The Structure of the WEC/ISDAT Data

WECdata Structure Working Group

Tony Allen, Chris Harvey,
Gunnar Holmgren, Anders Lundgren

Bars indicate changes since issue 1, revision 3
(except for appendices)

Contents

Document Status Sheet			
1. Document Title: **The Structure of the WEC/ISDAT Data**			
2. Document Reference Number: **CWD–OBSPM–DD–001**			
3. Issue	4. Revision	5. Date	6. Reason for Change
1	draft	Sept. 11, 1994	First issue
1	1	October 6	Ch. 1, explain relation with file `Dbdata.h` Ch. 2, introduce the "data brick' Ch. 2, introduce half-integer rank Ch. 2, clarify timing Ch. 3 completely rewritten Ch. 4, new Chapter
1	2	October 14	Ch. 2, philosophical statement Ch. 3, describe options (a), (b), (c) ... Ch. 3, introduce type (real or complex) Ch. 3, introduce tracability code Ch. 3, remove ref. to non-integral rank Sect. 4.1, new section Sect. 4.6, explain better sensor coordinates Ch. 5, Despin Operator forms a new chapter
1	3	October 28	Sect. 3.1, introduce the sensor location Sect. 3.3, introduce particle mass & charge Sect. 3.3, introduce the sampling window Sect. 3.4, introduce geographical coordinates Sect. 3.4, define default sensor orientation Sect. 4.2, expand discussion of coord. systems Sect. 4.3, discussion of sensor location Sect. 4.5, discussion of sampling window
1	4	December 29	Sect. 3.4, add mechanical build coordinates Sect. 4.2, describe mechanical build coordinates Appendix 2, corrections
1	5	March 16, 1995	Chapter 5 becomes sect. 5.2 Rest of Ch. 5 completely new Appendices 1 and 2 updated
1	6	March 24, 1995	Appendix 2 revised

1 Introduction

It is trite, but nevertheless true, to say that the ease of processing and analysing data is directly related to the suitability of the way it is stored and accessed. The processing and analysis are determined by physical considerations, and so too should be the optimisation of the storage and access architecture.

Space physics data is characterised by being organised in long strings of objects in chronological order. For each value of the time, the corresponding data object consists of one or more values which are all determined simultaneously, being either:–

- A single structure, consisting of either:
  1. a value which is "stand-alone", like the density or temperature, or
  2. values which are "naturally associated", like the three components of a "true vector" or the nine components of a "true tensor", upon which standard operations can be performed including, in particular, transformation to some other coordinate system.
- A sets of several such structures, all of which are structurally identical, have the same spatial characteristics, but which nevertheless have some other characteristic which distinguishes them from each other: examples are the different frequency channels of a power spectrum or a cross-spectral matrix, or the different energies of a particle spectrum.

All data needs its meta-data to be associated with it in a simple and logical way. The meta-data specifies units, the coordinate system, and other information which is essential to be able to process or scientifically interprete the data. It also includes attributes such as the processing software version and calibration set version used to derive the data. The meta-data changes more slowly than the data itself.

Information about the nature of the data objects must be immediately available to applications (clients, filters, etc.), and is part of the meta-data.

WEC has decided to structure the scientific data so as to separate in a uniform and well-defined way:

- the time dimension, in which averaging and more complicated processes like re-sampling or Fourier transformation can be performed;
- the space dimension, in which coordinate transformations may be applied;
- other possible (non-space, non-time) dimensionality of the data set.

The purpose of this document is to explain the WEC Data Structure or, more precisely, to describe in plain English the contents of the `Dbdata.h` header file used by ISDAT. This header file is what is actually used by the software to implement ISDAT. Therefore, in case of inconsistency with the present document, precedence is taken by the file `Dbdata.h`. Clearly, all such inconsistencies are blunders which must be corrected at the earliest opportunity. Therefore the authors kindly request that all such inconsistencies, or any other difficulties of comprehension of the present document, be reported to Gunnar Holmgren or Chris Harvey.

2 General Overview

The purpose of the WEC/ISDAT data structure is to make each data object completely self-describing; that is, the data object contains

- the data itself, in a format convenient for the widest possible range of applications, and
- all the information needed for the scientific analysis of that data.

Furthermore, this is true for any data object, whether it comes straight from the data-base handler, or is the result of preliminary scientific processing.

More explicitly, the data will be structured as follows:

1. **At the lowest level** (*i.e.*, adjacent in memory) is the "data brick". We use the term brick rather than item, unit, element or matrix to avoid confusion, as the latter terms are all used elsewhere in a different context. The data brick is a value or a set of values which are either:–

   - Related to the same physical observable. A physical observable is something which is fully defined whatever the system of spatial coordinates used: in other words, it is either a scalar (which is independent of the coordinate system), or the complete set of components of a vector or a tensor, which transform in a standard and well-defined way upon rotation or (non-relativistic) displacement of the coordinate system. The elements of vectors and tensors are stored in memory in the order of the C programming language. Standard observables can be handled by standard operators and graphical clients.
   - Elements of an incomplete observable. This may be, for example, one or more components of a vector, or the diagonal elements of a tensor, etc.. None of the standard WEC coordinate transformation operators can be applied to an incomplete observable, and the management of associated elements of an incomplete observable (*i.e.*, two elements of a vector) must be handled by special operators and graphical clients.

   The organisation of data at this level is defined by the concept of "rank". For a complete physical observable, this is the dimensionality (in the mathematical sense). By "complete", we mean an observable for which all the elements necessary to rotate or translate to another coordinate system are present: the three components of a vector, the nine elements of a tensor, *etc.*. "Complete" observables have ineteger rank. It is, however, convenient to enlarge the concept of rank to include objects which are not true observables in the physical sense *e.g.*, the two components of the spin-plane electric field, as explained in section 4.7.

