HIRDLS

SW-HIR-2008

Originator: Charles Cavanaugh

Date: 10 Dec. 2012

Subject/Title:

The Design of the HIRDLS Level 2 Preprocessor

Description/Summary/Contents:

The purpose of this document is to create a design specification for the High Resolution Dynamics Limb Sounder (HIRDLS) Level 2 Preprocessor, hereby known as L2 Preprocessor. This design specification will extend the architecture of L2 Preprocessor at a level of detail sufficient to facilitate implementation. It is assumed that the reader of this document has already read and understood the documented architectural plan of L2 Preprocessor.

Keywords:

Purpose of this Document:

Oxford University
Atmospheric, Oceanic &
Planetary Physics
Parks Road
OXFORD OXI 3PU
United Kingdom

University of Colorado, Boulder
Center for Limb Atmospheric Sounding
3450 Mitchell Lane, Bldg. FL-0
Boulder, CO 80301

EOS

The Design of the HIRDLS Level 2 Preprocessor

Charles Cavanaugh

Table of Contents

Table of Contents (continued)

List of Figures

List of Abstractions

1 Document Purpose and Goal

The purpose of this document is to create a design specification for the High Resolution Dynamics Limb Sounder (HIRDLS) Level 2 Preprocessor, hereby known as L2 Preprocessor. This design specification will extend the architecture of L2 Preprocessor at a level of detail sufficient to facilitate implementation. It is assumed that the reader of this document has already read and understood the documented architectural plan of L2 Preprocessor.

The goal of this document is to create a design that correctly models the overviewed task, and lays out a plan that distributes system intelligence as evenly as possible, is easy to understand and implement, and easy to extend or revise, if or when necessary in the future.

2 Design Notation and Goal

The notation used in this document will be the Unified Modeling Language (UML). However, the method used to specify each abstraction will be based on the work of Ward Cunningham and Kent Beck, and detailed in Designing Object-Oriented Software (Wirfs-Brock, et.al, 1990). This method places a high degree of importance on finding each abstraction's responsibilities and collaborations. C++ syntax will be used to specify detailed abstraction information (such as public contract interface). The goal of the design is to create the simplest system to correctly accomplish the task. Though effort will be made to make abstractions reusable throughout the system, no effort will be made to make abstractions reusable outside of the system, i.e. no functionality or data will exist that is not used by the system, unless that functionality makes for a more extensible system.

3 Design Considerations

As first mentioned in the L2 Preprocessor Requirements document, L2 Preprocessor is a stand-alone, non-graphical, non-embedded scientific application, and as such, resource usage has become a primary design consideration. Through negotiations with the HIRDLS Program Manager and HIRDLS Data Manager, available data storage size is not a concern. Application memory size, though, is a concern, and efficient usage of memory must be planned. The L2 Preprocessor memory management plan is twofold: 1) create a design that minimizes memory complexity; and 2) utilize tools during implementation to help find memory use flaws. The latter part of the plan is beyond the scope of this document. The former part has four approaches: 1) favor stack memory over heap memory; 2) implement allocation failure recovery; 3) reduce the scope of heap allocated memory; and 4) force safe memory parameter passing. Though the last three approaches are more implementation issues than design issues, all four approaches are addressed further in this section.

3.1 Favor Stack Memory

Data created with stack memory has the benefit of being unwound when the data's scope is terminated. Heap memory persists until explicitly terminated, and is therefore highly prone to leaking. If an abstraction is needed within a method, prefer to allocate it on the stack. If the method is called many, many times, consider having the needed abstraction as a private data member of the employing abstraction.

3.2 Recover From Failure

Stack memory use, though favored over heap memory use, will not be exclusive to a system with the size and complexity of the HIRDLS L2 Preprocessor. Every time an attempt is made to allocate heap memory, the status of the allocation must be checked before using the memory. If there was a failure, the system should recover in a consistent and graceful way. Immediately conveying failure information to the system user and exiting from the system is acceptable, and preferred.

3.3 Reduce Memory Scope

Again, there will be times when using heap memory is unavoidable (as when creating a vector of abstractions – vectorizing the address of the abstraction is much more efficient than vectorizing the entire abstraction). The scope of the accessibility of the memory must be reduced to its lowest practical point, but never higher than abstraction-wide scope. That is, the most preferred way to use heap memory is to allocate it, use it and destroy it within the same abstraction method. When it is not possible to destroy it within the same method as allocated (as with the example above), the memory must be destroyed in another method *of the same abstraction*.

3.4 Force Safe Passing

This aspect of the plan disallows destruction of memory passed into a method via the parameter list. If it is necessary to pass allocated memory to a method via the parameter list, that memory *must still exist in its original form* when the method terminates. The memory passed in is owned by another method, and it is incumbent on the owning method to destroy the memory, and any changes to the memory, or aggregation of the memory, is disallowed by the called method. If the language supports compile-time checking (such as forcing constant pointers), this must be used to verify the safeness of parameter memory.

4 Design Representations

The L2 Preprocessor Architecture document introduced various packages from which L2 Preprocessor will be built. Those packages will be enumerated in more detail in this document. Included in the detail will be the various abstractions that make up a given package. As mentioned in Section 2, UML notation will be used to show a package's internal mechanisms, including abstractions, dependencies, collaborations, responsibilities and contracts.

4.1 Packages

The packages that comprise L2 Preprocessor are logical encapsulations of a collection of abstractions that are homogeneous in purpose, with that purpose reflected in the package name. Packages are the building blocks for a system, and are the "whats" in that system. Figure 1 shows the hierarchy of the L2 Preprocessor packages introduced in the L2 Preprocessor Architecture document (though the names have been altered to fit this document's notation that packages have a singular name). The goal of that document was to identify the "whats" or packages necessary to fulfill the requirements of L2 Preprocessor. The goal of this document is to identify the "hows" of each package. The Diagnostics package is at the lowest level, and is accessible to all other packages. The Service package, which is to present to L2 Preprocessor packages a simplified front-end to SDP Toolkit, is a level higher than Diagnostics (which means Service can use Diagnostics), and is accessible to all other packages. The File package is introduced here, and its purpose is to provide a building block for file input/output. The Math package is also introduces here, and its purpose it to encapsulate system-wide mathematical calculation abstractions. The other packages shown in Figure 1 have only explicit accessibility to those packages to which each package's emanating arrows point.

4.2 Abstractions

Three different types of abstractions are used in L2 Preprocessor: process, control and data. Where packages are homogeneous *in purpose*, process abstractions are homogeneous *in action*, control abstractions are homogeneous *in idea*, and data abstractions are homogeneous *in content*. Process abstractions are about action, so they have "actiony" names, such as Scan Transformer or File Writer. Control abstractions are about idea, so they have "ideay" names, such as PCF Service or HIRDLS1R File. Data abstractions are about content, so they have "contenty" names, such as HIRDLS1C Scan or HIRDLS1R Scan. All abstractions, regardless of type, are to fully encapsulate all functionality needed to accomplish their specified task, and to present to L2 Preprocessor the simplest interface possible. Fully encapsulate does not mean an abstraction must be totally self-contained and needing no other abstractions. Fully encapsulate means that no other abstraction need know how it does its job, only that it does its job.

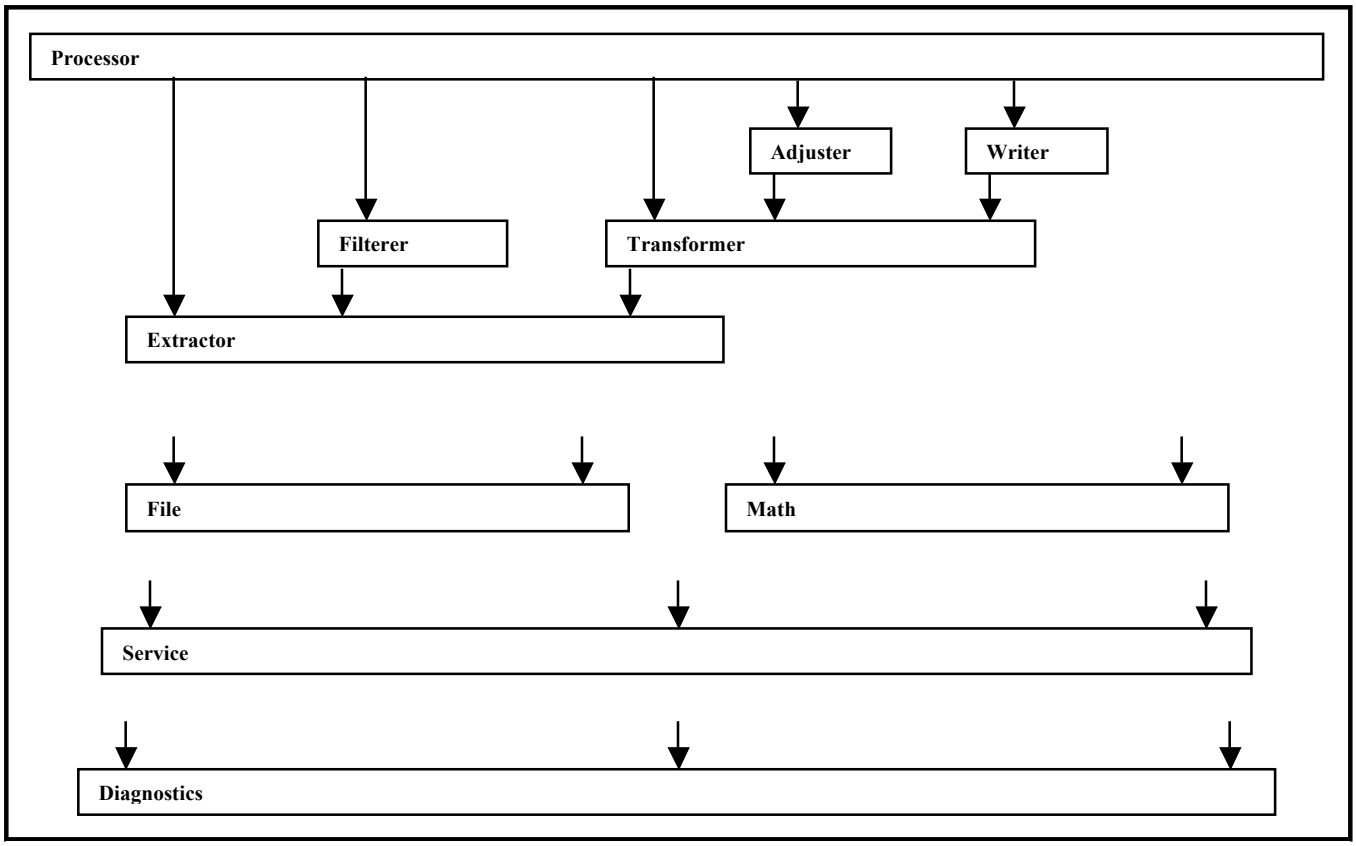

Figure 1 L2 Preprocessor Hierarchy of Packages

4.3 Dependencies

A package is dependent on another package when, obviously, the employing package needs something from the employed package. The exception to this is when two packages communicate via a data abstraction, which then makes the two packages dependent on the data abstraction. Because this approach minimizes inter-package dependencies within the system, or at the very least localizes the dependencies, communicating via data abstractions is the preferred way to handle inter-package dependencies in L2 Preprocessor. This is in agreement with the desire for *low coupling* in a system[1]. In a complex system, unplanned dependencies can get circular and unmaintainable, and one of the goals of L2 Preprocessor is to maximize maintainability. The L2 Preprocessor Architecture document shows that packages communicate with each other via data aggregations, and therefore L2 Preprocessor packages can be independently implemented and tested, making the system less circular and more maintainable. In the case(s) where a package encapsulates other packages, inter-package dependencies cannot be eliminated, but can be minimized by planning no inter-package dependencies amongst the encapsulated packages. In this document, abstraction dependency and hierarchy figures are interchangeable (similar to the packages in Figure 1).

4.4 Collaborations

Identifying and planning inter-package and inter-abstraction collaborations is one of the two important tasks for this L2 Preprocessor Design document (responsibility is the other important task, and that will be detailed in Section 4.5). Section 4.3 has already begun the discussion on collaborations, because collaborations, in the package or abstraction sense, are one-way streets and, therefore, dependencies arise. In the world of human interaction, two-way collaborations are considered

[1] W. Stevens, G. Myers, L. Constantine, "Structured Design", IBM Systems Journal, 13 (2), 115-139, 1974.

desirable, but in the logical, computer world, two-way collaborations are impossible, as you must first define a "thing" (package, abstraction, data, etc.) before some other "thing" can make use of it. Collaboration figures will be used by this document to show how abstractions interact to accomplish their respective tasks. These figures will employ UML notation to show aggregation, inheritance and simple collaboration (dependency without encapsulation).

