " OR ";

$index=$index+1;

}

}

$JoinClause = $JoinClause .") "

}

#Figure out sort by

if ($KinstSortBy eq 'Name') {

$JoinClause = $JoinClause . " ORDER BY tbl\_instrument.INST\_NAME";

} elsif ($KinstSortBy eq 'Prefix') {

$JoinClause = $JoinClause . " ORDER BY tbl\_instrument.PREFIX";

} else {

$JoinClause = $JoinClause . " ORDER BY tbl\_instrument.KINST";

}

#Figure out asc or desc

if ($KinstAscDesc eq 'Descending') {

$JoinClause = $JoinClause . " DESC";

} else {

$JoinClause = $JoinClause . " ASC";

}

#Finish the mysql statement

$SqlClause = $MainClause . $FromClause . $WhereClause . $JoinClause . ";' $CEDARDB";

# Jose Garcia /Debug

# $tt2=@Parameter;

# print "<p>$tt2</p>\n";

# print"<p>$SqlClause</p>\n";

#Open the SQL

open (FINDNAMES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for CreateKinstList");

#Loop through until done

while (<FINDNAMES>) {

#Get the current line ($\_) and parse it

($InstName,$Prefix,$KinstCode) = split /%/, $\_;

chop($KinstCode);

#Create the list item

print "<OPTION VALUE='" . $KinstCode . "'";

#See if the item has been selected

if ($ClearInstruments eq "") {

if ($Kinst eq $KinstCode) {

print " SELECTED";

}

}

#Specify the order of display

print ">" . $KinstCode . " - " . $Prefix . " - " . $InstName . "</OPTION>", "\n";

}

close (FINDNAMES);

#Finish the HTML for the select

print "</SELECT>", "\n";

#For testing

if ($DebugSQLClause eq "TRUE") {

print "<BR>\n" . $SqlClause . "\n";

}

}

sub DoKindats {

# Create the page for the Kindat selection

#

# &DoKindats;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Create a table

print "<TABLE BORDER='0' WIDTH='100%' CELLPADDING='1' CELLSPACING='1'>", "\n";

print "<TR>", "\n";

#For the left cell, put a header, the kindats table, and a "more info" button

print "<TD WIDTH='60%' VALIGN='top' ALIGN='left'>", "\n";

print "<P>", "\n";

print "Select one record type,<BR>", "\n";

print "or leave the box blank to select them all:</P>", "\n";

print "<P>Note: you need to click a button on the left to process your selection</P>", "\n";

print "<P>", "\n";

&CreateKindatList;

print "</P>", "\n";

print "</TD>", "\n";

#Add the right main page with "sort by", "Ascending/Descending", and "show" combo boxes,

#buttons to re-sort and clear the list, and a button to go to the world map

print "<TD WIDTH='40%' VALIGN='top' ALIGN='left'>", "\n";

#Create a "sort by" combo box

print "<P>", "\n";

print "Sort by:<BR>", "\n";

print "<SELECT NAME='KindatSortBy' SIZE='1'>", "\n";

print "<OPTION VALUE='Code'";

if ($KindatSortBy eq 'Code') {

print " SELECTED";

}

print ">Instrument code</OPTION>", "\n";

print "<OPTION VALUE='Kindat'";

if ($KindatSortBy eq 'Kindat') {

print " SELECTED";

}

print ">Record type code</OPTION>", "\n";

print "</SELECT>", "\n";

print "<BR>", "\n";

#Create an "Ascending/Descending" combo box

print "<SELECT NAME='KindatAscDesc' SIZE='1'>", "\n";

print "<OPTION VALUE='Ascending'";

if ($KindatAscDesc eq 'Ascending') {

print " SELECTED";

}

print ">Ascending</OPTION>", "\n";

print "<OPTION VALUE='Descending'";

if ($KindatAscDesc eq 'Descending') {

print " SELECTED";

}

print ">Descending</OPTION>", "\n";

print "</SELECT>", "\n";

print "<BR>", "\n";

#Create a "show" combo box

print "<SELECT NAME='KindatShow' SIZE='1'>", "\n";

print "<OPTION VALUE='All'";

if ($KindatShow eq 'All') {

print " SELECTED";

}

print ">All record type codes</OPTION>", "\n";

print "</SELECT>", "\n";

print "</P>", "\n";

#Give a hard break

print "<HR>", "\n";

#Let the user resort or clear the list

print "<P>", "\n";

print "<INPUT NAME='ResortKindats' VALUE='Re-sort list' TYPE='submit'>", "\n";

print "<BR>", "\n";

print "<INPUT NAME='ClearKindats' VALUE='Clear list' TYPE='submit'>", "\n";

print "</P>", "\n";

#Finish the right cell

print "</TD>", "\n";

#Close up the table

print "</TR>", "\n";

print "</TABLE>", "\n";

# Print the legend

print "<b>Legend:</b>\n";

print "<ul>\n";

print "<li>Nu=uncorrected electron density (Ne) from fine ht resolution power profiles.</li>\n";

print "<li>ACFs produce Ne, Te and Ti (Tr=Te/Ti) in long and short pulse lengths.</li>\n";

print "<li>The longer pulse lengths cover more altitude, and are better in the F region.</li>\n";

print "<li>Ni=ion density, which is either assumed or calculated from ACFs. </li>\n";

print "<li>CF=ACFs in EISCAT records (huge); use kindat+1000 for records w/o ACFs < 1994.</li>\n";

print "<li>Nn=any kind of neutral density except for Na (sodium) and Fe (iron). </li>\n";

print "<li>Tn,Vn=neutral temperature and winds.</li>\n";

print "<li>Vi=ion winds, related to electric fields (Ef), and electric potential (Ep), and electric current density (Je) and Joule heating (Qj).</li>\n";

print "<li>Sg=sigma or conductances and particle heating (Qp) are related to Ne</li>\n";

print "</ul>\n";