   The low level data brick is homogeneous. For each data set of physically different nature, it is necessary to create a separate logical instrument, so as to benefit from the full power of ISDAT (see section 4.6 below)

2. **At the intermediate level**, a "data structure" is a set of the above lowest-level data bricks which are sampled simultaneously or sequentially; for example, a set of scalars, vectors, or tensors as a function of some other parameter(s) which do(es) not transform on rotation of axes. Examples of such parameters are:

   - the frequencies of a spectrum. The brick at the lowest level may represent an observable which is:
     - a scalar, for example, a spectral power density, in which case the intermediate-level structure is a power spectrum;

  - a tensor of rank 2, such as a spectral power matrix, the intermediate-level structure then being a cross-spectral matrix.
- the the components in velocity space of a particle flux; then the data brick is a scaler (particle flux density), while the intermediate-level structure is a particle distribution function.
- *etc.*.

To form the intermediate level structure, the data bricks are stored in the order of the parameters(s) of which they are a function. The organisation of data at this level is defined by the concept of "dimension": this is simply the number of parameter domains used for ordering the low-level bricks associated with the same time-tag.

3. **At the highest level**, successive intermediate-level structures are ordered chronologically. The overall size of the data object is determined by the number of "samples".

   For each intermediate-level structure there is an associated time stamp. For uniform sampling, it is sufficient to specify the start time, the time-step (*i.e.*, the increment), and the number of samples. For non-uniformly sampled data, time is supplied via time stamps which are listed in a table of length equal to the number of samples in the data object.

4. The meta-data is intimately associated with the overall data object.

   There are two broad categories of meta-data:

   (a) Information is essential for operators and clients to be able to work at all. The architecture of the data structure comes into this category.

   (b) Information which is not essential, but which is nevertheless useful to simplify, or render more robust, the different operators and clients, or as a service to the end user. Nevertheless, client software could be designed so as not to require this information, and the end user could obtain his information from other documents. As an example, units and limit values for plotting come into this category.

   Not surprisingly, the borderline between these two categories of meta-data is not well defined.

3 The ISDAT Data Structure

The WEC data structure is described below, proceeding from the highest (most general) to the lowest level of the hierarchy (*i.e.*, in the reverse order of the preceding section. In this description

- roman type is used for essential features, all of which described are (being) implemented,
- sans serif is used for non-essential features which have, however, been implemented, and
- *slanted script is used for non-essential features which have not yet been implemented.*

Note that operators and clients MUST test all meta data values and pointers which concern them. Therefore, if a field containing information required by some client has not been filled, any attempt to use that logical instrument by the clients requiring this information will result in an error message.

In the following sections, all items with roman labels ((a), (b), (c), ...) are options. When terminated by "*etc.*", the list is not exhaustive, *i.e.*, user-defined options are acceptable.

3.1 General

There will be a list containing the meta-data. This can include pointers, although they need not all be filled.

The following information will be provided:

1. The nature of the parameter being measured, for example:
   (a) electron density
   (b) magnetic field
   (c) magnetic field power spectral density
   (d) electric/magnetic field cross-power spectral density
   (e) *etc.*
2. The units in which it is measured, for example,
   (a) cm^{-3}
   (b) nT
   (c) $nT^2\ Hz^{-1}$
   (d) $\mu V/m\ nT\ Hz^{-1}$
   (e) *etc.*
3. Its type, which is either real or complex. For the above examples:
   (a) through (c) are real, while
   (d) is complex.
4. The numerical "fill" value, *i.e.*, the value used to fill data gaps.
5. The flight instrument sampling frequency (in Hz)
6. The cut-off frequency of the flight instrument pre-detector low-pass filter (in Hz).
7. *The oversampling factor. The default value is unity; the value will be greater if the data has been interpolated, for example, during joining.*
8. The version number of the programme and the calibration software used to produce the data. This will be in the form of a tracability code, which describes exactly how the data has been obtained (see section 4.1).
9. The sensor location with respect to the origin of coordinates (see section 4.3).
10. The known WEC coordinate system in which this sensor location is expressed (see section 4.2).
11. A flag to indicate if the data is in telemetry units. Data may be requested in this form for troubleshooting; it must be flagged.
12. *The name and e-mail address of the contact person.*
13. *The name of the PI.*
14. *Other information, still TBD, which may be required by WEC or by CDF.*

3.2 Timing

The time of the data samples is specified

1. In all cases, by

   - The time of the first data sample,
   - the time interval covered (i.e., the difference between the times of the last and the first data samples), and
   - the number of samples.

   For uniformly sampled data, this is all that is required; such data is called "segmented" data.

2. For non-uniformly sampled data, there is a pointer to the array containing the time-stamps; this pointer will be null in the case of segmented data. The structure of the data as described in the following sections is totally unaffected by this choice of timing.

3. *Further timing information may be provided at a lower level: see section 3.3, items 5 and 6 , and also section 4.5.*

3.3 Dimension

The dimension of the data structure must be specified. This is an integer greater than or equal to zero, which indicates the number of non-spatial domains associated with the sampling of the data.

For each domain contributing to the dimension, it is necessary to specify:

1. The physical nature of the domain. This may be (for example):

   (a) the frequency of a spectrum analyser;
   (b) the energy of a particle detector;
   (c) the angular elevation of a particle detector;
   (d) the azimuth of a particle detector;
   (e) the mass of the particles being counted;
   (f) the charge of the particles being counted;
   (g) *etc.*

2. The physical units used in the domain; for the above examples (a) through (d), these could be:
   (a) Hz
   (b) keV
   (c) and (d), degrees
   (e) proton masses
   (f) electronic charges
   (g) *etc.*

3. The "number" of discrete values in the domain (each value yielding a data brick); for the above examples, this could be respectively:
   (a) the number of frequency channels of the analyser;
   (b) the number of energy channels of the particle detector;
   (c) and (d), the number of bins (angular ranges);
   (e) the number of particle masses resolved;
   (f) the number of charge states resolved;
   (g) *etc.*.
   The total number of data bricks (as defined in section 3.4 below) per structure is the product of the numbers for each of the different domains.