4.5 Responsibilities

As first mentioned in Sections 2 and 4.4, identifying a package's or abstraction's responsibilities is one of the two responsibilities of this document. In much the same way the members of a software team have their own responsibilities, so too do packages and abstractions in a system. It is important to note that abstractions and packages come from responsibilities, and not vice-versa. As software system construction starts with a Requirements document, package and abstraction identification starts with responsibilities. The L2 Preprocessor Requirements and Architecture documents have already begun the process, and have identified numerous packages to carry out the system's distributed responsibilities. This document will extend the depth of those responsibilities and identify the abstractions that will need to be created to fulfill the newer, and more focused, responsibilities. The responsibilities of an abstraction will be enumerated in that abstraction's respective section, but will not be displayed in any figure.

4.6 Contracts

Up until now, the L2 Preprocessor documents have expounded on finding the "whats" in the system. The contracts of an abstraction detail the "hows". And where packages and abstractions are the nouns, contracts are the verbs. This document will enumerate the contracts for the abstractions, in their respective sections, in L2 Preprocessor. As mentioned in Section 4.3, we have the goal of minimizing inter-package and inter-abstraction dependencies, and using data abstractions for communication does that. With many abstractions, specifically the process and control abstractions, the contracts will detail that dependency minimization. With data abstractions, however, it is often the case that this method is very inefficient, and we would have to create a data abstraction that exists solely to update, for instance, another data abstraction. We therefore minimize dependency on these data abstractions by creating contracts that use only language primitives. A data abstraction, by definition, encapsulates data, not process or control, so there are no "internal workings" that we would want to hide from the system.

Contracts have the obligation to detail how they handle failure. Most often, this will involve returning a status Boolean to detail if they were able to successfully (true) or unsuccessfully (false) carry out their responsibilities. How to represent success and failure is dependent on the implementation language, but must be consistent throughout the system. If the language has a Boolean primitive, using it is preferred over the system creating its own. In cases where failure within a contract is catastrophic to the system (e.g., a memory creation call is unsuccessful) and the system needs to abort processing, the contract must specify that it has system abortion authorization (noted as 'SAA' in the contract tables, all of which are listed in Appendix A). Contracts that do have SAA might still return status Booleans, as the contract could still fail, though not catastrophically.

5 Design Methodology

As the L2 Preprocessor documentation has progressed from requirements to architecture to design, we have been employing a top-down methodology to further decompose the system. These documents have also talked about elements of L2 Preprocessor being "building blocks" upon which to build other elements, which is the methodology used in bottom-up synthesis. Figure 1 shows three "building block" packages (File, Service, Diagnostic), while showing all other packages in L2 Preprocessor, which were derived from top-down analysis. The File and Service packages exist to reduce complexity in the system, and are tasked to contain no more "usefulness" than is needed by L2 Preprocessor. The Diagnostic package exists to pass meaningful information from the system to the user, and will most likely grow or shrink long after the first production version of L2 Preprocessor has been delivered. Therefore, while the majority of the elements of L2 Preprocessor are derived using a top-down decomposition, an eye is still open to find where elemental building blocks can be best utilized.

6 Package Enumeration

L2 Preprocessor package designs will be enumerated, in bottom-up order, in the following sections. The intent of enumerating in this order is to have a package or abstraction well understood before inserting it into the workings of other packages and/or abstractions. The remainder of this document is left to the detail design of each previously introduced package.

7 Diagnostics Package

The Diagnostics package has the responsibility to provide the system a consistent means to report system diagnostics, including termination information. As discussed in Section 4.1, this package is the lowest level package in the system and does not depend on any other package. This non-dependency is a design constraint detailed in the Section 7.1. Figure 2 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

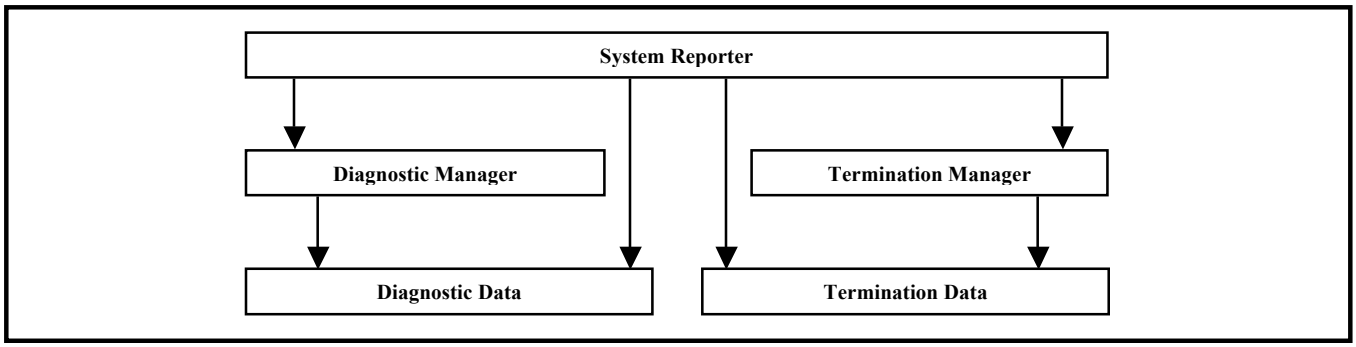

Figure 2 Diagnostics Package Hierarchy

7.1 System Reporter Abstraction

System Reporter is a control abstraction, and has the responsibility to accumulate system diagnostic and termination information, and generate a standardized summary report. To fulfill this responsibility, this abstraction collaborates with Diagnostic Manager, Termination Manager, Diagnostic Data and Termination Data, and presents an interface of Add and GetReporter contracts, as shown in Figure 3. The Add contracts allow the system to add diagnostic and termination information to the report. For the manager abstractions to work correctly, System Reporter must aggregate them and keep them persistent during its lifetime. This abstraction, in turn, must also be persistent to work correctly. This abstraction is specified to be globally accessible and fail-safe. Fail-safe means the report must be generated, even if the process terminates abnormally, and output to some device (file or screen), but employ no memory creation functionality or outside subsystem dependencies. This specification also applies to the abstractions with which System Reporter aggregates. The intent of this report is to give the operator some indication of what happened during a system run, so if this report is not generated, the run will have failed. There must be exactly one instance of this abstraction in the system, and therefore it is specified that this abstraction be a *Singleton creational pattern*[2], and the GetReporter contract is to return that instance. The classic implementation of a Singleton has the abstraction encapsulating a pointer to itself, but this breaks this abstraction's "no memory creation" requirement. Making the pointer a global stack pointer (guaranteed to be unwound off the stack at program termination), rather than a heap pointer, is allowable in this one exception. The public contracts of this abstraction must use only primitives (due to the no subsystem dependency requirement), and must not "fail" in the sense that processing should stop. Note that there is no contract to generate the status report. The report will be generated when the abstraction is destroyed.

[2] As detailed in Design Patterns, Elements of Reusable Object-Oriented Software by Gamma, et.al., 1995

7.2 Diagnostic Manager Abstraction

Diagnostic Manager is a control abstraction, and has the responsibility to accumulate and retrieve the reported system diagnostics. To fulfill this responsibility, this abstraction collaborates with Diagnostic Data, and presents an interface of Add and Retrieve contracts, as shown in Figure 4. The Add contract allows System Reporter to add diagnostic information to the report, and the Retrieve contract allows System Reporter to retrieve the added diagnostic information. Diagnostic Manager has the same "no memory creation" requirements as System Reporter, and therefore must aggregate a constant number of Diagnostic Data abstractions, and keep them persistent during its lifetime. This abstraction needs to be persistent to work correctly.

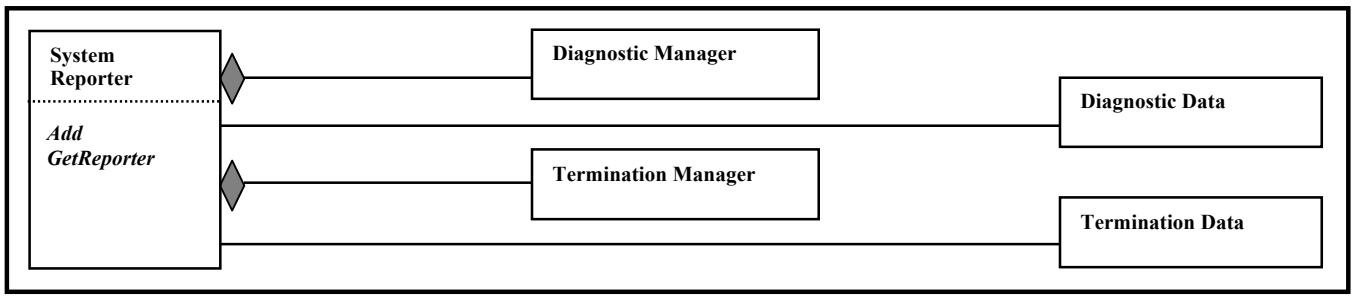

Figure 3 System Reporter Abstraction

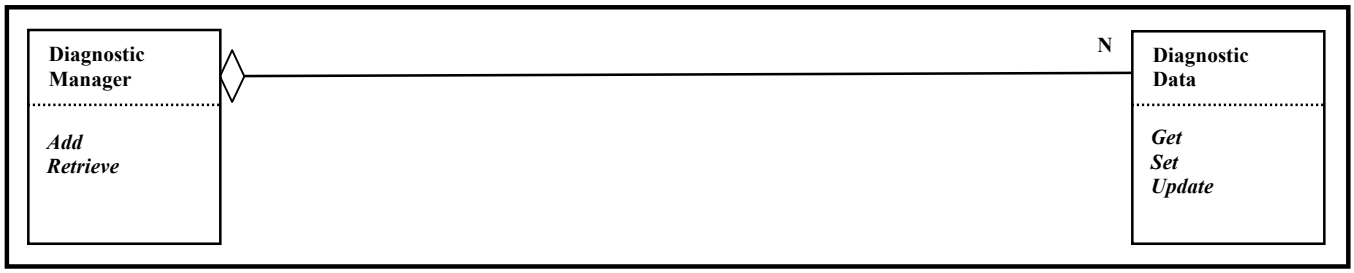

Figure 4 Diagnostic Manager and Diagnostic Data Abstractions

7.3 Diagnostic Data Abstraction

Diagnostic Data is a data abstraction, and has the responsibility to encapsulate information specific to one system diagnostic. To fulfill this responsibility, this abstraction presents an interface of Get, Set and Update contracts, as shown in Figure 4. The copy constructor, assignment operator, or Set contract can be used by Diagnostic Manager to initialize the abstraction. The Get contract returns data encapsulated by the abstraction. The Update contract updates the occurrence counter of an initialized abstraction.

7.4 Termination Manager Abstraction

Termination Manager is a control abstraction, and has the responsibility to store and retrieve the system termination. To fulfill this responsibility, this abstraction collaborates with Termination Data, and presents an interface of Add and Retrieve contracts, as shown in Figure 5. The Add contract allows System Reporter to add termination information to the report, and the Retrieve contract allows System Reporter to retrieve the added termination information. Termination Manager has the same "no memory creation" requirements as System Reporter, and therefore must aggregate one Termination Data

abstraction (since termination is binary – either normal or abnormal), and keep it persistent during its lifetime. This abstraction needs to be persistent to work correctly.

7.5 Termination Data Abstraction

Termination Data is a data abstraction, and has the responsibility to encapsulate information specific to the system termination status. To fulfill this responsibility, this abstraction presents an interface of Get and Set contracts, as shown in Figure 5. The copy constructor, assignment operator, or Set contract can be used by Termination Manager to initialize the abstraction. The Get contract returns data encapsulated by the abstraction.

