PL-TR-92-2240

PREPARATION FOR ANALYSIS OF DATA FROM THE LANGMUIR PROBE INSTRUMENT ON THE CRESS SATELLITE

Forrest S. Mozer

University of California
Space Sciences Laboratory
Berkeley, CA 94720

10 September 1992

Final Report
17 March 1987 - 17 June 1992

PHILLIPS LABORATORY
Directorate of Geophysics
AIR FORCE MATERIAL COMMAND
HANSCOM AIR FORCE BASE, MA 01731-5000

This technical report has been reviewed and is approved for publication

[illegible]

MICHAEL SMIDDY
Contract Manager

[illegible]

NELSON C. MAYNARD
Branch Chief

[illegible]

WILLIAM SWIDER
Deputy Division Director

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
9/10/92

3. REPORT TYPE AND DATES COVERED
final: 3/17/87 - 6/17/92

4. TITLE AND SUBTITLE
Preparation for Analysis of Data from the Langmuir Probe Instrument on the CRRES Satellite

5. FUNDING NUMBERS
F19628-87-K-0016
PE 62101F
PR 7601
TA 18
WU AJ

6. AUTHOR(S)
Forrest S. Mozer

7. PERFORMI"G ORGANIZATION NAME(S) AND ADDRESS(ES)
Space Sciences Laboratory
University of California
Berkeley, CA 94720

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Phillips Laboratory
Hanscom AFB, MA 01731-5000
Contract Manager: Michael Smiddy/GPSG

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
PL-TR-92-2240

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

With support from Phillips Laboratory (formerly AFGL) the Langmuir Probe/Electric Field Experiment was designed, built, and flown on the CRRES satellite to obtain high quality measurements of plasma density, electric fields, and waves in the inner magnetosphere. A data reduction and display program was written and initial results were reported. The spacecraft failed after more than one year of successful data collection.

The Principal Investigator for this project was Professor Forrest Mozer. The Project Scientist was Dr. John Wygant. The Project Engineer was Mr. Peter Harvey, and the Project Mechanical Engineer was Dr. David Pankow. This work was performed in collaboration with Dr. N. Maynard, Dr. H. Singer, Dr. M. Smiddy, and Mr. W. Sullivan of Phillips Laboratory, and Mr. P. Anderson of Boston University.

14. SUBJECT TERMS
electric field, Langmuir probe, CRRES

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	SAR

Discussion

The CRRES Electric Field/Langmuir Probe instruments consisted of a main electronics package on the spacecraft body and two pairs of orthogonal sensors with tip-to-tip separations of about 100 meters in the spin plane of the spacecraft. One pair of sensors were spherical probes and the other pair were cylindrical antennas. The instrument provided measurements of the quasistatic two-dimensional electric field in the spin plane of the spacecraft at a rate of 32 samples/s, a sensitivity of better than 0.1 mV/m, and a dynamic range of 1000 mV/m. The spherical probes were periodically swept in either current or voltage to determine plasma density and temperature through ground analysis of the resulting Langmuir characteristic curve. The measurement was accurate for densities between 0.1 and 10,000 electrons/cm^3 and electron temperatures ranging from a fraction of an eV to 100 eV. The instrument also had a programmable burst memory which provided selected high time resolution data at cumulative rates up to 50,000 samples/s.

The CRRES orbit, from 258 km at perigee to 33,584 km at apogee, allowed extensive measurements, at all local times, throughout the radial extent of the plasmasphere, ring current, and radiation belts of the Earth. The spacecraft also spent significant periods of time in the near-Earth plasmasheet during magnetically disturbed periods. The instrument measured electric fields associated with injection of plasmasheet particles into the inner magnetosphere; the wave mode responsible for particle loss through precipitation into the ionosphere; the role of electric fields in radiation belt particle energization; and plasma processes which couple along the magnetic field line to produce acceleration of electrons to form auroral arcs at lower altitudes. In addition, the instrument provided measurements during both high- and low-altitude barium and lithium releases. Some examples of the data that have been analyzed are presented below.

An example of early data from the CRRES instrument is presented in Figure 1. Illustrated in the top panel are density fluctuations measured by spherical double probes, in the middle panel are electric field fluctuations measured by the cylindrical double probes, and the lower panel illustrates magnetic field fluctuations measured by an instrument provided by the University of Iowa. This example is a whistler wave that is modulated on a 30 ms time scale and that appears in density cavities.

In situ measurements of electric and magnetic fields were obtained during a large ionospheric barium release on July 19, 1991. While previous measurements of chemical releases have shown that Alfvèn waves are excited, these CRRES measurements provided the first direct evidence for the magnetic braking of a plasma cloud through the formation of intense Alfvèn waves standing in the release frame. The electric field measurements show a 300 mV/m perturbation associated with the motion of the release plasma relative to the ambient medium (see Figure 2). Coincident with the electric field variation, an intense magnetic field perturbation (600 nT) was observed and attributed to a field-aligned current system. We estimate that the $J \times B$ force due to the perpendicular closure of this field-aligned current (0.5 A/m) can transfer momentum from the release to the ambient medium at a rate consistent with the measured deceleration time scale (1-2 seconds) obtained from LANL ground-based optical observers. The ratio, $\delta E/\delta B$, is about equal to the Alfvèn velocity. In addition, as displayed in Figure 3, other features of this release include observations of intense low-frequency electrostatic fluctuations (100 mV/m) polarized perpendicular to the magnetic field at the release boundary, and a variety of Alfvènic fluctuations observed inside and outside the release.

Similar data were obtained during barium releases at altitudes between the ionosphere and L = 7.25. A summary of the measured duration of the perturbed electric field as compared to a theoretical estimate of the cloud deceleration time in an Alfvèn wing deceleration model is given in Figure 4. The proportionality between theory and experiment suggests that, at all altitudes, the barium cloud is braked by Alfvèn wave interactions.

Another interesting example from the CRRES data set involves an encounter of the Earth's magnetosphere with an interplanetary shock on March 23, 1991, at about 3:41 UT. At this time, the spacecraft was at about L=2.6 and 3:00 MLT near the equatorial plane. During the sudden commencement associated with this shock, for approximately 90 seconds, the electric and magnetic field detectors observed some of the largest variations ever observed in the inner magnetosphere. These observations coincided with the formation near L=2.6 of a new electron radiation belt with fluxes 3-4 orders of magnitude larger than the previously observed electron fluxes at 15 MeV. The time scale associated with the formation of the new belt was a fraction of a 15 MeV drift period. As has been previously reported, the observed fluxes of electrons in the 15-30 MeV range dominated over pre-existing belt fluxes by two orders of magnitude for the remaining six months of the CRRES lifetime. The electric field measurements are the first such observations of a shock-induced sudden commencement in the inner magnetosphere. The 60 mV/m (peak-to-peak) variations in the electric field observed during this event are an order of magnitude larger than the largest sudden-commencement field observed by the GEOS electric field experiment. They are two to three orders of magnitude larger than the typical electric field seen at L=2.6 and also two orders of magnitude larger than the 0.1-1 mV/m electric field fluctuations thought to drive the stochastic processes of radial diffusion and particle energization in the inner magnetosphere over much longer periods of time. Since radial diffusion coefficients scale with the square of the electric field fluctuation amplitude, these large amplitude electric fields which resulted in a "single step" acceleration of particles were probably four to six orders of magnitude more efficient in transporting and energizing particles than multi-step radial diffusion processes. A summary of the electric field and particle measurements is presented in Figure 5.

Another significant result from the CRRES research program was the development of general purpose software for processing, analyzing, and displaying electric and magnetic field data. The status of this effort is:

1. Survey data files and plots at spin-time resolution have been created for all 1045 orbits which contain Langmuir Probe telemetry.

2. A number of derived science quantities have been added to the current interactive CRRES viewing program. These include:

   Software-generated E-field spin fits
   $\bar{V} \times \bar{B}$ subtraction
   Spin period
   Angle between cylinder E-field and sphere E-field
   Angles between E-field and $\bar{V} \times \bar{B}$
   Angle between Y-MGSE and booms
   Ratios of components of E-field and $\bar{V} \times \bar{B}$ subtractions
   Sphere and cylinder boom positions in ECI
   X-component of E-field by use of $\bar{E} \bullet \bar{B} = 0$
   Sine-wave subtraction

3. Software least-squares spin fits have been performed and stored in files for every orbit with Langmuir Probe telemetry.

4. The CRRES viewing program outputs ASCII files with an arbitrary selection of the instrument and physics quantities that are available for plotting.

5. The CRRES viewing program operates in unattended batch mode to create hardcopy plots or ASCII file output.

6. A new version of the interactive CRRES viewing program is under development and is currently 80% complete. It is described in the following documents, attached as Appendices A and B, respectively: "Functional Description of the New Version of the 'Fiche' Software" and "Conceptual Design of the CRRES Software."

Ongoing activities at the end of this contract are completion of the automated data reduction and presentation program and continuing analyses of the average DC electric field inside L=7, ion cyclotron waves in the ring current, the formation of trapped particle radiation belts due to sudden-commencement electric fields, relation of the plasmapause location to penetrating electric fields, the normal component of the electric field at the plasmasheet boundary layer, and small-scale electric fields near the inner edge of the ring current.

A list of publications resulting from the work completed to date is attached as Appendix C.

Fig. 1. Observations of whistler wave packets imbedded in 10-100 second density depletions from the UCB/AFGL Langmuir Probe/Electric Field Experiment on CRRES. The data was obtained through instrument burst memory at a data rate of 10^3 samples/sec.

Fig. 2. *In situ* measurements of electric fields, spacecraft potential, and magnetic fields over a ten-second interval containing a large barium chemical release at ionospheric altitudes. Both electric and magnetic field data are presented in a sensor coordinate system rotating with the spacecraft. The data is also presented in the spacecraft frame.

Fig. 3. This figure shows an enlarged view of the electric field component nearly perpendicular to the spacecraft velocity and the local magnetic field direction (panel 1) and the magnetic field measurement along the spacecraft velocity vector. The electric and magnetic field measurements provide evidence for an Alfvèn wave propagating along $\vec{B}$ which act to decelerate the release on a time scale of 1-2 seconds.

Fig. 4. Comparison of the measured deceleration time scale of six plasma clouds to that predicted (t*) from Alfvèn wings.

Fig. 5. Measurement of the YGSE component of the electric field and electron count rates from four energetic particle detectors ranging in amplitude from >6 MeV to 10 - 50 MeV. The spacecraft was located at L = 2.5 and MLT = 2.5. The spacecraft observed large-amplitude electric fields and drift echo electrons as the first step in formation of a new radiation belt on March 24, 3:41 UT when the Earth encountered an interplanetary shock.

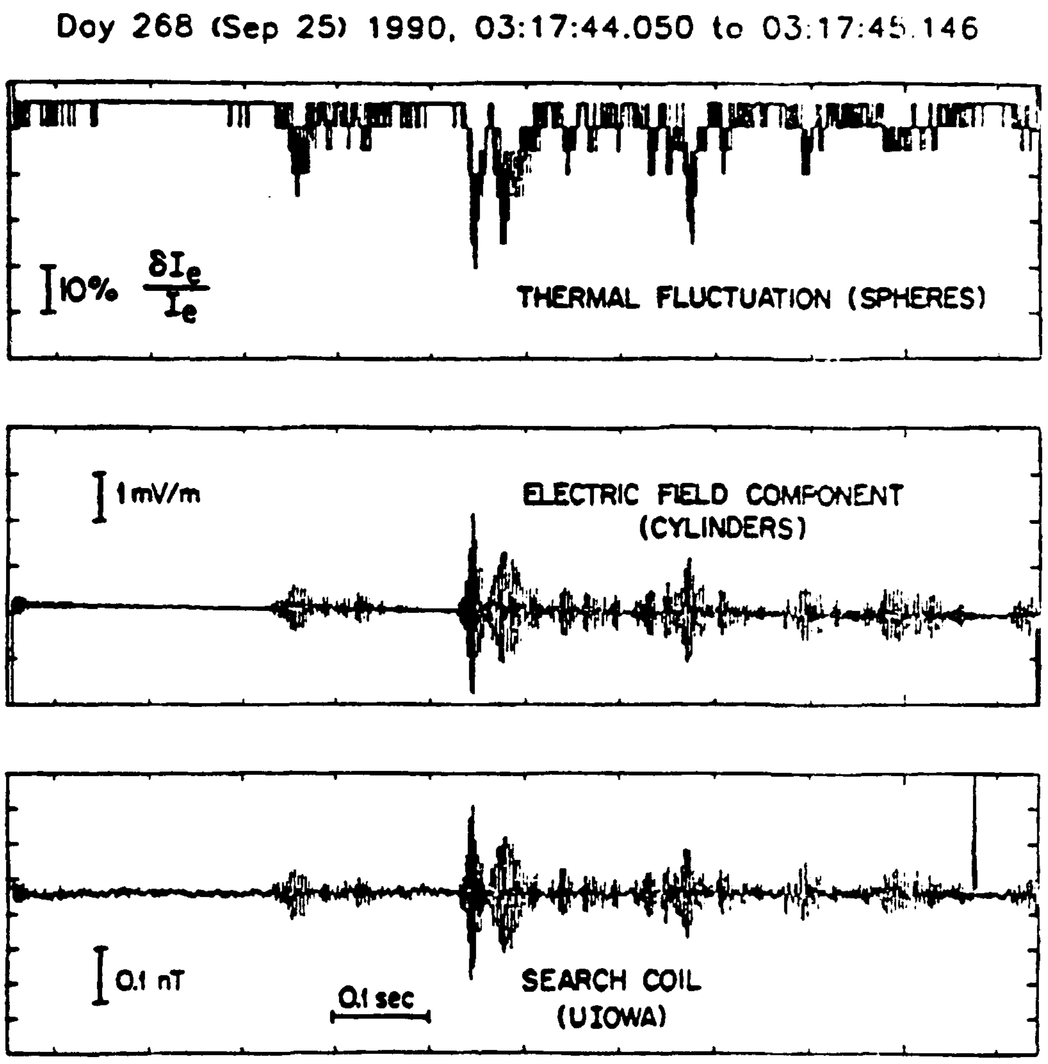
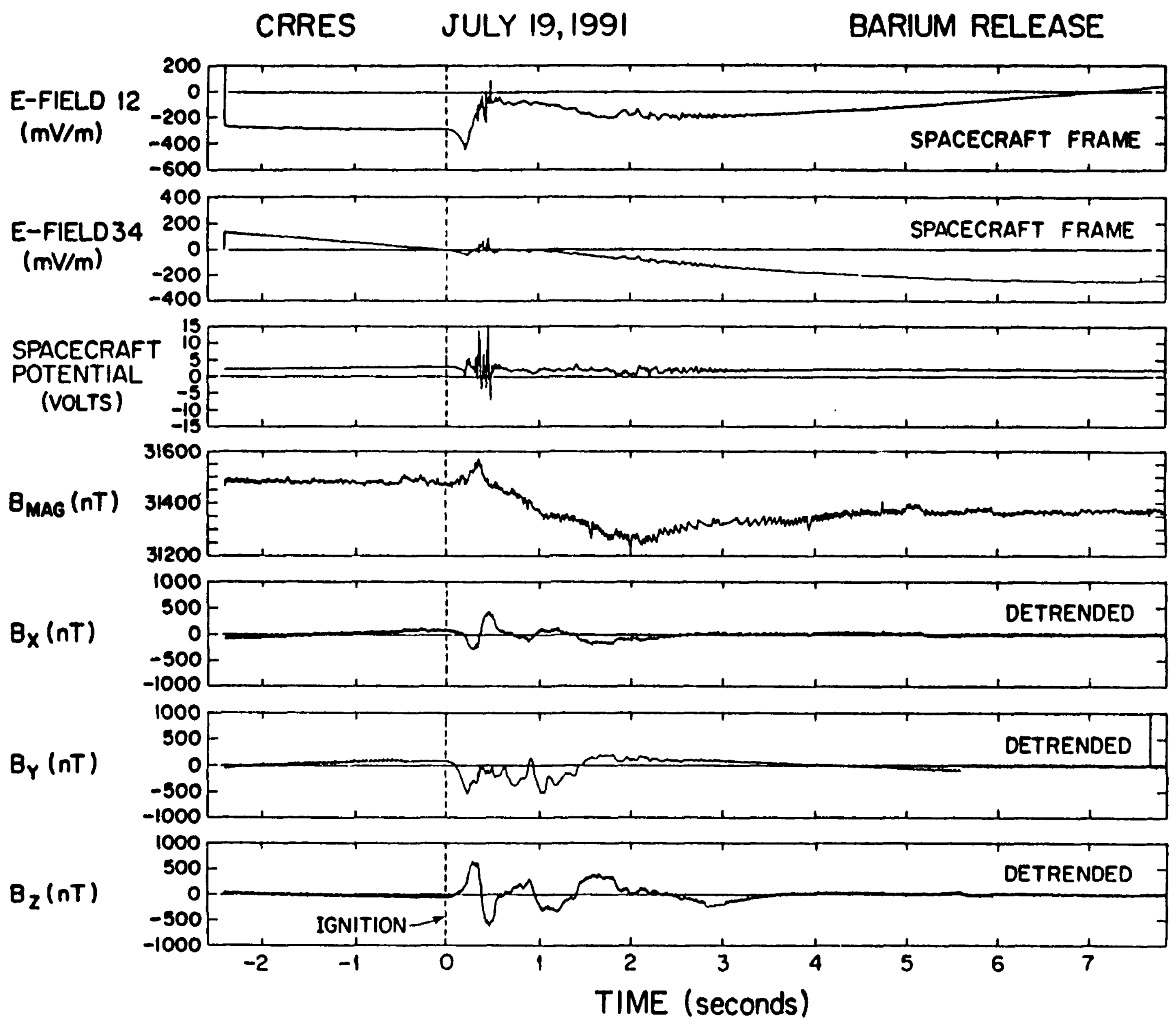