4. A map. This is a table of length "number" which specifies, for each discrete value, the values (in the specified units) of the physical variable for successive data bricks. For our examples, these would be
   (a) a table of frequencies;
   (b) a table of energies;
   (c) a table of angular elevations;
   (d) a table of azimuthal directions;
   (e) a table of particle masses;
   (f) a table of charge states;
   (g) *etc.*.

5. *An "offset" table. This is a table of length "number" which specifies, for each discrete value, the offset in time of sampling of the data brick with respect to the first time-tag in that domain (see section 4.5).*

6. *A "window" table. This is a table of length "number" which specifies, for each discrete value, the time duration of the sampling window (see section 4.5).*

7. It is useful (required by CDF, useful, but not essential for the operation of ISDAT) to provide information for plotting in each domain. The information which should be provided is:
   - the lower limit of the plot scale;
   - the upper limit of the plot scale;
   - the most suitable type of plot scale: linear or logarithmic.

All the above items (1 through 6) exist for each domain. Therefore, if "dimension"> 0, the total size of the "map" and "offset" tables (items 4 and 5) is equal to the total number of data bricks in the structure (*i.e.*, bricks per time-tag), which is

$$\prod_{i=1}^{\text{“dimension”}} \text{“number”}_i$$

where "number"$_i$ is the number of values (item 3 above) associated with domain i.

3.4 Rank

Rank concerns the nature of the physical variable: scalar, vector, tensor of rank 2 or higher, and other known types as described in sections 4.6 and 4.7.

The following information must be provided:

1. The data brick type. This must be (see section 4.7) either

   (a) complete, or

   (b) incomplete.

   This information is essential for operators to know how to handle the data structure.

2. Whatever the type, it is essential to give further information, as follows:

   If **complete**, the rank of must be one of the following:

   (a) a tensor of rank 0, *i.e.*, a scalar;

   (b) a tensor of rank 1, *i.e.*, a vector;

   (c) a tensor of rank 2, (*e.g.* pressure);

   (d) a tensor of rank 3, (*e.g.* heat flux).

   Arrays are stored in memory in the order of the C programming language. Complex logical instruments (sect. 3.1, item 3) have the real and imaginary parts of each separate element of the array stored in adjacent memory locations, as in the FORTRAN programming language.

   If **incomplete**, another standard known type must be specified:

   (a) a 2D vector (see section 4.7);

   (b) a 3×2 "tensor" with has three elements in on direction and only two in the other, as explained in section 4.7;

   (c) the diagonal elements of a tensor

   (d) a particle counts object (TBD)

   (e) an inhomogeneous vector object (this is a "catch-all" option which, however, is not recommended);

   (f) etc. (new standard types can be defined as and when necessary).

   Note that each logical instrument contains only one type of data brick (see section 4.6 below).

3. Except for objects of rank 0 (*i.e.*, scalars), the coordinate system with respect to which the typed data objects are represented must be indicated by a flag (see section 4.8 below). Known WEC coordinate systems are:

   (a) satellite mechanical-build coordinates;

   (b) despun (*i.e.*, inertial) satellite coordinates;

   (c) geocentric solar ecliptic (GSE) coordinates;

   (d) geocentric terrestrial coordinates (this includes longitude and latitude).

4. Except for objects of rank 0, the orientation of the sensor axes must be described by means of the direction cosines of each of the sensor axes with respect to the known coordinate system specified by item 3 above, as described in section 4.8.

(a) For data coming directly from sensors aboard the spacecraft, the components will, in general, not be aligned with the axes of the spacecraft coordinate system (the spin axis and the two mutually perpendicular directions, the first defined to be in the meridian containing the sun pulse generator). Therefore it is necessary to specify:

- the direction cosines in body-build coordinates of the effective direction of each of the components of the sensor. These directions may not be mutually orthogonal, see section 4.8.

(b) For inertial coordinate systems,

- the unitary matrix required to rotate to one of the known WEC/ISDAT coordinate systems.

In either case, if not defined explicitly this matrix will be assumed to be the identity matrix.

4 Miscellaneous Explanations

4.1 Tracability

All data must be tracable origin. The WEC data group has decided that for data coming from the database handler, tracability will be assured (item 8 of section 3.1) by a code

`aa.bb.cc.dd`

in which the different fields have the following significance:

`aa` : the ISDAT software version number;
`bb` : the ISDAT software revision number;
`cc` : the WEC instrument software version number;
`dd` : the WEC calibration set version number.

It is clear that further thought needs to be given to the tracability code attached to data which has been joined or otherwise processed.

4.2 Coordinate Systems

All vector or tensor data, as well as the sensor location, must be specified with respect to some recognised coordinate system. WEC/ISDAT presently recognises five orthonormal Cartesian coordinate systems:

Mechanical-build coordinates. This is the coordinate system used to build the spacecraft; it is defined in the Experiment Interface Document (ref. 1). It is the system with respect to which all instrument sensors are mechanically aligned, and with respect to which the actual physical axis of each directional sensor can be defined (item 4 of section 3.4). The Cluster Project has designed the spacecraft to spin about the mechanical-build Ox axis, with the $+Oy$ axis parallel to the direction of the radial boom which supports the STAFF search coil. Furthermore, it is important to note that the "WEC" E_y and E_z axes are respectively in the mechanical-build (0, 1, 1) and (0, –1, 1) directions (ref. 2, section 2, page 100). This 45° offset must be included in the definition of the sensor reference axes.

Despun Satellite coordinates. The O3 axis is in the direction of the satellite spin axis, and the O1 axis is in the meridian defined by the direction of the Sun.

