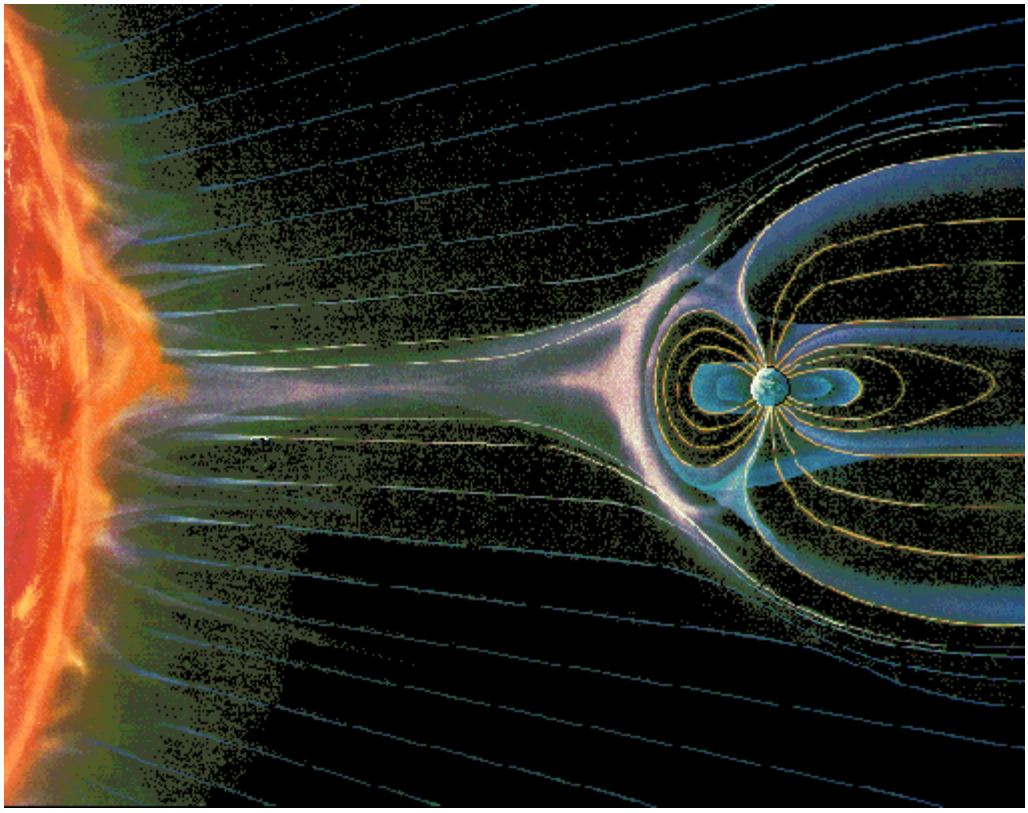
Space Physics Dataset Browser
Searching Distributed Datasets



Matthew Marquissee
Undergrad, Mathematics and Computer Science
University of Illinois at Urbana-Champaign
Achieving Competency in Careers in Engineering and Space Sciences (AAAS)
NASA Goddard Space Flight Center
Code 632 – Space Physics Data Facility
Advisor: Robert M. Candey

Abstract

The Space Physics Data Facility (SPDF) manages and distributes a variety of data products geared toward the study of plasma around the Earth and its interaction with solar phenomena. Such products include CDAWeb, a user-friendly web service that plots many variables over time, and SSCWeb, a service that can locate a certain spacecraft in the magnetosphere. These two systems alone give researchers in physics enormous insight into the turbulent frontier between Earth and our Sun. My mentor has proposed a new service as part of a future Space Physics Virtual Observatory, designed to incorporate and augment already existing systems. The goal is to combine abstraction with depth. The user of this system will not need to know where or how the source data is stored. Given the name of a spacecraft 's dataset, a time range, and a desired output format, it is the system's task to search through as much source data as possible, combine it, and present it in the best possible way. A researcher can focus on space physics research and ignore the implementation details.

However, the servers containing the source data, owned by several different organizations, are not synchronized. My project was to develop a flexible schema, using XML, describing how to find desired files in a specific dataset. In addition to providing a means to generate XML files for each data site, I was to use these files to limit search time and server load. The XML files, once generated, provide an instant tree hierarchy to search, performing most of the work without any impact on the server itself.

The heart of my project, dubbed the "Space Physics Dataset Browser", consists of four major phases. First, input is gathered from the user interface, which allows a user to navigate through the entire XML tree structure with ease. Second, a dataset descriptor and time range is sent into the search method. This search returns a full path to the desired dataset on a remote server. Third, directory listings on the remote server are scanned, and the URLs to files with desired data are returned. Lastly, these URLs are displayed to the user or exported in other formats.

There is still much work to be done before going operational. My project is just one fundamental part of the Observatory. Another key issue being solved by the SPDF is efficient conversion between file formats. The overall project will greatly increase physicists' potential to understand the Sun's impact on our outer atmosphere.

Table of Contents

1 Introduction

1.1 Background

The interaction between the Earth's upper atmosphere and the dynamic Sun is a crucial balance, and it is pertinent that scientists monitor and study these phenomena. Physicists have been analyzing minute changes in the magnetosphere region with the aide of special "eyes" in orbit. Satellites including Geotail, Polar, ACE, and numerous others, allow them to monitor Earth's shield of plasma with carefully calibrated sensors. However, the raw data must first be processed and made easily available before the researchers can incorporate it into their research. The Space Physics Data Facility (SPDF) in Code 632 performs part of this crucial task.

The SPDF "has its primary intent to lead in the definition, development, operation and promotion of collaborative efforts in the collection and utilization of space physics data and models" (SPDF). The two central data gathering services currently available to researchers are CDAWeb and SSCWeb. The Coordinated Data Analysis Web (CDAWeb) service offers a dazzling array of data comparison and plotting capabilities for numerous datasets from orbiting spacecraft. Physics researchers simply select data sources and specify variables, and the results are plotted in the appropriate graphical form. The Satellite Situation Center Web (SSCWeb) provides a way to determine through which magnetospheric region a certain spacecraft has or will pass. This information determines, among other things, whether solar activity has interfered with results or will cause damage to spacecraft. With access to CDAWeb and SSCWeb, research scientists around the world gain significant insight into the inner workings of the upper atmosphere, ionosphere, and magnetosphere.

