UPOS-AB3-03

August 2004

Version 2.0

By
Janice Schofield
Simon Wing
# Table of Contents

1. INTRODUCTION .................................................................................. 2

1.1 Overview .......................................................................................... 2
1.2 Summary of Architecture ................................................................. 2
1.3 Statement of Purpose ...................................................................... 3

2. PRODUCT STRUCTURE ................................................................... 4

2.1 Executive Script Software calling hierarchy .................................... 4
2.2 Dst Forecasting Software calling hierarchy ..................................... 4

3. OBJECT DESCRIPTIONS .................................................................. 6

3.1 Executive Script Software ................................................................. 6
  3.1.1 CSH Objects ............................................................................. 6
  3.1.2 BASH Objects ......................................................................... 6
  3.1.3 PERL Objects ......................................................................... 6
  3.1.4 C Objects ................................................................................ 12
  3.1.5 HTML Objects ........................................................................ 12
  3.1.6 rplot Objects ......................................................................... 12

3.2 Prediction Software ......................................................................... 13
  3.2.1 Java Objects ............................................................................. 13

4. EXTERNAL INTERFACE ................................................................. 17

4.1 Input Data ....................................................................................... 17
4.2 Operating System Services ............................................................ 19
4.3 Output ............................................................................................. 19
4.4 Other software programs or libraries ............................................ 20

5. RELATED DOCUMENTATION ....................................................... 21

A. ACRONYMS AND ABBREVIATIONS .............................................. 23
B. SOURCE CODE ............................................................................... 24
C. SAMPLE OUTPUT PLOTS ................................................................. 25
1 Introduction

1.1 Overview

The primary purpose of the Dst Forecast system is to provide AFWA the capability to predict Dst. Dst (disturbance storm time index) is an hourly index that gives a measure of the strength of the ring current that, in turn, provides a measure of the geomagnetic storm [Dessler and Parker, 1959]. Because of its global nature, Dst is often used as one of the several indices that represent the state of the magnetosphere. Other indices include AE and Kp ([http://sd-www.jhuapl.edu/UPOS/ForecastingKP](http://sd-www.jhuapl.edu/UPOS/ForecastingKP)). When energetic ions are injected into the Earth’s inner magnetosphere, they drift westward around the Earth, forming the ring current. The Dst is derived from a composite of the H component of several low-latitude ground magnetometers after the quiet day H component has been removed [Kivelson and Russell, 1995]. The increase in the ring current, which usually occurs during a magnetic storm, reduces the horizontal (H) component of the magnetic field near the equatorial region on the surface of the Earth. Hence, the Dst is depressed or reduced during periods of magnetic disturbances, typically in the time scale of a few hours. This is followed by a recovery or an increase in Dst that is gradual, in the order of several days. The storm morphologies and Dst behavior during magnetic storms has been the subject of many studies [e.g., Mayaud, 1980; Kamide et al., 1997; Akasofu and Chapman, 1972].

Many models for the near-Earth space environment need Dst to predict various parameters such as Magnetospheric Specification Forecast Model [Freeman et al., 1993], T96 magnetic field model [Tsyganenko and Stern, 1996], etc. Unfortunately, Dst indices are published with significant time delay. The purpose of the Forecasting Dst project is to provide a predicted Dst using inputs from the ACE satellite.

The Forecasting Dst application consists of a series of scripts and programs that download data from the NOAA Space Weather and World Data Center for Geomagnetism sites, process the data into one hour averages, input the data into a neural network, and create images of the ACE data and forecasted Dst. The user can then view these images via a web browser.

1.2 Summary of Architecture

The system architecture is based on software that is currently used for similar purposes at JHU/APL. Figure 1.1 shows a diagram of the architecture and data flow.
Once the Dst Forecasting system is started, it runs in the background. A perl script downloads the ACE data from the NOAA Space Weather site and the Quicklook Dst from the Geomagnetism site, compiles this data into 1-hour averages, and then feeds the data into a neural network. The results from this algorithm are then displayed on a web page along with the graphs of the IMF values and solar wind data.

1.3 Statement of purpose

The purpose of this document is to describe the software architecture and to identify the components of the system.
2 Product Structure

The Forecasting Dst system is composed of two major components. The first component is implemented in perl and managed by the Executive Script. This piece of the system runs every 15 minutes by a cron job and controls data acquisition, processing, and plotting. The second piece of the Forecasting Dst system is implemented in Java to perform the prediction process. This software runs at the command of the user as a background process.