Geocentric Solar Ecliptic coordinates. GSE coordinates have the O3 axis in the direction of the north solar ecliptic pole, and the O1 axis in the meridian containing the direction of the Sun.

Geocentric Geographic coordinates. A Cartesian system with the O3 axis in the direction of the geographic north pole and O1 axis in the Greenwich meridian. This system is mainly used by ground-based experiments.

Most spacecraft measurements are made with respect to the mechanical-build frame of reference, and GSE is probably the inertial coordinate system which will be most frequently used by the Cluster community. The despun satellite coordinate system is also used (for example, by the STAFF-SA instrument) and, furthermore, it is an unavoidable point of passage between the satellite system and any inertial coordinate system. Ground-based experiments are intimately related to geographical coordinates.

These four known coordinate systems are adequate for the vast majority of STSP logical instruments; but others are available, or can be added if and when required. A complete list of currently available coordinate systems is provided in ref. 5.

Two other coordinate systems are of particular importance within ISDAT, because of the key role they play in time varying coordinate transformations (see ref. 5)

The Spin Reference coordinate system. The passage from mechanical-build to despun satellite coordinates involves the spin reference coordinate system, which has its O3 axis along the direction about which the spacecraft actually spins (this is subtly different from the axis about which it was designed to spin), and its O1 axis in the mechanical build O12 plane. For further details, see ref. 5.

Geocentric Equatorial Inerital, or astronomical coordinates. This is as near as one can get to a truly inertial system. It is the coordinate system in which the direction of the spacecraft spin axis is delivered (in terms of right ascension α and declination δ) by ESOC; its corresponding Cartesian direction cosines are $\cos\delta\ \cos\alpha$, $\cos\delta\ \sin\alpha$, $\sin\delta$.

All these coordinate systems are, of course, available for general scientific use.

4.3 The sensor location

For every logical instrument it is possible to specify the position of the sensor. A satellite instrument sensor may be placed on a boom or other spacecraft structure; when not specified, this usually means that the position is unimportant or self-evident: but this in itself is scientific information. For a ground-based instrument, it is essential to define the latitude and longitude of the observation site.

The location of the sensor will be measured with respect to the origin of the known coordinate system (section 4.2) (unless this is despun satellite coordinates, as for STAFF-SA, but in this case the sensor location has no meaning). For the sake of homogeneity, it is proposed that the sensor position always be specified in Cartesian coordinates. The use of a Cartesian system for defining latitude and longitude offers the possibility of defining the sensor altitude too (*e.g.*, for a geostationary satellite).

4.4 The number of dimensions

It will be noted that much of the same information is required both for the physical variable measured by the logical instrument itself (section 3.1, item 1), and for the variables associated with each of the physical domains contributing to the dimension (section 3.3, item 2).

The reason for treating the logical instrument variable differently from the "dimension" variables is that the former is continuously variable (within the limitations of the telemetry (8, 12 or 16-bit words), whilst the latter are clearly quantised, as defined by "number" and "map".

4.5 High resolution timing

Within a given logical instrument, the different "domains" may not all be sampled simultaneously; this is particularly true for instruments which scan. A swept-frequency radio receiver is an example, as is any particle experiment which either sweeps in energy or scans in azimuth and/or elevation. For many purposes, the timing as described in section 3.2 is adequate, but for highest resolution studies the exact value of the time of sampling is are required. These are provided by the time offset table described in section 3.3, item 5.

The time during which the particular domain is being sampled is generally not equal to the time between the sampling of successive domains. For example, a sweep-frequency wave receiver requires some dead-time for its oscillators to stabilise, a particle detector acquires counts during a shorter time. The table described in section 3.3, item 6 allows the sampling window to be determined precisely.

4.6 Homogeneity of the Rank

The rank of the data set is essential information
- for operators and clients to be able to use the data at all, and
- to determine which filters and/or clients can be used on any particular data set

Because of the fundamental nature of rank, the WEC/ISDAT data structure requires logical instruments to be of homogeneous rank. The dimension and numbers (item 3 of section 3.3) together indicate the number of bricks contained in the data structure: these bricks are all *identical.* Unlike the CDF, it is not possible to mix scalars and tensors. A WECdata logical instrument cannot include, for example, both density (rank 0) and vector magnetic field (rank 1). The reason for this difference is that WEC/ISDAT is an analysis tool, while CDF is primarily a storage tool.

We may note that the mixing of different data objects is totally compatible with the concept of the logical instrument. Data objects with different structure are necessarily from physical observables of different nature. Therefore they cannot share the same clients, and should be kept separate, in different logical instruments.

4.7 Complete Rank and Incomplete Rank

A physical variable is said to be "complete" if its representation includes all the information required to transform it to another coordinate system. In particular, a complete vector three components, and a complete tensor of rank 2 has nine components, and so on. Many, if not most, of the vectors and tensors handled by WEC/ISDAT will be complete; this is certainly true for the CSDS Prime and Summary Parameters.

There is an important category of observation for which only two components of the vector are observed. With one component missing, the observation is not a complete vector (rank 1), but neither is it a scalar (rank 0); it lies somwhat between the two. There is one example of such a logical instrument which is particularly important for WEC/ISDAT: the electric field. Such a vector of can be rotated only in its plane of measurement, *i.e.*, about the spacecraft spin axis. Such rotation is rather important, because it permits the transformation from the rotating spacecraft coordinate system to an inertial frame.

Between a vector of rank 1 and a tensor of rank 2, it is possible to have a brick with one dimension complete (3 elements) and the other with only two elements; the Cluster/STAFF cross-correlation between the magnetic and electric fields is an example. It too can be rotated only about the axis perpendicular to the plane of the 2-vector.

It is also possible to have a 2×2 brick, for example, the Cluster/STAFF cross-correlation of the electric field. Likewise, it can be rotated only about the axis perpendicular to the plane of the 2-vector.