#Create some hidden variables

print "<INPUT TYPE=HIDDEN NAME='StartYear' VALUE=",$StartYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMonth' VALUE=",$StartMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartDay' VALUE=",$StartDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartHour' VALUE=",$StartHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMinute' VALUE=",$StartMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartSecond' VALUE=",$StartSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndYear' VALUE=",$EndYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMonth' VALUE=",$EndMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndDay' VALUE=",$EndDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndHour' VALUE=",$EndHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMinute' VALUE=",$EndMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndSecond' VALUE=",$EndSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kinst' VALUE=",$Kinst,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstSortBy' VALUE=",$KinstSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstAscDesc' VALUE=",$KinstAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstShow' VALUE=",$KinstShow,">", "\n";

foreach $par(@Parameter)

{

print "<INPUT TYPE=HIDDEN NAME='Parameter' VALUE=$par>\n",

}

print "<INPUT TYPE=HIDDEN NAME='ParameterSortBy' VALUE=",$ParameterSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterAscDesc' VALUE=",$ParameterAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterShow' VALUE=",$ParameterShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterSearch' VALUE=",$ParameterSearch,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='filter' VALUE=",$Filter,">", "\n";

}

sub CreateKindatList {

# Uses the MySQL meta-database to create a list of KINDAT codes

# at runtime.

#

# &CreateKindatList;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# all page variables

#Set up the Kinst list

print "<SELECT NAME='Kindat' SIZE='";

print "15";

print "'>", "\n";

#Create the mysql statement

$MainClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT concat(tbl\_record\_type.KINST,\"%\",tbl\_record\_type.KINDAT,\"%\",tbl\_record\_type.DESCRIPTION)";

$FromClause = " FROM tbl\_record\_info,tbl\_record\_type";

$WhereClause = " WHERE tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_record\_info.RECORD\_TYPE\_ID";

$JoinClause = "";

#See if the user has selected a date

if (($StartIndex != 1) || ($EndIndex != $LastIndex)) {

#Add to the FromClause

$FromClause = $FromClause . ",tbl\_file\_info,tbl\_cedar\_file,tbl\_date\_in\_file";

#Add to the WhereClause

$WhereClause = $WhereClause . " AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_file\_info.RECORD\_TYPE\_ID AND tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_file\_info.RECORD\_IN\_FILE\_ID=tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID";

#Add to the JoinClause

$JoinClause = " AND (tbl\_date\_in\_file.DATE\_ID >= " . $StartIndex. ") AND (tbl\_date\_in\_file.DATE\_ID <= " . $EndIndex .")";

}

#See if the user has specified any instruments

if ($Kinst ne "") {

$JoinClause = $JoinClause . " AND (tbl\_record\_type.KINST=" . $Kinst . ")";

}

#See if the user has specified any parameters

$tt7=@Parameter;

if ($tt7>0) {

$JoinClause = $JoinClause . " AND (";

$index=1;

foreach $par(@Parameter)

{

$JoinClause = $JoinClause . "(tbl\_record\_info.PARAMETER\_ID=" . $par . ")";

if($index<$tt7)

{

$JoinClause = $JoinClause . " OR ";

$index=$index+1;

}

}

$JoinClause = $JoinClause .") "

}

#Figure out sort by

if ($KindatSortBy eq 'Kindat') {

$JoinClause = $JoinClause . " ORDER BY tbl\_record\_type.KINDAT,tbl\_record\_type.KINST";

} else {

$JoinClause = $JoinClause . " ORDER BY tbl\_record\_type.KINST,tbl\_record\_type.KINDAT";

}

#Figure out asc or desc

if ($KindatAscDesc eq 'Descending') {

$JoinClause = $JoinClause . " DESC";

} else {

$JoinClause = $JoinClause . " ASC";

}

#Finish the mysql statement

$SqlClause = $MainClause . $FromClause . $WhereClause . $JoinClause . ";' $CEDARDB";

#Jose Garcia / Debug

#print "<p>$SqlClause</p>\n";

#Open the SQL

open (FINDNAMES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for CreateKindatList");

#Loop through until done

while (<FINDNAMES>) {

#Get the current line ($\_) and parse it

($KinstCode,$KindatCode,$KindatDesc) = split /%/, $\_;

chop($KindatDesc);

#Create the list item

print "<OPTION VALUE='" . $KinstCode . "/" . $KindatCode . "'";

#See if the item has been selected

if ($ClearKindats eq "") {

if ($Kindat eq ($KinstCode . "/" . $KindatCode)) {

print " SELECTED";

}

}

#Specify the order of display

print ">" . $KinstCode . " - " . $KindatCode . " " . $KindatDesc . "</OPTION>", "\n";

}

close (FINDNAMES);

#Finish the HTML for the select

print "</SELECT>", "\n";

#For testing

if ($DebugSQLClause eq "TRUE") {

print "<BR>\n" . $SqlClause . "\n";

}

}

sub DoParameters {

# Create the page for the parameter selection

#

# &DoParameters;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Create a table

print "<TABLE BORDER='0' WIDTH='100%' CELLPADDING='1' CELLSPACING='1'>", "\n";

print "<TR>", "\n";