Robert Candey, my advisor, has proposed a new addition to the services provided by the SPDF that offers even wider access to data and more capability for its analysis. This service will search through a wealth of data and ultimately present it in any desired science data format such as CDF or HDF or FITS and plot it through the CDAWeb service. A researcher's job will be greatly simplified, as this one interface would combine all of the data from multiple sources, which he or she would have had to consult

before. One fundamental piece of a future Virtual Observatory, which I was responsible for, is this collection of the data from multiple sources for further processing. It is this part on which I will now focus.

1.2 Issues and Motivations

In considering the data gathering process, one major issue is the synchronization of the sources of raw satellite data. With more data comes different ways to store data. Certain questions surface: How is the hierarchy of directories structured on a remote server? What file naming conventions do files in each and every dataset follow? In which data format are the files stored? If the data servers were all managed by a single organization, this could be done fairly easily. However, this is certainly not the case. Some important space physics data stores are managed by external organizations, not by NASA. One example is the TIMED satellite that is run out of the Advanced Physics Laboratory at Johns Hopkins University. If these crucial sources were incorporated into the easy-to-use Observatory framework, the researcher's perspective would be much broadened. The key to science is the availability to consult multiple sources of quality data, and this was considered.

Another major motivation of the project is abstraction. The user of this system should not be required to know where or how the source data is stored. Given the name of a spacecraft's dataset, a time range, and a desired output format, it is the system's task to search through as much source data as possible, combine it, and present it in the best possible way. A researcher can then focus on space physics theory and ignore the implementation details. Additionally, the user would not be given the opportunity to tamper, accidentally or otherwise, with the data site. This is a fundamental aspect of data modeling.

It is crucial to keep any data model representing distributed datasets as simple as possible but also to make it flexible. Keeping this in mind, the most important attributes of a dataset have been considered, and the way each attribute is stored preserves flexibility. This simplicity and flexibility allows both data users and maintainers to perform their tasks more quickly and easily. In considering this and the other issues and motivations, suitable software tools have been selected for the project.

1.3 Tools and Resources

For a simple, flexible way to synchronize various distributed datasets, the natural choice was to use metadata in the form of Extensible Markup Language (XML). XML is a series of nested "tags" that describe the way that certain pieces of data are structured. For my project, I decided to use XML to describe a remote server's collection of space physics datasets. A benefit of XML is its natural tree structure (root, parent, child), which can be used to organize the process. This kind of methodology is given the name of Document Object Model (DOM), and is an official specification of the W3C. I used this in both the User Interface (Browser) and the actual data server search algorithm. More about this will be presented below under "The Schema and Data Model" and in the discussion of the Interface.

As far as which programming language suits the task, I chose to use Python (version 2.3). The latest version of Python, combined with the useful PyXML and Tix modules, offers multiple methods with which to process XML efficiently. It also offers an object-oriented framework that makes extensive use of inheritance and polymorphism. Functional programming constructs such as mapping a function onto a list, lambda forms of functions (i.e. f x = x + 1, instead of f(x) = x + 1), and list comprehensions also simplify programs written in the language. The most important feature of Python is the short learning curve and ease with which development can be accomplished. Java development generally takes more time and strategy.

There are three main parts of the project: the Dataset Browser, the Dataset Finder, and the Dataset Editor. During their development, I was able to test my ideas with datasets available through anonymous FTP and then extend this to as many datasets as possible. All in all, the project's development process went quickly with these tools and resources. Now, I shall proceed to describe the process below.

2 The Schema and Data Model

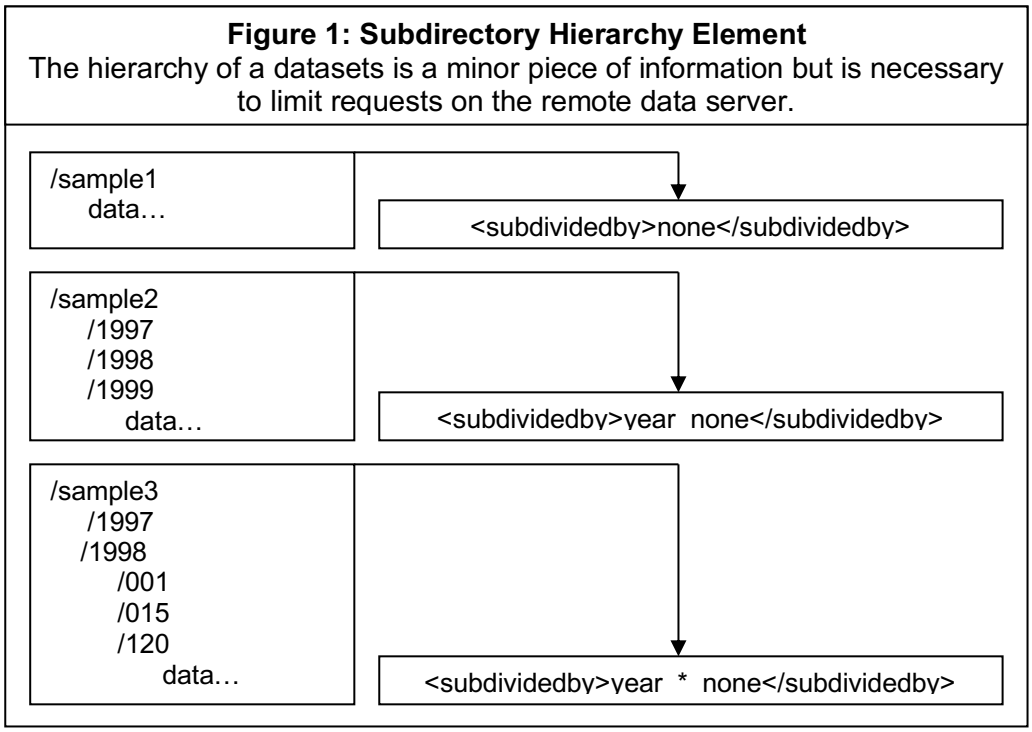
2.1 Dataset Element

Any data model considers the basic attributes needed to represent a data object. Name and location are good starting attributes for the model of a dataset; these are crucial to any kind of search.