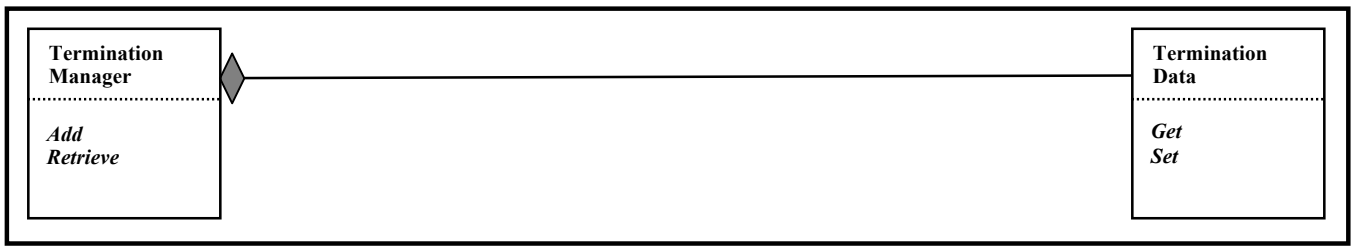

Figure 5 Termination Manager and Termination Data Abstractions

8 Service Package

The Service package has the responsibility to encapsulate all potentially system-wide service abstractions necessary to fulfill the system requirements. If a service abstraction is specific to one package, then it belongs in that package, otherwise it belongs in this package. As discussed in Section 4.1, this package is the second lowest package in the system, and can depend on the Diagnostics package. Figure 6 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

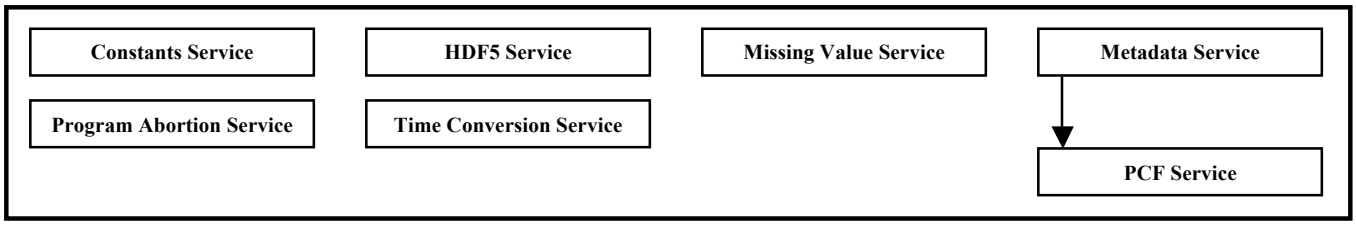

Figure 6 Service Package Hierarchy

8.1 Constants Service Abstraction

Constants Service is a control abstraction, and has the responsibility to provide a single access point to instrument-specific or useful processing constants. To fulfill this responsibility, this abstraction presents an interface of one size constant, two useful math constants, and one contract to test the validity of the size index, as show in Figure 7. This abstraction is not meant to be instantiated, but instead provide non-dynamic data into the global space. Persistency is not an issue.

Figure 7 Constants Service Abstraction

8.2 HDF5 Service Abstraction

HDF5 Service is a control abstraction, and has the responsibility to provide simple and coherent access to HDF5 services provided via the SDP Toolkit. To fulfill this responsibility, this abstraction collaborates with the SDP Toolkit, and presents an interface of many file and field access contracts, as shown in Figure 8. The CreateFile contract creates a new HDF5 file, and the OpenFile contract opens an existing HDF5 file. The CloseFile contract closes an HDF5 file. The CreateSwath contract creates a new swath within the newly created HDF5 file, and the OpenSwath contract opens an existing swath. The CloseSwath contract closes access to a swath. The DefineDimension contract defines new dimensions within a newly created HDF5 file, the DefineField contracts provide multiple ways to define data and geolocation fields within a newly created HDF5 file, and DefineCompression defines the compression characteristics of a newly defined data or geolocation field. The GetDimensionSize contract returns a defined dimension size, and the GetFieldFillValue contract returns the fill value of an already defined field. The WriteField contract writes data to a field, and the WriteAttribute contract writes a file-level attribute. The ReadField contract reads a field. This abstraction need not be persistent to work correctly, though this abstraction does return data that needs to stay persistent to work correctly. Therefore, the abstraction that uses this abstraction must either aggregate the returned data, or begin and end service access within a persistent scope.

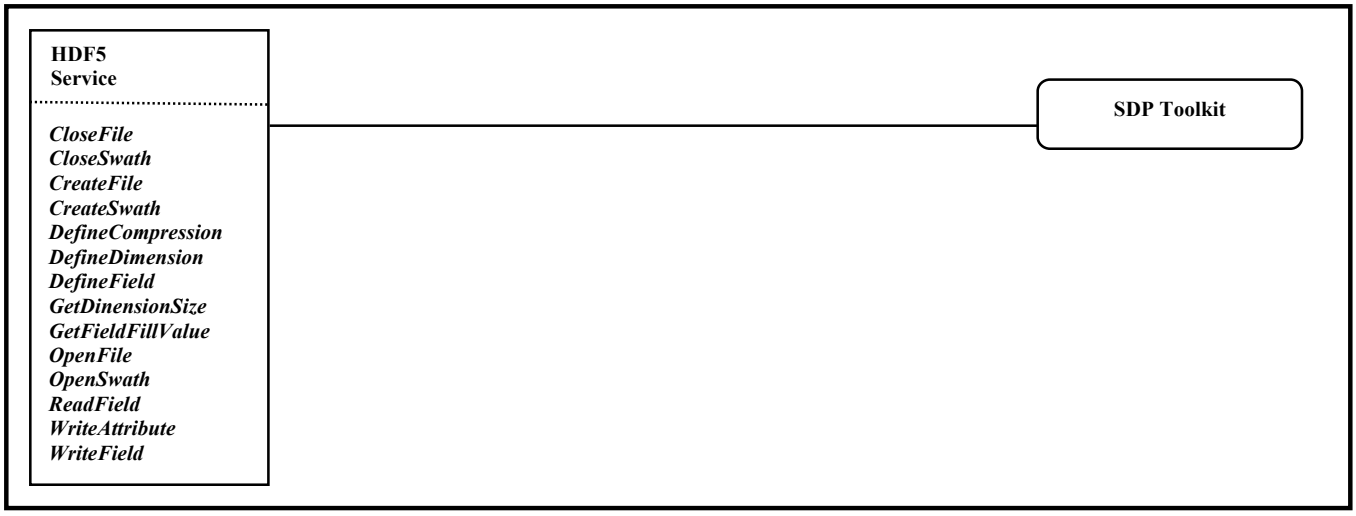

Figure 8 HDF5 Service Abstraction

8.3 Missing Value Service Abstraction

Missing Value Service is a control abstraction, and has the responsibility to provide a single access point for missing value representation and comparison checking. To fulfill this responsibility, this abstraction presents an interface of missing value

retrieval contracts and missing value comparison contracts, as shown in Figure 9. The Get*MissingValue contracts all return the primitive-specific representation of system-wide missing value, and the IsMissingValue contracts tests whether a value is the missing value. This abstraction need not be persistent to work correctly.

Figure 9 Missing Value Service Abstraction

8.4 Program Abortion Service Abstraction

Program Abortion Service is a control abstraction, and has the responsibility to provide a consistent means to abort the program. To fulfill this responsibility, this abstraction collaborates with System Reporter, and presents an interface of Abort contracts, as shown in Figure 10. These contracts add an abnormal termination notice to the Report Generator abstraction, and then abort the system with a failure indication. This abstraction need not be persistent to work correctly. This abstraction has not been shown in any collaboration figures, but of course is available to any abstraction that needs this access.

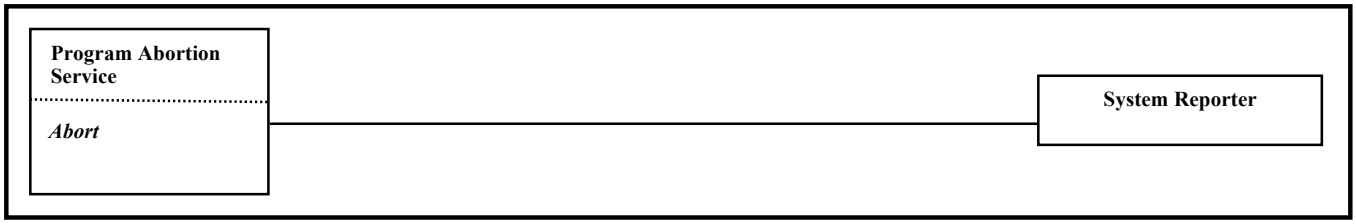

Figure 10 Program Abortion Service Abstraction

8.5 Time Conversion Service Abstraction

Time Conversion Service is a control abstraction, and has the responsibility to provide simple time format conversion service. To fulfill this responsibility, this abstraction collaborates with the SDP Toolkit, and presents an interface of conversion contracts, as shown in Figure 11. Each contract does as its name specifies, converting a value from one time format to another. This abstraction need not be persistent to work correctly.

Figure 11 Time Conversion Service Abstraction

8.6 PCF Service Abstraction

PCF Service is a control abstraction, and has the responsibility to provide simple and coherent access to process control file (PCF) access services provided via the SDP Toolkit. To fulfill this responsibility, this abstraction collaborates with the SDP Toolkit, and presents an interface of GetFilename and GetParameter contracts, as shown in Figure 12. The GetFilename contracts provide multiple ways to retrieve the name of a file listed in the PCF. The GetParameter contract provides a means of retrieving an input parameter listed in the PCF. This abstraction need not be persistent to work correctly.

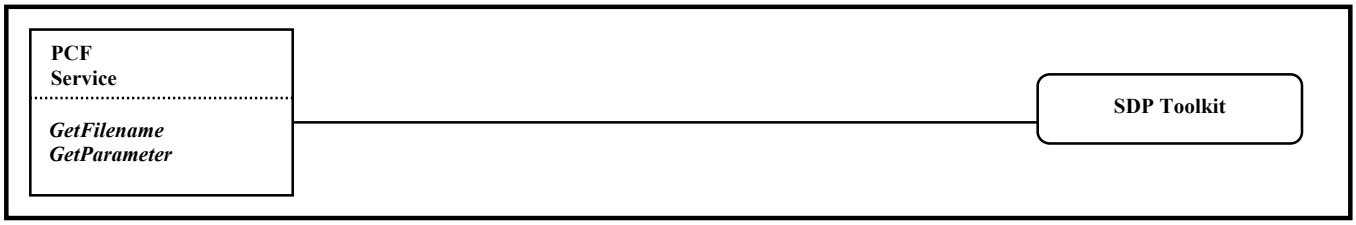

Figure 12 PCF Service Abstraction

8.7 Metadata Service Abstraction

Metadata Service is a control abstraction, and has the responsibility to provide simple and coherent access to ECS metadata services provided via the SDP Toolkit. To fulfill this responsibility, this abstraction collaborates with PCF Service and the SDP Toolkit, and presents an interface of Set and Write contracts, as shown in Figure 13. The Set contracts allow setting of ECS metadata parameters, and the Write contract writes the ECS metadata to the file with which it is connected. ECS service access needs to be initialized and terminated, but that should happen upon abstraction instantiation and destruction, respectively. This abstraction needs to be persistent to work correctly.

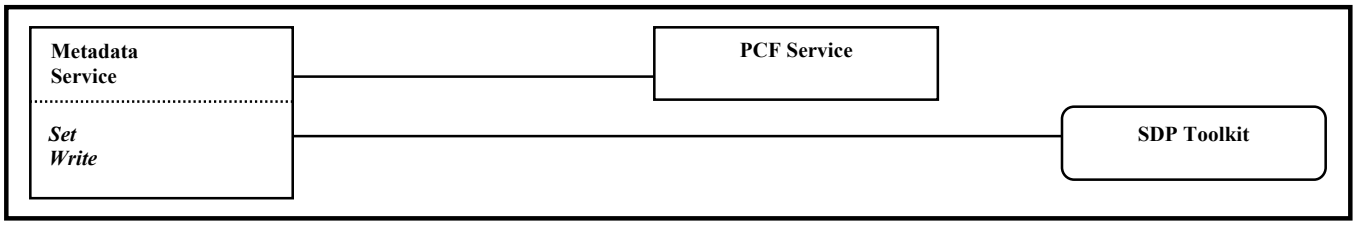

Figure 13 Metadata Service Abstraction

9 File Package

The File package has the responsibility to encapsulate all abstractions necessary to provide the system a consistent means to interact with data files. As discussed in Section 4.1, this package is the third lowest package in the system, and can depend on the Diagnostics and Service packages. Figure 14 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

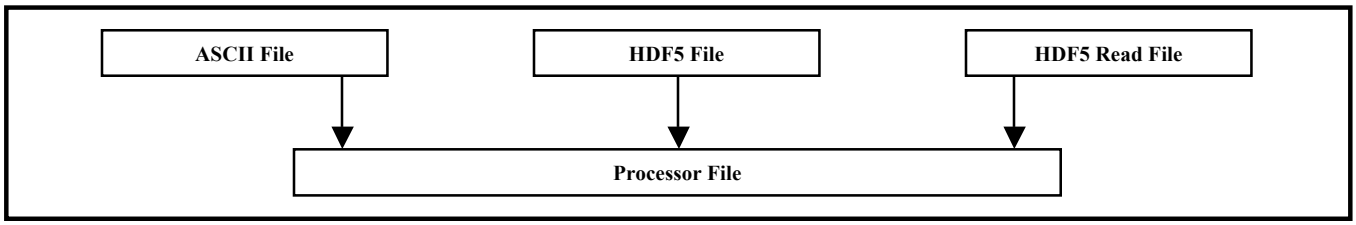

Figure 14 File Package Hierarchy

9.1 Processor File Abstraction

Processor File is a control abstraction, and has the responsibility to manage low-level access to all files within the system, but only as a pure virtual abstraction intended to be used as a base for all file abstractions in the system. To fulfill this responsibility, this abstraction collaborates with PCF Service, and presents an interface of IsValid, GetLogical and GetName contracts, as shown in Figure 15. The IsValid contract determines if the file is valid, he GetLogical contract returns the logical ID of the file, and the GetName contract returns the name of the file. This abstraction does not handle opening and closing, as those are specific to a type of file. For this abstraction to be used in a realistic manner, the abstraction that is derived from it needs to be persistent, unless this abstraction is used solely for the purpose of testing the validity of a file.