#For the left cell, put a header, the parameter table, and a button for more info

print "<TD WIDTH='60%' VALIGN='top' ALIGN='left'>", "\n";

print "<P>", "\n";

print "Select one or more parameters,<BR>", "\n";

print "or leave the box blank to select them all:</P>", "\n";

print "<P>Note: you need to click a button on the left to process your selection</P>", "\n";

&CreateParameterList;

print "</P>", "\n";

print "</TD>", "\n";

#Add the right main page, with a kindat combo box and and button for more info,

#combo boxes for "sort by", "Ascending/Descending", and "show", a label for the

#search text box, the text box and its "go" button, and buttons to resort or clear the list

print "<TD WIDTH='40%' VALIGN='top' ALIGN='left'>", "\n";

#Create a "sort by" combo box

print "<P>", "\n";

print "Sort by:<BR>", "\n";

print "<SELECT NAME='ParameterSortBy' SIZE='1'>", "\n";

print "<OPTION VALUE='Code'";

if ($ParameterSortBy eq 'Code') {

print " SELECTED";

}

print ">Parameter code</OPTION>", "\n";

print "<OPTION VALUE='Madrigal'";

if ($ParameterSortBy eq 'Madrigal') {

print " SELECTED";

}

print ">Madrigal name</OPTION>", "\n";

print "<OPTION VALUE='Description'";

if ($ParameterSortBy eq 'Description') {

print " SELECTED";

}

print ">Description</OPTION>", "\n";

print "</SELECT>", "\n";

print "<BR>", "\n";

#Create an "Ascending/Descending" combo box

print "<SELECT NAME='ParameterAscDesc' SIZE='1'>", "\n";

print "<OPTION VALUE='Ascending'";

if ($ParameterAscDesc eq 'Ascending') {

print " SELECTED";

}

print ">Ascending</OPTION>", "\n";

print "<OPTION VALUE='Descending'";

if ($ParameterAscDesc eq 'Descending') {

print " SELECTED";

}

print ">Descending</OPTION>", "\n";

print "</SELECT>", "\n";

print "<BR>", "\n";

#Create a "show" combo box

print "<SELECT NAME='ParameterShow' SIZE='1'>", "\n";

print "<OPTION VALUE='All'";

if ($ParameterShow eq 'All') {

print " SELECTED";

}

print ">All Parameters</OPTION>", "\n";

print "</SELECT>", "\n";

print "</P>", "\n";

#Create a searching text box

#print "<P>", "\n";

#print "Search (with wildcards):<BR>", "\n";

#print "<INPUT TYPE='text' NAME='ParameterSearch' SIZE='10' VALUE='" . $ParameterSearch . "'>", "\n";

#print "<INPUT NAME='ParameterReSearch' VALUE='Search' TYPE='submit'>", "\n";

#print "</P>", "\n";

#Give a hard break

print "<HR>", "\n";

#Let the user resort or clear the list

print "<P>", "\n";

print "<INPUT NAME='ResortParameters' VALUE='Re-sort list' TYPE='submit'>", "\n";

print "<BR>", "\n";

print "<INPUT NAME='ClearParameters' VALUE='Clear list' TYPE='submit'>", "\n";

print "</P>", "\n";

#Finish the right cell

print "</TD>", "\n";

#Close up the table

print "</TR>", "\n";

print "</TABLE>", "\n";

#Create some hidden variables

print "<INPUT TYPE=HIDDEN NAME='StartYear' VALUE=",$StartYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMonth' VALUE=",$StartMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartDay' VALUE=",$StartDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartHour' VALUE=",$StartHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMinute' VALUE=",$StartMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartSecond' VALUE=",$StartSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndYear' VALUE=",$EndYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMonth' VALUE=",$EndMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndDay' VALUE=",$EndDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndHour' VALUE=",$EndHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMinute' VALUE=",$EndMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndSecond' VALUE=",$EndSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kinst' VALUE=",$Kinst,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstSortBy' VALUE=",$KinstSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstAscDesc' VALUE=",$KinstAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstShow' VALUE=",$KinstShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kindat' VALUE=",$Kindat,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatSortBy' VALUE=",$KindatSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatAscDesc' VALUE=",$KindatAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatShow' VALUE=",$KindatShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='filter' VALUE=",$Filter,">", "\n";

}

sub CreateParameterList {

# Uses the MySQL meta-database to create a list of parameters

# at runtime.

#

# &CreateParameterList;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# all page variables

#Set up the parameter list

print "<SELECT NAME='Parameter' SIZE='";

print "15";

print "' MULTIPLE>", "\n";

#Create the mysql statement

$MainClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT concat(tbl\_parameter\_code.PARAMETER\_ID,\"%\",tbl\_parameter\_code.MADRIGAL\_NAME,\"%\",tbl\_parameter\_code.LONG\_NAME)";

$FromClause = " FROM tbl\_parameter\_code";

$WhereClause = " WHERE (1)";

$JoinClause = "";

#See if the user has chosen anything yet

if (($Kinst ne "") || ($Kindat ne "") || ($StartIndex != 1) || ($EndIndex != $LastIndex)) {

#Add to the clauses

$FromClause = $FromClause . ",tbl\_record\_info,tbl\_record\_type";

$WhereClause = $WhereClause . " AND tbl\_parameter\_code.PARAMETER\_ID=tbl\_record\_info.PARAMETER\_ID AND tbl\_record\_info.RECORD\_TYPE\_ID=tbl\_record\_type.RECORD\_TYPE\_ID";