The search algorithm needs to know exactly at which directory to start and exactly what data item is requested. If this were not provided, a remote server would receive many more queries than needed to find the requested dataset. Once the exact path to a dataset and its unique descriptor are known, the subdirectories must be searched. The remote server is not intelligent at all, so the schema must provide it with more basic instructions.

One set of instructions is the hierarchy of a dataset's subdirectories, stored in the **subdividedby** element. The data could all be clumped into the single base directory or could be separated by year and then day of year as an example. This piece of information is encoded into a string where a list of underscore-separated keywords is stored. Among keywords are *year*, *month*, *day* for directories, *none* for the stopping directory, and an asterisk as a wild card. Figure 1 gives several examples of subdirectory hierarchies.

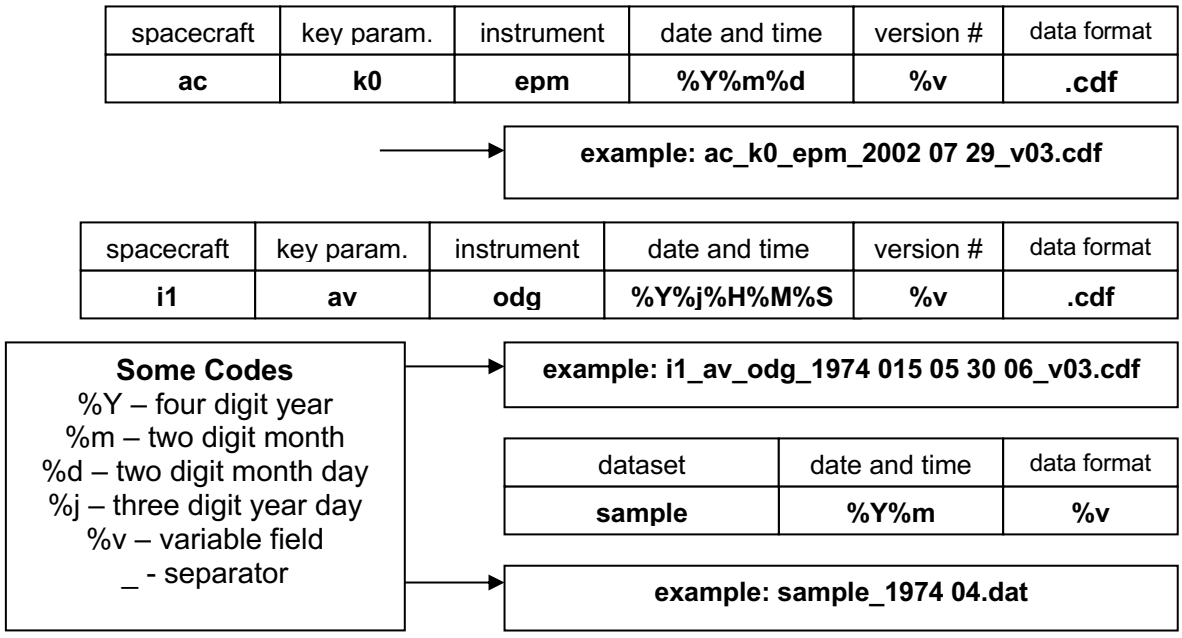
Figure 1: Subdirectory Hierarchy Element
The hierarchy of a datasets is a minor piece of information but is necessary to limit requests on the remote data server.



Next, the search algorithm needs to know how files in a certain dataset directory are named. In a perfect world, all files would follow the same conventions, but this is not the case. Datasets use store data in various time intervals from seconds to days to Bartel rotations (27 days) to even entire years at once. In order to handle this, a **filename** element is stored in the XML dataset. This element, summarized in Figure 2, uses regular expressions to match a specific file name pattern given by a coded string. A file must match this exactly or it will not be considered. Now, every alphanumeric character and some symbols are matched as is, but special codes, created by a percent sign then letter, fill in dates and other

variable fields. This element is very flexible. For instance, if the data has different data formats, a ".%v" can pick up data files with any extension provided that the rest matches. This is summarized in Figure 2 below.

Figure 2: File Naming Conventions
The file naming conventions that are followed by data stores vary greatly. The <filename> tag encodes this flexibly into the XML, using codes from the strptime module



A third piece of helpful dataset information is the time range for which data is available from a specific dataset. This is stored in the **timerange** element, which consists of a single tag with three attributes. These attributes are **start** time, **stop** time, and optional time **units**. The start and stop times are strings in either "%Y%m%d", "%Y%j", "%Y%m%d%H%M%S", or "%Y%j%H%M%S" (note that spaces and hyphens are ignored). The units attribute is only relevant when you use units other than these standards, such as **Bartels.** Bartels are used to represent a single solar rotation and approximate to a period of 27 days. Some datasets are still operational and near current data is available. For this, the string "LATEST" will allow users to see new files without regenerating XML each time. In addition to providing the search algorithm with useful information, the timerange element defines an upper and lower limit on the data that the client program will search. For example, if an instrument was not calibrated

correctly at first and data earlier than a certain date is invalid, this element will tell the finder to ignore that data. This is summarized in Figure 3 below.