2.1 Executive Script Software calling hierarchy

1.1 ace.csh
  1.1.1 createHist.pl
  1.1.2 plot_day
    1.1.2.1 day_julian_to_hour.pl
    1.1.2.2 rplot day.rp
  1.1.3 plot_week
    1.1.3.1 week_julian_to_day.pl
    1.1.3.2 rplot week.rp
  1.1.4 ftp.pl
  1.1.5 ace.pl
    1.1.5.1 getMonthYear
    1.1.5.2 getACEData
      1.1.5.2.1 getDay
      1.1.5.2.2 getWeek
      1.1.5.2.3 parseLocation
      1.1.5.2.4 process_sw_data
      1.1.5.2.5 process_imf_data
      1.1.5.2.6 average_data
      1.1.5.2.7 create_nn_input
  1.1.5.3 getNowcast
    1.1.5.3.1 getJulian

2.2 Dst Prediction Software calling hierarchy

1.1 DstPredictor
  1.1.1 NNDataset
  1.1.2 NN_Dstmodel
    1.1.2.1 NNModel
      1.1.2.1.1 RecurrentData
      1.1.2.1.2 URLMappings
    1.1.2.2 ImfSwData
      1.1.2.2.1 URLMappings
      1.1.2.2.2 DataLoader
    1.1.2.3 DstData
      1.1.2.3.1 NowcastData
3 Object Descriptions

3.1 Executive Script Software

3.1.1 CSH Objects

3.1.1.1 ace.csh

This executive script controls the data acquisition and processing operation of the Forecasting Dst system. It calls the functions necessary to process the acquired data and create the plots of the data and predictions.

3.1.2 BASH Objects

3.1.2.1 plot_day

This script creates a GIF image plot of IMF, solar wind bulk speed, solar wind proton density, 1-Hr forecast Dst, and 4-Hr forecast Dst values over the last 24 hours.

3.1.2.2 plot_week

This script creates a GIF image plot of IMF, solar wind bulk speed, solar wind proton density, 1-Hr forecast Dst, and 4-Hr forecast Dst values over the last 7 days.

3.1.3 PERL Objects

3.1.3.1 createHist

This subroutine creates two files to contain the latest 1-Hr and 4-Hr forecast values, as output by the prediction software, along with the historical predictions.

**CALLING SEQUENCE:** createHist.pl (no arguments)

**RETURN:** None

3.1.3.2 day_julian_to_hour

This script converts the Julian dates associated with the 24-hour data into hours (in Coordinated Universal Time) for the purposes of labeling the 24-hour data plot.

**CALLING SEQUENCE:** day_julian_to_hour.pl (no arguments)

**RETURN:** None
3.1.3.3 week_julian_to_day

This script converts the Julian dates associated with the 7-day data into days of the month for the purposes of labeling the 7-day data plot.

CALLING SEQUENCE: week_julian_to_day.pl (no arguments)

RETURN: None

3.1.3.4 ftp

This script transfers the images created by plot_day and plot_week to the web server for display in a web page.

CALLING SEQUENCE: ftp.pl (no arguments)

RETURN: None

3.1.3.5 ace

This script downloads the ACE data from the NOAA site and formats the data for ingestion by the prediction software.

CALLING SEQUENCE: ace.pl (no arguments)

RETURN: None

3.1.3.6 getMonthYear

This subroutine gets the current system time to use for downloading ACE data.

CALLING SEQUENCE: getMonthYear (no arguments)

RETURN: month, year

Where

month = current month
year = current year

3.1.3.7 getACEData

This subroutine checks that all ACE data has been downloaded from the NOAA SEC web site. Note: The data is actually downloaded by an overall ftp script used for UPOS.
CALLING SEQUENCE: getACEData(month, year)

Where:

\[ month = \text{current month} \]
\[ year = \text{current year} \]

RETURN: login_result

Where:

\[ login\_result > 0 \text{ if successful} \]

3.1.3.8 getDay

This subroutine opens a given file and retrieves the date it was created.

CALLING SEQUENCE: getDay(filename)

Where:

\[ filename = \text{the filename which contains the date} \]

RETURN: day

Where \[ day = \text{current day} \]

3.1.3.9 getWeek

This subroutine takes the current year, month, and day and calculates the previous eight days and returns the result in an array.