}

#See if the user has selected a date

if (($StartIndex != 1) || ($EndIndex != $LastIndex)) {

#Add to the FromClause

$FromClause = $FromClause . ",tbl\_file\_info,tbl\_cedar\_file,tbl\_date\_in\_file";

#Add to the WhereClause

$WhereClause = $WhereClause . " AND tbl\_record\_type.RECORD\_TYPE\_ID=tbl\_file\_info.RECORD\_TYPE\_ID AND tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_file\_info.RECORD\_IN\_FILE\_ID=tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID";

#Add to the JoinClause

$JoinClause = " AND (tbl\_date\_in\_file.DATE\_ID >= " . $StartIndex. ") AND (tbl\_date\_in\_file.DATE\_ID <= " . $EndIndex .")";

}

#See if the user has specified any instruments

if ($Kinst ne "") {

$JoinClause = $JoinClause . " AND (tbl\_record\_type.KINST=" . $Kinst . ")";

}

#See if the user has specified any record\_type

if ($Kindat ne "") {

#Do the split

($KinstPart,$KindatPart) = split(/\//,$Kindat);

$JoinClause = $JoinClause . " AND ((tbl\_record\_type.KINST=" . $KinstPart . ") AND (tbl\_record\_type.KINDAT=" . $KindatPart . "))";

}

#Figure out sort by

if ($ParameterSortBy eq 'Description') {

$JoinClause = $JoinClause . " ORDER BY tbl\_parameter\_code.LONG\_NAME";

} elsif ($ParameterSortBy eq 'Madrigal') {

$JoinClause = $JoinClause . " ORDER BY tbl\_parameter\_code.MADRIGAL\_NAME";

} else {

$JoinClause = $JoinClause . " ORDER BY tbl\_parameter\_code.PARAMETER\_ID";

}

#Figure out asc or desc

if ($ParameterAscDesc eq 'Descending') {

$JoinClause = $JoinClause . " DESC";

} else {

$JoinClause = $JoinClause . " ASC";

}

#Finish the mysql statement

$SqlClause = $MainClause . $FromClause . $WhereClause . $JoinClause . ";' $CEDARDB";

#Open the SQL

open (FINDNAMES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for CreateParameterList");

#Loop through until done

while (<FINDNAMES>) {

#Get the current line ($\_) and parse it

($Id,$MadrigalName,$LongName) = split /%/, $\_;

chop($LongName);

#Create the list item

print "<OPTION VALUE='" . $Id . "'";

#See if the item has been selected

if ($ClearParameters eq "") {

foreach $par(@Parameter)

{

if ($par == $Id) {

print " SELECTED";

}

}

}

#Specify the order of display

print ">" . $Id . " - " . $MadrigalName . " - " . $LongName . "</OPTION>", "\n";

}

close (FINDNAMES);

#Finish the HTML for the select

print "</SELECT>", "\n";

#For testing

if ($DebugSQLClause eq "TRUE") {

print "<BR>\n" . $SqlClause . "\n";

}

}

sub DoGo {

# Create the page for the data to be returned

#

# &DoGo;

#

# INPUT:

# (none)

# RETURNS:

# (none)

# EXTERNALS:

# (none)

#Create a table with two columns

print "<TABLE BORDER='0' CELLPADDING='1' CELLSPACING='1' WIDTH='100%'>", "\n";

print "<TR>", "\n";

#Do the left column

print "<TD WIDTH='50%' VALIGN='top' ALIGN='left'>", "\n";

#Create a header

print "<P>You have chosen the following selections:</P>", "\n";

#Show the selections

print "<P>", "\n";

print "Starting Date\_Time: " . $EffectiveStartYear . "," . $EffectiveStartMonth . "," . $EffectiveStartDay . "," . $EffectiveStartHour . "," . $EffectiveStartMinute . "," . $EffectiveStartSecond . "<BR>", "\n";

print "Ending Date\_Time: " . $EffectiveEndYear . "," . $EffectiveEndMonth . "," . $EffectiveEndDay . "," . $EffectiveEndHour . "," . $EffectiveEndMinute . "," . $EffectiveEndSecond, "<BR>\n";

#Show the instruments

print "<P>", "\n";

print "Instrument(s):", "\n";

print "<UL>", "\n";

if ($Kinst ne "") {

print "<LI>" . $Kinst . "</LI>", "\n";

} else {

#Print "All instruments"

print "<LI>All instruments</LI>", "\n";

}

print "</UL>", "\n";

print "</P>", "\n";

#Show the KINDATs

print "<P>", "\n";

print "Record Type(s):" ,"\n";

print "<UL>", "\n";

if ($Kindat ne "") {

print "<LI>" . $Kindat . "</LI>", "\n";

} else {

#Print "All record types"

print "<LI>All record types</LI>", "\n";

}

print "</UL>", "\n";

print "</P>", "\n";

#Show the parameters

print "<P>", "\n";

print "Parameter(s):", "\n";

print "<UL>", "\n";

$tt8=@Parameter;

if ($tt8>0) {

foreach $par(@Parameter)

{

print "<LI>" . $par . "</LI>", "\n";

}

} else {

#Print "All parameters"

print "<LI>All parameters</LI>", "\n";

}

print "</UL>", "\n";

print "</P>", "\n";

#Tell the user how big the data file will be

#print "<P>", "\n";

#print "This query will return:", "\n";

#print "<UL>", "\n";