Day 268 (Sep 25) 1990, 03:17:44.050 to 03:17:45.146
10% $\frac{\delta I_e}{I_e}$
THERMAL FLUCTUATION (SPHERES)
1 mV/m
ELECTRIC FIELD COMPONENT (CYLINDERS)
0.1 nT
0.1 sec
SEARCH COIL (UIOWA)

Figure 1



CRRES
JULY 19, 1991
BARIUM RELEASE
E-FIELD 12 (mV/m)
200
0
-200
-400
-600
SPACECRAFT FRAME
E-FIELD34 (mV/m)
400
200
0
-200
-400
SPACECRAFT FRAME
SPACECRAFT POTENTIAL (VOLTS)
15
10
5
0
-5
-10
-15
B_MAG (nT)
31600
31400
31200
B_X (nT)
1000
500
0
-500
-1000
DETRENDED
B_Y (nT)
1000
500
0
-500
-1000
DETRENDED
B_Z (nT)
1000
500
0
-500
-1000
DETRENDED
IGNITION
-2
-1
0
1
2
3
4
5
6
7
TIME (seconds)

Figure 2

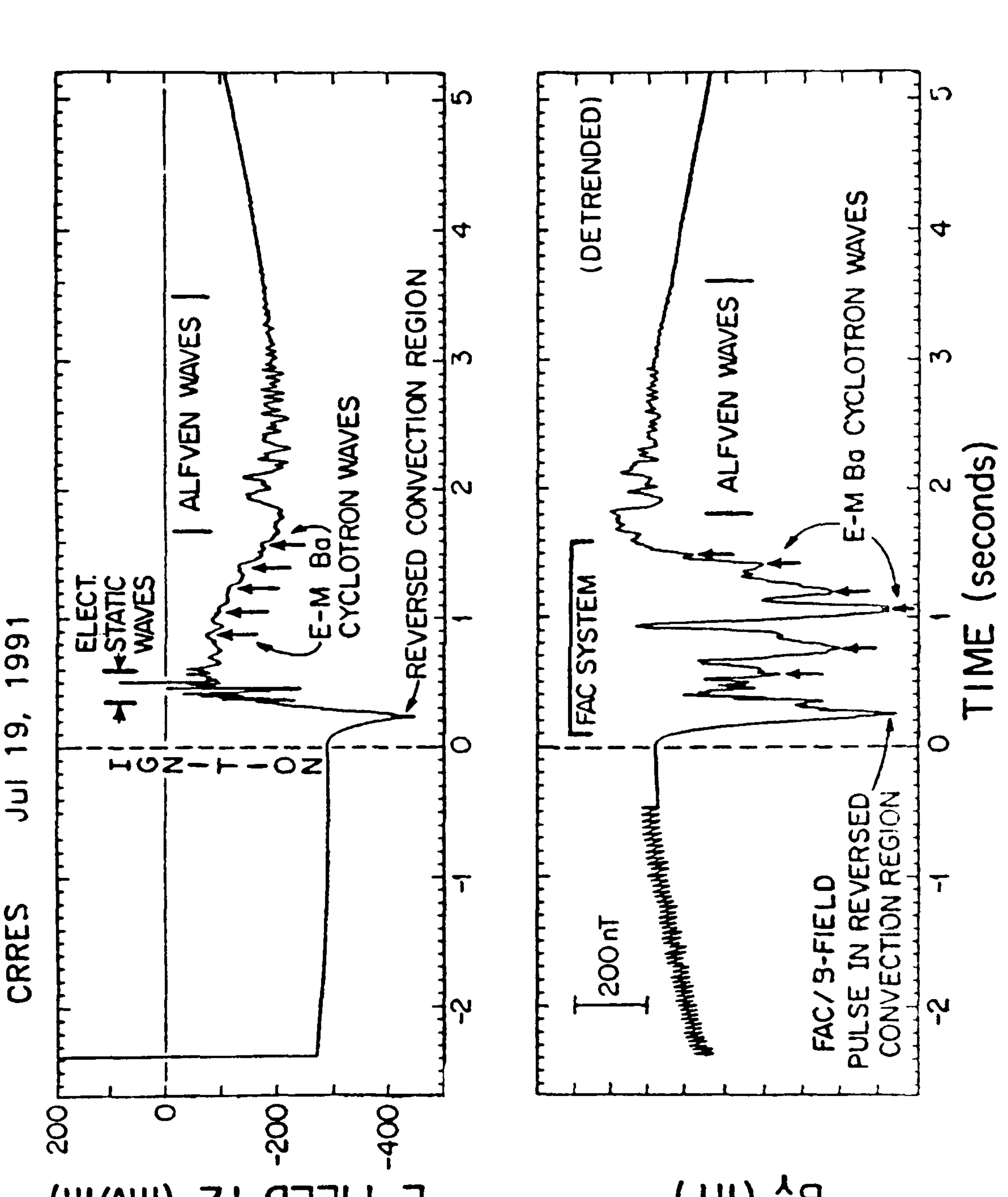

CRRES Jul 19, 1991
E-FIELD 12 (mV/m)
200
0
-200
-400
IGNITION
ELECT. STATIC WAVES
ALFVEN WAVES
E-M Ba CYCLOTRON WAVES
REVERSED CONVECTION REGION
By (nT)
200nT
FAC SYSTEM
(DETRENDED)
ALFVEN WAVES
E-M Ba CYCLOTRON WAVES
FAC/B-FIELD PULSE IN REVERSED CONVECTION REGION
-2
-1
0
1
2
3
4
5
TIME (seconds)

Figure 3

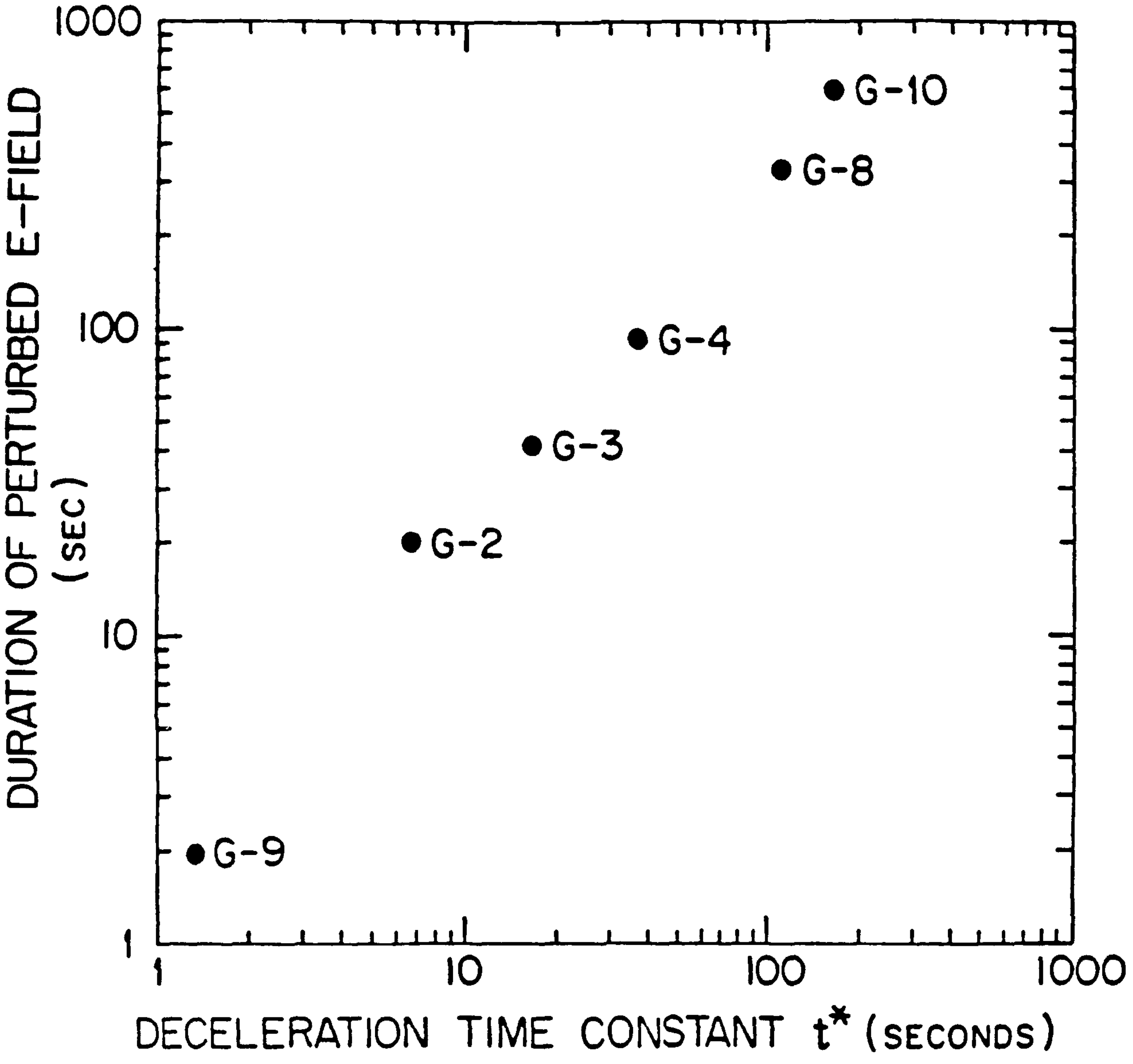

1000
100
10
1
1
10
100
1000
DURATION OF PERTURBED E-FIELD (SEC)
DECELERATION TIME CONSTANT t* (SECONDS)
G-10
G-8
G-4
G-3
G-2
G-9