CALLING SEQUENCE: getWeek(year, month, day)

Where:

\[ year = \text{current year} \]
\[ month = \text{current month} \]
\[ day = \text{current day} \]

RETURN: week

Where \[ week = \text{multi-dimensional array of months, days and years} \]

3.1.3.10 parseLocation
This subroutine parses the location file for locations of the ACE satellite for dates later than the date given.

**CALLING SEQUENCE:** parseLocation(year, month, day, hour)

Where:

- *year* = year
- *month* = month
- *day* = day
- *hour* = hour

**RETURN:** *location*

Where *location* is an array of locations by date

### 3.1.3.11 process_sw_data

This subroutine reads the solar wind data file and creates a file of the solar wind data for the day and a file of the solar wind data for the week.

**CALLING SEQUENCE:** process_sw_data(sweepam_count, ace_location)

Where:

- *sweepam_count* = number of lines in data file
- *ace_location* = current location data

**OUTPUT:** sweday.dat & sweweek.dat

With Columns:

- j_date_unprop
- j_date_prop
- year
- month
- day
- hhmm
- modified_julian_day
- seconds
- status
- density
- speed
- temperature
- ace_location

**RETURN:** None
3.1.3.12 process_imf_data

This subroutine reads the IMF data file and creates a file of the IMF data for the day and a file of the IMF data for the week.

CALLING SEQUENCE: process_imf_data(imf_count, ace_location)

Where:

imf_count = number of lines in data file
ace_location = current location data

OUTPUT: imfday.dat & imfweek.dat

With Columns:

j_date_unprop
j_date_prop
year
month
day
hhmm
modified_julian_day
seconds
status
bx
by
bz
b
latitude
longitude
velocity
density

RETURN: None

3.1.3.13 average_data

This subroutine averages the latest propagated solar wind and magnetic field data and combines them into a single file.

CALLING SEQUENCE: average_data()

OUTPUT: data_prop_ave.dat

With Columns:

j_date_prop
bx
by
bz
density
speed

RETURN: None

3.1.3.14 create_nn_input

This subroutine copies the propagated, averaged solar wind and IMF data into the input file to be used by the neural network.

CALLING SEQUENCE: create_nn_input()

OUTPUT: nn_input.dat

With Columns:
  j_date_prop
  bx
  by
  bz
density
  speed

RETURN: None

3.1.3.15 getNowcast

This subroutine retrieves Quicklook (Nowcast) Dst data from the World Data Center for Geomagnetism web site.

CALLING SEQUENCE: getNowcast (no arguments)

RETURN: None

3.1.3.16 getJulian

This subroutine converts a time in “normal” format (year, month, day, hour) to a time in Julian format.

CALLING SEQUENCE: getJulian (year, month, day, hour)

Where:

year = 4-digit year to be converted
month = month to be converted, spelled out (e.g., “SEPTEMBER”)
day = day (of month) to be converted
hour = hour (of day) to be converted

RETURN: jdate

Where:

jdate = Julian date

3.1.4 C Objects

3.1.4.1 rplot

This program produces a plot image, given a data specification and dataset.

CALLING SEQUENCE: rplot specfile.rp

Where: specfile.rp is an ASCII text file that specifies the plot format and the data to be plotted

RETURN: None

3.1.5 HTML Objects

3.1.5.1 dstday.html

This HTML file displays the results of a 24-hour period.

3.1.5.2 dstweek.html

This HTML file displays the results of a 7-day period

3.1.5.3 dst.html

This HTML file is the main display for the Forecasting Dst process.

3.1.6 rplot Objects

3.1.6.1 day.rp

This ASCII text file specifies the format of the 24-hour plot and the location of the data to plot.

3.1.6.2 week.rp

This ASCII text file specifies the format of the 7-day plot and the location of the data to plot.
3.2 Prediction Software

3.2.1 Java Objects

This section contains a summary of each Java class and the major functions used in the Forecasting Dst System. A more detailed description of all attributes and methods in each class is provided in the Javadoc for this software (see Appendix B for information regarding the source code/Javadoc).

3.2.1.1 DstPredictor

This is the main class for the Dst Prediction software. This class initializes all of the prediction models, and begins a looping routine which first checks for new data and then runs each model and outputs the predictions once new data arrives. The loop will only be exited at the user’s command or upon an irreconcilable error.