Figure 3: Time Range
The time range element defines an upper and lower limit on the datasets searchable time window.

start	stop	units (optional)
2002-10-31	LATEST	days

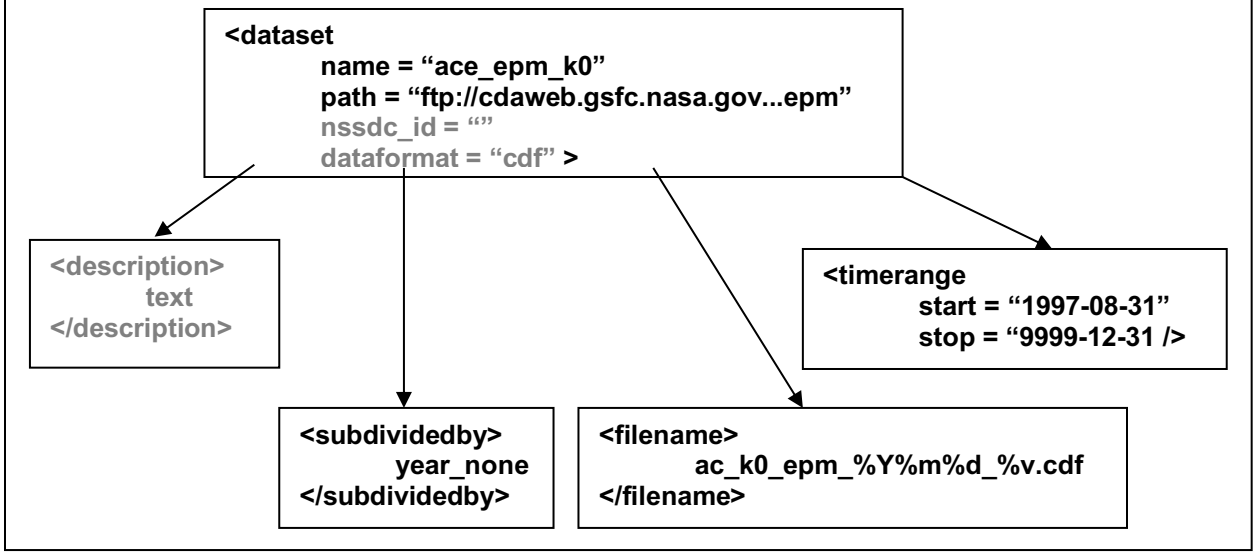
<timerange start="2002-10-31" stop="LATEST" />

start	stop	units (optional)
2240	2300	bartels

<timerange start="2240" stop="2300" units="bartels"/>

Now that we have this required information, another useful option is to include a **description** element and a couple more attributes. These optional attributes do not have to be anything in particular. My work has included the dataset's data format (or file extension) and its ID in the NSSDC Master Catalog, a database of NASA missions and datasets. However, I am not currently using these for anything although the capability certainly exists. This data is combined into the **dataset** element, summarized in Figure 4. The **dataset** node of the XML tree is a leaf (no children), contains the **description**, **subdividedby**, **filename**, and **timerange** elements, and has the two required attributes of **name** and **path**.

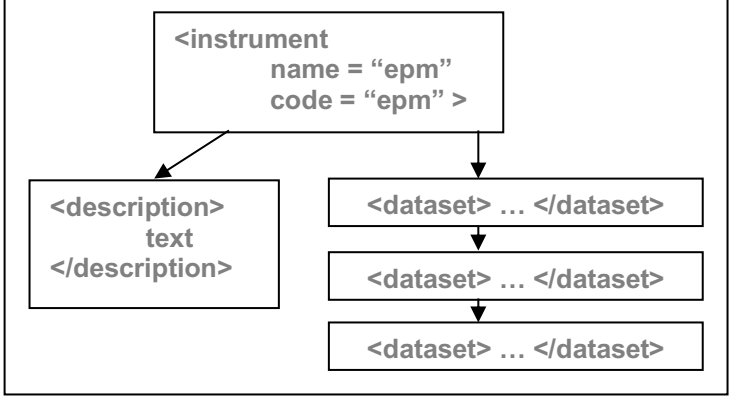
Figure 4: Dataset Element
This is a summary of the dataset element (gray denotes optional attributes).



2.2 Instrument Element

The dataset element is the fundamental building block, and the XML file could contain a list of dataset elements alone. However, to accommodate users who wish to browse around more or do not know the specific dataset descriptor, I constructed a more complete hierarchy. Some datasets are grouped into a larger group classified by instrument or instrument type. Perhaps a researcher only cares about magnetometer data or wants the data from ACE's Cosmic Ray Isotope Spectrometer. The **instrument** element allows the data to be organized in this way and is summarized in Figure 5. It contains an optional **description**, a list of **dataset** children, and **name** and **code** attributes.

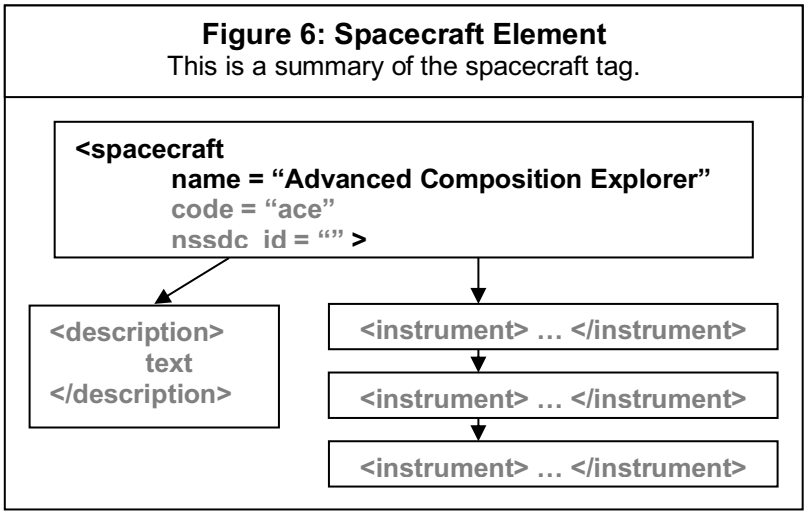
Figure 5: Instrument Element
This is a summary of the optional instrument element.



2.3 Spacecraft Element

At the next level upward, the **spacecraft** element groups either instrument elements or dataset elements, depending on the existence of instruments. The only two major pieces of information needed are the **name** attribute of the spacecraft and its list of children (datasets or instruments). Additional attributes such as its acronym and NSSDC Master Catalog ID and a **description** can also be included. This is summarized in Figure 6.