#print "<LI>0 MB ASCII</LI>", "\n";

#print "</UL>", "\n";

#print "</P>", "\n";

#Give the user a button to redo this query

print "<FORM ACTION='" . $cgi\_site . "/cedarweb.pl' METHOD='POST'>", "\n";

print "<INPUT NAME='Modify' VALUE='Modify Query' TYPE='submit'>", "\n";

#Add the hidden variables

print "<INPUT TYPE=HIDDEN NAME='StartYear' VALUE=",$StartYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMonth' VALUE=",$StartMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartDay' VALUE=",$StartDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartHour' VALUE=",$StartHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartMinute' VALUE=",$StartMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='StartSecond' VALUE=",$StartSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndYear' VALUE=",$EndYear,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMonth' VALUE=",$EndMonth,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndDay' VALUE=",$EndDay,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndHour' VALUE=",$EndHour,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndMinute' VALUE=",$EndMinute,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='EndSecond' VALUE=",$EndSecond,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kinst' VALUE=",$Kinst,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstSortBy' VALUE=",$KinstSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstAscDesc' VALUE=",$KinstAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KinstShow' VALUE=",$KinstShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='Kindat' VALUE=",$Kindat,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatSortBy' VALUE=",$KindatSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatAscDesc' VALUE=",$KindatAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='KindatShow' VALUE=",$KindatShow,">", "\n";

foreach $par(@Parameter)

{

print "<INPUT TYPE=HIDDEN NAME='Parameter' VALUE=$par>\n",

}

print "<INPUT TYPE=HIDDEN NAME='ParameterSortBy' VALUE=",$ParameterSortBy,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterAscDesc' VALUE=",$ParameterAscDesc,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterShow' VALUE=",$ParameterShow,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='ParameterSearch' VALUE=",$ParameterSearch,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='filter' VALUE=",$Filter,">", "\n";

#End the form

print "</FORM>", "\n";

#Give the user a button to clear the query

print "<FORM ACTION='" . $cgi\_site . "/cedarweb.pl' METHOD='POST'>", "\n";

print "<INPUT NAME='Clear' VALUE='Clear Query' TYPE='submit'>", "\n";

print "</FORM>", "\n";

print "</P>", "\n";

#Finish the cell

print "</TD>", "\n";

#Do the right column

print "<TD WIDTH='50%' VALIGN='top' ALIGN='left'>", "\n";

#Put all of this into a form

print "<FORM ACTION='" . $cgi\_site . "/Submit.pl' METHOD='POST'>", "\n";

#Create a selection combo for the filter

print "<P>", "\n";

print "Return data as (INFO= header/catalog, TAB= ascii similar to DB ascii with labels):<BR>", "\n";

print "<SELECT NAME='filter' SIZE='1'>", "\n";

print "<OPTION VALUE='INFO'";

if ($Filter eq "INFO") {

print " SELECTED";

}

print ">INFO - Information about the data</OPTION>", "\n";

print "<OPTION VALUE='DAS'";

if ($Filter eq "DAS") {

print " SELECTED";

}

print ">DAS - DODS Attribute Service</OPTION>", "\n";

print "<OPTION VALUE='DDS'";

if ($Filter eq "DDS") {

print " SELECTED";

}

print ">DDS - DODS Descriptor Service</OPTION>", "\n";

print "<OPTION VALUE='DODS'";

if ($Filter eq "DODS") {

print " SELECTED";

}

print ">DODS - DODS Data Service (binary)</OPTION>", "\n";

print "<OPTION VALUE='ASC'";

if ($Filter eq "ASC") {

print " SELECTED";

}

print ">ASC - DODS Data Service (ascii)</OPTION>", "\n";

print "<OPTION VALUE='TAB'";

if (($Filter eq "") || ($Filter eq "TAB")) {

print " SELECTED";

}

print ">TAB - DODS Data Service (tab delimited)</OPTION>", "\n";

print "<OPTION VALUE='HELP'";

if ($Filter eq "HELP") {

print " SELECTED";

}

print ">HELP - DODS Help File</OPTION>", "\n";

print "<OPTION VALUE='VER'";

if ($Filter eq "VER") {

print " SELECTED";

}

print ">VER - DODS Version file</OPTION>", "\n";

print "</SELECT>", "\n";

print "</P>", "\n";

#Create a button to actually get the file

print "<P>", "\n";

print "<INPUT NAME='GetFile' VALUE='Get the data file(s) with these options' TYPE='submit'>", "\n";

print "</P>", "\n";

print "<P>", "\n";

print "NOTE: If you use 'Back' on your browser, you will get a ";

print "<b>'Data Missing'</b>";

print " message from your browser since the access procedure disallows caching.";

print " Click on the 'Reload' button in your browser to restore.";

print " Also, the URL itself can be changed in the 'Location' edit area in your browser.";

print "</P>", "\n";

#Create the hidden variables

$FullStart = $EffectiveStartYear . "," . $EffectiveStartMonth . $EffectiveStartDay . "," . $EffectiveStartHour . $EffectiveStartMinute . "," . $EffectiveStartSecond;

$FullEnd = $EffectiveEndYear . "," . $EffectiveEndMonth . $EffectiveEndDay . "," . $EffectiveEndHour . $EffectiveEndMinute . "," . $EffectiveEndSecond;