Figure 4

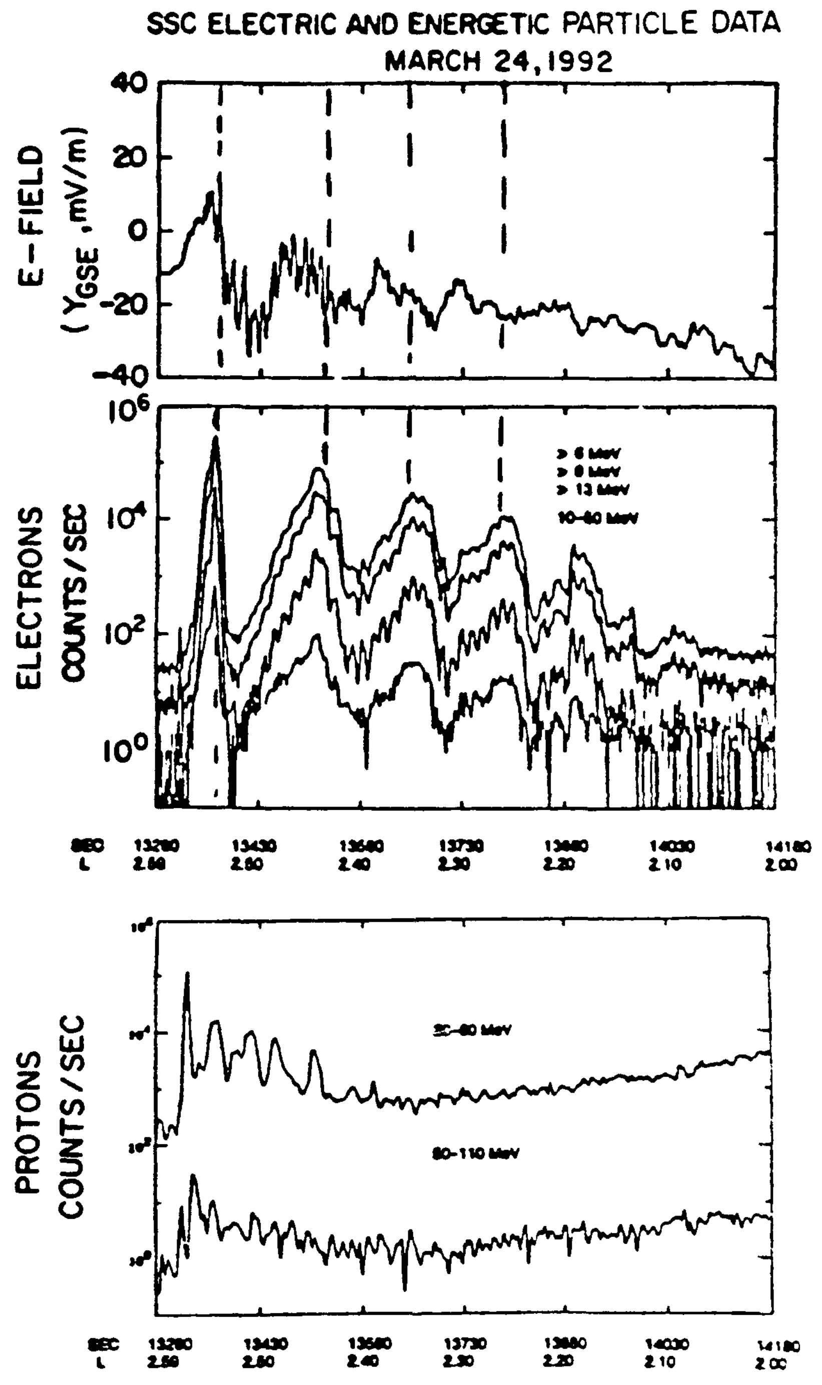

SSC ELECTRIC AND ENERGETIC PARTICLE DATA
MARCH 24,1992
E-FIELD
(Y_{GSE} ,mV/m)
40
20
0
-20
-40
ELECTRONS
COUNTS/SEC
10^6
10^4
10^2
10^0
> 6 MeV
> 8 MeV
> 13 MeV
10-60 MeV
SEC
L
13280
2.68
13430
2.60
13580
2.40
13730
2.30
13880
2.20
14030
2.10
14180
2.00
PROTONS
COUNTS/SEC
20-80 MeV
80-110 MeV

Figure 5

Appendix A

Functional Description of the New Version of the "Fiche" Software

I. Facilities and Expectations

The New Version of the "Fiche" Software running on a Sun SparcII Workstation using CRRES satellite data and its associated science algorithms will allow a user to view either raw instrument data quantities (see APPENDIX I for a listing) or data which results from CRRES related S ience Algorithms (see APPENDIX II for a listing) in an interactive manner.

The user will accomplish this by starting the CRRES "Fiche" Software. The user will then be expected to select an appropriate timespan (corresponding to a data file or files containing the orbit or orbits) and to indicate a set of quantities one wishes to view. This may be accomplished piece-meal or by selecting a previously defined configuration file. At this point the decommutator will begin decoding all of the raw data from the orbit files followed by the computation of requested science quantities. As the raw instrument quantities are unpacked and as the scientific quantities are computed the data will be displayed on in the user interface plot panels in a "real-time" manner.

The user may then pan through the data, zoom-in, zoom-out, scale the data up or down for finer granularity, select different numbers of points, select a new timespan for viewing, add new plot panels, delete existing plot panels, rearrange the current plot list, save the current configuration, select a new configuration, or otherwise set his/her view of the data; in short the user can perform any of the existent functions of the current version of the "Fiche" Software.

The user can realistically expect instant replotting of CRRES raw data quantities as long as one stays within the requested timespan. This is possible since all of the raw data is decommutated at the time of timespan request and is stored in contiguous memory arrays. These can be accessed very quickly by the software and can therefore provide this performance. When a new timespan is requested, however, the decommutator will decode all of the raw data at that time. The user can expect this, again, to take several minutes. Thus the user optimizes his/her time by selecting judiciously the timespan in which he/she is interested. Note that the larger the time-span the larger the amount of memory needed to hold the data. The user will be restricted at the upper end of length of allowable timespan by the amount of available system memory.

The CRRES Scientific Computation data quantities, on the other hand, are calculated on an as requested basis for a particular viewing timespan. This is the case because the computations are generally very time consuming compared to the decommutation of raw data and the amount of available system memory is far to small to hold all of the available "Science Quantitites". Computations on all points for an hour's wort. of Electric Field data (32 points/second) could take on the scale of (30)? seconds. Furthermore, if a user is zoomed in to a particular level and the user chooses another viewing timespan (larger or smaller) by zooming out/in, the science quantities will be recomputed for the new viewing timespan. Since the time needed to yield the desired scientific quantities is considerable and they will be recomputed often, an option will exist to cull down the number of points computed for a given timespan. This will, in turn, proportionally reduce the amount of time required to complete the computations. This can be used to optimize one's use of time. Thus the user, here, optimizes his/her time by setting the number of points to a small value until he/she closes in on the particular data he/she wishes to view. It is important to note here that the memory resident raw instrument data, along with the restructuring of the software should be expected to improve performance of even these time consuming Science Algorithms over the present version.

The New Version of the "Fiche" Software will also provide facilities for getting output such as printed plots & ascii dumps and for running plots in a batch mode.

II. Computing Standards

The New Version of the "Fiche" Software will be developed on a Sun Sparc II Workstation with 64 MegaBytes of memory and a 2 GigaByte hard disk running Sun UNIX. The goal of the developers will be to develop portable software which will run on most Standard UNIX Workstations or Minis which have enough memory (RAM or virtual) to hold the application data (>50 MegaBytes for CRRES) and enough disk space (>35 MegaBytes) to support Sun Mapped Memory plus whatever is necessary to hold the data files and to support the virtual memory requirements.

More specifically, the developers will support at a minimum any of these machines running standard UNIX which supports Shared and/or Mapped Memory, TCPIP, and Sockets or TLI (or Sun RPC). For Graphics support, the New Version of the "Fiche" Software will support Xlib and eventually either MOTIF or OpenLook.

This Software will be developed primarily in the ANSI standard 'C' programming language. In addition to this, the developers hope to expand this base and implement portions of the code in a 'C' superset, C++, where a clear cut advantage exists to using the C++ object oriented style of programming.

III. Extensibility

Since the developers of the "Fiche" Software cannot foresee all the possible applications and configurations for this software, the above computing standards represent the minimum baseline targets for initial development. This is not to say that further support and development for future configurations or applications will not be available, but rather that these are the minimum facilities which will be supported for the initial implementation of the New Version of the "Fiche" Software. Future support and development will be carried on in several areas; namely addition of more advance and needed science algorithms by either CRRES developers or by independent programmers, support for future instruments, portability of this software to other platforms, and possibly a version that may be run over a network. These extensions and advancements will be taken into consideration and the software will be designed to facilitate the addition of these capabilities.

The first of these extensions will be the addition of further Science Algorithms. There will be two separate ways of accomplishing this; one will be to simply add the new algorithm to the existing CRRES Science Algorithm Module and the other will be to essentially create another versions of the Science Algorithm Module which will look and behave exactly like it, but which will implement different computations. This second type of interface may support Fortran, and possibly, far in the future, IDL.

Certainly the use of the main portion of this software for future satellites, namely POLAR, CLUSTER, and FAST, is required Although the new satellites will have their own decommutator decoding scheme and its own associated science computations, the user interface, Science Algorithm and Decommutator Super structure will be identical. This will leave any modification and development for future satellites to those sections which are specific to that instrument. As a possibility, other instruments may be supported (e.g. rockets, file formats, etc.). At a minimum, very well defined interfaces will exist which will allow developers from other instruments to create their own software to attach to the main portion of the CRRES "Fiche" Software and provide its facilities.

In the above case it may be desirable to port this software to other types of systems (hardware and OS). The developers will attempt to support this portability in the following limited way: The developers will attempt to use the ANSI standard 'C' language and a standard library (Xlib, etc.) to promote easy conversion to other compilers. The support of Shared or Mapped memory will have to be rewritten by the users whose systems cannot support it.

The last extension which will be discussed is the possibility of running the separate functional parts of the software over a network for more beneficial resource allocation (i.e. running the decommutator and science algorithms from a powerful machine and leaving the lesser tasks of the user interface for a local terminal). This will be supported by the separation of the software into large functional blocks of standalone processes. These will be able to run independently of the (location of) the other programs and thus will be more easily configured to meet the requirements of network operation.

These are the facilities and goals for the New Version of the "Fiche" Software. Our aim is to provide the excellent facilities for data analysis which are available in the current software while providing opportunities for easy modification, adaptation, and expansion in the future.

APPENDIX I

CRRES Raw Instrument Data

Fast Digital Monitor
FDM Current/Voltage mode
FDM Command Error Count
FDM Test Calibrate mode
FDM Bias Sweep
FDM Playback on/off
FDM Playback Main/Burst
FDM Burst condition (1=search,2=collect,3=wait)
V3 (cylinder) (mux 3)
V2 (sphere) (mux 4)
V1 - voltage mode only (sphere) (mux 5)
V1 both voltage and current mode (mux 6)
V12 voltage difference spheres (mux 8)
V12 through BP Filter F3 (2048 Hz center freq) (mux 9)
V12 through BP Filter F2 (256 Hz center freq) (mux 10)
V12 through BP Filter F1 (32 Hz center freq) (mux 11)
V4 (cylinder) (mux 12)
V12 unfiltered (mux 14)
V34 voltage difference cylinders (mux 15)
RI2 current from sphere 2 (mux 4)
Search Coil (mux 5)
RI1 current from sphere 1 (mux 8)
RI1 through BP Filter F3 (2048 Hz center freq) (mux 9)
RI1 through BP Filter F2 (256 Hz center freq) (mux 10)
RI1 through BP Filter F1 (32 Hz center freq) (mux 11)
RI1 current from sphere 1 - unfiltered (mux 14)
Magnetometer BZ
Magnetometer BY
Magnetometer BX
Burst BZ (bmux 0)
Burst BX (bmux 1)
Burst BY (bmux 2)
Burst V3 (bmux 3)
Burst V4 (bmux 4)
Burst V34 (bmux 5)
Burst V34 AC (bmux 6)
Burst V1 - voltage mode (bmux 7)
Burst V12 AC (bmux 8)
Burst V2 (bmux 9)
Burst V1 (bmux 10)
Burst V12 (bmux 11)
Burst Direct AGC (bmux 12)
Burst AGC unfiltered (bmux 13)
Burst Guard 1 (bmux 14)
Burst Stub 1 (bmux 15)
Burst Search Coil (bmux 7)
Burst RI1 AC (bmux 8)
Burst RI2 current (bmux 9)
Burst RI1 (bmux 10)
DSC Number signed ###
DSC Number unsigned ###
DSC Sun angle
DSC Sun period
Raw Mux Channel ###
Bias value (Voltage mode)
V1 Bias Sweep
V2 Bias Sweep
V3 Bias Sweep
V4 Bias Sweep
Bias value (Current mode)
I1 Bias Sweep
I2 Bias Sweep
Diagnostic Bias stepping (RAM quantity)
Spheres: Spin Fit\nCoefficient AHI
Spheres: Spin Fit\nCoefficient ALO
Spheres: Spin Fit\nCoefficient B
Spheres: Spin Fit\nCoefficient C
Spheres: Spin Fit\nStd. Deviation
Spheres:\nPoints in Spin Fit
Spheres:\nSQRT (B\u2 + C\u2)
Cylinders: Spin Fit\nCoefficient AHI
Cylinders: Spin Fit\nCoefficient ALO
Cylinders: Spin Fit\nCoefficient B
Cylinders: Spin Fit\nCoefficient C
Cylinders: Spin Fit\nStd. Deviation
Cylinders:\nPoints in Spin Fit
Cylinders:\nSQRT (B\u2 + C\u2)
Ey\nSphere
Ez\nSphere
Ey\nCylinder
Ez\nCylinder
Spheres:32pt Spin Fit\nCoefficient AHI
Spheres:32pt Spin Fit\nCoefficient ALO
Spheres:32pt Spin Fit\nCoefficient B
Spheres:32pt Spin Fit\nCoefficient C
Spheres:32pt Spin Fit\nStd. Deviation
Spheres:32pt\nPoints in Spin Fit
Spheres:32pt\nSQRT (B\u2 + C\u2)
Spheres:32pt Spin Fit\nCoefficient D
Cylinders:32pt Spin Fit\nCoefficient AHI
Cylinders:32pt Spin Fit\nCoefficient ALO
Cylinders:32pt Spin Fit\nCoefficient B
Cylinders:32pt Spin Fit\nCoefficient C
Cylinders:32pt Spin Fit\nStd. Deviation
Cylinders:32pt\nPoints in Spin Fit
Cylinders:32pt\nSQRT (B\u2 + C\u2)
Cylinders:32pt Spin Fit\nCoefficient D