Figure 15 Processor File Abstraction

9.2 ASCII File Abstraction

ASCII File is a control abstraction, and has the responsibility to manage access to a read-only, sequential-access ASCII file, but only as a pure virtual abstraction intended to be used as a base for file abstractions that model ASCII files. To fulfill this responsibility, this abstraction collaborates with Processor File, and presents an interface of Open, Close, Read and GetToken contracts, as shown in Figure 16. The Open contract opens the existing file for reading, the Close contract closes the opened file, and the Read contract reads the next line in the opened file. The GetToken contracts are provided to help parse a file line in the usual way: to extract a particular token from the line. For this abstraction to work correctly across multiple Read calls, the abstraction that is derived from this abstraction needs to be persistent.

Figure 16 ASCII File Abstraction

9.3 HDF5 File Abstraction

HDF5 File is a control abstraction, and has the responsibility to manage access to a write-only, HDF5-formatted file, but only as a pure virtual abstraction intended to be used as a base for file abstractions that model HDF5 files. To fulfill this responsibility, this abstraction collaborates with Processor File and HDF5 Service, and presents an interface of creating, closing, defining and writing contracts, as shown in Figure 17. The Create contract creates a new HDF5 file, and the Close contract closes a newly created HDF5 file. The DefineDimension contract is used to define dimensions of a newly created HDF5 file, and the DefineField contracts are used to define fields in a newly created HDF5 file. The WriteField contract writes a field's data to the newly created HDF5 file, and the WriteAttribute writes a file-level attribute to a newly created HDF5 file. For this abstraction to work correctly across its multiple calls, the abstraction that is derived from this abstraction needs to be persistent.

Figure 17 HDF5 File Abstraction

9.4 HDF5 Read File Abstraction

HDF5 Read File is a control abstraction, and has the responsibility to manage access to a read-only, HDF5-formatted file, but only as a pure virtual abstraction intended to be used as a base for file abstractions that model HDF5 files. To fulfill this responsibility, this abstraction collaborates with Processor File and HDF5 Service, and presents an interface of opening, closing and reading contracts, as shown in Figure 18. The Open contract opens an existing HDF5 file, and the Close contract closes an opened HDF5 file. The GetDimensionSize contract retrieves the size of a dimension in an opened HDF5 file. The ReadField contracts read fields of specified data types. For this abstraction to work correctly across its multiple calls, the abstraction that is derived from this abstraction needs to be persistent.

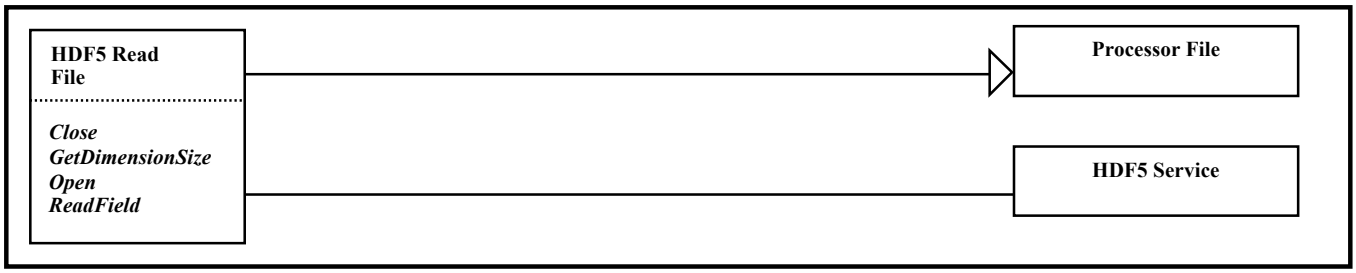

Figure 18 HDF5 Read File Abstraction

10 Math Package

The Math package has the responsibility to encapsulate all potentially system-wide mathematical abstractions necessary to fulfill the system requirements. If a mathematics abstraction is specific to one package, then it belongs in that package, otherwise it belongs in this package. As discussed in Section 4.1, this package is the third lowest package in the system, and can depend on the Diagnostics and Service packages. Figure 19 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

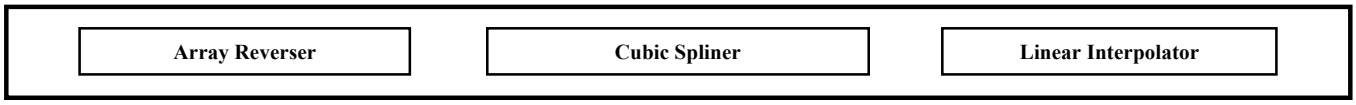

Figure 19 Math Package Hierarchy

10.1 Array Reverser Abstraction

Array Reverser is a process abstraction, and has the responsibility to perform in-place array reversal. To fulfill this responsibility, this abstraction presents an interface of Reverse contracts, as shown in Figure 20. These contracts allow the system a means to perform in-place array reversal for an array of the primitives used in the system. This abstraction is not meant to be instantiated, but instead provide non-dynamic methodology into the global space. Persistency is not an issue.

Figure 20 Array Reverser Abstraction

10.2 Cubic Spliner Abstraction

Cubic Spliner is a process abstraction, and has the responsibility to perform cubic splining. To fulfill this responsibility, this abstraction presents an interface of Spline contracts, as shown in Figure 21. These contracts allow the system a means to perform cubic splining for all combinations of ordinate and abscissa arrays of the primitives used in the system. This abstraction is not meant to be instantiated, but instead provide non-dynamic methodology into the global space. Persistency is not an issue.

Figure 21 Cubic Spliner Abstraction

10.3 Linear Interpolator Abstraction

Linear Interpolator is a process abstraction, and has the responsibility to perform linear interpolation. To fulfill this responsibility, this abstraction presents an interface of Interpolate contracts, as shown in Figure 22. These contracts allow the system a means to perform linear interpolation of data points and arrays of the primitives used in the system. This abstraction is not meant to be instantiated, but instead provide non-dynamic methodology into the global space. Persistency is not an issue.

Figure 22 Linear Interpolator Abstraction

11 Extractor Package

The Extractor package has the responsibility to encapsulate all abstractions necessary to provide the system a means to extract scans from the HIRDLS L1C file. As shown in Figure 1, this package is used by the Processor, Filterer and Transformer packages, and has access to the File, Math, Service and Diagnostics packages. Figure 23 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

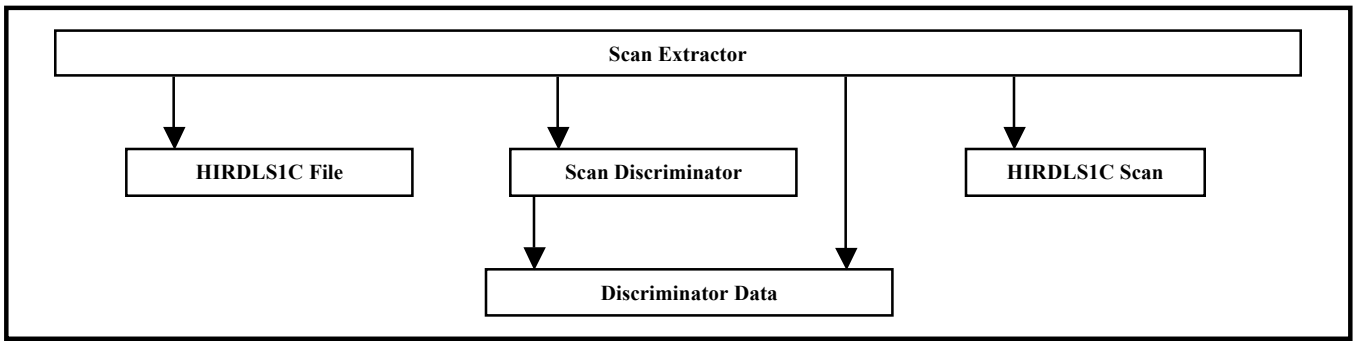

Figure 23 Extractor Package Hierarchy

11.1 Scan Extractor Abstraction

Scan Extractor is a process abstraction, and has the responsibility to extract scans from the input HIRDLS1C file, and return those scans to the system. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C File, Scan Discriminator, Discrimination Data and HIRDLS1C Scan, and presents an interface of one Extract contract, as shown in Figure 24. The Extract contract is to return the *next* scan in the file. It is specified here that Scan Extractor, in the interest of efficiency, read in from HIRDLS1C File all the data from all the fields necessary to fill a HIRDLS1C Scan. Also specified here is that Scan Extractor must manage the scan sequence, and return a Boolean when it can not return the next scan in sequence, whether from error or from a lack of further scans. Due to its responsibility, this abstraction needs to be persistent to work correctly.

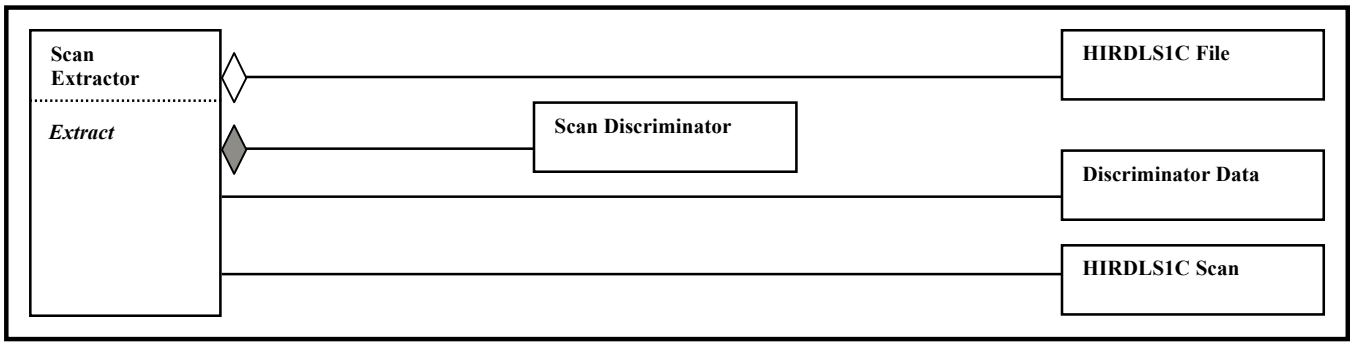

Figure 24 Scan Extractor Abstraction

11.2 HIRDLS1C File Abstraction

HIRDLS1C File is a control abstraction, and has the responsibility to manage all access to a HIRDLS1C file. To fulfill this responsibility, this abstraction collaborates with HDF5 Read File, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 25. The GetFile contract returns an instance of a HIRDLS1C File abstraction. At this time, this instance is not considered a Singleton[3], but the contract for returning an instance of this abstraction is left to look like a Singleton, in the case that that is what is implemented, or changed to during testing and/or maintenance. The Read contract returns the contents of a given HIRDLS1C field. This contract is to return a Boolean status to the caller, indicating if it was able to retrieve the field or not. This abstraction needs to be kept persistent to work correctly, except in the case that the instantiated file is opened, read completely, and closed, in one structural calling sequence.