#Send the date as 1990,0130,2015,0000,1991,0220,2359,5999

$FullDate = $FullStart . "," . $FullEnd;

print "<INPUT TYPE=HIDDEN NAME='date' VALUE=" . $FullDate,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='instrument' VALUE=",$Kinst,">", "\n";

print "<INPUT TYPE=HIDDEN NAME='record\_type' VALUE=",$Kindat,">", "\n";

foreach $par(@Parameter)

{

print "<INPUT TYPE=HIDDEN NAME='Parameter' VALUE=$par>\n",

}

#End the form

print "</FORM>", "\n";

#Finish the cell

print "</TD>", "\n";

#Finish the table

print "</TR>", "\n";

print "</TABLE>", "\n";

}

sub FindLastDayOfMonth

{

# Given a month and year, find the last day of that month

#

# &FindLastDayOfMonth;

#

# INPUT:

# $month = the month we are checking (with two digits)

# $year = the year we are checking (with all four digits)

# RETURNS:

# A number representing the last day of the month (28--31)

# If the data is invalid, 0 will be returned

# EXTERNALS:

# (none)

my $inmonth = shift;

$mymonth = 0 + $inmonth;

my $inyear = shift;

$myyear = 0 + $inyear;

if (($mymonth == 1) || ($mymonth == 3) || ($mymonth == 5) || ($mymonth == 7) || ($mymonth == 8) || ($mymonth == 10) || ($mymonth == 12)) {

#There are 31 days

return (31);

} elsif (($mymonth == 4) || ($mymonth == 6) || ($mymonth == 9) || ($mymonth == 11)) {

#There are 30 days

return (30);

} else {

#See if that year is a leap year

if (&IsLeapYear($myyear)) {

#It is

return (29);

} else {

# It is not a leap year

return (28);

}

}

}

sub FindDay

{

# Given a month and a year, find which day of the month it is (0-6)

#

# &FindDay

#

# INPUT:

# $month = the month we are checking (with two digits)

# $year = the year we are checking (with all four digits)

# RETURNS:

# A number representing the day of the week (0-6)

# EXTERNALS:

# (none)

my $inmonth = shift;

$mymonth = 0 + $inmonth;

my $inyear = shift;

$myyear = 0 + $inyear;

#Use the localtime command to find the date

$TempDate = parsedate($mymonth."/01/".$myyear);

@MyDate = localtime($TempDate);

$MyWeekDay = @MyDate[6];

#This is the value (0 to 6)

return ($MyWeekDay);

}

sub IsLeapYear

{

# Cute little program to figure out is the passed-in year is a leap year

#

# &IsLeapYear;

#

# INPUT:

# $year = the year we are checking (with all four digits)

# RETURNS:

# 0 = the year is not a leap year

# 1 = the year \*is\* a leap year

# EXTERNALS:

# (none)

my $inyear = shift;

$myyear = 0 + $inyear;

$leap = ($myyear%4 == 0 && $myyear%100 != 0 || $myyear%400 == 0) ;

return $leap;

}

sub DoExit {

# Cute little program to exit smoothly

#

# &DoExit;

#

# INPUT:

# $ErrMessage = Message to print to error file

# RETURNS:

# (none)

# EXTERNALS:

# (none)

my $ErrMessage = shift;

print $ErrMessage . "\n";

#Open up a file

#if (open (ERRFILE,">error.log")) {

# print ERRFILE $ErrMessage;

#}

#close (ERRFILE);

exit(1);

}

1;

## Submit.pl

#!/opt/local/bin/perl

#This is the user portal for

#the Cedarweb interface

#Patrick Kellogg pkellogg@hao.ucar.edu x1544

#Version 1.0

#Oct 12, 1999

#Essential lines for security

use CGI;

$ENV{PATH} = '/bin:/usr/bin:/cedar/e/mysql/bin';

$ENV{IFS} = "" if $ENV{IFS} ne "";

#Global variables

#

#program\_name: name of this program.

$program\_name=$0;

#

#Log\_file: name of the file for dumping error messages.

$log\_file=$program\_name.".log";

#

#mysql\_home: home directory for MySQL binary files.

$mysql\_home="/cedar/e/mysql/bin/";

#

#mysql\_client: name of the MySQL client program.

$mysql\_client=$mysql\_home."mysql";

#

#user: Defines the user ID to connect to mysqld.

$user="madrigal";

#

# pass: The password for the user which is getting connected to mysqld.

$pass="madrigal";

#

# host: Defines the host where mysqld is running.

$host="cedar";

#

# CEDARDB: the database to connect

$CEDARDB = "CEDAR\_CATALOG";

# DODS: Gives the location of DODS

$DODS = "http://cedarweb.hao.ucar.edu/cedarweb1-cgi-bin/nph-cedar/data/stage/";

#Create a new query object (see CGI.pl)

$query = new CGI;

#Parse the fields

$date = $query->param('date');

$instrument = $query->param('instrument');

$record\_type = $query->param('record\_type');

@parameters = $query->param('Parameter');

$filter = $query->param('filter');

#Parse the date, too

($StartYear,$StartMonthDay,$StartHourMinute,$StartSecond,$EndYear,$EndMonthDay,$EndHourMinute,$EndSecond) = split /,/, $date;

#Use substr(EXPR, OFFSET, LENGTH)

$StartMonth = substr($StartMonthDay, 0, 2);

$StartDay = substr($StartMonthDay, 2, 2);

$EndMonth = substr($EndMonthDay, 0, 2);

$EndDay = substr($EndMonthDay, 2, 2);

#Create an error flag and clear it

@error = "";