4.8 The Data Coordinates

For any logical instrument which is not scalar, it is essential to specify completely the coordinate system in which it is measured.

The coordinate system used to represent a vector or tensor quantity must be specified with respect one of the known WEC/ISDAT coordinate systems (section 4.2). Once the most suitable known WEC system has been chosen, the logical instrument coordinate system is specified in terms of the direction cosines of its axes with respect to the known WEC system. For example, these may be :–

- The direction cosines (in satellite coordinates) of the effective axes of the sensors; these are rarely aligned with the satellite coordinate axes.
- The direction cosines (in GSE coordinates) of the coordinate axes used for the data analysis; for example, the axes of the boundary normal coordinate system being used to study a particular region of strong spatial gradient.

Note that the effective axis of a sensor may deviate significantly from its geometrical axis; this is certainly true for electric and magnetic sensors, which are sensitive to perturbations of the near field due to physical obstructions. Therefore the coordinate system in which the data is presented may be significantly non-orthogonal. This is not serious: the complete set of direction cosines provides sufficient information to be able to inverse the matrix and rotate a complete observable to any arbitrary (orthogonal or otherwise) coordinate system. The same is only partially true for an incomplete observable.

5 Coordinate Transformations

The transformation of data from one coordinate system to another is a major requirement for the WEC/ISDAT system. Such transformation will be

- required to despin the data from satellite coordinates to inertial coordinates,
- used by several WEC logical instruments, and
- necessary to compare data from different instruments or different spacecraft. Indeed,
- the joining of data sets will be a major activity during the Cluster data analysis.

Since every data set is self describing, any change of coordinate system of a data set requires that the meta-data associated with the data set be modified too. Furthermore, there is a multitude of useful coordinate systems (see ref. 5 for a selection of possibilites). Due to the need to handle the meta-data in a homogeneous way, there is one common coordinate transformation operator, as described in ref. 5.

5.1 General transformations

Fortunately, the information specifying the transformation to be effected can be handled as a data set in its own right; it can be completely described using the data structure of section 3. This has a further advantage: the rotation operator can operate not only on a science data structure but also on the data structure describing the rotation to be performed. In this way, it is possible to use the general rotation operator plus the associative property of matrices to reduce two or more successive rotations to one single composite rotation which is all that need be applied to the actual physical data.

The rotation data structure consists of the three base vectors of the "old" coordinate system represented in the "new" coordinate system; thus the rotation data set is a structure of rank 1 (vectors) and dimension 1 (three of them). For further details, the reader is referred to ref. 5.

5.2 The Despin Operator

The coordinate transformation from satellite (rotating) to inertial coordinates has the particularity that it is changing very rapidly, in some cases on a time-scale comparable with the sampling interval of the data itself. Furthermore:

- The rotation matrix required to despin the data is varying sinsoidally (*i.e.*, not monotonically), and therefore linear interpolation using the joining algorithm of ref. 3 is not possible (unless the matrix is provided with unnecessarily high time resolution).
- The rotation matrix cannot form part the meta-data of the logical instrument, because this would create the necessity of joining inhomogenous data types (the data itself, plus the rotation matrix), an eventuality which has already been eliminated (section 4.6).

Therefore a special despin operator is required. This operator is used as and when required by the common coordinate transformation operator, as described in ref. 5.

6 Meta Data and Status Data

Meta data is information which is intimately attached to the physical data (*e.g.*, units, coordinate system, possibly limit values), and which is supplied "attached to the data object" as described in section 3. It must be joined when the physical data is joined.

Status data is information which is separated from the instrument data to which it refers, for use independently of the instrument physical data. In particular, it may be used by other experiments, to determine the conditions for their own observations. It has an inherent time granularity, which is somewhat arbitrary. Depending upon the precise definition of the status, it may be more or less degraded when joined. It is preferable not to join status files when they are changing rapidly (which is also when the are most interesting).

7 Reference Documents

1. Experiment Interface Document, Part A, ESTEC document CL–EST–RS–0002/EID A, Issue 2, Rev. 0, 1993/06/01.
2. Experiment Interface Document, Part B, ESTEC document CL–EST–RS–0451/EID B, Issue 2, Rev 0, 1994/06/30.
3. The Joining of WEC/ISDAT Data, WEC/ISDAT document CWD–OBSPM–TN–001.
4. ISDAT 2.0 Definition of Filter, WEC/ISDAT document CWD-IRF-TN-001.
5. The WEC/ISDAT Coordinate Transformations, WEC/ISDAT document CWD–OBSPM–TN–002.

APPENDIX 1

The file Dbdata.h

The file /wecftp/doc/drafts/IRFU/Dbdata.h

```
  20 -rw-rw-r--   1 79       100        10028 Mar 16 11:08 Dbdata.h
```