APPENDIX II

CRRES Data from Scientific Calculations

LSQ Fit and Subtract from Current Plot
Bias Sweep Linear Fit
  Temp
  Density
Bias Sweep Exp. Fit
  Temp
  Density
Bias Setting
Power Spectrum (Fourier)
B components, MAG
B components, SC
B components, ECI
B components, GSE
B components, MGSE
B components, ECI, model subtracted
B components, GSE, model subtracted
B components, MGSE, model subtracted
mag of B (MAG coords)
mag of B (SC coords)
mag of B (ECI coords)
mag of B (GSE coords)
mag of B (MGSE coords)
mag of B-model (ECI)
mag of B-model (GSE)
vXB-x (MGSE)
vXB-y (MGSE)
vXB-z (MGSE)
vcorotXB-x (MGSE)
vcorotXB-y (MGSE)
vcorotXB-z (MGSE)
vXB(mod)-x (MGSE)
vXB(mod)-y (MGSE)
vXB(mod)-z (MGSE)
Sph SpinFit-vXB(mod)-y
Sph SpinFit-vXB(mod)-z
Cyl SpinFit-vXB(mod)-y
Cyl SpinFit-vXB(mod)-z
sqrt(vXB-y^2+vXB-z^2)
Sphere SpinFit-vXB-y
Sphere SpinFit-vXB-z
Cylinder SpinFit-vXB-y
Cylinder SpinFit-vXB-z
mag(Ecyl)/mag(Esph)
Ang Between Ecyl and Esph
SphEy - VxB_y / |VxB_y|
SphEz - VxB_z / |VxB_z|
CylEy - VxB_y / |VxB_y|
CylEz - VxB_z / |VxB_z|
sqrt((Ey-VBy)\u2+(Ez-VBz)\u2)
sqrt((Ey-VBy)\u2+(Ez-VBz)\u2)
Sph (E, VxB) angle
Cyl (E, VxB) angle
Angle(Ymgse, Sph-Boom1)
Angle(Ymgse, Cyl-Boom1)
Angle(Ymgse, Sph-Boom1)
Angle(Ymgse, Cyl-Boom1)
Sphere boom spin time
Cylinder boom spin time
Sphere boom1_X
Sphere boom1_Y
Sphere boom1_Z
Cylinder boom3_X
Cylinder boom3_Y
Cylinder boom3_Z
Agmod Spinperiod
(V1+V2)/2
(BV1+BV2)/2
(V1+V2)/2, (V3+V4)/2, SC_Vel (MGSE)
V12, V34, Boom1 Angle (to MGSE-Y)
Soft SF of V12
Soft SF of V34

APPENDIX B

Conceptual Design of the CRRES Software

Space Sciences Laboratory
The University of California, Berkeley

January 16, 1992

Table of Contents

Appendix B

Conceptual Design of the CRRES Software

1. Introduction:

It is the purpose of this document to specify the concepts used in the design and implementation of the CRRES viewing software. It serves two main purposes

- It serves as an introduction to the software for anyone with programming responsibilities on the software.

- It serves as an initial guide for anyone who wishes to write extensions to the existing CRRES software.

A more detailed, implementation-level description of the software can be found in the document entitled: "Implementation Notes of the CRRES Software".

2. Description of the Original Version of the CRRES Viewing Program

2.1 Program Design Constraints:

The original CRRES program was designed and implemented in 1990, 1991 by Jack Vernetti, John Caron, and Chris Sparhawk of SSL in the "C" programming language. It was to serve the same purpose as micro-fiche were to older data sets (e.g. ISEE), hence the name "fiche". At the user's request, it was to produce time-series plots on a display screen for selected data quantities, all plots over a common time period.

The following were the major design considerations of the original version:

- The CRRES software was to be portable between Sun UNIX and large extended-memory DOS machines.

- The input data included orbit-by-orbit telemetry, attitude, and ephemeris files supplied by AFGL. The telemetry and ephemeris files were binary, defined at byte level. The attitude file was ASCII. These files were to be readable by both the Sun and PC versions of the program. The telemetry file was defined with a structure very similar to the raw agency format and included an appended index for use in random access. It was not assumed that processed, secondary data sets would be available.

- The program was to be very flexible with regard to which quantities, which time spans, and what vertical scaling could be viewed. A user could view up to 32 plots at once (out of over 100 available) for any time span ranging from a few milliseconds to an entire orbit (10 hours) with the vertical scaling independently set for each plot.

- Only single orbits could be viewed at one time: there was no facility for looking at time spans overlapping two orbits.

- The program was to view, not only processed data from agency tapes, but data from the CRRES GSE as well

2.2 Program Components:

The DOS requirement immediately impelled use of a single, large program which would be responsible for every aspect of the processing (since DOS is not a multi-tasking OS).

Portability between UNIX and DOS also precluded direct use of graphics and user-interface toolkit standard software packages. This required building in-house packages to handle various graphics, user-interface, and system-level operations. These "covering"-libraries included both UNIX and DOS versions of each of the following:

tool1	(basic system operations)
toolm	(basic mathematics routines)
gsh	(graphics primitives, both "display" and "hardcopy")
gob	(user-interface ("graphical objects") routines)

In addition, there was the CRRES decommutation library:

decom

It is this library which knows how to decode the CRRES telemetry, ephemeris, and attitude files. Note that the AFGL "AGMOD" software was incorporated into "decom" and is used for decoding the attitude files.

The application source itself ("fiche") includes four main areas:

fiche	(main user-interface controller and plot generator)
inst	(creates instrument (raw) quantity data buffers)
phys	(creates derived "physics" quantity buffers from raw data).
OP	(handles "operations")

2.3 Program Implementation:

The requirement of supporting DOS recommended an "event-driven" approach to the software. Because memory is so valuable on the PC (even 8 Mbytes of extended-memory is smaller than a packed orbit of non-time-tagged CRRES data), it was not possible to store complete, time-tagged arrays of raw quantities (we assumed that users would often request ten hours of data).

The alternative was to perform as much processing "on-the-fly" as possible when individual data points were returned to the application from the decommutator. Thus, a single raw data point would often be used, as soon as it entered the application, in 5 or 10 different physics computations. The raw data point itself would not be stored in any buffer unless it was to be directly used on one of the displayed plots.

The following indicates how a decommutation session within the program operated:

a. The program ("fiche" module) tells the decommutator exactly which types of data it is interested in. It also indicates whether the decommutator is to send back all possible points of all requested data types or if it is to decrease the data density of all returned data types by limiting the number of points of each type to a positive integer "n" (the default was 2000) of evenly spaced points for any requested time span. It is this data density control which allowed plot data to be buffered whole, in reasonable amounts, for operation of the PC version.

b. The program then tells the decommutator to start sending back the data types mentioned in step (a) for a given time span. One of the arguments to this subroutine call is a data-return function pointer, which indicates where the decommutator is to send back the individual data points. This function is in the "inst" module.

c. Now the decommutator acts like the controlling program which sends individual data points to the application. This is what is meant by an "event-driven" architecture. The decommutator acts like an event-generator (the "events" are the time-tagged data points), sending each data point to the application, which responds in some way to those points. Note that the decommutator doesn't care how the application uses a data point.

d. A single data point from the decommutator would take the following path through the application:

The data-return function in the "inst" module receives the event from the decommutator and figures out what type of data it is.

The data-return function then loops through all of the currently defined plots and sees if this type of data is required for each of those plots (it is required for at least one of them because of step (a)).

When a relevant plot is found, the data is scaled and offset (the scaling and offset coefficients for the type of data was set up at program initialization time).

If this is a "raw" instrument plot, the scaled and offset data is passed directly to the plotting routine in the "fiche" module, which inserts the data into the buffer for that plot.

If this is a "physics" plot, the scaled and offset data is passed into the "phys" module, where it is routed, via a complicated switch statement, to the appropriate code for use in a scientific algorithm. Note that this will only sometimes send a "physics" point to the "fiche" module for plotting, but not always, since the algorithm may need more data before a physics plot point can be generated.

When all of the current plots have been accessed in the loop, control is returned to the decommutator for the next "event".

e. When the decommutator comes to the end of the time-span (from step b), it sends a special "event" to the "inst" module indicating that it is now finished. The decommutator is exited and control is passed up to the "fiche" module which outputs the plots, whose buffers have been set up by the previous steps, to the display.

It should be noted the "OP" software acted directly on currently existing "plot" buffers as stored in the "fiche" module. It was not directly affected by the event-driven nature of the program.

2.4 Program Presentation:

The application presented itself to a user with a large single window with an initial list of predefined user "configurations" to select from. Note that a configuration includes an orbit name, time span within the orbit, and a list of current plots. When the user selected a configuration, the program would start the decommutator for the given time span. When the plot buffers were complete, then the plots would be output and the program would present a set of popup menus that allow the user to change the current plot configuration, change the time span, create hardcopy, etc.

2.5 Program Deficiencies:

As stated above, the "event-driven" nature of the software requires that physics computations be performed on the fly as raw data points are returned from the decommutator. This requires a complicated but very delicate input filter structure which is extremely difficult and dangerous to maintain and extend. Indeed, many program bugs were introduced when adding new "physics" plots to the program's repertoire. Source code reuse and modularization also suffered from the use of this point-by-point filter. Furthermore, the science source code itself now has the responsibility of buffering appropriate data points for use. This is difficult and would be unnecessary if the raw points are already buffered, which would also allow the algorithm designer to concentrate on the scientific algorithm itself.

It also became apparent by the end of 1990 that DOS was an unworkable platform for this application simply from the standpoint of the familiar 640K DOS main memory limit. The program itself, through the addition of physics quantities, grew quite large for a DOS program. The extended memory was usable for data storage, but did not help in terms of executable code. The use of program overlays was not an option because of the programmer time involved in such a difficult process.

The event-driven nature created another problem with the program. Since everything had to be generated on a point-by-point basis, it was necessary to have the decommutator restrict its output to only those quantities required by the set of plots defined by the user's configuration at a given time. In this way much time was saved because no extraneous data would be sent around the program. However, it also meant that if extra plots were requested later or the current time span was increased, the decommutator had to be called again! It should only be necessary to decommutate data once! This is a very complicated and time consuming part of the processing and should be restricted to a single session if possible.

Another deficient area of the original program is in burst processing. Since very large data arrays were not a viable program option and since all data had to be processed on a point-by-point basis, the fact that burst data is played back significantly later than actual burst occurrence made it very difficult to insure correct time-appearance of burst data in the program plots. In fact, some bursts are played back in later orbits than their actual occurrence. A much more robust form of burst handling would entail pre-buffering and time-tagging all the bursts for an orbit before any plotting occurs.

Finally, in general, it is difficult and dangerous to have anyone but those programmers intimately knowledgeable about the single-program architecture make additions to the software. This is unacceptable since it should be reasonably simply for a scientist, or anyone else familiar as a user but not well-versed in the internals of the software, to create useful extensions.

3. Architecture of the New Version of the CRRES Viewing program

3.1 Constraints of the New Version:

The following are the new design constraints:

- Only UNIX will be supported. However, the software must be portable to a large class of UNIX platforms (not just SUN!).
- The software will be optimized for large physical memory machines (at least 64 MBytes).
- All of the flexibility of the older version will be maintained. In addition, time spans will no longer to be constrained to be within a single orbit. Time spans crossing orbit boundaries will be supported.
- The design will support easy "connection" to data sources other than the CRRES AFGL agency files. This includes other CRRES data file formats (exported files) or data from other instruments like POLAR, CLUSTER, or FAST.
- Scientific extensions to the software or entire new applications must be easy to implement for someone not versed in the program internals. In fact, the extension capability must support both "C" and "Fortran".
- The software must also be designed so that realtime data applications (e.g. GSE, quick-look software) can be easily implemented. In particular, the FAST project will need this capability.

3.2 System Design Considerations

This section introduces the concepts that are of importance in the design of the new software at a high system level.

3.2.1 Single-Program vrs. Object Oriented:

In order to confront the deficiencies of the original version, and consider the new design constraints above, it is useful to look at the general programming paradigms that are available.

Single-Program:

The original version, as stated, was written as single, procedural program. It is well known in the software world that large, single- architecture programs invariably become nightmarish in terms of maintainability and extensibility. It is inevitable that as a program grows in size, the number of internal, module-by-module dependencies increase. At the same time program documentation falls behind. New programmers on an application find that they must spend very large amounts of time just learning where various sub-tasks within the program are actually performed. Adding extensions require very delicate and dangerous alterations of the inter-module interfaces. It is rare, when such changes are made, that there is enough time for complete re-testing.

Object-Oriented:

The new paradigm (and it has become something of an industry buzzword) is "object-oriented" programming. There are many descriptions of this term in the literature but the important concept with regard to this software is to create separate programs which perform relatively small and well-defined tasks. Of course, it is imperative that these individual programs communicate with each other using well-specified protocols on a common inter-program communication system in order to cooperate in performing useful, large-scale applications. This requires a robust and sophisticated operating system.

There are two immediate benefits to using the "object-oriented" paradigm. Since each program has a very specialized responsibility, it is much easier to design, code, and test than a large single-architecture program. Secondly, since the programs are designed to perform specialized tasks independent of a specific application, they are reusable (i.e. they can be used in more than a single application).

A major caveat when designing an object-oriented system is that decisions must be made as to the level of decomposition of tasks into separate programs. Too many very small and simple programs would use an inordinate amount of inter-process communication, reducing program responsiveness.

The typical object-oriented model is called "client/server". A "server" is a program which performs a task requested by another program (called the "client" program). Note that a program can be a client to one program and a server to another.

In order to meet the design constraints in section 3.1, it is clear that the refurbished CRRES program should be designed in the object-oriented way. Since only UNIX now need be supported by the software, this is can be done. UNIX supports all of the inter-process communications (IPC) requirements for the object-oriented paradigm.

3.2.2 Use of Large Data Buffers:

As stated in the description of the original version of the program, the "on-the-fly" techniques used in the scientific quantity computations were very troublesome. The more traditional approach is to delay computation of derived quantities until the entire data set of the "raw" quantities are easily accessible as arrays in memory. Availability of these raw data arrays greatly simplify the design, coding, and testing of the scientific algorithms.

The new version of the CRRES software will be designed to make use of large data arrays. That is why large memory machines are considered as an essential requirement in section 3.1.

Data buffers are to be used in the following way: The application will request the data for a given time-span from the decommutator. The decommutator will then buffer all of the data for the time-span in the appropriate arrays (one array for each data quantity type, with individual points stored in time-increasing sequence). When the decommutator finishes the time span, the requested science algorithms will be invoked, using the raw data buffers as input. They will produce data array buffers of their own, which will then be plotted.

3.2.3 Use of Shared-Memory and Memory-Mapped Files:

The new design envisions use of specialized and separate programs, as well as large data arrays in memory. This requires a method of sharing data buffers between different concurrent programs.

The UNIX operating system on the Sun supports two methods of sharing data arrays between programs: "shared-memory" and "memory-mapped files".