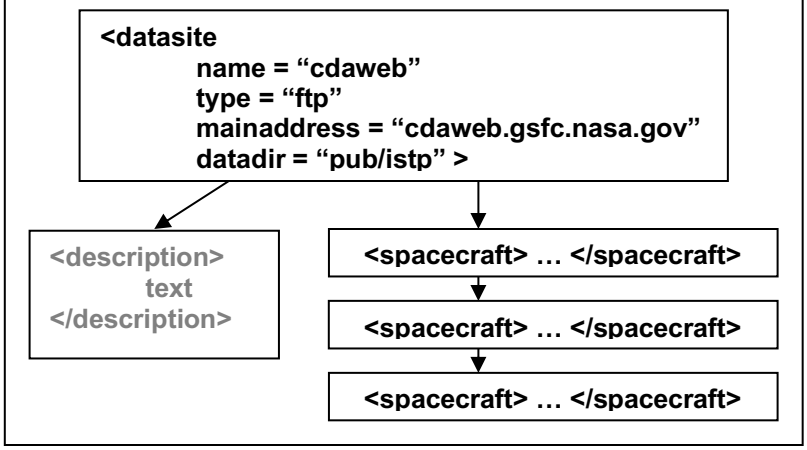
Figure 6: Spacecraft Element
This is a summary of the spacecraft tag.



2.4 Datasite Element

Finally, the spacecraft elements are grouped into a **datasite** element. This element represents a complete remote data server where many datasets are stored. Of course, the datasite must have attributes of **name**, server **type**, **mainaddress** (its URL)**,** and **datadir** (where data is stored). In the future, the type could be extended to use protocols other than FTP; however, that capability does not yet exist. The element also includes a list of its **spacecraft** children and an optional **description**. I would suggest that the description elements be used, as it is an easy way to provide extra useful information to a client program.

Figure 7: Datasite Element
This is a summary of the datasite element.



2.5 Summary

The data model for these distributed datasets makes strong use of the XML concept of the Document Object Model (DOM). Elements are treated as nodes in a large tree structure with a list of children: elements, attributes, text, and other types. The root of this model is the datasite element with some attributes and data elements along with a list of children: spacecraft elements. The spacecraft element has informational items along with a list of children: instrument or dataset elements. The optional instrument element has informational items along with a list of children: dataset elements. Finally, the dataset element has no children but contains attributes, a description, a subdividedby string, a filename pattern, and a timerange element.

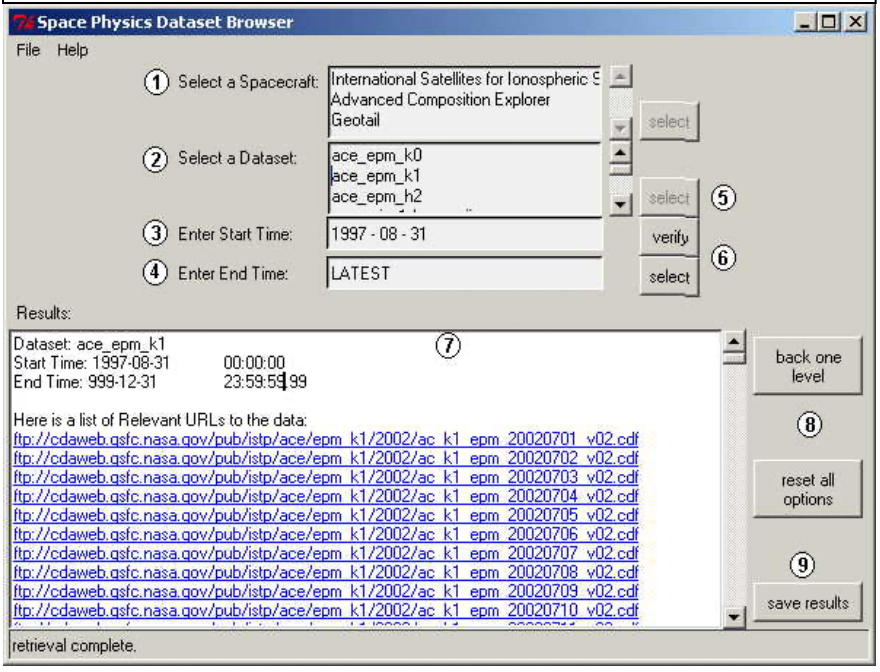
3 Space Physics Dataset Browser

3.1 Phase 1: Getting Input from the Interface

The Space Physics Dataset Browser (Browser) is the front-end interface to the Dataset Finder (Finder). It is written in Python using the Tk interface module (Tkinter). The Finder module does the actual searching of the remote server using the XML files generated by another program called the Editor, which is discussed later. The first step in the search process is to get input from the user. All that the Finder requires is the specific dataset descriptor and two strings representing some form of time. However, the Browser interface is designed to be much more user-friendly, relatively simple and intuitive. It loads the XML files into DOM objects in memory, so that it is faster to navigate the hierarchy of elements: datasite, spacecraft, instrument, and dataset. First, a list of datasite XML files defined in the "datasites.pref" file is loaded and the spacecraft elements are listed in a selection box (Fig. 8-1). Additionally, general information is loaded into the result text box (Fig. 8-7). Note that the user does not need to know anything about the datasite elements at all. This is abstraction at work.

Once the spacecraft has been selected, a list of its datasets populates the second selection box (Fig. 8-2). Once a dataset is selected and the "select" button is pressed, the time fields become active. At this point, the program stores the dataset node in question; no other nodes are relevant at this time. Now, the user fills in

Figure 8: Space Physics Dataset Browser
This screenshot shows the result of a typical search.



the start time (Fig. 8-3) and stop time (Fig. 8-4). Most of what applied to the timerange element in Section 2.1 applies. "LATEST" and "BEGINNING" are key words signifying the latest data and the minimum allowable date respectively. The formats acceptable for input are the same ones as in the timerange element. If the user wishes to use Bartels, he or she could simply type a "b" followed by the rotation number, but the default is year-month-day. Now, the "verify" button (Fig. 8-6) will check to see if the time range is valid and that the stop time comes after the start time. Any time that the verify routine does not recognize is set to BEGINNING or LATEST, depending on the field. Once this is done, the user clicks the final select button (Fig 8-6) and the search is performed. If the search is successful, the resulting list of URLs is displayed and can be saved (Fig. 8-9). Two additional buttons (Fig. 8-8) allow the user to navigate back up the tree. Also note that the instrument element is not currently used in the Browser. Finally, the required input is sent on to the Space Physics Dataset Finder.