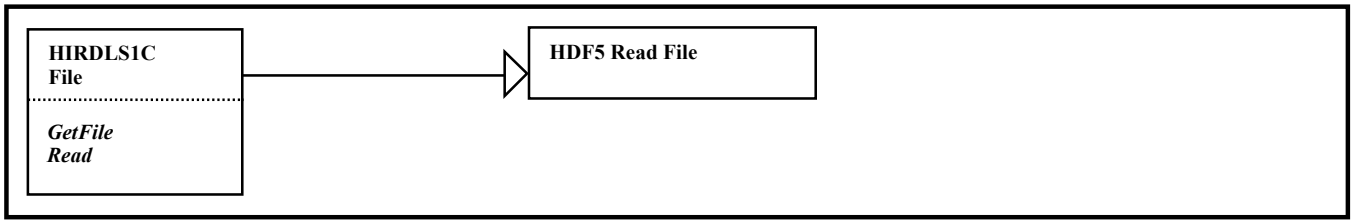

Figure 25 HIRDLS1C File Abstraction

[3] See Section 7.1

11.3 Scan Discriminator Abstraction

Scan Discriminator is a process abstraction, and has the responsibility to discriminate the scans from a HIRDLS1C file. To fulfill this responsibility, this abstraction collaborates with Discriminator Data, and presents an interface of one Discriminate contract and one GetNext contract, as shown in Figure 26. The Discriminate contract initializes the abstraction. The GetNext contract returns a Discriminator Data abstraction that represents the next scan, and is to return a Boolean status to the caller, indicating if there was another scan to return. This abstraction needs to be kept persistent to work correctly.

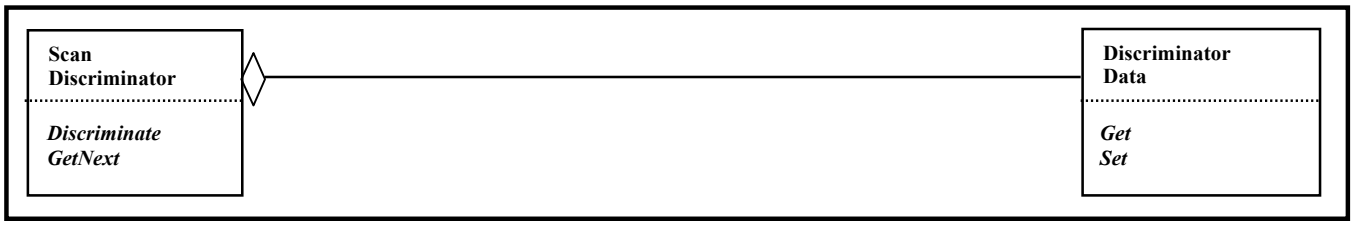

Figure 26 Scan Discriminator and Discriminator Data Abstractions

11.4 Discriminator Data Abstraction

Discriminator Data is a data abstraction, and has the responsibility to encapsulate information specific to scan discrimination. To fulfill this responsibility, this abstraction presents an interface of Get and Set contracts, as shown in Figure 26. The copy constructor, assignment operator, or Set contract can be used by Scan Discriminator to initialize the abstraction. The Get contract returns data encapsulated by the abstraction.

11.5 HIRDLS1C Scan Abstraction

HIRDLS1C Scan is a data abstraction, and has the responsibility to manage access to a HIRDLS1C scan's data. To fulfill this responsibility, this abstraction presents an interface of constants describing data sizes, contracts to retrieve specific data fields, one SetRadiances contract, one Set contract, and various Get contracts, as shown in Figure 27. The Set contract is used to initialize the abstraction. The Get contracts are used to retrieve various "sets" of scan data (rather than retrieve all fields when only a portion is needed). The SetRadiances contract is necessary for use by Radiance Filterer, to reset the radiance data after filtering. This abstraction needs to be persistent to work correctly, unless the Set contract, change contracts and Get contract are all called during the same procedural calling sequence.

Figure 27 HIRDLS1C Scan Abstraction

12 Filterer Package

The Filterer package has the responsibility to encapsulate all abstractions necessary to provide the system a means to filter the radiances in a HIRDLS1C scan. As shown in Figure 1, this package is used by the Processor package, and has access to the Extractor, File, Math, Service and Diagnostics packages. Figure 28 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

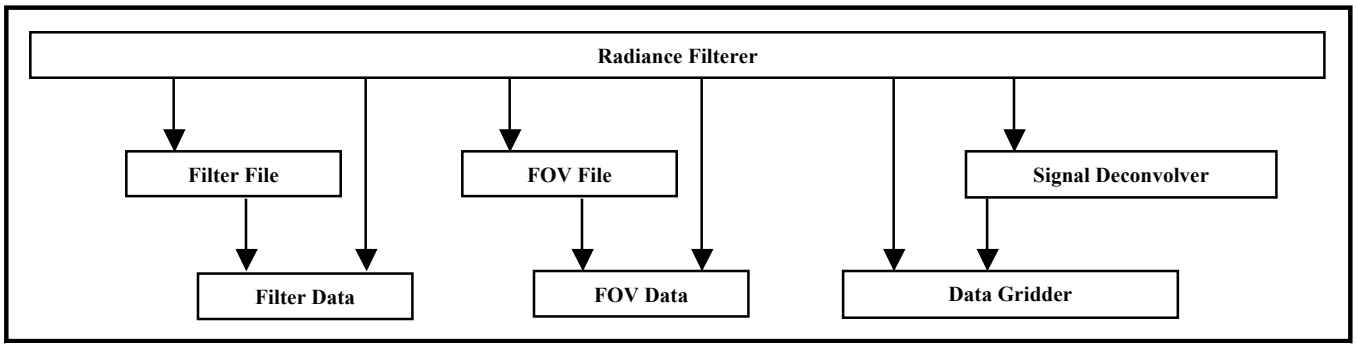

Figure 28 Filterer Package Hierarchy

12.1 Radiance Filterer Abstraction

Radiance Filterer is a process abstraction, and has the responsibility to filter radiances in a HIRDLS1C scan. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Scan, Filter File, Filter Data, FOV File, FOV Data, Data Gridder and Signal Deconvolver, and presents an interface of one Filter contract, as shown in Figure 29. The Filter contract extracts, filters, and resets the radiances in a HIRDLS1C Scan. Due to initializing itself with data read in from two files, this abstraction needs to be persistent to work efficiently.

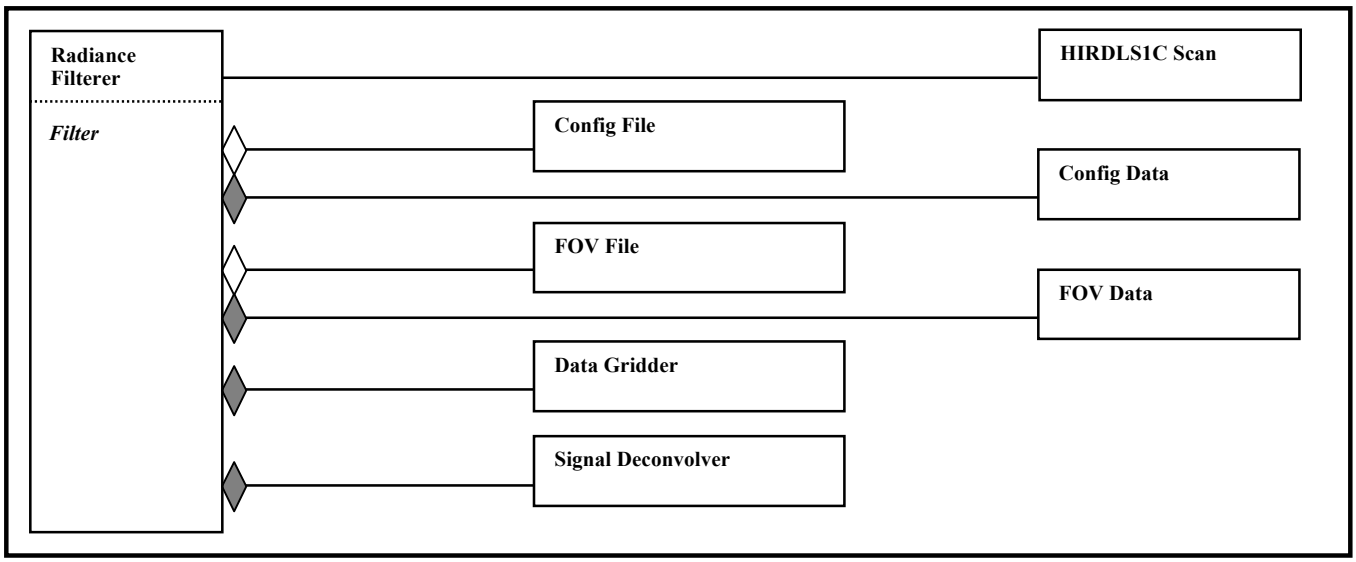

Figure 29 Radiance Filterer Abstraction

12.2 Config File Abstraction

Config File is a control abstraction, and has the responsibility to manage all access to a filter configuration file. To fulfill this responsibility, this abstraction collaborates with ASCII File and Config Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 30. The GetFile contract returns an instance of a Config File abstraction. At this time, this instance is not considered a Singleton[4], as configuration files are read-only files, but the contract for returning an instance of this abstraction is left to look like a Singleton, in the case that that is what is implemented, or changed to during testing and/or maintenance. The Read contract reads the data from the file and stores it into a Config Data instance. This contract is to return a Boolean status to the caller, indicating if it was able to read the file or not. This abstraction needs to be kept persistent to work correctly, except in the case that the instantiated file is opened, read completely, and closed, in one structural calling sequence.

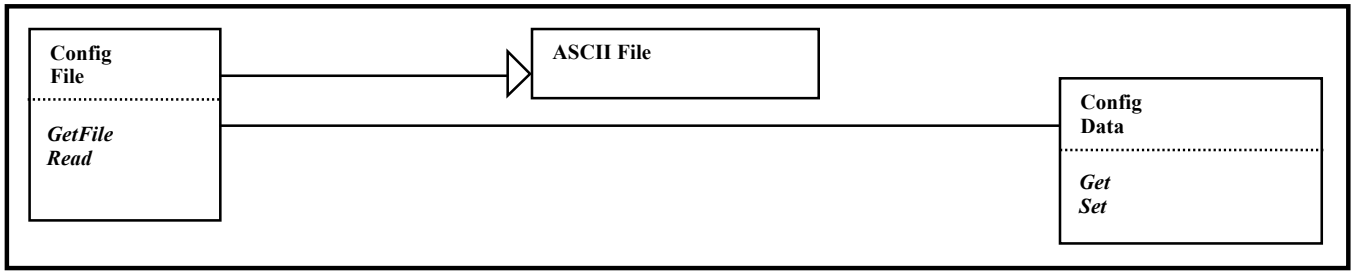

Figure 30 Config File and Config Data Abstractions

12.3 Config Data Abstraction

Config Data is a data abstraction, and has the responsibility to manage access to the data read from a Config File. To fulfill this responsibility, this abstraction presents an interface of one Set contract and one Get contract, as shown in Figure 30. The copy constructor, assignment operator, or Set contract can be used by Config File to initialize the abstraction. The Get contract allows retrieval of all the data.

12.4 FOV File Abstraction

FOV File is a control abstraction, and has the responsibility to manage all access to a FOV response function file. To fulfill this responsibility, this abstraction collaborates with HDF5 Read File and FOV Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 31. The GetFile contract returns an instance of an FOV File abstraction. At this time, this instance is not considered a Singleton[5], as the FOV file is a read-only file, but the contract for returning an instance of this abstraction is left to look like a Singleton, in the case that that is what is implemented, or changed to during testing and/or maintenance. The Read contract reads the data from the file and stores it into an FOV Data instance. This contract is to return a Boolean status to the caller, indicating if it was able to read the file or not. This abstraction needs to be kept persistent to work correctly, except in the case that the instantiated file is opened, read completely, and closed, in one structural calling sequence.

12.5 FOV Data Abstraction

FOV Data is a data abstraction, and has the responsibility to manage access to the data read from an FOV File. To fulfill this responsibility, this abstraction presents an interface of one Set contract and one Get contract, as shown in Figure 31. The

[4] See Section 7.1
[5] See Section 7.1

copy constructor, assignment operator, or Set contract can be used by FOV File to initialize the abstraction. The Get contract allows retrieval of all the data.