Shared-Memory:

Shared-memory is a standard UNIX System V mechanism which allows one program to grab control of a segment of UNIX virtual memory. Other programs may then request access to this memory segment. UNIX "semaphores" (another System V feature) can be used to synchronize requests for reads and writes to this memory between programs.

One disadvantage in the use of shared-memory is that standard UNIX configurations only support shared-memory "segments" of 1 Mbyte or less (although 100 or more "segments" are allowed at one time). This is an important limitation because CRRES V12, V34, and raw mag-field data, for one orbit, will each require 5 to 10 Mbyte segments. It is not acceptable to split a data quantity buffer among several segments - this will not allow simple, array-indexing into the buffer by an application. This limitation is easy to fix (we have done so on our own machines) but it requires re-configuring the UNIX kernel. This will slightly increase the difficulty of support to non-SSL users of the CRRES software.

Memory-Mapped Files:

This option is available on Sun and many other UNIX platforms, but is not a standard System V mechanism. The main advantage of this mechanism over shared-memory is that it does not require a UNIX kernel re-configuration. It does, however, require backing-store disk space (since mapped-memory is considered an extension of the basic system virtual memory). For an entire CRRES orbit, this would require 40 Mbytes of free disk space on the local machine for a margin of safety.

Provision for support of both methods will be implemented into the new version of the application. This will allow users to run the software as best suits their working environment.

3.2.4 Use of InterProcess Communications:

Since the CRRES software will be split into a number of different programs, it will be necessary to provide a message/request transport mechanism for sending information and commands between the programs. Note that this will not be used for actual science data transmission (which is handled by use of shared-memory or memory-mapped files) except in the case where the CRRES decommutator is explicitly requested to use "streaming"-mode (described in section 3.3 of this document). The standard CRRES software will not use streaming mode.

Since UNIX provides many forms of InterProcess Communications (hereby referred to as "IPC"), a decision had to be made as to what form to use. It is very tempting to use SUN's high-level RPC (Remote Procedure Call) mechanism because of ease of use. However, this is not a standard UNIX mechanism, although it is supported on a number of other UNIX platforms. The highest level of IPC protocol supported on almost all UNIX platforms is TCP/IP. It is this protocol that will be used in the CRRES software. This provides adequate support for communications between programs running on the same machine as well as between programs running on separate machines on a network. Use of TCP/IP in the CRRES program will be through a UCB-designed library specializing in the IPC needs of the software.

TCP/IP is a protocol, not a transmission mechanism. There are two possible transport mechanisms on UNIX: Berkeley (BSD) sockets and SystemV's Transport Layer Interface (TLI). TLI is supposed to become the UNIX standard IPC transport mechanism. However, almost all network programming is currently done using BSD sockets. We will use BSD sockets with the intention of writing a TLI version of the IPC library later.

3.3 Overview of the Software Design:

This section will describe the main components of the CRRES program.

The new CRRES software will take advantage of the UNIX InterProcess Communication (IPC) capability and split the software into distinct programs, with well-defined interfaces. The main programs in the CRRES software will be:

- User Interface Module (UI)
- CRRES Decommutation Module
- Science Control Module and separate science programs (SCM)
- File Export and Import Facilities
- Data Quantity Handler Module (DQH)

There will be several other subsidiary programs which need not be discussed at this conceptual level. They will be described in the Implementation Document

Here is a brief description of the main programs:

3.3.1 User Interface Module.

This program will handle interactive user-control of the software. It is from this program that time-spans for viewing are set, plots are added/deleted from the current viewing configuration, hard copy is output, etc. It will be mouse and menu-driven. It is also responsible for producing plot output to a display or hardcopy device.

3.3.2 CRRES Decommutation Module:

This program decodes CRRES data files (telemetry, attitude, and ephemeris) and provides the time-tagged, decoded data to other programs. It also has the responsibility of informing other programs in the overall software as to the names and attributes of the data produced by the CRRES instrument.

3.3.3 Science Control Module and customized science programs:

The science-algorithm portion of the software is constructed to allow science application programmers easy access into the application. It is envisioned that separate, customized programs will be written, following the standard conventions (from the "Implementation Document"), and connected to a single, controlling program called the "Science Control Module". This program has the responsibility of ensuring access by the science sub-programs to the appropriate data buffers controlled by the Data Quantity Handler, and communicating the data produced by the sub-programs to the User Interface for plotting. There will be a standard science package which computes all of the derived quantities of the original version of the program, as well as the "operator" commands.

3.3.4 File Export and Import Facilities:

The ability to input and output a variety of data files is crucial to the ongoing utility of this software. Hence, the new version of the CRRES "Fiche" software will provide facilities for file export and file import of ASCII files, flat files, the CDF format files, and CRRES data format files which will enable users to generate plottable data to a file and recall it quickly.

The export and import facilities will be provided in two modes. The user interface mode will allow the user to export and import files while he/she waits. The batch mode will allow the user to run the software over-night or at a convenient down time. This will be useful for longer executions and computations.

Some of the file formats listed above are not intended to be supported in the primary development schedule. These will however be included in the secondary development effort.

3.3.5 Data Quantity Handler Module.

This program maintains the list of all current quantity buffers for access by the other modules of the CRRES software system. Note that these buffers will be generated by this program as either shared-memory or mapped-memory files. Note also that the module is capable of handling input sources other than just the CRRES decommutator and Science module. It is conceivable that, at a given instant, it may be handling data from two separate spacecraft which would allow an application communicating with it to compare the data from the two instruments in some way.

3.4 Data Flow Description:

This section provides a summary of the data flow between the programs described in the previous section.

There are four main categories of "data" which need to be considered conceptually, where "data" means any information flowing between programs. They are:

- Commands and requests for service from one program to another
- Quantity Description information
- Memory buffer pointers
- Data in the usual sense (telemetry, attitude, ephemeris)

The description of commands and requests is very detailed and best left for the "Implementation Document". It suffices to mention at this level that these commands are used mostly to trigger the flow of the other three types of information between the appropriate programs.

Quantity descriptors exist as a mechanism to break any dependence that the UI, DQH, and SCM would normally have on the specifics of CRRES data. By their use, the UI can be written with no pre-knowledge of the CRRES data set, the DQH will work with the data for any instrument (at least in the POLAR, CLUSTER, FAST series), and the science programs can be written with no need to know the internals of the CRRES decommutator and its associated data types. They are defined in detail in the "Implementation Document". These descriptors are passed between the following pairs:

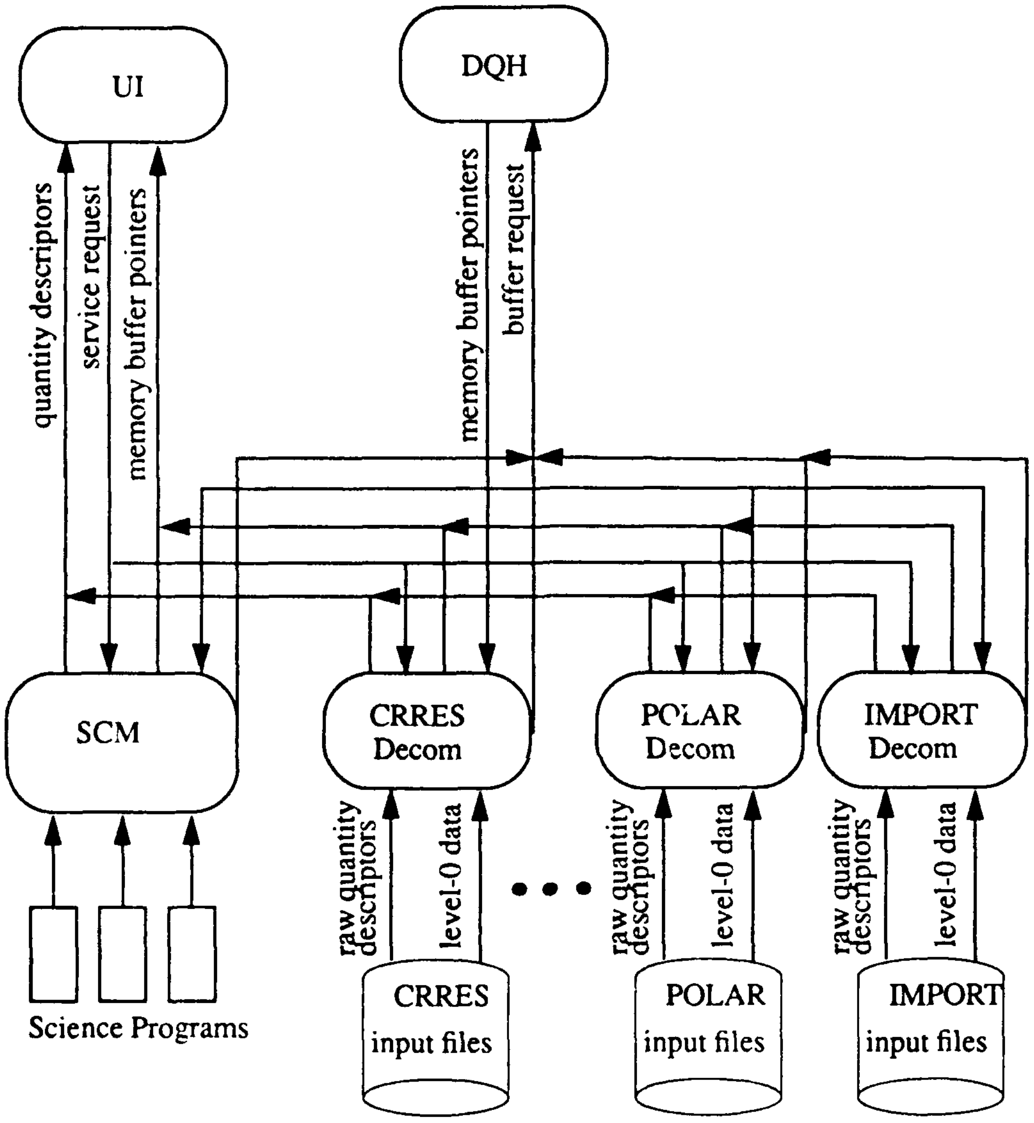
CRRES Decom to DQH (raw data quantity descriptors - piece-meal as buffers are requested)
SCM to DQH (quantity descriptors defined by any of the science plug-in modules - piece-meal as buffers are requested)
CRRES Decom to UI (raw data quantity descriptors)
SCM to UI (quantity descriptors defined by any of the science plug-in modules)

Memory buffer pointers are maintained by the DQH and are used by the UI to set plot data arrays and by the SCM and science programs for raw data array access. Here are the interprogram paths for transmission of memory buffer locations:

SCM to UI
Decommutator to UI

Note that the raw data is never sent directly out of the decommutator. The DQH informs it as to the location of the appropriate raw data buffers, which Decom fills as it decodes. These buffers are then made available to the Science modules and User Interface by the Decommutator.

The following diagram summarizes the major data flow of the software:



The following section describes the main programs in more detail.

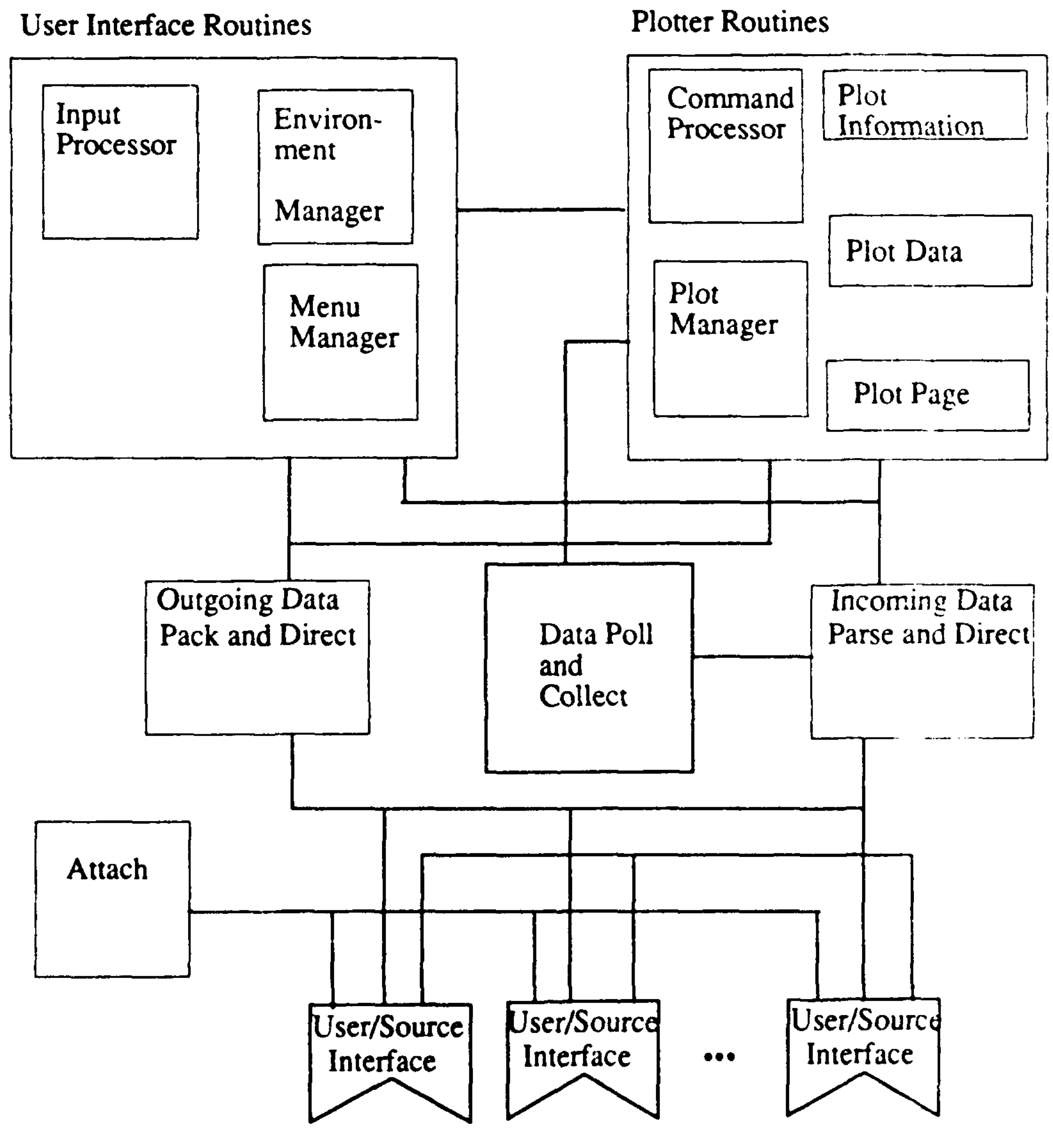
4. Internal Structure of the Major Software Modules.