#If there is an error, show the user a page

if ($error ne "") {

#Create the page

print "Content-type: text/html", "\n\n";

print "<HTML>", "\n\n";

print "<HEAD>", "\n";

#Don't let the server cache this page

#print "<META HTTP-EQUIV='Pragma' CONTENT='no-cache'>", "\n";

#print "<META HTTP-EQUIV='Expires' CONTENT='Mon, 01 Jan 1990 00:00:01 GMT'>", "\n";

#Give the title

print "<TITLE>Error Using the CEDAR Database</TITLE>", "\n";

print "</HEAD>", "\n\n";

#Start the body section

print "<BODY BGCOLOR='#C0C0C0'>", "\n";

#Give the user a warning message

print "<P ALIGN='center'>Error passing parameters to the CEDAR database</P>", "\n";

#Do the footer

print "<HR>", "\n";

#Create a table

print "<TABLE BORDER='0' CELLPADDING='1' CELLSPACING='1' WIDTH='100%'>", "\n";

print "<TR>", "\n";

print "<TD WIDTH='50%' VALIGN='top' ALIGN='left'>", "\n";

#Do the left side (return and copyright)

print "<P><A HREF='http://cedarweb.hao.ucar.edu'><IMG SRC='/icons/home03.gif' WIDTH=36 HEIGHT=24></A><A HREF='http://cedarweb.hao.ucar.edu'> <I>Return to the CEDAR homepage</I></A><BR>", "\n";

print "<A HREF='http://www.hao.ucar.edu/public/home/copyright.html'>Copyright 1999, NCAR. </A></P>", "\n";

print "</TD>", "\n";

#Do the right side (approval and mailto)

print "<TD WIDTH='50%' VALIGN='top' ALIGN='right'>", "\n";

print "<P>-Approved by Peter Fox<BR>", "\n";

print "<I>-Version 2.0 by </I><A HREF='mailto:pkellogg\@hao.ucar.edu'><I>Patrick Kellogg</I></A></P>", "\n";

print "</TD>", "\n";

print "</TABLE>", "\n";

print "</BODY>", "\n\n";

print "</HTML>", "\n";

} else {

#Redirect the user to the correct page

print "Content-type: text/html", "\n\n";

print "<HTML>", "\n\n";

print "<HEAD>", "\n";

#Don't let the server cache this page

#print "<META HTTP-EQUIV='Pragma' CONTENT='no-cache'>", "\n";

#print "<META HTTP-EQUIV='Expires' CONTENT='Mon, 01 Jan 1990 00:00:01 GMT'>", "\n";

#Give the title

print "<TITLE>Results From the CEDAR Database</TITLE>", "\n";

print "</HEAD>", "\n\n";

#Start the body section

print "<BODY BGCOLOR='#C0C0C0'>", "\n";

#Give the user a message

print "<P>Date: " . $date . "</P>", "\n";

print "<P>Instrument: " . $instrument . "</P>", "\n";

print "<P>Record type: " . $record\_type . "</P>", "\n";

print "<P>Parameters: ";

$number\_of\_parameters\_in\_list=@parameters;

if ($number\_of\_parameters\_in\_list > 0)

{

$index=1;

foreach $par(@parameters)

{

print $par;

if($index<$number\_of\_parameters\_in\_list)

{

print ",";

$index=$index+1;

}

}

print "</P>", "\n";

}

print "<P>Filter: " . $filter . "</P>", "\n";

#Query the mysql database to find the starting and ending date indicies

$StartClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DATE\_ID from tbl\_date WHERE (YEAR = " . $StartYear . ") AND (MONTH = " . $StartMonth . ") AND (DAY = " . $StartDay . ");' $CEDARDB";

#Open the SQL

open (STARTINDEX, "$StartClause|") || DoExit ("MySQL could not open MySQL for StartIndex");

$StartIndex = 1;

while (<STARTINDEX>) {

$StartIndex = $\_;

chop($StartIndex);

}

$EndClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DATE\_ID from tbl\_date WHERE (YEAR = " . $EndYear . ") AND (MONTH = " . $EndMonth . ") AND (DAY = " . $EndDay . ");' $CEDARDB";

#Open the SQL

open (ENDINDEX, "$EndClause|") || DoExit ("MySQL could not open MySQL for EndIndex");

$EndIndex = 1;

while (<ENDINDEX>) {

$EndIndex = $\_;

chop($EndIndex);

}

##Find the index of the very last record in the database

#$LastClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DATE\_ID from tbl\_date WHERE (YEAR = " . $final\_year . ") AND (MONTH = 12) AND (DAY = 31);' $CEDARDB";

##Open the SQL

#open (LASTINDEX, "$LastClause|") || DoExit ("MySQL could not open MySQL for EndIndex");

#$LastIndex = 1;

#while (<LASTINDEX>) {

# $LastIndex = $\_;

# chop($LastIndex);

#}

#Create the mysql statement

$SqlClause = $mysql\_client . " -s -B -u$user -p$pass -e'SELECT DISTINCT tbl\_cedar\_file.FILE\_NAME";

$SqlClause = $SqlClause . " FROM tbl\_date\_in\_file,tbl\_cedar\_file,tbl\_file\_info,tbl\_record\_type";

$SqlClause = $SqlClause . " WHERE tbl\_date\_in\_file.RECORD\_IN\_FILE\_ID=tbl\_file\_info.RECORD\_IN\_FILE\_ID and tbl\_file\_info.FILE\_ID=tbl\_cedar\_file.FILE\_ID AND tbl\_cedar\_file.FILE\_ID=tbl\_file\_info.FILE\_ID AND tbl\_file\_info.RECORD\_TYPE\_ID=tbl\_record\_type.RECORD\_TYPE\_ID";