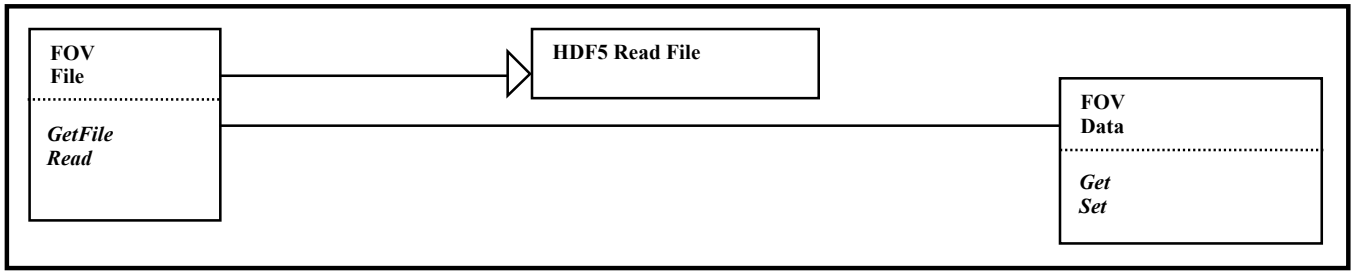

Figure 31 FOV File and FOV Data Abstractions

12.6 Data Gridder Abstraction

Data Gridder is a process abstraction, and has the responsibility to grid (and ungrid) data from a HIRDLS1C scan. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Scan, and presents an interface of a size constant, one Grid contract and one Ungrid contract, as shown in Figure 32. The Grid contract splines data from a scan-specific grid onto a new static grid. The Ungrid contract does the reverse. This abstraction need not be persistent to work correctly.

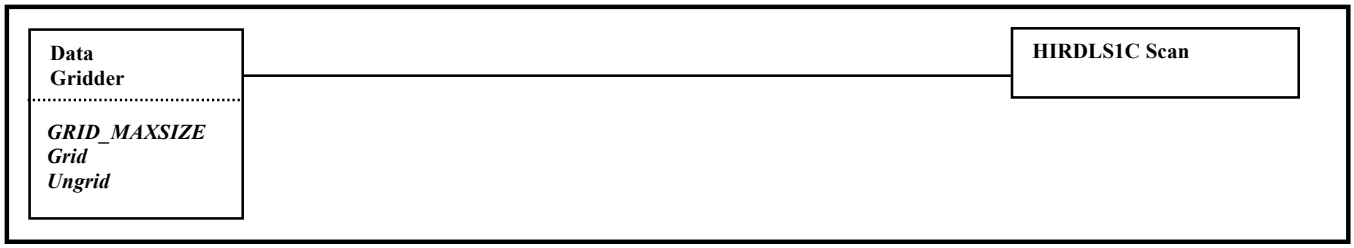

Figure 32 Data Gridder Abstraction

12.7 Signal Deconvolver Abstraction

Signal Deconvolver is a process abstraction, and has the responsibility to deconvolve the radiances in a HIRDLS1C scan. To fulfill this responsibility, this abstraction collaborates with Data Gridder, and presents an interface of one Initialize contract and one Deconvolve contract, as shown in Figure 33. The Initialize contract initializes the abstraction with data not accessible to the abstraction upon instantiation. The Deconvolve contract deconvolves the radiances. This abstraction needs to be persistent to work correctly.

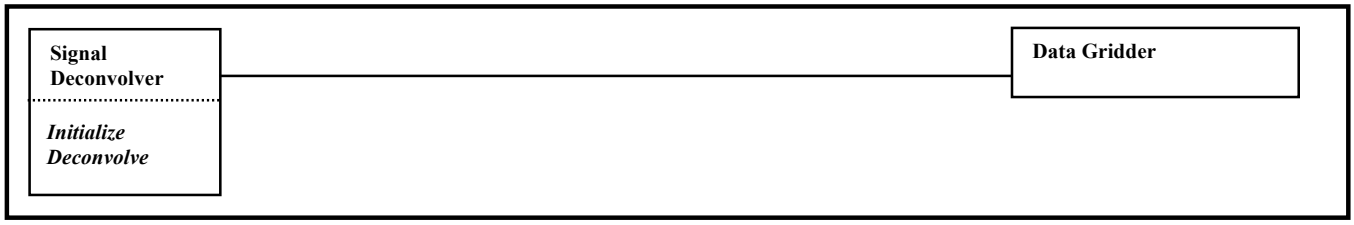

Figure 33 Signal Deconvolver Abstraction

13 Transformer Package

The Transformer package has the responsibility to encapsulate all abstractions necessary to provide the system a means to transform a HIRDLS1C scan into a HIRDLS1R Scan. As shown in Figure 1, this package is used by the Processor, Adjuster and Writer packages, and has access to the Extractor, File, Math, Service and Diagnostics packages. Figure 34 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

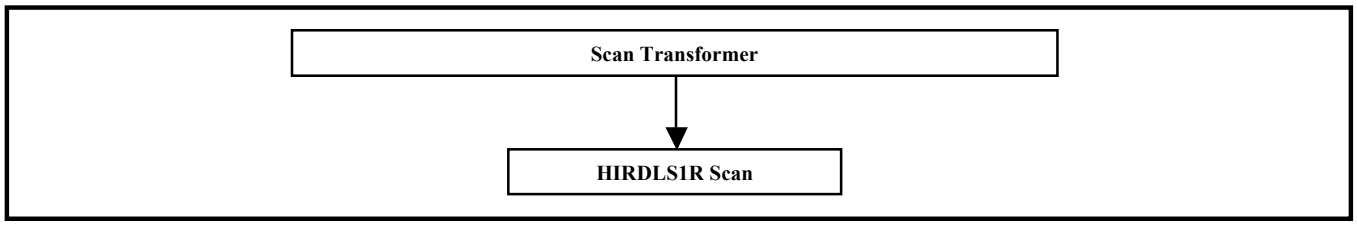

Figure 34 Transformer Package Hierarchy

13.1 Scan Transformer Abstraction

Scan Transformer is a process abstraction, and has the responsibility to transform a HIRDLS1C Scan into a HIRDLS1R Scan. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Scan and HIRDLS1R Scan, and presents an interface of one Transform contract, as shown in Figure 35. The Transform contract retrieves the appropriate data from the HIRDLS1C Scan, transforms it into HIRDLS1R form, and then fills a HIRDLS1R Scan with that data. This abstraction need not be persistent to work correctly.

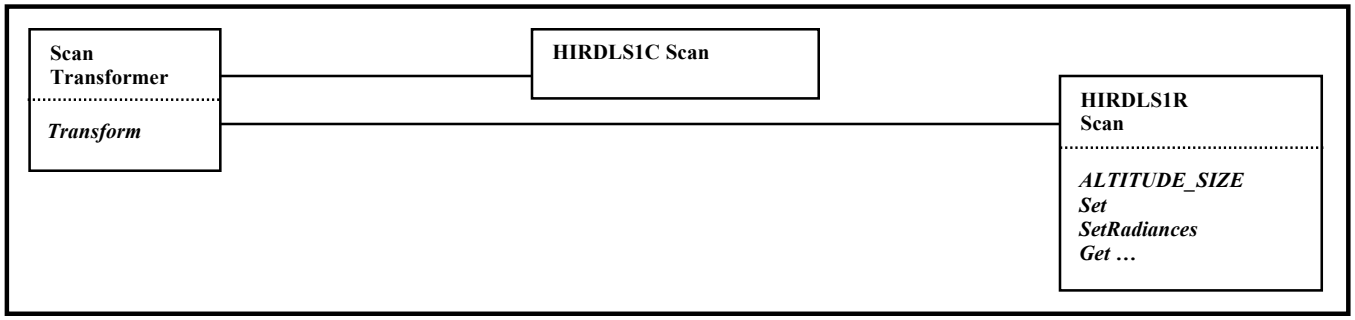

Figure 35 Scan Transformer and HIRDLS1R Scan Abstractions

13.2 HIRDLS1R Scan Abstraction

HIRDLS1R Scan is a data abstraction, and has the responsibility to manage access to a HIRDLS1R scan's data. To fulfill this responsibility, this abstraction presents an interface of constants describing data sizes, one SetRadiances contract, one Set contract, and various Get contracts, as shown in Figure 35. The Set contract is used to initialize the abstraction. The Get contracts are used to retrieve various data fields (see Appendix A for a detailed listing of the Get contracts). The SetRadiances contract is necessary for use by Radiance Adjuster, to reset the radiance data after adjustmet.

14 Adjuster Package

The Adjuster package has the responsibility to encapsulate all abstractions necessary to provide the system a means to adjust the radiances in a HIRDLS1R scan. As shown in Figure 1, this package is used by the Processor package, and has access to the Transformer, File, Math, Service and Diagnostics packages. Figure 36 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

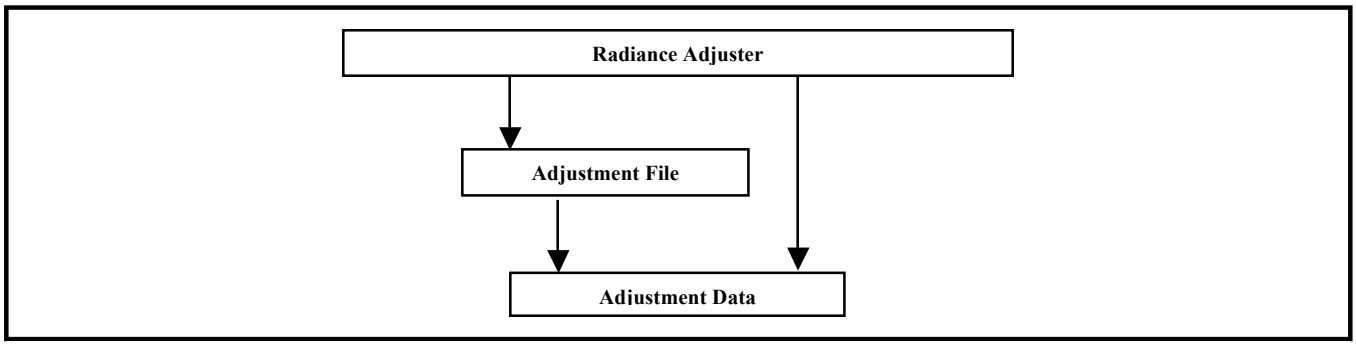

Figure 36 Adjuster Package Hierarchy

14.1 Radiance Adjuster Abstraction

Radiance Adjuster is a process abstraction, and has the responsibility to adjust radiances in a HIRDLS1R scan. To fulfill this responsibility, this abstraction collaborates with HIRDLS1R Scan, Adjustment File and Adjustment Data, and presents an interface of one Adjust contract, as shown in Figure 37. The Adjust contract extracts, adjusts, and resets the radiances in a HIRDLS1R Scan. Due to initializing itself with data read in from a file, this abstraction needs to be persistent to work efficiently.

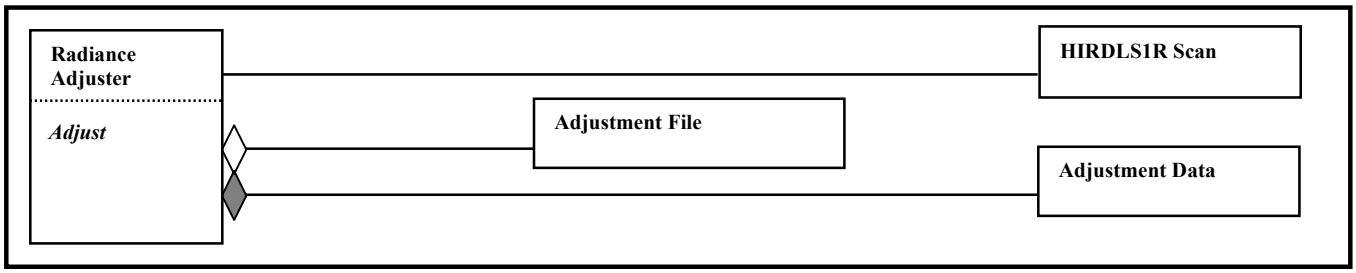

Figure 37 Radiance Adjuster Abstraction

14.2 Adjustment File Abstraction

Adjustment File is a control abstraction, and has the responsibility to manage all access to a radiance adjustment file. To fulfill this responsibility, this abstraction collaborates with ASCII File and Adjustment Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 38. The GetFile contract returns an instance of an Adjustment File abstraction. At this time, this instance is not considered a Singleton[6], as adjustment files are read-only files, but the contract for returning an instance of this abstraction is left to look like a Singleton, in the case that that is what is implemented, or changed to during testing and/or maintenance. The Read contract reads the data from the file and stores it into an Adjustment Data instance. This contract is to return a Boolean status to the caller, indicating if it was able to read the

[6] See Section 7.1

file or not. This abstraction needs to be kept persistent to work correctly, except in the case that the instantiated file is opened, read completely, and closed, in one structural calling sequence.