3.2.1.2 NNDataSource

This class contains the blocking method for the main DstPredictor class. After initialization, the DstPredictor begins a loop that runs each model and outputs the predictions, after first calling the checkNewData() method contained in the NNDataSource class. This method will not return until new data (nowcast and ACE) is present, thereby blocking the DstPredictor from further processing.

3.2.1.3 NN_dstNow, NN_dstFuture

The DstPredictor runs two different neural network models, each contained in a separate class but all working in a similar way and extending the NNModel class. The DstPredictor initializes each of the neural network classes. This initialization sets up a NNModel with the data to be used by the neural net, the weightfile, the transformer file, the number of nodes for the neural network, and the output transformer index value. The initialization of the neural network also sets up an array to hold the most recent predictions made by the model. The getPrediction() method is the only public method available from the neural network classes. The method initializes the data files and sets the values for the input and output data at the current time, calls the getInitialPrediction() method of the NNModel class, and adjusts the prediction accordingly. The getPrediction() method returns an instance of the Prediction class for that model.

3.2.1.4 NNModel

Each model’s class (e.g. NN_dstNow) contains an instance of the NNModel class. This class is initialized for the specific model with the data to be used by the neural net, the weightfile, the transformer file, the number of nodes for the neural network, and the output transformer index value. After setting up these
parameters and establishing the weights and transformer values, the network is initialized for the NNModel. The description of the neural network and how it operates is beyond the scope of this document.

Two public methods are available for use by an instance of NNModel. The public method setVals(double inVals[], double outVals[]) takes the current input and output data and sets up a new RecurrentData instance. The method getInitialPrediction() uses the RecurrentData instance and the neural network to provide a prediction.

3.2.1.5 RecurrentData

This class is part of the neural network and sets up DataArrayTransformers for the input and output data for the neural network. The public methods of this class are used by the getInitialPrediction() method of the NNModel class to input values into the neural network in order to get a prediction.

3.2.1.6 UCB_dstNow

In addition to the NN_dstNow and NN_dstFuture models, the DstPredictor also runs the UCB model. The initialization of the UCB model sets up an array to hold the most recent predictions made by the model. The getPrediction() method is the only public method available from the class. The method initializes the data files and sets the values for the input and output data at the current time, calls the getInitialPrediction() method of the UCBModel class, and adjusts the prediction accordingly. The getPrediction() method returns an instance of the Prediction class for the UCB model.

3.2.1.7 UCBModel

This class contains methods that perform the calculations used by the UCB model in order to get a prediction. In addition to these methods, two public methods are available for use by an instance of UCBModel. The public method setVals(double inVals[], double outVals[]) takes the current input and output data and checks the validity of the values. The method getInitialPrediction() runs the calculations and returns the prediction.

3.2.1.8 Prediction

This class is used to store a prediction value and its time. The public methods getValue() and getTime() allow access to the attributes of the current prediction for a model.
3.2.1.9 DstPredictionData

This class contains an array of the most recent predictions made by the Dst model. Each Dst model runs initially using the actual Dst values as input to the neural network. After a specified period of time, however, the neural network must begin to receive the last three predictions as the Dst input values. This array allows for storage of those values required by the neural network, and bumps one value out each time a new prediction is made. The public methods isFull() and isEmpty() allows the model to determine the status of the array. Other public methods, such as getPredictions(), getPrediction(double Jtime), and updatePredictionArray(Prediction p) allow for the model to grab any values from the array or to update the array with the latest prediction.

3.2.1.10 ImfSwData

This class establishes the interface to the data file containing the current ACE IMF and solar wind data. Once the data is loaded into an array by the DataLoader class, the public methods of the ImfSwData class provide access to each of the fields available in the data. The class stores the index of all the fields available. Data can be accessed at any time, given the Julian time via the getData(String var, double Jtime) method. The methods getBfield(), getVbs(), and getPressure() perform calculations on the data to get these values.

3.2.1.11 DstData

This class extends the NowcastData class to provide methods specifically for accessing the Dst values. Methods are provided to check if the Dst is valid and to get the Dst value for specific times.

3.2.1.12 NowcastData

This class establishes the interface to the data file containing the current nowcast data. Once the data is loaded into an array by the DataLoader class, the public methods of the NowcastData class provide access to each of the fields available in the data. The class stores the index of all the fields available. Data can be accessed at any time, given the Julian time via the getNowcast(double Jtime) method.