3.2 Note: How does the time range get represented?

At this point, you may be wondering how a string with characters becomes a time range object and how the time range is implemented. First, the start and stop time strings are parsed using a pattern (defined using the exact same notation as the filename element!). A module called "strptime" handles this with the exception of a few special cases. This method produces a datetime object, which is an improved time object, new in Python 2.3. The available dates range from Jan. 1, 0001 to Dec. 31, 9999, whereas the old time object followed the limited C/Unix epoch time implementation. As a further improvement, I wrote the timeRange class that represents a specific time interval. If the interval of the user input overlaps with the interval of a file, the file will most likely contain data that the user is requesting. The timeRange module will also extract times out of data URLs and handle useful conversions. This approach turned out to be a simple and powerful strategy to use.

3.3 Phase 2: Navigating the Tree

The user has submitted a dataset descriptor and a time range to the Finder through some type of interface, not necessarily the Browser described above. It is now the Finder's job to return a list of data file URLs matching those criteria. First, the Finder traverses the list of dataset elements in the desired

XML document and searches using the dataset descriptor. Next, it calls upon the timeRange module to convert the time input from a pair strings into a timeRange object. Once the dataset element and desired time range are known, the program analyzes the main attributes of the dataset: remote path, subdirectory hierarchy, time range, and file naming string. Up until this point, no remote server access has been performed. The Finder now sends this information onto the Datasite Walk (discussed below) module, which returns a list of file names from the remote server. This list is then scanned, and a time range is extracted from each file name that matches the dataset's filename element. If this range overlaps with the requested time range, that means that the file contains desired data. Finally, the file names with overlapping time ranges are now passed back to the user.

3.4 Note: Walking the Data Site

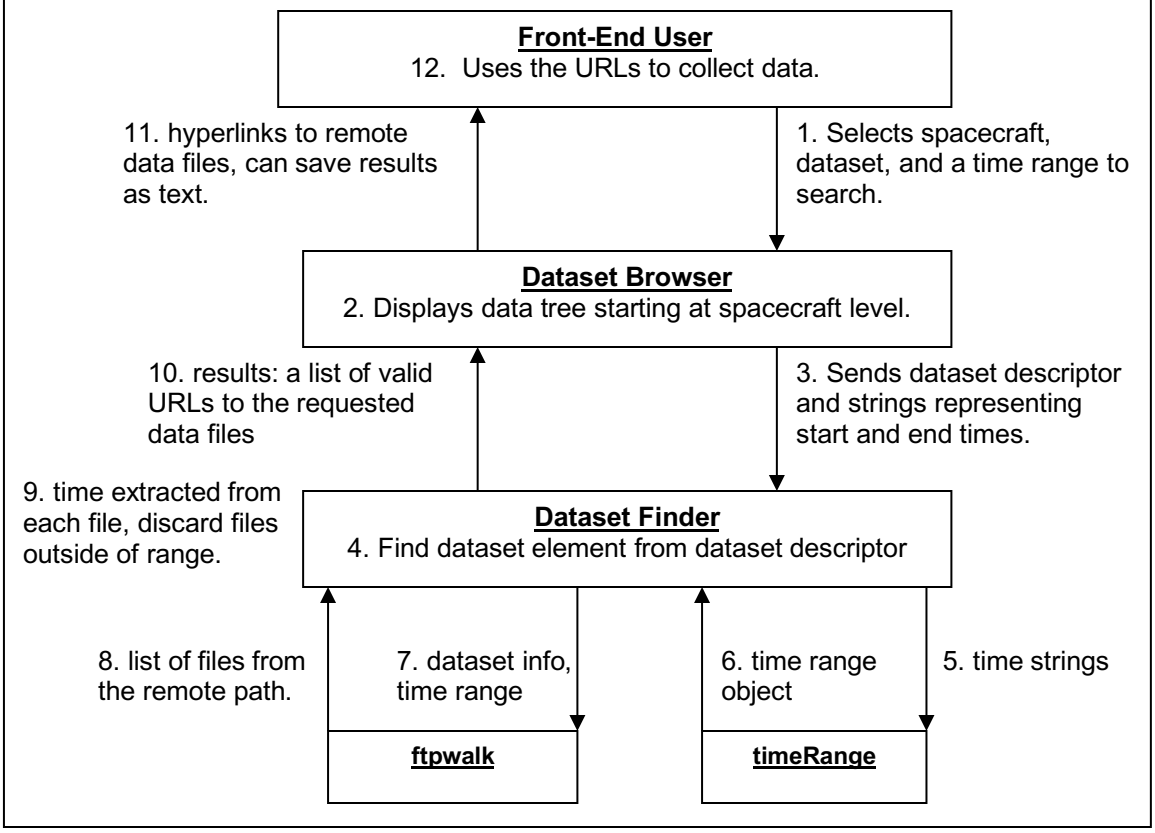
The Datasite Walk module requires four main pieces of data: the path where it should start, the levels of subdirectories, a start time, and an end time. From the path, the algorithm can determine to which server it should connect, how it should connect (currently, just FTP), and where the dataset directory is. Once it arrives at the base directory, the routine traverses to the bottom-most level ("none") as defined in the hierarchy element. It does this by initiating a directory listing command (LIST for FTP) and parsing out the names of files and subdirectories. Now, the time parameters are used to implement searching shortcuts. For instance, if a subdirectory is divided by year, only years in the user-defined time range are deemed relevant. This walk routine will not process this list; the filenames are simply passed back to the caller, which will perform any processing. I chose to write this module in this way because it is lightweight and relatively straightforward. Currently, this algorithm only works using File Transfer Protocol; however, the layers of abstraction allow for the addition of other protocols.

Phase 3: Using the Results and Beyond

From the Dataset Finder, the user receives a list of URLs to the data he or she requested. Equipped with this information, the user could simply download the file from the remote data server. However, another layer of processing could prove to be even more powerful. Such a program could gather these data files and plot them through CDAWeb's toolbox of plotting functions or convert the data

into any format desirable. The goal is, then, the efficient collection of data files so that any higher-level program would not need to concern itself with accessing numerous servers just to find data. The data will be easily accessible to any researcher. Figure 9 summarizes the different layers of abstraction.