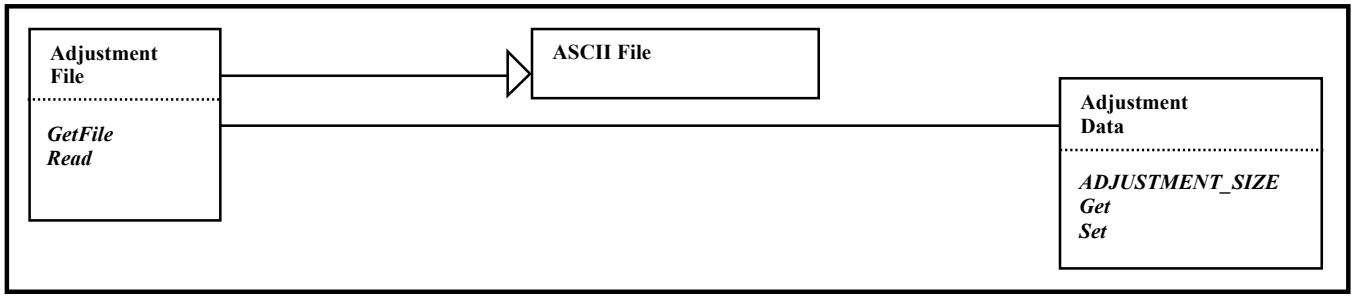

Figure 38 Adjustment File and Adjustment Data Abstractions

14.3 Adjustment Data Abstraction

Adjustment Data is a data abstraction, and has the responsibility to manage access to the data read from an Adjustment File. To fulfill this responsibility, this abstraction presents an interface of a size constant, one Set contract and one Get contract, as shown in Figure 38. The copy constructor, assignment operator, or Set contract can be used by Adjustment File to initialize the abstraction. The Get contract allows retrieval of all the data.

15 Writer Package

The Writer package has the responsibility to encapsulate all abstractions necessary to provide the system a means to write HIRDLS1R scans to a HIRDLS1R file. As shown in Figure 1, this package is used by the Processor package, and has access to the Transformer, File, Math, Service and Diagnostics packages. Figure 39 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

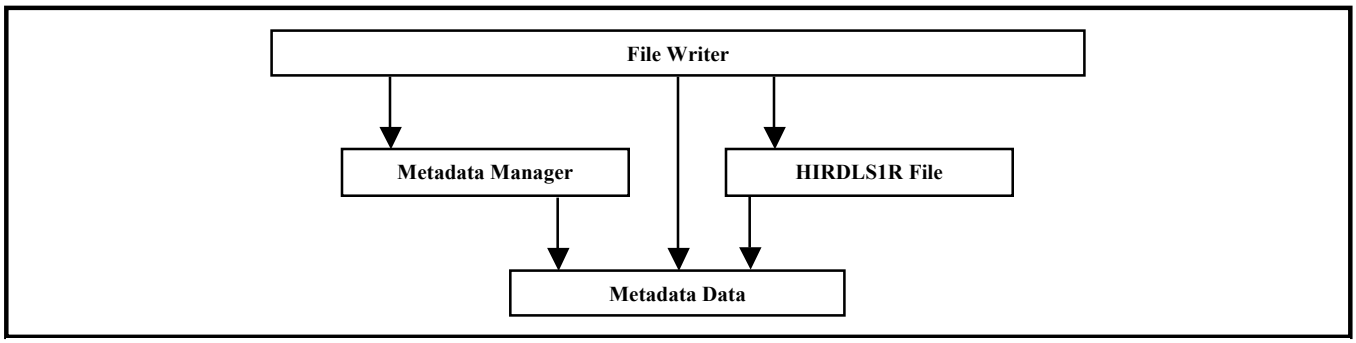

Figure 39 Writer Package Hierarchy

15.1 File Writer Abstraction

File Writer is a process abstraction, and has the responsibility to write scan data to a HIRDLS1R file. To fulfill this responsibility, this abstraction collaborates with HIRDLS1R Scan, Metadata Manager, Metadata Data and HIRDLS1R File, and presents an interface of one Write contract, as shown in Figure 40. The Write contract receives a HIRDLS1R scan and writes the scan to a HIRDLS1R file, or buffers the scan to write many scans at once. Those details are left for implementation. This abstraction must also manage the metadata that needs to be written to the HIRDLS1R file. Some of

the metadata cannot be determined until all of the scans have been written, and that status will only be known upon destruction. Therefore, this abstraction's destructor must acquire the metadata from the manager, and pass that to the HIRDLS1R file for writing. This abstraction needs to be persistent to work correctly.

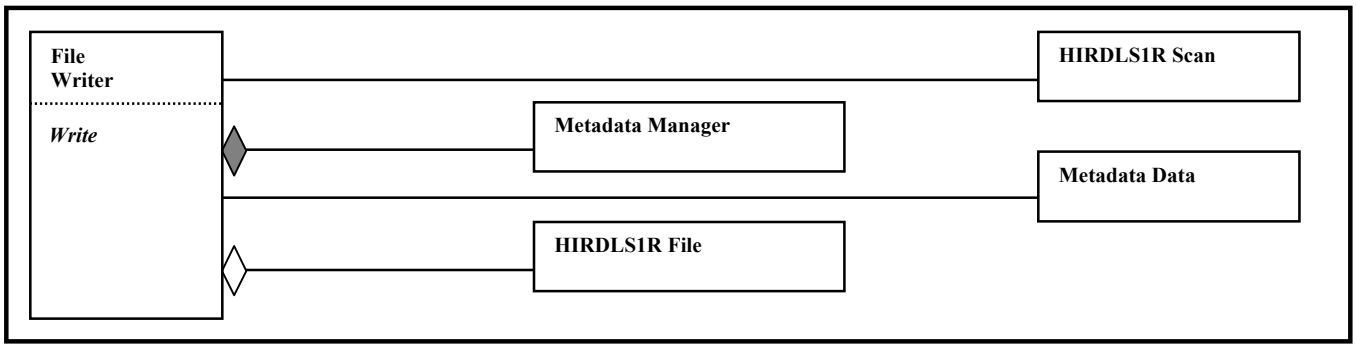

Figure 40 File Writer Abstraction

15.2 Metadata Manager Abstraction

Metadata Manager is a control abstraction, and has the responsibility to store and retrieve the HIRDLS1R file metadata. To fulfill this responsibility, this abstraction collaborates with HIRDLS1R Scan and Metadata Data, and presents an interface of Add and Retrieve contracts, as shown in Figure 41. The Add contract takes a HIRDLS1R Scan, extracts what it needs, and adds/overwrites that information to Metadata Data. The Retrieve contract the information aggregated in Metadata Data. This abstraction needs to be persistent to work correctly.

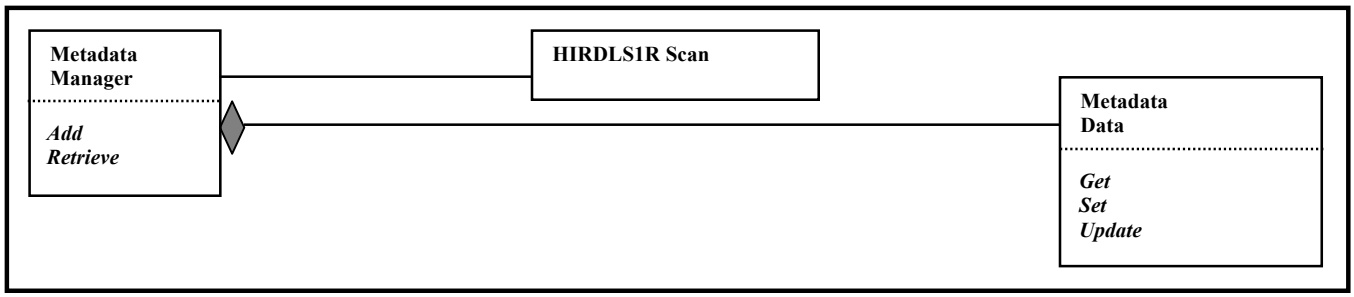

Figure 41 Metadata Manager and Metadata Data Abstractions

15.3 Metadata Data Abstraction

Metadata Data is a data abstraction, and has the responsibility to encapsulate information specific to HIRDLS1R file metadata. To fulfill this responsibility, this abstraction presents an interface of Get, Set and Update contracts, as shown in Figure 41. The copy constructor, assignment operator, or Set contract can be used by Metadata Manager to initialize the abstraction. The Get contract returns data encapsulated by the abstraction. The Update contract updates the appropriate data within the abstraction.

15.4 HIRDLS1R File Abstraction

HIRDLS1R File is a control abstraction, and has the responsibility to manage all access to a HIRDLS1R file. To fulfill this responsibility, this abstraction collaborates with HDF5 File, and presents an interface of one GetFile contract, one Write contract and one WriteMetadata, as shown in Figure 42. The GetFile contract returns an instance of a HIRDLS1R File abstraction. This instance is considered a Singleton[7], as there must only be one output file instance. The Write contract writes a HIRDLS1C scan to the file. This contract is to return a Boolean status to the caller, indicating if it was able to write the field or not. The WriteMetadata contract reads the metadata from a Metadata Data abstraction, and then writes that metadata to the file. This abstraction needs to be kept persistent to work correctly.

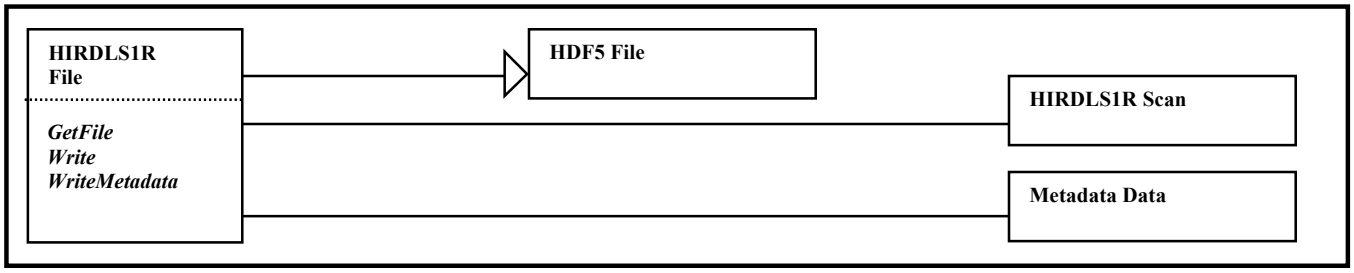

Figure 42 HIRDLS1R File Abstraction

16 Processor Package

The Processor package has the responsibility to encapsulate all abstractions necessary to preprocess L2 data. As shown in Figure 1, this package is the highest-level package in the system, and has access to the Extractor, Filterer, Transformer, Adjuster, Writer, Math, File, Service and Diagnostics packages. The only abstraction in this package is L2 Preprocessor, and it is detailed further in this section.