has been recovered from the IRFU server.

```
#ifndef DBDATA_H
#define DBDATA_H

/*
#IN Include name: $Id: Dbdata.h,v 1.1 1994/12/20 21:53:09 al Exp $
#IA Anders Lundgren, IRFU

#ID Include description:
#ID Type definitions to interface to the DbGetData() function
#ID and related function prototypes.
*/

/* rank constants */
#define DbRANK_0 0
#define DbRANK_1 1
#define DbRANK_2 2
#define DbRANK_3 3
#define DbRANK_DIAG 20
#define DbRANK_2D 21

/* complete constants */
#define DbRANK_INCOMPLETE 0
#define DbRANK_COMPLETE 1

/* type constants */
#define DbTYPE_REAL_FLOAT 1
#define DbTYPE_REAL_DOUBLE 2
#define DbTYPE_COMPLEX_FLOAT 3
#define DbTYPE_COMPLEX_DOUBLE 4
#define DbTYPE_REAL_UCHAR 6
#define DbTYPE_REAL_SHORT 7
#define DbTYPE_REAL_INT 9
#define DbTYPE_STRING 15

/* coordinate system constants */
#define DbCOORD_SENSOR 1
#define DbCOORD_PLATFORM 2
#define DbCOORD_DESPUN 3
#define DbCOORD_GSE 4

/*
 * type definitions for the data hierarchy specification
 */
typedef struct _DbDataSpec { /* data hierarchy description
```

```
   (logical instrument) */
    int project;
    int member;
    int instrument;
    int sensor;
    int signal;
    int channel;
    int parameter;
} DbDataSpec;

typedef struct _DbSpecName {
    char project[32];
    char member[32];
    char instrument[32];
    char sensor[32];
    char signal[32];
    char channel[32];
    char parameter[32];
} DbSpecName;

/*
 * type definitions for the data object
 */

typedef struct _DbDataRF0 { /* real float scalar */
    float re;
} DbDataRF0;

typedef struct _DbDataRF1 { /* real float vector (eg. magnetic field) */
    float re[3];
} DbDataRF1;

typedef struct _DbDataRF2 { /* real float tensor of rank 2
   (eg. pressure, cross correlations) */
    float re[3][3];
} DbDataRF2;

typedef struct _DbDataRF3 { /* real float tensor of rank 3
   (eg. heat flux) */
    float re[3][3][3];
} DbDataRF3;

typedef struct _DbDataRF2D { /* real float 2D vector */
    float re[2];
} DbDataRF2D;

typedef struct _DbDataRD0 { /* double scalar */
    double re;
} DbDataRD0;

typedef struct _DbDataRD1 { /* double vector (eg. magnetic field) */
    double re[3];
} DbDataRD1;

typedef struct _DbDataRD2 { /* double float tensor of rank 2
   (eg. pressure, cross correlations) */
```

```
    double re[3][3];
} DbDataRD2;

typedef struct _DbDataCF0 { /* complex float scalar */
    float re; /* real part */
    float im; /* imaginary part */
} DbDataCF0;

typedef struct _DbDataCF1 { /* complex float vector */
    float re[3]; /* real part */
    float im[3]; /* imaginary part */
} DbDataCF1;

typedef struct _DbDataCF2 { /* complex float tensor of rank 2 */
    float re[3][3]; /* real part */
    float im[3][3]; /* imaginary part */
} DbDataCF2;

typedef struct _DbDataCF3 { /* complex float tensor of rank 3 */
    float re[3][3][3]; /* real part */
    float im[3][3][3]; /* imaginary part */
} DbDataCF3;

typedef struct _DbDataCF2D { /* complex float 2D vector */
    float re[2];
    float im[2];
} DbDataCF2D;

typedef struct _DbDataCD0 { /* complex double scalar */
    double re; /* real part */
    double im; /* imaginary part */
} DbDataCD0;

typedef struct _DbDataRUC0 { /* real unsigned char data */
    unsigned char re;
} DbDataRUC0;

typedef struct _DbDataRS0 { /* real short scalar */
    short re;
} DbDataRS0;

typedef struct _DbDataRI0 { /* real int scalar */
    int re;
} DbDataRI0;

typedef struct _DbDataString { /* text data */
    int type; /* optional type value */
    char *s; /* text string */
} DbDataString;


typedef struct _DbDataSegment {
    IsTime start; /* time of first sample */
    IsTime interval; /* time interval covered */
    int samples; /* number of samples */
    void *data; /* pointer to an array of:
```

```
   DbDataF0   (rank = 0, type = DbTYPE_FLOAT)
   DbDataF1   (rank = 1, type = DbTYPE_FLOAT)
   DbDataF2   (rank = 2, type = DbTYPE_FLOAT)
   DbDataF3   (rank = 3, type = DbTYPE_FLOAT)
   DbDataCF0  (rank = 0,
       type = DbTYPE_COMPLEX_FLOAT)
   DbDataCF1  (rank = 1,
       type = DbTYPE_COMPLEX_FLOAT)
   DbDataCF2  (rank = 2,
       type = DbTYPE_COMPLEX_FLOAT)
   DbDataCF3  (rank = 3,
       type = DbTYPE_COMPLEX_FLOAT)
   DbDataF1   (rank = DbRANK_DIAG,
               type = DbTYPE_FLOAT)
   DbDataCF1  (rank = DbRANK_DIAG,
               type = DbTYPE_COMPLEX_FLOAT)
   DbDataF2D  (rank = DbRANK_2D,
       type = DbTYPE_FLOAT)
   DbDataCF2D (rank = DbRANK_2D,
       type = DbTYPE_COMPLEX_FLOAT) */
    IsTime *time; /* pointer to samples number of time tags,
   time may be a NULL pointer */
} DbDataSegment;

typedef struct _DbDataInfo {
    int units; /* units (eg. DbUN_VOLT) */
    int quantity; /* quantity (eg. DbQTY_AMPLITUDE) */
    int scaleType; /* scale type (DbSCALE_LIN, DbSCALE_LOG,
   DbSCALE_IRREGULAR) */
    double scaleMin; /* min value to be used in plot */
    double scaleMax; /* max value to be used in plot */
    double samplingFreq; /* sampling frequency used in the instrument */
    double filterFreq; /* filter frequency used in the instrument */
    char unitString[32]; /* units string */
    char quantityString[32]; /* quantity string */
    char conversion[80]; /* SI conversion string, only valid if units
   is set to DbUNDEF */
} DbDataInfo;

typedef struct _DbCoordinate {
    int reference; /* the reference coordinate system used */
    int system; /* coordinate system of returned data,
   DbCOORD_SENSOR, DbCOORD_PLATFORM,
   DbCOORD_DESPUN or DbCOORD_GSE */
    DbDataRD2 rot; /* rotation matrix with respect
   to DbCOORD_PLATFORM */
    DbDataRD1 location;
    DbDataRD1 direction;
} DbCoordinate;

typedef struct _DbDataObject {
    int rank; /* 0, 1, 2, 3, DbRANK_2D or DbRANK_DIAG */
    int complete; /* DbRANK_COMPLETE if rank is complete,
   DbRANK_INCOMPLETE if rank is uncomplete
   ie. is not a true tensor.
   If a tensor is not complete the missing
```

```
   elements will be filled with NaN */
    int dataType; /* DbTYPE_FLOAT, DbTYPE_COMPLEX_FLOAT,
   DbTYPE_SHORT, DbTYPE_BYTE or
   DbTYPE_STRING */
    int dimension; /* dimensionality of the phase space,
   0, 1, 2, ... */
    int *n; /* number of identical data types in each
   dimension (phase space dimension)
   per sample,
   the size of this array is dimension,
   n is a NULL pointer if dimension = 0 */
    int pack; /* data pack mode used (DbPACK_SEGMENT,
   DbPACK_TIMETAG or DbPACK_FILL) */
    int reduction; /* type of reduction performed (DbRED_NONE,
   DbRED_AVERAGE, DbRED_MIN or DbRED_MAX */
    int gapFill; /* how gaps are handled (DbFILL_NAN,
   DbFILL_INTERPOL, DbFILL_ZERO), only valid
   if pack is DbPACK_FILL */
    int segments; /* number of segments in segment table */
    DbDataSegment *seg; /* table of segment descriptors,
   each segment contains a contiguous data set,
   the table has segments number of entries */
    int samples; /* total number of samples in all segments */
    /* meta data */
    DbDataInfo *info; /* info table with info for each dimension,
   the size of this array is (dimension + 1) */
    int mapType; /* the data type of each map value,
   DbTYPE_FLOAT or DbTYPE_STRING */
    void **map; /* info map values for each dimension,
   the size of each array is n[0], n[1],
   ..., n[dimension - 1],
   map is a NULL pointer if dimension = 0 */
    float **timeOffset; /* time offset values for each dimension,
   the size of each array is n[0], n[1],
   ..., n[dimension - 1],
   timeOffset is a NULL pointer if
   dimension = 0 or if no time offset
   values are needed */
    float **timeWindow; /* time window values for each dimension,
   the size of each array is n[0], n[1],
   ..., n[dimension - 1],
   timeWindow is a NULL pointer if
   dimension = 0 or if no time window
   values are needed */
    DbCoordinate *coord; /* coordinate system data, NULL pointer
   if not available */
    DbDataSpec spec; /* data hierarchy description
   (logical instrument) */
    char title[32]; /* title string to be used in plot */
    char message[64]; /* optional message */
    char version[32]; /* version string */
    /* request status */
    unsigned int warning; /* warning bit mask */
} DbDataObject;
```

```
/*
 * type definitions for the data request DbGetData()
 */

typedef struct _DbDataRequest {
    IsTime start; /* start time of requested data */
    IsTime interval; /* time interval of requested data */
    DbDataSpec spec; /* data hierarchy description
   (logical instrument) */
    int units; /* requested units (DbUN_PHYS, DbUN_CORR,
   DbUN_TM or DbUN_META) */
    int reduction; /* type of reduction to perform (DbRED_NONE,
   DbRED_AVERAGE, DbRED_MIN or DbRED_MAX */
    int samples; /* maximum number of samples to return if
   reduction is requested */
    int gapFill; /* how to treat gaps (DbFILL_NONE, DbFILL_NAN,
   DbFILL_INTERPOL, DbFILL_ZERO) */
    int pack; /* data pack mode (DbPACK_SEGMENT,
   DbPACK_TIMETAG or DbPACK_FILL) */
} DbDataRequest;


/*
 * type definitions for the coordinate transform
 * matrix request DbGetCTMatrix()
 */

typedef struct _DbCTMatrixRequest {
    IsTime start; /* start time of requested data */
    IsTime interval; /* time interval of requested data */
    DbDataSpec spec; /* data hierarchy description
   (logical instrument) */
    int fromSystem; /* the present coordinate system */
    int toSystem; /* the wanted coordinate system */
    int segments; /* number of segments in segment table */
    DbDataSegment *seg; /* table of segment descriptors,
   the data part will not be used,
   the table has segments number of entries */
    int samples; /* total number of samples in all segments */
    int pack; /* data pack mode (DbPACK_SEGMENT,
   DbPACK_TIMETAG or DbPACK_FILL) */
} DbCTMatrixRequest;

#endif /* DBDATA_H */
```