This section describes the internal structure of each of the major programs in the software system. Specific subroutine descriptions and interface protocols are left to the "Implementation Document".

4.1 User Interface Module:

The User Interface Module's purpose will be simply to provide the user with facilities to input a series of requests to external data sources for data and to provide a convenient interface for manipulating that data for display. This interface will be completely generic in that none of the internal menus will be instrument dependent. The User Interface Module will contain neither instrument (satellite, rocket, file, etc.) specific operations nor any data computational functions.

In the "Fiche" application the "Fiche" User Interface Module will be provided with data from both the CRRES Science Algorithm Module and the CRRES Raw Data Source. These three programs will "attach" together and act in unison to provide the full, extant facilities of the "Fiche Program".



The process of linking up will occur something like this. The CRRES Science Algorithm

Module will attach to the "Fiche" User Interface, identify itself and pass to the User Interface a list of data which the CRRES Science Algorithm Module can make available upon request. The CRRES Raw Data (Decommutator) Module will do the same. These two programs will be referred to as "Sources".

When the User Interface Module passes a request for a certain data quantity (using the list of available plots for that Source which was passed to it on attachment) to the appropriate Source, the Source will immediately send back all information the plotting routine will need to set up the plot frame. The Source will then obtain or create the data buffers required for the requested quantity and will return the data to the User Interface Module via the DQH data buffers asynchronously as it becomes available. The User Interface Module will plot this data as it is received in a real-time fashion.

At this time we should mention that the actual redesign of the User Interface Module is somewhat in question due to possible time constraints. The User Interface Module, as it stands now, is not well suited to fit in with the concept of breaking the "Fiche" software apart into separate executable portions. The redesign to allow the User Interface Module to take full advantage of the new software structure could be quite time consuming depending on the level of reconstruction which is undertaken. Several possibilities exist. The quickest solution would be to simply split the existing User Interface off into its own executable package and stick the interprocess communications on to it as an appendage. A very basic, but functional, User Interface could be built in a relatively short period of time, but this would leave the User Interface disorganized and unmanageable for future modifications. This is not the optimal solution. The optimal solution would be to completely redesign the User Interface to use the advantages gained by splitting the software and to build it either on the existing libraries which are semi-portable or on a standard library (MOTIF or OpenLook). This option might take only marginally longer.

Again, this is a consideration based on available time. We will, however, proceed with this section as if this were the path we would pursue. The goals and functionality of the User Interface Module discussed in the above paragraphs will be met in the redesigned version by breaking the User Interface Module down into several functional components which will work together to accomplish the set goals. These functional components are listed below and will be discussed in detail in the appropriate section.

4.1.1. User/Source Interface
4.1.2. Incoming Data Parse and Direct
4.1.3. Outgoing Data Pack and Direct
4.1.4. Data Poll and Collect
4.1.5. User Interface
4.1.6. Plotter

4.1.1. User/Source Interface

The User/Source Interface will be a connection which will be used to send small amounts of data between what is known as a "source", a provider of plottable information, and what is known as a "user", a process which utilizes the data for some sort of user output. Basic sorts of messages which will be sent from the User to the Source will include information the Source requires from the user to operate correctly and requests for plot information/data. The sorts of messages which will be sent from the Source back to the User will include an initiation list of quantities available from the Source and a Source name, Specific Plot information, and the asynchronous plottable data itself. The latter will comprise the majority of information transferred via this interface.

4.1.2. Incoming Data Parse and Direct

The Incoming Data Parse and Direct Module will be responsible for receiving all input from all of the attached sources. This routine will then decode the input and pass this off to the appropriate portion of the User Interface, Data Poll and Collect, and Plotter. The types of incoming data will include the Source name, list of available plots, and the Source's needs, plot information, and plot data.

4.1.3. Outgoing Data Pack and Direct

The Outgoing Data Pack and Direct Module will be responsible for packing up all of the outgoing messages and data into the appropriate format and sending them to the correct Source. The types of outgoing elements this routine will handle will include needed data from the User Interface for the Sources and plot requests.

4.1.4. Data Poll and Collect

The Data Poll and Collect Routine receives the buffer information originating from a data Source from the Incoming Data Parse and Direct Routine. It then uses this buffer information to poll the data acquisition of the source and, at defined intervals, update the plots in real-time. This routine may also be responsible for building the actual plottable buffers from the DQH data buffers for use by the plot routines.

4.1.5. User Interface

The User Interface Routines along with the Plotter Module are the most important (and largest) portions of the User Interface Module. The User Interface Routine will provide the user an environment for inputting different types of data and data requests. An Environment Manager Routine will control what the user sees and what data being input does to the displays. The Menu Manager will provide the Environment Manager with its needed menus to prompt the user for input. Finally the Input Processor will receive input and take the appropriate actions. Each of these separate functions is discussed below.

4.1.5.1. Environment Manager

The Environment Manager is responsible for what the user sees in the menu area, what menus are displayed, moving up and down the menu structure and keeping track of what inputs correspond to what input areas on the menu list. When the program is started the Environment Manager will display a start-up screen (logo type thing), a list of available Sources and a list of options such as select a configuration, enter the needs of the Sources, start building a new configuration, etc. As the user makes selections the Environment Manager will adjust the display by requesting a menu from the Menu Manager and using the returned menu to prompt further user input. As any input is necessary the Environment Manager will prompt the user, get this input and pass this on to the Input Processor. When a string of menu selections leads to a specific command, the Environment Manager will pass this on to the Input Processor.

4.1.5.2. Menu Manager

The Menu Manager will read an ASCII file which contains a predetermined menu structure. It will maintain this by adding and deleting from it modified values that are received from the attached Sources. For example, when the program begins no Sources or associated available plots will be listed. When the Sources attach, they will pass the Menu Manager through the User/ Source Interface and Incoming Data Parse and Direct Module their name and list of available plots (e.g. CRRES Science Algorithms, Bmgse/Bgse/and so on). The Menu Manager will add the name (CRRES Science Algorithms) to the available source menu selection and the list of plots (Bmgse, Bgse, and so on) to the available plot list. Thus, no instrument dependent information will need to reside in the actual User Interface Module. When a request for a menu is received from the Environment Manager, the Menu Manager will dynamically create a menu structure and return it to the Environment Manager. If a need request is received from a Source the Menu Manager will queue this up and at the next request for a menu will provide a menu for input of the need first then the requested menu.

4.1.5.3. Input Processor

The Input Processor will be responsible for receiving completed input or commands from the Environment Manager and directing them to the correct place. The received inputs and commands will include Plot additions and deletions, display commands such as panning and zooming and Need data for Sources. When the Input Processor receives any plot additions, deletions, or Need data for Sources it will pass these out to the Outgoing Data Pack and Direct Module to send out to the appropriate Source. In addition it will send any display commands or deletion requests to the Plotter Module.

4.1.6. Plotter

The Plotter is responsible for all plot panel display of data. It is only a receiver of data programmatically speaking; it does not send any data to any external process. It receives incoming plot information and data from Sources and delete and display commands from the User Interface routines. This data is processed by six routines which comprise the Plotter Module. A discussion of these follows.

4.1.6.1. Plot Manager

The Plot Manager simply maintains a list of the plots which are being displayed and the parameters which determine the status of the display (plots per page, displayed timespan, data timespan, and so on). This information corresponds identically with what is called the Configuration. When a set of Plot Information is received, the Plot Manager adds this plot to its plot list and calls the Plot Information Routine to put up this new plot. When a delete command is received from the User Interface, the plot is deleted from its list. The maintained list is used by the Plot Data Routine to determine which plot panel incoming data belongs to and by the Plot Page Routine to determine which panels to create plots for.

4.1.6.2. Plot Information

The Plot Information Routine is responsible for establishing a plot panels existence. It plots no data. The Plot Information Routine goes to the plot list maintained by the Plot Manager and gets the appropriate information (title, scales, labels, limits, messages, etc.). It then uses a set of library routines to create the actual panel frame, put the title up, label the plot, and establish values for the scale.

4.1.6.3. Plot Data

The Plot Data Routine is responsible for collecting incoming data and placing this data in two places, the panel of the appropriate plot and a contiguous data buffer. The Plot Data Routine will receive an amount of data and will plot this data asynchronously and immediately on the panel. This will allow the user to view real-time the data as it is created. The Plot Data Routine will create and maintain contiguous buffers for this plottable data. These buffers will be used for redrawing the plots when zooming out or paging up and down.

4.1.6.4. Plot Page

The Plot Page Routine will be responsible for creating a fresh page of plot panels. It will do this by First clearing the plot area and calling a set of library routines to place the plot title at the top of the page and the legend on the bottom and any other pertinent information. It will then determine which plots will be on the given page as indicated by the number of plots per page and number of plots requested. It will loop through these and call Plot Information and Plot Data for each one to create that plot panel.

4.1.6.5. Command Processor

The Command Processor will be responsible for receiving, parsing, and executing of commands from the User Interface Routine. The types of commands this will include are plot delete requests, zoom and pan requests, changing the number of plots per page, and so on. When the Command Processor receives one of these commands it will call the necessary above routines to appropriately execute the given command (e.g. Delete Plot - call Plot Manager to delete from list and call Plot Page Routine to refresh without deleted plot).

This is the breakdown of the design for the User Interface Module. It is believed that this design will not only provide all the desired performance and functions required of the User Interface, but will provide flexibility and organization for expansion and refinement in the future.

4.2 CRRES Decommutation Module:

The design of the CRRES decom program will establish a standard data "connection" mechanism which the later projects (POLAR, CLUSTER, and FAST) can utilize to directly hook into the new version of the CRRES program. This standard will be described in technical detail in the "Implementation Document". What follows here is a more general description of the design features of the CRRES decommutator.

4.2.1 *The Decommutator in the Original CRRES Program:*

The original program included the decommutator as a "C"-callable library, an architecture which precluded use of other data streams (from other instruments or other CRRES data files). This created an undesirable dependency between the higher levels of the program and the CRRES decommutator output itself, which must be avoided in the new version. However, the basic telemetry decoding algorithm will remain intact in the new version.

4.2.2 Input Files:

The basic input files to the new CRRES decommutator are the following:

Telemetry file
Attitude file
Ephemeris file
Quantity Descriptor file

The first three are the data inputs into the original version of the program and are available on an orbit-by-orbit basis from AFGL. The attitude and ephemeris files are exactly as produced by AFLG whereas the telemetry file has an appended index to aid in random accessing. The attitude file is ASCII and the telemetry and ephemeris files are binary.

The Quantity Descriptor file is a new ASCII file which will describe the types of instrument data which this decommutator provides, as well as the scaling and offset coefficients for those quantities.

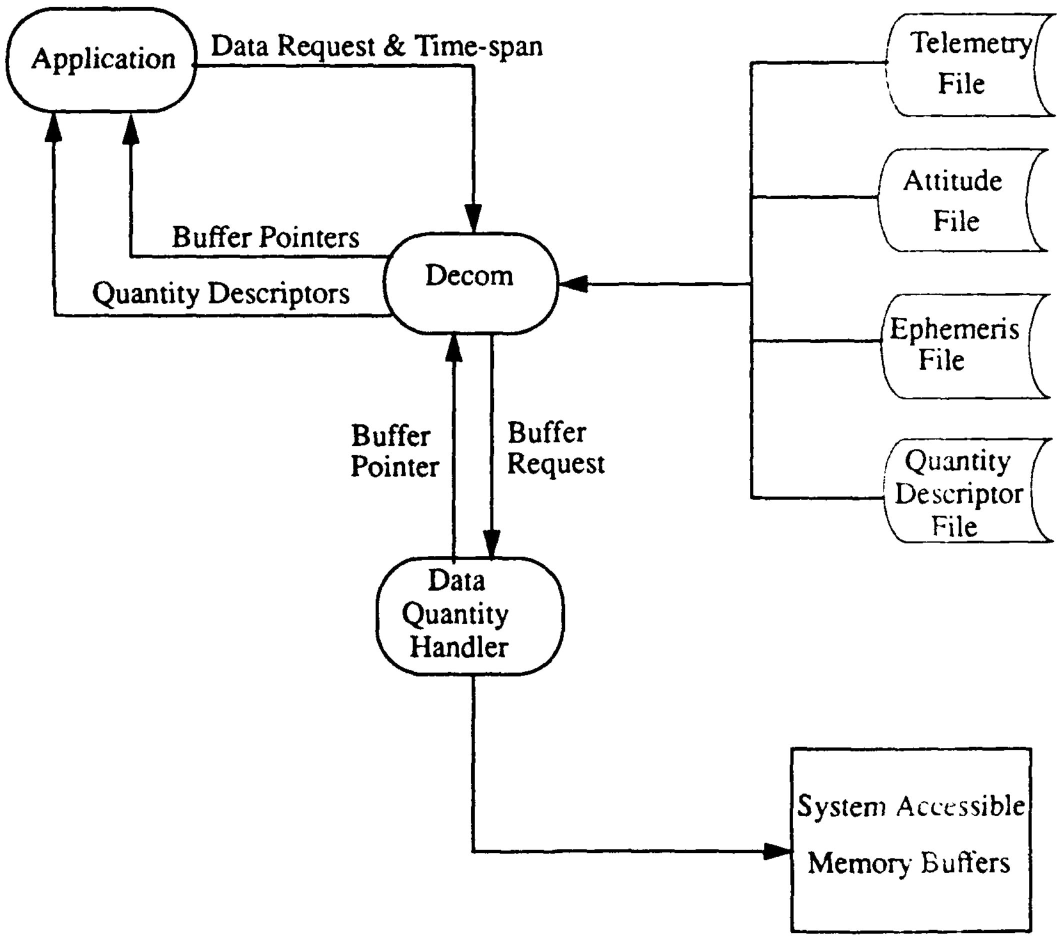
4.2.3 Modes of Operation:

This decommutator is called as a service by anyone requesting CRRES data from the data files used by the original version. It is in charge of decoding CRRES telemetry for given date/time intervals as well as providing spacecraft attitude and ephemeris information for that same time span. Since the decommutator must be able to service many applications, it must support two different modes of operation: "buffered mode" and "streaming mode".