3.2.1.13 URLMappings

This class sets up the URL values for the data files, by reading the urls_for_data.prop file. Each entry of the data file contains a string value (e.g. “IMF_SW_URL”) and the location of the data directory or file the string represents. The public method getURL(String dataName) returns the location of the data as a URL instance, given the string value of the data.
3.2.1.14 DataLoader

This class reads a given file and sets up a multi-dimensional array of the data contained in that file. The public method getDataArray (int index) allows users to get a single array of a specific data field given the index. The method findDataPoint (int dataIndex, double value) returns the index of a specific value if it exists in the data field array. If a data value for a specific time does not exist in the current data array, the method getInterpolatedValue(double data[], double time[], double t) will calculate that value by interpolation and return it.

3.2.1.15 JulianTime

This class contains several static methods for use with Julian dates.

3.2.1.16 NNOutput

This class is used by the DstPredictor to output the 1-Hr and 4-Hr forecast values into separate data files. The public methods used to do this are output1HrForecast (Prediction p) and output4HrForecast(Prediction p). These output files are used by the plotting tools of the Forecasting Dst system.

3.2.1.17 NNLogs

This class is used by the DstPredictor to log the predictions for all models. Each model is run simultaneously, but not all predictions may be output for use by the plotting tools of the Forecasting Dst system. A separate log file for each model is created for each day to store all predictions from that model.

3.2.1.18 LogFile

This class is used by the NNLogs class to set up a log file for a specific model. Public methods are provided to both append and access data in the log file.

3.2.1.19 ErrorLog

This class provides a single static method reportError(String error) to output error messages from the prediction software into an error file.
4 External Interface

4.1 Input Data

The Forecasting Dst system takes as input real-time ACE SWEPAM, MAG, and location data as provided by NOAA. The system also utilizes the Quicklook Dst data as output by the World Data Center for Geomagnetism site.

Below is a sample of an input file for the ACE SWEPAM data. This file is ingested from the NOAA/SEC ftp server, where a new file is made available every five minutes. The header has comment lines starting with either a “#” or a “:” character. The file includes six date columns: year, month, day, time of day as HHMM, Julian day, and seconds within the day. The status flag and the ion temperature columns are ignored. The proton density and bulk speed are propagated, averaged and merged by the perl software with the ACE MAG data to create a single data file for use by the Dst prediction Java software. Note that the fill value for this data is –9999.9.

The Dst Forecasting software also obtains and uses the ACE MAG real-time data. A sample file is shown below for the magnetometer data. The header has comment lines starting with either a “#” or a “:” character. The data columns are the same as those for the ACE SWEPAM data: year, month, day, time of day as HHMM, Julian day, and seconds within the day. The status flag, Bt, latitude, and longitude fields are ignored. The Bx, By, and Bz fields are propagated, averaged and then merged by the perl software with the ACE SWEPAM data to create the input file used by the Dst prediction Java software. Note that the fill value for this data is –9999.9.

---

The Forecasting Dst system takes as input real-time ACE SWEPAM, MAG, and location data as provided by NOAA. The system also utilizes the Quicklook Dst data as output by the World Data Center for Geomagnetism site.

Below is a sample of an input file for the ACE SWEPAM data. This file is ingested from the NOAA/SEC ftp server, where a new file is made available every five minutes. The header has comment lines starting with either a “#” or a “:” character. The file includes six date columns: year, month, day, time of day as HHMM, Julian day, and seconds within the day. The status flag and the ion temperature columns are ignored. The proton density and bulk speed are propagated, averaged and merged by the perl software with the ACE MAG data to create a single data file for use by the Dst prediction Java software. Note that the fill value for this data is –9999.9.

The Dst Forecasting software also obtains and uses the ACE MAG real-time data. A sample file is shown below for the magnetometer data. The header has comment lines starting with either a “#” or a “:” character. The data columns are the same as those for the ACE SWEPAM data: year, month, day, time of day as HHMM, Julian day, and seconds within the day. The status flag, Bt, latitude, and longitude fields are ignored. The Bx, By, and Bz fields are propagated, averaged and then merged by the perl software with the ACE SWEPAM data to create the input file used by the Dst prediction Java software. Note that the fill value for this data is –9999.9.
The ACE spacecraft location data is also obtained by the Forecasting Dst System for use in propagation of the ACE MAG and SWEPAM data. A sample of this file is shown below. The header has comment lines starting with either a “#” or a “:” character. The date columns are the same as those for the ACE data: year, month, day, time of day as HHMM, Julian day, and seconds within the day. The X-coordinate of the GSE is utilized by the data processing script. Note the fill value for this data is –9999.9.