APPENDIX 1

Examples of Logical Instrument Structures

The current version of the examples are taken from the following message

```
From:   SMTP%"al@irfu.se" 21-MAR-1995 14:38:03.59
To:     harvey@megash.obspm.fr
CC:
Subj:   data structure examples
```

Example 1 - EFW waveform data sampled at 25 Hz

```
Dbdata data structure example 1.
================================

EFW waveform data sampled at 25 Hz.


dataObject
    rank = 0
    complete = DbRANK_COMPLETE
    dataType = DbTYPE_REAL_FLOAT
    dimension = 0
    n[] = NULL
    pack =  DbPACK_TIMETAG
    reduction = DbRED_NONE
    gapFill =
    segments = 1
    seg[] = see below
    samples = 4
    info[] = see below
    mapType =
    map[][] = NULL
    timeOffset[][] = NULL
    timeWindow[][] = NULL
    title = ""
    message = ""
    version = "2.0.2.23"
    warning = 0

seg[0]
    start = xxxx
    interval = 0.16
    samples = 4
    data[] = see below
    time[] = see below

info[0]
    units = DbUN_VOLT
    quantity = DbQTY_AMPLITUDE
    scaleType = DbSCALE_LIN
```

```
    scaleMin = -30.0
    scaleMax = 30.0
    samplingFreq = 25.0
    filterFreq = 10.0

data[] (array of type DbDataRF0)
    0 = -3.5
    1 = 2.5
    2 = 10.0
    3 = 30.0

time[] (array of type IsTime)
    0 = xxxx.00
    1 = xxxx.04
    2 = xxxx.08
    3 = xxxx.12
```