Buffered Mode:

Buffered mode will be the operational mode of the new CRRES software. In this mode, the decommutator will not send data back to the client. Instead, the DQH will transmit the locations of the memory buffers for each raw data quantity to the decommutator which it then fills as it decodes the data for a given time span. The client will query the Decommutator for data and the Decommutator will ask the DQH for pointers to these arrays. These pointers will be returned to the client and the client can then access the buffers directly. Note that the DQH will define the memory arrays as either shared-memory or memory-mapped files, depending on the needs of the application.

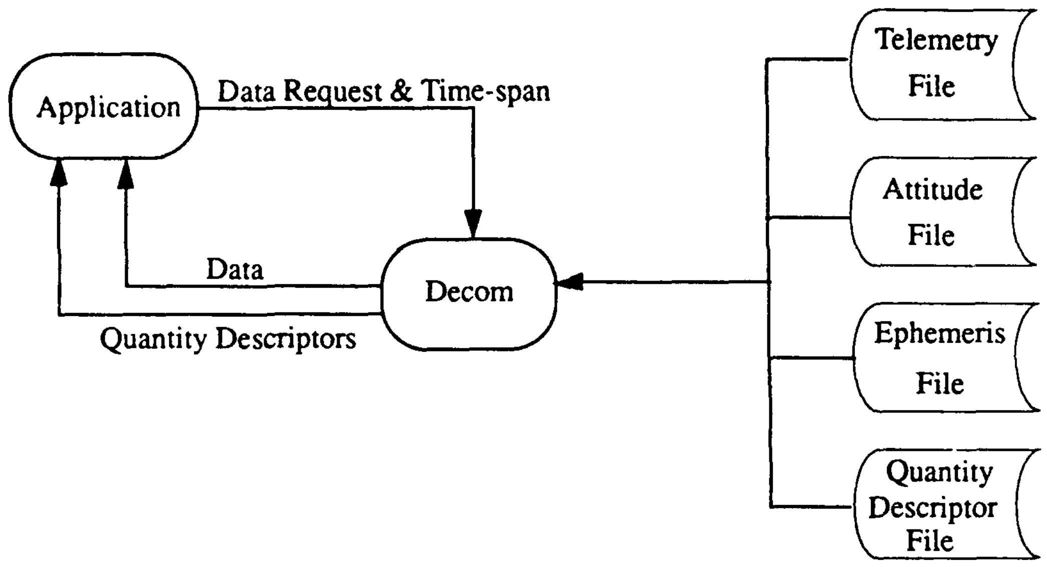
The following diagram indicates the basic data flow in buffered mode:



Note that the Application uses the Quantity Descriptors to set up its list of available plots.

Streaming Mode:

Streaming mode will be most useful for quick-look, realtime GSE software. In this mode, the decommutator does send data directly to the client application which would probably plot the data points as soon as they are received. Note that attempts to write science analysis software in this mode would be open to the difficulties which burdened the original program (whose decommutator did use the streaming mode). The following diagram indicates the data flow:



4.2.4 Sub-modules of the CRRES Decommutator:

The following are the sub-modules of the Decommutator:

- InterProcess Communications Handler
- Command/Request Interpreter
- Data Output Handler
- Quantity Descriptor Handler
- Telemetry Decoder
- Attitude Decoder
- Ephemeris Decoder

The InterProcess Communications Handler (IPC Handler) will be provided as a standard library of 'C' and Fortran callable subroutines which read and output to the UNIX IPC mechanism used by the CRRES software (TCP/IP transported by BSD sockets). This library will be used by all programs and sub-modules in the CRRES application. It does nothing more than read and output incoming and outgoing IPC transmissions.

The Command/Request Interpreter unpacks and parses incoming IPC messages for command/request input. Once a command has been parsed, the submodule passes control to the appropriate subroutine within the decommutator.

The Data Output Handler packages outgoing data points and sends them to the IPC handler for transmission to another program. Note that the decommutator sends data to other processes using IPC only in streaming mode.

The Quantity Descriptor Handler reads the Quantity Descriptor file and, if so requested, packages the information and sends it to the IPC Handler for transmission.

The above submodules are really quite independent of the CRRES data stream itself and thereby form a reusable subset of this particular decommutator for application in POLAR, CLUSTER, and FAST. The following submodules are CRRES dependent and will be composed for the most part by the current decommutator.

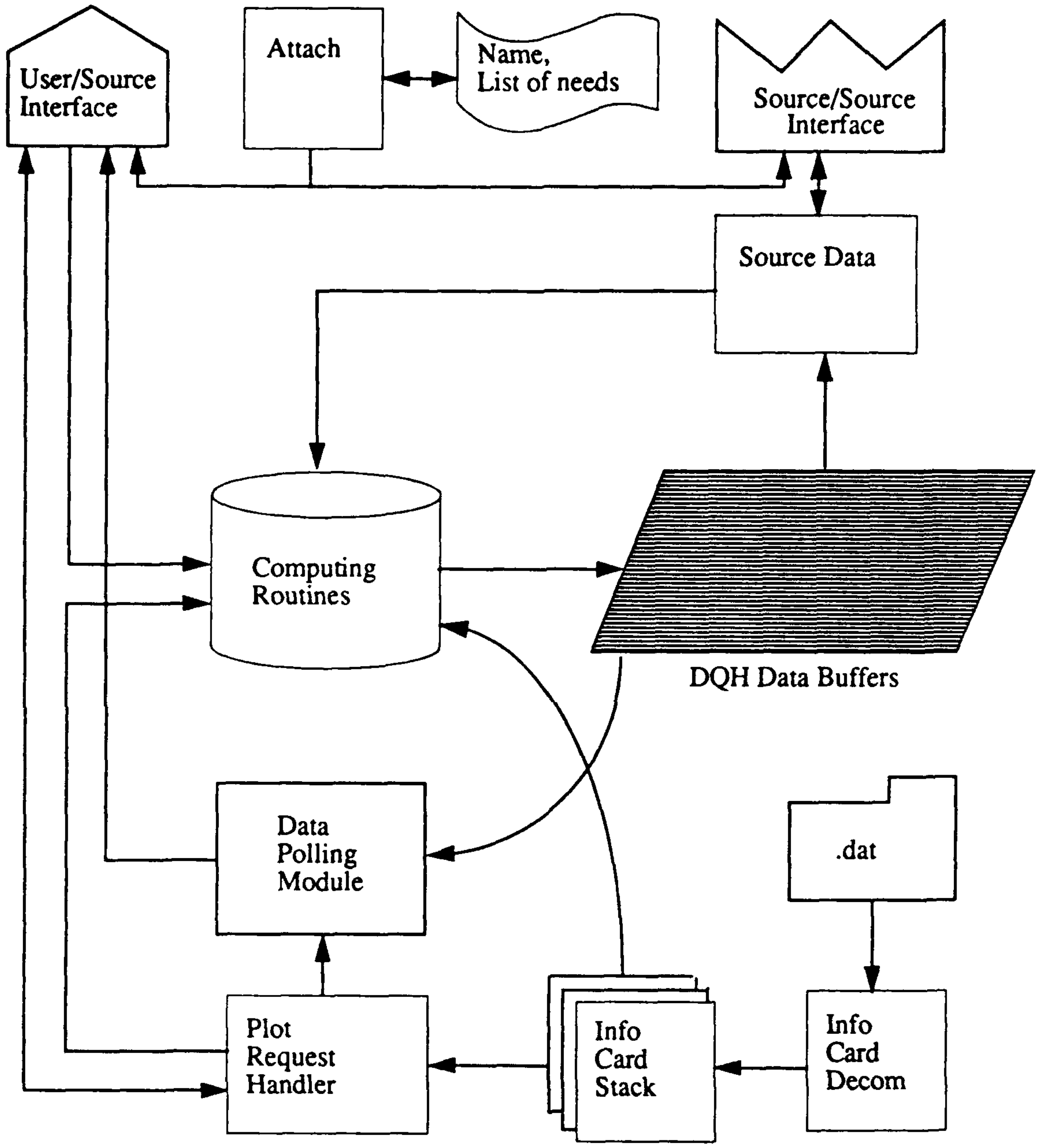
The Telemetry Decoder unpacks and interprets the Telemetry files, sending data out on a point-by-point basis, for a pre-specified time span. In buffered mode, the data points are inserted *directly into the appropriate shared-memory or mapped-memory data arrays by this sub-module.* In streaming mode, the data points are sent to the Data Output Handler for IPC transmission to another program.

The Attitude and Ephemeris Decoders read the Attitude and Ephemeris files and make this information accessible to other programs. They work a bit differently from the Telemetry Decoder in that they read and decode the entire attitude or ephemeris file when an orbit is initialized. The attitude and ephemeris data can then be accessed at will by the Telemetry Decoder or others without resort to disk I/O.

The Telemetry, Attitude, and Ephemeris Decoding sub-modules are also responsible for handling orbit overlap when time spans crossing orbit boundaries are specified. They must insure that redundant data is not made available to other programs.

4.3 Science Control Module:

(See figure for context)



The purpose of the CRRES Science Algorithm Source is to provide computed science quantities to user programs which request such information. In the "Fiche" application the "Fiche" User Interface will attach to the CRRES Science Algorithm Source and the CRRES Science Algorithm

Source will in turn attach to the CRRES Raw Data Source. These three programs will act in unison to provide the full, extant facilities of the "Fiche Program".

The process of linking up will occur something like this. The CRRES Science Algorithm Source will attach to the "Fiche" User Interface, identify itself and pass to the User Interface a list of data which the CRRES Science Algorithm Source can make available upon request. When the User Interface passes a request for a certain science quantity, the CRRES Science Algorithm Source will immediately send back all information the plotting routine will need to set up the plot frame.

The CRRES Science Algorithm Source will in turn attach to the CRRES Raw Data Source and obtain the raw data buffers required for the requested quantity. Then it will perform the necessary calculations on that data and create a buffer of the requested quantity. This quantity data will be returned to the User Interface asynchronously as it becomes available so that it can be plotted immediately.

The proposed organization of the CRRES Science Algorithm Source is shown in figure 2. The major components of the CRRES Science Algorithm Source number 10 and are listed below. Each is explained briefly in the sections which follow.

4.3.1. User/Source Interface
4.3.2. Source/Source Interface Plug
4.3.3. Source/Source Interface Socket
4.3.4. Source Data Module
4.3.5. Attach Module
4.3.6. Info Card Decommutator
4.3.7. Plot Request Handler
4.3.8. Computing Module
4.3.9. Data Buffers
4.3.10. Data Polling Module

4.3.1. User/Source Interface

The User/Source Interface will be a connection which will be used to send small amounts of data between what is known as a "source", a provider of plottable information, and what is known as a "user", a process which utilizes the data for some sort of user output. Basic sorts of messages which will be sent from the User to the Source will include information the Source requires from the user to operate correctly and requests for plot information/data. The sorts of messages which will be sent from the Source back to the User will include an initiation list of quantities available from the Source and a Source name, Specific Plot information, and the asynchronous plottable data itself. The latter will comprise the majority of information transferred via this interface.

4.3.2. Source/Source Interface Plug

The Source/Source Interface Plug is a receiving "port" for data from another Source (i.e. the CRRES Raw Data Source). This port will attach to available Source(s) and make available to the

CRRES Science Algorithm Source data from the attached source.

A key element of this Plug is that it will provide for three "modes" of operation; Shared Memory, which is the primary option, Mapped Memory, and Streamed Data. The difference between the three options is but a configuration decision. The use of Shared Memory would require system administrators to recompile their UNIX kernel (operating system) to allow shared memory segments of ten megabytes. Mapped Memory does not require this, but requires instead that a 35 megabyte file be created on one's harddisk. Streamed Data would circumvent both of these downfalls, but would also cut performance considerably.

4.3.3. Source/Source Interface Socket

The Source/Source Interface Socket is quite similar to the Source/Source Interface Plug except that it provides access for other Source processes to the CRRES Science Algorithm Source data. Another Process would attach to this, say, if it wished to take CRRES Science data and perform further calculations before passing the data on to a User.

4.3.4. Source Data Module

The Source Data Module's sole purpose is to provide a common interface for the CRRES Science Algorithms to other Source data independent of the Source/Source Interface mode of operation.

It is desirable to always be able to access data buffers by getting some limit information about the amount of data available and a pointer to an established data buffer. In this manner the CRRES Science Algorithms are freed from having to perform unnecessary and messy data collection (event driven data collection) which caused so many problems in the previous version of the "Fiche" software.

In particular, this is not a problem in the cases of using Shared or Mapped Memory. A pointer to those common memory locations is passed and the data is accessed trivially. In the case of Streamed Data, however, data will be coming in small portions. The Source Data Module will assemble these pieces into a contiguous data buffer and header information and return the header and pointer information which will identically match the Shared and Mapped Memory case.

Note: there will be NO event driven data computations!

4.3.5. Attach Module

The Attach Module has a relatively small, but important role in the outlined scheme. It's sole purpose is to attach to any other matching process. The term attach implies that it will first of all establish a firm data connection between this Source and other Sources or User programs. Once this has been established the Attach Module will exchange with that process any necessary information for the two to operate in unison. As an example, Attach routine will recognize the "Fiche" User Interface process and form a (series of) data link and handshaking lines with it. It will then pass the User Interface its name and a list of the quantities that it can make available.

Once this is done it will tell the User Interface that it must have a timespan before any data can be requested. Once this is done, the attach module goes back to waiting for other User processes to start up which it will need to attach to. Any further processed which start-up will be attached to in turn in a similar manner.

4.3.6. Info Card Decommutator

The Info Card Decommutator has but one function and that is to parse from an ASCII file the list of quantities and quantity specific plot information that will be needed to provide to any User process to which this attaches and place this information where the CRRES Science Algorithm Source can access it. This location will be known as the Info Card Stack.

4.3.7. Plot Request Handler

The Plot Request Handler will receive from the User/Source Interface Data Requests which originated from the "Fiche" User Interface. It will immediately gather the plot specific information for the requested data quantity and send this to the User/Source Interface to be forwarded to the "Fiche" User Interface. It will then relay this Data Request to two places; the Computing Module so that the desired quantity(ies) can begin to be calculated and to the Data Polling Module so that as the data is calculated it can be sent off to the "Fiche" User Interface for 'real-time' plotting. It then waits for another Data Request to come in.