Figure 9: Summary of the Dataset Browser Process
Each module (box) can act independently of the other modules.



4 XML Generation: Space Physics Dataset Editor

4.1 The Problem

Up until this point, this discussion has assumed the presence of local XML files containing metadata about datasites, spacecraft, instruments, and datasets. However, the focus will now shift to the actual XML generation process. The tools for manipulating XML are drawn from the same sources as before. Many different operations are now required in addition to parsing the XML into a DOM tree: editing attributes and fields, adding elements as children, creating and importing new files, cloning

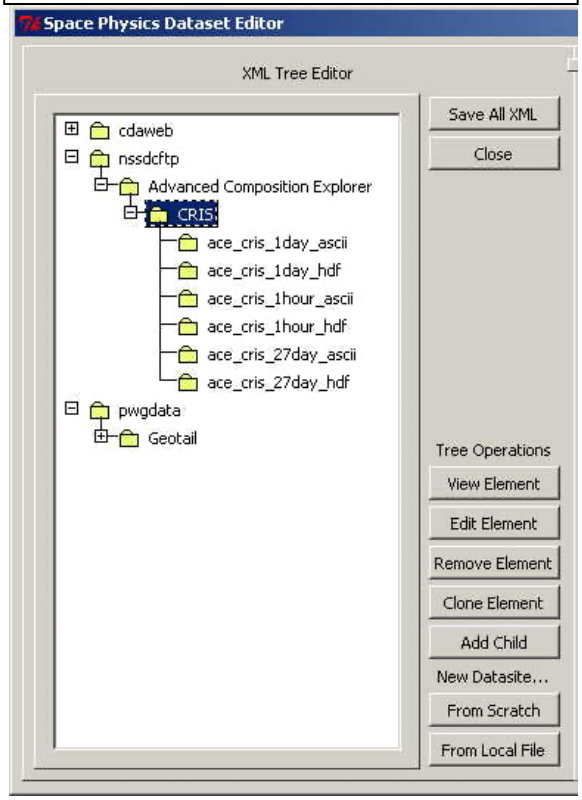
elements, and providing a quick method to browse this tree. In addition, it would be important to display information from the remote server at the same time so that no other program (i.e. Internet Explorer) is required. Some kind of auto-completion for elements (using remote server data) also proves to be helpful.

4.2 The Solution

To solve this, I created a companion to the Browser and Finder: the Space Physics Dataset Editor. The best approach for this application was to start with two adjacent tree view interfaces, one for the local XML data files and one for the remote server, that have expansion capabilities. In addition, context-sensitive buttons allowed for quick manipulation of the XML Tree. Inheriting the Tree widget from the Tix (addition to Tkinter) module, I created a class for each type of tree viewer and for a generic dialog box that edits any element in the tree. This allows for much more code reuse.

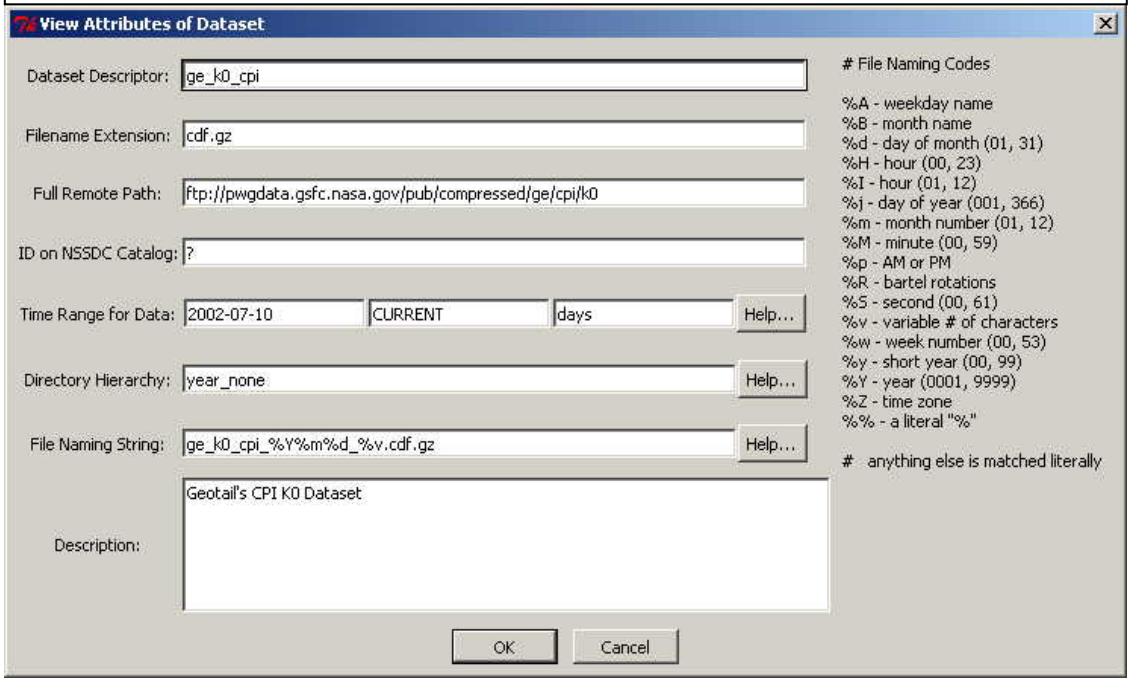
First, the XML Tree Display (fig. 10) displays the hierarchical structure of the local XML files (defined in "datasites.pref"). Elements can be expanded and contracted at will simply by clicking the plus/minus buttons. The "Tree Operations" buttons all perform an operation on the selected element while the "New Datasite" buttons import new data files. Buttons are fairly self-explanatory. "View" and "Edit" both pop up the node-editing dialog (fig. 11), a generic method to change an element's attributes. "Remove" will also delete datasite elements, but will not physically remove the XML file from memory. "Clone" generates a full copy of the selected element.

Figure 10: XML Tree Editor
This tool simplifies XML manipulation.