Below is a sample of a few lines from the file nn_input.dat, which is used as one of the input files for the Dst prediction Java software. It is the result of merging the propagated, averaged ACE SWEPAM and MAG data together. The columns consist of the Julian Time, Bx, By, Bz, density, and speed. The location of this file is specified in the configuration file, urls_for_data.prop.

Along with the ACE real-time data, the Dst Forecasting System obtains the Quicklook Dst from the World Data Center for Geomagnetism website. A sample of this file is shown below. The file is an HTML file and has comment lines.
starting with "<!--" character set and ending with "-->". The data is separated in each row according to the day and the hour. The script extracts the Dst data into a .dat file and then transfers the Quicklook Dst value along with the time in Julian format to a file for use by the Dst prediction Java software. Note that the fill value for this data is 9999.

Below is a sample of a few lines from the file nowcast.dat, which is used as one of the input files to the Dst prediction Java software. It is the result of converting the Quicklook Dst data file to a file consisting of only the columns Julian Time and Dst.

```
53126.500000 -17
53126.541667  -17
53126.983333  -27
```

The location of this file is specified in the configuration file, urls_for_data.prop.

### 4.2 Operating System Services

The Forecasting Dst system, implemented in Java, will run on any machine with at least Java 1.3 installed. The shell scripts and perl scripts that supervise and handle the acquisition and processing of the data, along with the generation of data plots, will run on any machine where certain utilities are present (bash, cron, ftp).

### 4.3 Output

The output of the Forecasting Dst system includes:

- A current 1-Hr forecast
• A current 4-Hr forecast
• A log of all predictions made by each model
• An error log
• GIF plots of the daily and weekly values of Bx, By, and Bz
• GIF plots of the daily and weekly values of the solar wind speed
• GIF plots of the daily and weekly values of the solar wind proton density
• GIF plots of the daily and weekly Dst 1-Hr and 4-Hr forecast values

The plots are all linked into web pages, included with the system, that display the 24-Hour plots and 7-Day plots. Examples of these plots are shown in Appendix C.

4.4 Other Software Programs or Libraries

The Forecasting Dst system was implemented in Java, and so runs on any Java-enabled operating system. The software relies on neural net library math routines, which were created in-house. Finally, an in-house Java library is used for generic functionality such as simple math and string operations. All required libraries are provided with the Forecasting Dst system. Although development was done on both Windows 2000 Professional and Solaris 2.7, the software is not limited to these platforms, but should run on any platform supporting Java and perl. Also, a GNU gcc compiler should be present on the target system to compile the C programs.
5 Related Documentation


### Appendix A  Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>Advanced Composition Explorer</td>
</tr>
<tr>
<td>AE</td>
<td>Auroral Electrojet Index</td>
</tr>
<tr>
<td>AFWA</td>
<td>Air Force Weather Agency</td>
</tr>
<tr>
<td>APL</td>
<td>Applied Physics Laboratory of Johns Hopkins University</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>Bx, By, Bz</td>
<td>Components of the Interplanetary Magnetic Field</td>
</tr>
<tr>
<td>Dst</td>
<td>Disturbance Storm Time Index</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphic Interchange Format</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>IMF</td>
<td>Interplanetary Magnetic Field</td>
</tr>
<tr>
<td>JHU</td>
<td>Johns Hopkins University</td>
</tr>
<tr>
<td>MAG</td>
<td>Magnetometer (ACE Instrument)</td>
</tr>
<tr>
<td>NOAA</td>
<td>National Oceanic and Atmospheric Administration</td>
</tr>
<tr>
<td>SWEPAM</td>
<td>Solar Wind Electron, Proton, and Alpha Monitor (ACE Instrument)</td>
</tr>
<tr>
<td>UCB</td>
<td>University of California at Berkeley</td>
</tr>
<tr>
<td>UPOS</td>
<td>University Partnering for Operational Support</td>
</tr>
<tr>
<td>VBs</td>
<td>Solar wind electric field</td>
</tr>
</tbody>
</table>
Appendix B  Source Code

The source code is available upon request from the UPOS Transition Team. An on-line request can be made via the feedback link on the JHU/APL UPOS web page: http://sd-www.jhuapl.edu/UPOS/index.html.
Appendix C  Sample Output Plots

24-Hour Plots