4.3.8. Computing Module

The Compute Module is the main stay of the CRRES Science Algorithm Source. It provides the computational engine for deriving the desired quantities. The Compute Module will receive timespan information from the "Fiche" User Interface and pass this on to the decommutator to make available the raw data for the Scientific Calculations. It will receive data requests from the User Interface and maintain a list of all Science Quantities to be calculated, and it will maintain the buffers for all final Science Quantities and Intermediate steps so that redundant computations will not occur. All of these tasks can be broken down into several functional parts (see figure 3). These are

4.3.8.1. Data Manager
4.3.8.2. Data List
4.3.8.3. Prepare Data
4.3.8.4. Build All Buffers
4.3.8.5. Data Quantity Handler

Each part and its function will be discussed in the following sections.

4.3.8.1. Data Manager

The Data Manager Routine will be responsible for receiving data requests (and the terminations thereof) from the User/Source Interface which originated in the "Fiche" User Interface and will use these to maintain a list of all scientific quantities which must be computed. In addition to this task, when a new quantity (a previously unrequested quantity which is neither the result nor an intermediate step to another previously requested science quantity) is requested, the Data Manager will signal the Prepare Data Routine to compute this quantity.

4.3.8.2. Data List

The Data List will be maintained by the Data Manager. This list will include all information which will be required to compute the given science quantity. This list will be referenced by the Build All Buffers routine which will scroll through all the quantities to be computed and do so.

4.3.8.3. Prepare Data

By far the most important routine of all, The Prepare Data routine is responsible for creating a requested science quantity either from scratch or from an intermediate step which has been previously computed for another science quantity for this timespan. The aim of this routine is to eliminate the possibility of redundant science computations.

The first step (figure 4) is that the Prepare Data routine will receive the requested quantity from the calling routine. It will then check with the Data Quantity Handler to see if it exists. If it does, then the action is trivial; simply return the pointer to the existing buffer. If it does not exist then it will get the list of building blocks (intermediate steps which are computed on the way to building this requested quantity). It will then go through this list and call Prepare Data itself for each Building Block Quantity. Prepare Data will return the pointers to each of the building block quantity buffers as it is called in the manner that is being described here. Once all the building block quantities for this requested quantity are gotten (and the respective data buffers built) Prepare Data will call the "Compute Function" for this particular quantity.

The "Compute Function" corresponds to the actual science algorithm calculation (e.g. Rotate Electric Field, Rotate B into MGSE, Free form Operators and so on). The environment of the Compute Function is that it receives complete buffers for the timespan for which it is concerned and can access these in a completely random fashion. This allows great flexibility and speed for any sort of science algorithm whether it simply performs a point by point computation or whether it is dependent upon past and future values for the given computational time. The Compute Function will be the focus for a person who is adding a science quantity. The only other step will be setting up the description in an ASCII file for the building block list and plotting information.

4.3.8.4. Build All Buffers

The Build All Buffers will be responsible for re-creating the current Data List of requested quantities from scratch when necessary (i.e. a new timespan is received from the User Interface). It will do this by first using the Data Quantity Handler to throw away all existing buffers. It will then go through the Data List and request Prepare Data to compute each of the requested quantities.

4.3.8.5. (DQH) Data Quantity Handler
see section 4.4 Data Quantity Handler (DQH) Buffer Module

The Data Quantity Handler may be either a separate program or simply a set of library routines which will maintain the Data Buffers. Ideally, the same code should be used to maintain the CRRES Science Algorithm Source's Data Buffers as is used to maintain the CRRES Raw Data Source's Data Buffers.

The Data Quantity Handler will provide facilities for adding, deleting, filling, emptying, reading, and indexing Data Buffers. All values (i.e. size, label) should be dynamic (passed in from the outside world).

No other routines will directly access the data buffers except for direct random access reading.

4.3.9. Data Buffers

The Data Buffers will be created and maintained by a buffer manager library. This library will be called by the Computing Module to set up and store the Data Buffer quantities for use by either other Sources through the Source Data Module or to the User Interface through the Data Polling Module.

4.3.10. Data Polling Module

The Data Polling Module receives a Data Request from the Plot Request Handler which has been forwarded down from the "Fiche" User Interface. This routine requests a buffer of the appropriate size from the DQH Module and returns this pointer to the calling routine so that the data can be accessed as it is created by the Computing Module.

All of these sections will be combined in one process which will be accessed as an entity only through the Source/Source Interface or the User/Source Interface. No other access will be allowable nor necessary to properly use and manipulate the CRRES Science Algorithm Source.

4.4 File Export and Import Facilities

The File Export and Import Facilities will be a series of separate modules which are as separate in software as they are in function.

Each of the file formats will have a module which will encode that particular file format. Each of these in turn will be called from a more general file export module. The general file export module will interface with the Decommutator, Science, and File "sources" in the same manner as the User Interface does; that is it will utilize the same Inter-Process Communication protocol and methodology as the UI. This general file export module can be called up from the User Interface or the Batch Processor and will then be spawned off as its own process.

Likewise, the Import Facilities will have a module which will decode each particular file format. These specific decoders will be called from a more general file import module. The File

Import Module will attach in the same manner and behave to the software exactly the same as any other "source" namely, the CRRES Decommutator and the Science Control Module.

4.5 Data Quantity Handler (DQH) Buffer Module:

This program has the responsibility of providing access to shared data between the various separate programs in the CRRES software package. It will also be used in the same way for POLAR, CLUSTER, and FAST data handling.

It would be possible to have each separate program (the CRRES decommutator and the science programs) be responsible for their own use of memory. However, it is architecturally more sound and simple to focus this responsibility on a single program for many of the reasons listed in section 2. It also opens the possibility of multiple data streams into a CRRES user session. One would be able to compare CRRES data with, say, POLAR data.

4.5.1 DQH Inputs:

There are no input files to this program. The only inputs are quantity descriptors, which the DQH uses to keep track of the names of each data buffer. These are transmitted as data buffers are requested to the program (via IPC) from two main sources: the raw instrument quantity descriptors from the decommutator and the derived scientific quantity descriptors from the Science Control Module (SCM). These programs must also specify how large the associated arrays must be for each quantity.

4.5.2 How the DQH Works:

This subsection describes the sequence of events which result in system-available data arrays in memory for use by other programs.

The DQH will be invoked as a service when the CRRES software is started on a workstation. One of the parameters used to invoke the DQH is an indication of which of the two storage methods (shared-memory or mapped-memory files) should be utilized. The DQH is then ready for buffer requests.

The decommutator is given a time-span to decode (from the User Interface). It computes how many data points of each of the raw instrument quantities it will produce for the time span and then sends the required commands including the particular raw data descriptor to the DQH to have it allocate the necessary buffer storage for each raw quantity. The DQH will maintain a list of internal tables which store this information and allocate a buffer for each request. Similarly, the SCM will pass the array size information and science data descriptors to the DQH for allocation. Once these arrays have been allocated (either shared-memory or memory-mapped files), any program can request the location of these buffers from the DQH. The decommutator and SCM will request this information.

When the CRRES software requests a new time span from the decommutator, it is checked to see if the new time span is a subset of the old. If not, the decommutator commands the DQH to destroy its current CRRES buffers and the above cycle is repeated. If the new time span is a subset of the old (a "zoom-in") then there is no need for a re-decommutation cycle and the old buffers are still used. Even in a zoom-in, however, the SCM may wish to have the science quantity buffers destroyed, reallocated, and refilled. This can be controlled from the UI.

When the CRRES session is ended, the DQH is commanded to destroy the buffers and go back into a "sleep" state (waiting for the next service request).

4.5.3 Sub-modules of the DQH:

The following are the sub-modules of the Data Quantity Handler:

- InterProcess Communications Handler
- Command/Request Interpreter
- Buffer Allocation and Maintenance Handler

The InterProcess Communications Handler is the same set of library routines used by the CRRES decommutator. It reads and writes IPC messages.

The Command/Request sub-module unpacks and parses incoming IPC messages. It expects to see only commands and requests known by the DQH. When it parses a legitimate command, it passes control to the appropriate subroutine within the DQH.

The Buffer Allocation and Maintenance sub-module is the major portion of the program. It deals with quantity descriptors, shared-memory vs. mapped memory, allocation of memory, etc.

Contents of the CRRES Implementation Document:

The "CRRES Implementation Document" will be a programmer's level of documentation on the new version of the CRRES software. Its purpose is to document, as fully as possible, all aspects of the source code so that responsibility of program maintenance can be painlessly shifted to new people. This document will be generated as the software is implemented in the first part of 1992.

Here is a brief description of the contents of the document:

1. An introduction which will briefly describe the major and minor component programs of the software.

2. Detailed definitions of the important data abstractions in the software (quantity descriptors, plot descriptors, etc). The level of documentation will include actual "C"-structure listings, with a description of each field within these structures.

3. Definitions of the command/request functions to each of the programs in the software.

4. Detatiled definition of the inter-program communications protocol and functions of the software. TCP/IP will be used as the transmission mechanism but our own internal protocol will encode the specific commands and data transfer structures within TCP. This must be defined in detail.

5. A section which will describe, in great detail, control and data flow during a typical user session (how and when the various sub-programs are used, where the data really is, when computations are triggered, etc).

6. A description of the mechanism for allowing user's to write and connect new science programs into the software. The "User's Guide" will show examples of how this is done.

7. A large section describing the interface to all subroutines in the programs of the software. This section will be similar to the MicroSoft "C-Compiler Run Time Library Manual". Each subroutine will have about a page describing its calling sequence and an example of use. Those subroutines which are accessible to writers of science "plug-in" modules will have "C" and "Fortran" interfaces described.

8. Instructions on building the software and source-code control.

New CRRES Implementation Document

I. Introduction

The Implementation Document describes in detail the actual composition and interactions of the New Version of the "Fiche" Software. It accomplishes this by discussing the Startup and Control process of the software modules, discussing the inner-structure of each major module of the software (e.g. Decommutator, Science, Data Quantity Handler, User Interface, IPC), discussing the interactions of these modules in a typical user session, the methodology behind building and maintaining the software, the data objects which are used throughout the software, and finally the description in detail of each function used to comprise the whole. In this document, these sections are developed and expanded. The hope is that this document will provide all the necessary information for a new programmer or scientist with software experience to maintain, modify and expand this software.

II. StartUp and Control

This section will describe in detail the method in which the major software modules are started up and how one process (The User Interface) gains "control". The methods that have been discussed about starting the processes up are

1) starting all the necessary processes through a shell script
2) having the user interface have knowledge of the other modules and allow the user to start them up.
3) having a startup which looks and acts similar to the Frame Maker startup bar.

III. Module Section

This section will describe in detail the major modules of the "Fiche" Software. Here the Software Developers will detail the specific routines which comprise each module, reference them in the Function Listing Section, and show how they interact to provide complete functionality of that module. In addition the developers will describe the data passed between the routines. Finally, the developers will describe how each entire module interfaces with the other major modules and will mention the data abstractions which are used and reference them in the Data Abstraction Section

A. User Interface
B. Decommutator
C. Science
D. Data Quantity Handler
E. InterProcess Communications
F. In-House Libraries (gob, tool1, toolm, gsh, decom, and so on).

IV. User Session Description Section

The User Session Description Section will discuss the Above described interaction for the specific instance of a typical user session. In this, the developers will specify specific Data Abstractions and their values and show how these flow through the software as a whole from module to module and routine to routine. In addition, the developers will describe the process of starting the processes up and how the interact to form a functional whole.

V. Building and Maintaining the Software Section

This section will describe how a user should safe-guard the software with software control and how the user should go about building an executable version of the software for use in a variety of software configurations. Also this section will describe how to configure one's system to support this software.

VI. Data Abstraction Section

The Data Abstraction Section will describe all of the Data Objects/Quantities which are used universally throughout the software. This description will describe in detail the memory sizes of the data types and the contents of each byte.

VII. Function Listing Section

The Function Listing Section will contain an alphabetical listing of all of the functions in the software regardless of module affiliation. Each of the listings will contain a description of each function declaration and its calling parameters and return values. In addition there will be listed a pertinent example of use and a reference as to where it may be found in the actual source code files. This will provide future software developers and scientists a useful tool for comprehending and utilizing the software. Reference to source code.

Copy template from function.template:

function name

Summary

include files

function declaration
function parameter declaration

Description

description
caveats

Return Value

return value

See Also

related functions

Example

example source code file (main)

VIII. Conclusion: Additions and Simple Changes

The Conclusion to the Implementation Document will include a sort of programmer manual will discuss "how to" make simple changes and additions to the 'Fiche' software (i.e. add algorithms and plots, etc.) in a very regimented straight forward manner.

Appendix C

Publications Resulting from Research Supported Wholly or in Part by CRRES Contract F19628-87-K-0016

1992

Singer, H., W. Sullivan, P. Anderson, F. Mozer, P. Harvey, J. Wygant and W. McNeil, Fluxgate magnetometer instrument on the Combined Release and Radiation Effects Satellite (CRRES), *J. Spacecraft and Rockets* **29**, 599, 1992.

Wygant, J. R., P. R. Harvey, F. S. Mozer, N. Maynard, H. Singer, M. Smiddy, W. Sullivan and P. Anderson, The CRRES electric field/Langmuir probe instrument, *J. Spacecraft and Rockets* **29**, 601, 1992.

Submitted or In Press

Imhof, W., R. Robinson, H. Collin, J. Wygant and R. Anderson, Simultaneous equatorial measurements of waves and precipitating electrons in the outer belt, submitted to *Geophys. Res. Lett.*, 1992.

Wygant, J., F. Mozer, J. Blake, N. Maynard and H. Singer, CRRES observations of large amplitude electric and magnetic field signatures in the inner magnetosphere during an SSC-induced rapid formation of a new radiation belt, submitted to *Geophys. Res. Lett.*, 1992.

Wygant, J., P. Harvey, F. Mozer, M. Pongratz, D. Simons and H. Singer, Observations of CRRES electric and magnetic field measurements during the Alfvén wave braking of a barium release in the ionosphere, submitted to *Geophys. Res. Lett.*, 1992.