Example 2 - Viking V4 DFT data

```
Dbdata data structure example 2.
================================

Viking V4 DFT data.


dataObject
    rank = 0
    complete = DbRANK_COMPLETE
    dataType = DbTYPE_REAL_FLOAT
    dimension = 1
    n[] = see below
    pack = DbPACK_TIMETAG
    reduction = DbRED_NONE
    gapFill =
    segments = 1
    seg[] = see below
    samples = 2
    info[] = see below
    mapType = DbTYPE_REAL_FLOAT
    map[][] = see below
    timeOffset[][] = NULL
    timeWindow[][] = NULL
    title = ""
    message = ""
    version = "2.0.2.23"
    warning = 0

n[]
    0 = 256

seg[0]
    start = xxxx
    interval = 0.6
    samples = 2
    data[] = see below
    time[] = see below

info[0]
    units = DbUN_HZ
    quantity = DbQTY_FREQUENCY
    scaleType = DbSCALE_LOG
    scaleMin = 0.0
    scaleMax = 15600.0
    samplingFreq = 3.333
    filterFreq = 0.0 (not applicable)

info[1]
    units = DbUN_V_PER_M_SQR_PER_HZ
    quantity = DbQTY_POWER
    scaleType = DbSCALE_LOG
    scaleMin = 0.1
```

```
    scaleMax = 300.0
    samplingFreq = 30000.0
    filterFreq = 15600.0

map[0][]
    0 = 0.0
    1 = 3.5
    2 = 7.3
    :
    255 =  15600.0

data[] (array of type DbDataRF0)
    0-255 = spectra 0
    256-511 = spectra 1

time[] (array of type IsTime)
    0 = xxxx.0
    1 = xxxx.3
```

Example 3 - Particle Data

```
Dbdata data structure example 3.
================================

Particle data.


dataObject
    rank = 0
    complete = DbRANK_COMPLETE
    dataType = DbTYPE_REAL_FLOAT
    dimension = 2
    n[] = see below
    pack = DbPACK_TIMETAG
    reduction = DbRED_NONE
    gapFill =
    segments = 1
    seg[] = see below
    samples = 2
    info[] = see below
    mapType = DbTYPE_REAL_FLOAT
    map[][] = see below
    timeOffset[][] = NULL
    timeWindow[][] = NULL
    title = ""
    message = ""
    version = "2.0.2.23"
    warning = 0

n[]
    0 = 128
    1 = 32

seg[0]
    start = xxxx
    interval = 0.5
    samples = 2
    data[] = see below
    time[] = see below

info[0]
    units = DbUN_DEGREES
    quantity = DbQTY_ANGLE
    scaleType = DbSCALE_LIN
    scaleMin = 0.0
    scaleMax = 90.0
    samplingFreq = 4.0
    filterFreq = 0.0 (not applicable)

info[1]
    units = DbUN_DEGREES
    quantity = DbQTY_ANGLE
    scaleType = DbSCALE_LIN
```

```
    scaleMin = 0.0
    scaleMax = 30.0
    samplingFreq = 0.0 (not applicable)
    filterFreq = 0.0 (not applicable)

info[2]
    units = DbUN_HZ
    quantity = DbQTY_COUNTS
    scaleType = DbSCALE_LIN
    scaleMin = 0.0
    scaleMax = 1000.0
    samplingFreq = 0.0 (not applicable)
    filterFreq = 0.0 (not applicable)

map[0][]
    0 = 0.0
    1 = 7.0
    2 = 14.0
    :
    127 =  90.0

map[1][]
    0 = 0.0
    1 = 0.93
    2 = 1.86
    :
    31 =  30.0

data[] (array of type DbDataRF0)
    0-4095 = sweep 0
    4096-8191 = sweep 1

time[] (array of type IsTime)
    0 = xxxx.00
    1 = xxxx.25
```

Example 4 - The STAFF acB Instrument

```
Dbdata data structure example 4.
================================

CLUSTER STAFF acB instrument.


dataObject
    rank = 2
    complete = DbRANK_COMPLETE
    dataType = DbTYPE_COMPLEX_FLOAT
    dimension = 1
    n[] = see below
    pack = DbPACK_TIMETAG
    reduction = DbRED_NONE
    gapFill =
    segments = 1
    seg[] = see below
    samples = 2
    info[] = see below
    mapType = DbTYPE_FLOAT
    map[][] = see below
    timeOffset[][] = NULL
    timeWindow[][] = NULL
    title = ""
    message = ""
    version = "2.0.2.23"
    warning = 0

n[]
    0 = 27

seg[0]
    start = xxxx
    interval = 4.0
    samples = 2
    data[] = see below
    time[] = see below

info[0]
    units = DbHZ
    quantity = DbQTY_FREQUENCY
    scaleType = DbSCALE_LIN
    scaleMin = 8.0
    scaleMax = 4096.0
    samplingFreq = 0.25
    filterFreq = 0.0 (not applicable)

info[1]
    units = DbUN_NT_SQR_PER_HZ
    quantity = DbQTY_POWER
    scaleType = DbSCALE_LOG
    scaleMin = 0.1
```

```
    scaleMax = 1000.0
    samplingFreq = 9000.0
    filterFreq = 4000.0

map[0][]
    0 = 8.9797
    1 = 11.3137
    2 = 14.2544
    :
    26 =  3649.12

data[] (array of type DbDataCF2)
    0-26 = spectra 0
    27-51 = spectra 1

time[] (array of type IsTime)
    0 = xxx0.0
    1 = xxx4.0
```