$SqlClause = $SqlClause . " AND (tbl\_date\_in\_file.DATE\_ID >= " . $StartIndex. ") AND (tbl\_date\_in\_file.DATE\_ID <= " . $EndIndex . ")";

#Add the Instrument

if ($instrument ne "") {

$SqlClause = $SqlClause . " AND (tbl\_record\_type.KINST=" . $instrument . ")";

}

#Add the KINST/KINDATS

if ($record\_type ne "") {

@MyParse = split(/,/,$record\_type);

#Pop the first key

@ToAdd = split /\//, pop(@MyParse);

$ExpandIt = "((tbl\_record\_type.KINST=" . $ToAdd[0] . ") AND (tbl\_record\_type.KINDAT=" . $ToAdd[1] . "))";

while (@MyParse) {

@ToAdd = split /\//, pop(@MyParse);

$ExpandIt = $ExpandIt . " OR ((tbl\_record\_type.KINST=" . $ToAdd[0] . ") AND (tbl\_record\_type.KINDAT=" . $ToAdd[1] . "))";

}

$SqlClause = $SqlClause . " AND (" . $ExpandIt . ")";

}

#Finish the query

$SqlClause = $SqlClause . ";' $CEDARDB";

#Open the SQL

open (FINDFILES, "$SqlClause|") || DoExit ("MySQL could not open MySQL for FindFiles");

#Start listing the URLs

print "<HR>", "\n";

$TotalString = "";

$AnyFound = 0;

#Loop through until done

while (<FINDFILES>) {

#We found at least one URL

$AnyFound = ($AnyFound + 1);

#Get the current line ($\_) and parse it

$NextFile = $\_;

chop($NextFile);

#Create the URLs

$URL = $DODS . $NextFile . ".cbf." . (lc $filter) . "?";

$URL = $URL . "date(" . $date . ")";

#See if there are any record\_types

if ($record\_type ne "") {

$URL = $URL . ";record\_type(" . $record\_type . ")";

}

#See if there are any parameters

$number\_of\_parameters\_in\_list=@parameters;

if ($number\_of\_parameters\_in\_list > 0) {

$URL = $URL . ";parameters(";

$index=1;

foreach $par(@parameters)

{

$URL = $URL. $par;

if($index<$number\_of\_parameters\_in\_list)

{

$URL = $URL. ",";

$index=$index+1;

}

}

$URL = $URL . ")";

}

$TotalString = $TotalString . "<A HREF='" . $URL . "'>" . $URL . "</A><BR>", "\n";

}

#See if anything was found

if ($AnyFound == 0) {

#No files were found for that query

print "<P>No data found for those query parameters</P>", "\n";

} elsif ($AnyFound == 1 ) {

#Print a message for 1 URL

print "<P>Please click on the URL below to retrieve your data<BR>", "\n";

} else {

#Print a message for > 1 URL

print "<P>Please click on each of the URLs below to retrieve your data<BR>", "\n";

}

#Print the URLs

print "<ul>";

print "<li>On some browsers, using the left mouse button while holding down the SHIFT key or using the right mouse button and then selecting the option \"Save link as\" ";

print "will send the data to a user-specified file instead of to the browser window.", "\n";

print "<li>If you use 'Back' on your browser, you will get a ";

print "<b>'Data Missing'</b>";

print " message from your browser since the access procedure disallows caching.";

print " Click on the 'Reload' button in your browser to restore.";

print " <li>Also, the URL itself can be changed in the 'Location' edit area in your browser.";

print "</ul>";

print "<P>", "\n";

print $TotalString;

print "</P>", "\n";

#Give the user a button to do another query

print "<P>", "\n";

print "<FORM ACTION='http://cedarweb.hao.ucar.edu/cedarweb-cgi-bin/cedarweb.pl' METHOD='POST'>", "\n";

print "<INPUT NAME='DoAnother' VALUE='Do Another Query' TYPE='submit'>", "\n";

print "</FORM>", "\n";

print "</P>", "\n";

#Do the footer

print "<HR>", "\n";

#Create a table

print "<TABLE BORDER='0' CELLPADDING='1' CELLSPACING='1' WIDTH='100%'>", "\n";

print "<TR>", "\n";

print "<TD WIDTH='50%' VALIGN='top' ALIGN='left'>", "\n";

#Do the left side (return and copyright)

print "<P><A HREF='http://cedarweb.hao.ucar.edu'><IMG SRC='/icons/home03.gif' WIDTH=36 HEIGHT=24></A><A HREF='http://cedarweb.hao.ucar.edu'> <I>Return to the CEDAR homepage</I></A><BR>", "\n";

print "<A HREF='http://www.hao.ucar.edu/public/home/copyright.html'>Copyright 2001, NCAR. </A></P>", "\n";

print "</TD>", "\n";

#Do the right side (approval and mailto)

print "<TD WIDTH='50%' VALIGN='top' ALIGN='right'>", "\n";

print "<P>-Approved by Peter Fox<BR>", "\n";

print "<I>-Version 2.0 by </I><A HREF='mailto:pkellogg\@hao.ucar.edu'><I>Patrick Kellogg</I></A></P>", "\n";

print "</TD>", "\n";

print "</TABLE>", "\n";

#Close the html

print "</BODY>", "\n\n";

print "</HTML>", "\n";

}

exit(0);