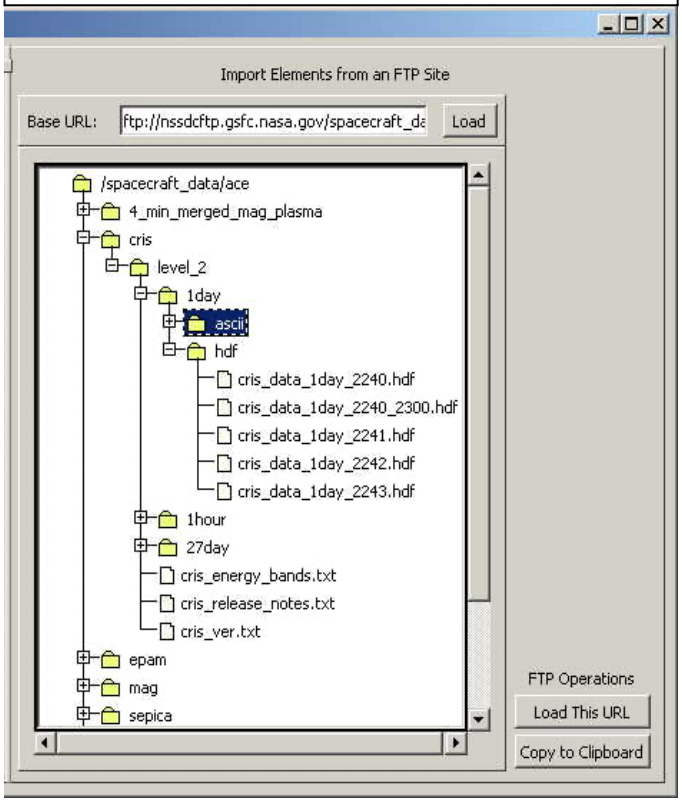
"Add Child" will create a new sub element from scratch; it will prompt the user to complete the fields. "From Scratch" asks for a new XML file in which to place a new datasite element while "From Local File" simply adds a file to "datasites.pref".

Figure 11: XML Node Dialog
This is where fields are modified. If a remote path is also selected, these are guessed.



The intelligent part of the Editor is the use of a remote server Path Viewer (fig. 12) to help auto-complete dialog fields. If both an XML element and a remote server path are selected and "Copy to XML" is activated, the same "Add Child" dialog box pops up. However, some fields will be already filled in with guessed values based on context. This feature is currently in the planning stage, but once completed, the result is an easy method

Figure 12: FTP Path Viewer
This tool gives the user a handy reference.



to generate the XML documents that are used by the Dataset Browser.

5 Conclusion

5.1 Project Status and Suggestions for Improvements

The Space Physics Dataset Browser system is flexible enough to allow for as high a level of usability as possible. What I have described is only the completed skeleton of its capabilities. One major addition to this could be more use of the metadata. Each element could include additional attributes for the benefit of Browser users. For example, spacecraft elements might incorporate a link to its website, the team in charge of it, and any useful fact. To accomplish this, descriptions could be generated from the NSSDC Mater Catalog or another database. Use of these XML files is not limited to the Dataset Browser, however. Anyone who uses the data would benefit from a file describing the structure of these data sites. By simply applying a style sheet to these files, instant documentation could be displayed for those browsing a data site. Another improvement involves a more automated way to generate the XML files. A routine could walk an entire remote server and analyze its structure every so often. New datasets would be discovered and incorporated into the system. However, some data sites are not organized well enough to do this, so some human verification should be involved in this process. Other future avenues of exploration include the selection of multiple datasets at once, bundling the resulting data into a single. tar.gzip file, support for VMS servers and HTTP, substituting a web service for the current Browser, and connecting it to SPDF's collection of Master CDF files. There are more than likely other ways to improve upon the Browser system's methods; these are only a few.

5.2 Virtual Observatories and Beyond

Through my experience with this project, I have realized how much of a demand there is for a large collection of distributed science data in one source. Obviously, this is simply not physically possible, even with today's availability of disk space. However, with such a tool as the vast worldwide network, this can be achieved virtually. As I have already mentioned, the future Space Physics Virtual

Observatory will be a powerful web service combining the CDAWeb system and Dataset Finder. There is still much work to be done before going operational. The Browser system is just one fundamental part of this. Another key issue is the efficient conversion from one data format to another. The software development team at the SPDF is currently making significant progress in these areas. The Observatory project will greatly increase physicists' potential to understand the Sun's impact on our near-Earth environment. Combined with other initiatives for Virtual Observatories in the science community, Space and Earth Science will continue to move into new frontiers of understanding.

Appendix A: Index of Figures

Schema and Data Model

Space Physics Dataset Browser

XML Generation: Space Physics Dataset Editor

Appendix B: Sources of More Information

CDAWeb:	http://cdaweb.gsfc.nasa.gov/
DOM:	http://www.w3.org/TR/DOM-Level-2-Core/ explains all the technical details.
Python:	Python.org is the bleeding edge source. Documentation is extensive and there is a whole community of devoted developers. Impressive little language.
PyXML:	Pyxml.sourceforge.net is the source. A great tutorial for processing XML with Python can be found at http://pyxml.sourceforge.net/topics/howto/xml-howto.html
SSCWeb:	http://sscweb.gsfc.nasa.gov/
SPDF (632):	http://spdf.gsfc.nasa.gov
TIMED	http://www.timed.jhuapl.edu/
Tkinter:	http://www.pythonware.com/library/tkinter/introduction/
Tix:	http://tix.sourceforge.net
XML:	http://www.w3.org/XML/

Appendix C: Terms and Acronyms

ACE	Advanced Composition Explorer
CDAWeb	Coordinated Data Analysis Web
DOM	Document Object Model
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
NSSDC	National Space Science Data Center
SSCWeb	Satellite Situation Center Web
SPDF	Space Physics Data Facility
TIMED	Thermosphere • Ionosphere • Mesosphere • Energetics and Dynamics
URL	Uniform Resource Locator
XML	eXtensible Markup Language

Final Technical Notes and Source Code Available Upon Request of Robert M. Candey (Robert.M.Candey@nasa.gov). Special Thanks to Robert Candey, Bob McGuire and the 632 Software Development Team, Dan Krieger, Michael Hartman, Laureen Summers and

LaTasha Mason from AAAS, Dillard Menchan, my new friends at Goddard, and my parents for years of support.