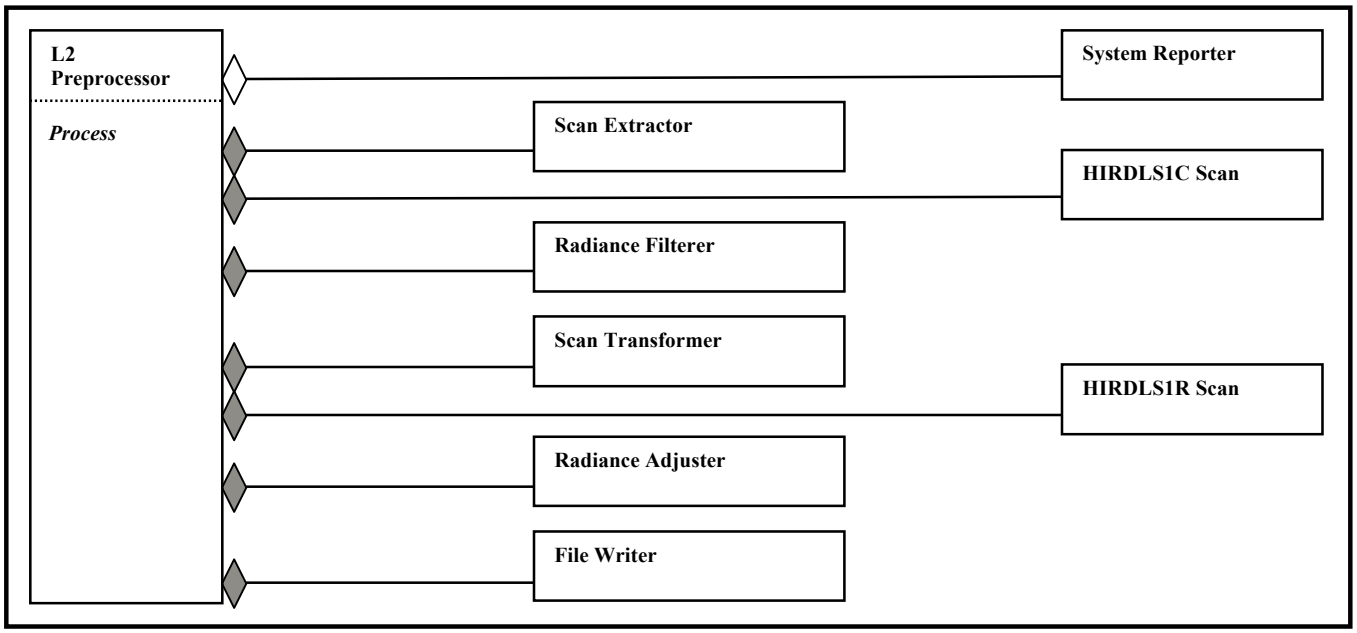

Figure 43 L2 Preprocessor Abstraction

[7] See Section 7.1

16.1 L2 Preprocessor Abstraction

L2 Preprocessor is a process abstraction, and has the responsibility to manage the preprocessing of HIRDLS L2 data. To fulfill this responsibility, this abstraction collaborates with System Reporter, Scan Extractor, HIRDLS1C Scan, Radiance Filterer, Scan Transformer, HIRDLS1R Scan, Radiance Adjuster and File Writer, and presents an interface of one Process contract, as shown in Figure 42. The Process contract first instantiates a System Reporter abstraction, and then calls, in sequence: Scan Extractor, Radiance Filterer, Scan Transformer, Radiance Adjuster and Data Writer, for each scan found in the input HIRDLS1C file. Since this abstraction fulfills its tasks in one call, it does not need to be persistent to work correctly.

Appendix A – Abstraction Interfaces

A.1 System Reporter

```
static system_reporter* GetReporter ()

~system_reporter ()

void Add (enum diagnostic_code code)
void Add (enum termination_code code, const string& message)
```

A.2 Diagnostic Manager

```
diagnostic_manager ()

~diagnostic_manager ()

static int const MAXIMUM_DIAGNOSTICS

void Add (enum diagnostic_code code)
bool Retrieve (enum diagnostic_code code, diagnostic_data& data) const
```

A.3 Diagnostic Data

```
diagnostic_data ()
diagnostic_data (const diagnostic_data& data)
diagnostic_data& operator= (const diagnostic_data& data)

~diagnostic_data ()

enum diagnostic_code {<the list of acceptable diagnostic codes>}

void Get (enum diagnostic_code& code, long& occurrences, string& message) const
void Set (enum diagnostic_code code, long occurrences, const string& message)
void Update ()
```

A.4 Termination Manager

```
termination_manager ()

~termination_manager ()

void Add (enum termination_code code, const string& message)
void Retrieve (termination_data& data) const
```

A.5 Termination Data

```
termination_data ()
termination_data (const termination_data& data)
termination_data& operator= (const termination_data& data)

~termination_data ()

enum termination_code {TERMINATION_NORMAL, TERMINATION_ABNORMAL}

void Get (enum termination_code& code, string& message) const
void Set (enum termination_code code, const string& message)
```

A.6 Constants Service

```
static int const CHANNEL_SIZE
static double const PI
static double const RADIANS_TO_DEGREES

static inline bool IsValidChannelIndex (int index)
```

A.7 HDF5 Service

```
hdf5_service ()

~hdf5_service ()

enum hdf5service_fieldtype {HDF5SERVICE_DATAFIELD, HDF5SERVICE_GEOFIELD}

bool CloseFile (long fileid) const
bool CloseSwath (long swathid) const
bool CreateFile (const string& filename, long& fileid) const
bool CreateSwath (long fileid, const string& swathname, long& swathid) const
bool DefineCompression (long swathid, int level, int count, const int dimensions[]) const
bool DefineDimension (long swathid, const string& name, long value) const
bool DefineField (enum hdf5service_fieldtype type, long fileid, const string& fieldname,
                  const string& dimensionnames, char fillvalue, long& swathid) const
bool DefineField (enum hdf5service_fieldtype type, long fileid, const string& fieldname,
                  const string& dimensionnames, short fillvalue, long& swathid) const
bool DefineField (enum hdf5service_fieldtype type, long fileid, const string& fieldname,
                  const string& dimensionnames, int fillvalue, long& swathid) const
bool DefineField (enum hdf5service_fieldtype type, long fileid, const string& fieldname,
                  const string& dimensionnames, long fillvalue, long& swathid) const
bool DefineField (enum hdf5service_fieldtype type, long fileid, const string& fieldname,
                  const string& dimensionnames, float fillvalue, long& swathid) const
bool DefineField (enum hdf5service_fieldtype type, long fileid, const string& fieldname,
                  const string& dimensionnames, double fillvalue, long& swathid) const
bool GetDimensionSize (long swathid, const string& name, long& size) const
bool GetFieldFillValue (long swathid, const string& name, void* value) const
bool OpenFile (const string& name, long& id) const
bool OpenSwath (long fileid, const string& name, long& id) const
bool ReadField (long swathid, const string& name, int rank, const long starts[],
                const long edges[], void* data) const
bool WriteAttribute (long fileid, const string& name, const string& value) const
bool WriteField (long swathid, const string& fieldname, int dimensioncount,
                 const long starts[], const long edges[], const void* data) const
bool WriteSWAttribute (long swathid, const string& name, int value) const
bool WriteSWAttribute (long swathid, const string& name, float value) const
bool WriteSWAttribute (long swathid, const string& name, double value) const
bool WriteSWAttribute (long swathid, const string& name, const string& value) const
```

A.8 Time Conversion Service

```
static bool ConvertTAI58ToTAI93 (double tai58, double& tai93) const
static bool ConvertTAI93ToUTC (double tai93, string& utc) const
static bool ConvertUTCToJD (const string& utc, double& jd) const
static bool ConvertUTCToTAI93 (const string& utc, double& tai93) const
```

A.9 Missing Value Service

```
static inline double GetDoubleMissingValue ()
static inline float GetFloatMissingValue ()
static inline int GetIntMissingValue ()
static inline long GetLongMissingValue ()
```

```
static inline short GetShortMissingValue ()
bool IsMissingValue (double value) const
bool IsMissingValue (float value) const
bool IsMissingValue (int value) const
bool IsMissingValue (long value) const
bool IsMissingValue (short value) const
```

A.10 Program Abortion Service

```
static void Abort () const
static void Abort (const string& message) const
```

A.11 PCF Service

```
static bool GetFilename (int id, string& name) const
static bool GetFilename (int id, int version, string& name) const
static bool GetParameter (int id, string& parameter) const
```

A.12 Metadata Service

```
ecs_service (int datafilelogical, int ecsfilelogical)

~ecs_service ()

bool Set (const string& name, int value)
bool Set (const string& name, const void* value)
bool Set (const string& name, const string& value)
bool Write ()
```

A.13 Processor File

```
processor_file (int id)
processor_file (int id, int version)

~processor_file ()

inline int GetLogical () const
inline string GetName () const
# inline bool IsValid () const
```

A.14 ASCII File

```
ascii_file (int id)

~ascii_file ()

# void Close ()
# bool GetToken (const string& line, int number, double& value) const
# bool GetToken (const string& line, int number, float& value) const
# bool GetToken (const string& line, int number, int& value) const
# bool GetToken (const string& line, int number, string& value) const
# bool Open ()
# bool Read (string& line)
# bool Read (string& line, char commentchar)
```

A.15 HDF5 File

```
hdf5_file (int id, const string& swathname)

~hdf5_file ()

# void Close ()
# bool Create ()
# bool DefineDimension (const string& name, long value)
# bool DefineField (enum hdf5file_fieldtype type, const string& name,
                     const string& dimensionnames)
# bool DefineField (enum hdf5file_fieldtype type, const string& name,
                     const string& dimensionnames, int compressiondimensioncount,
                     int compressiondimensions[])
# bool WriteAttribute (const string& name, const string& value)
# bool WriteField (const string& name, int dimensions, const long starts[],
                   const long edges[], const void* data)
# bool WriteSWAttribute (const string& name, int value)
# bool WriteSWAttribute (const string& name, float value)
# bool WriteSWAttribute (const string& name, double value)
# bool WriteSWAttribute (const string& name, const string& value)
```

A.16 HDF5 Read File

```
hdf5_readfile (int id, const string& swathname)

~hdf5_readfile ()

# void Close ()
# bool GetDimensionSize (const string& dimensionname, long& value) const
# bool Open ()
# bool ReadField (const string& name, int dimensions, const long start[], const long edge[],
                   short& fillvalue, short* data) const
# bool ReadField (const string& name, int dimensions, const long start[], const long edge[],
                   float& fillvalue, float* data) const
# bool ReadField (const string& name, int dimensions, const long start[], const long edge[],
                   int& fillvalue, int* data) const
# bool ReadField (const string& name, int dimensions, const long start[], const long edge[],
                   double& fillvalue, double* data) const
```

A.17 Array Reverser

```
static void Reverse (int size, int array[])
static void Reverse (int size, float array[])
static void Reverse (int size, double array[])
```

A.18 Cubic Spliner

```
static bool Spline (int n, const int x[], const float y[], int m, const int xspline[],
                     float yspline)
static bool Spline (int n, const int x[], const double y[], int m, const int xspline[],
                     double yspline[])
static bool Spline (int n, const float x[], const float y[], int m, const double xspline[],
                     double yspline[])
static bool Spline (int n, const double x[], const int y[], int m, const double xspline[],
                     int yspline[])
static bool Spline (int n, const double x[], const double y[], int m, const double xspline[],
                     int yspline[])
static bool Spline (int n, const double x[], const float y[], int m, const double xspline[],
                     float yspline[])
static bool Spline (int n, const double x[], const double y[], int m, const double xspline[],
```

```
                    double yspline[])
static bool Spline (int n, const double x[], const double y[], int m, const double xspline[],
                    double yspline[], double yderiv[])
```

A.19 Linear Interpolator

```
static double Interpolate (double preabscissa, double preordinate, double postabscissa,
                           double postordinate, double targetabscissa)
static void Interpolate (double preabscissa, double preordinate, double postabscissa,
                         double postordinate, double targetabscissa, double& targetordinate)
static void Interpolate (double preabscissa, double preordinate, double prevariance,
                         double postabscissa, double postordinate, double postvariance,
                         double targetabscissa, double& targetordinate,
                         double& interpolatedvariance)
static void Interpolate (int valuesize, const int valueabscissa[],
                         const double valueordinate[], int interpsize,
                         const int interpabscissa[], double interpordinate[])
static void Interpolate (int valuesize, const double valueabscissa[],
                         const double valueordinate[], int interpsize,
                         const double interpabscissa[], double interpordinate[])
```

A.20 Scan Extractor

```
scan_extractor ()

~scan_extractor ()

bool Extract (hirdls1c_scan& scan)
```

A.21 HIRDLS1C File

```
static hirdls1c_file* GetFile ()

~hirdls1c_file ()

enum hirdls1c_code {<the list of acceptable diagnostic codes>}

bool Read (enum hirdls1c_code code, void* data)
```

A.22 Scan Discriminator

```
scan_discriminator ()

~scan_discriminator ()

void Discriminate (long size, const short scannumbers[])
bool GetNext (discriminator_data& data)
```

A.23 Discriminator Data

```
discriminator_data ()
discriminator_data (const discriminator_data& data)
discriminator_data& operator= (const discriminator_data& data)

~discriminator_data ()

void Get (int& firstrev, int& lastrev, int& scannumber) const
void Set (int firstrev, int lastrev, int scannumber)
```

A.24 HIRDLS1C Scan

```
hirdls1c_scan ()

~hirdls1c_scan ()

static int const SCANREVS_SIZE
static int const CHOPPERREV_SIZE
static int const MINORFRAME_SIZE

inline int GetScanLength () const
inline bool IsUpScan () const
void Get (bool& isupscan, int& scanlength, int altitudes[][SCANREVS_MAXSIZE],
          float radiances[][SCANREVS_MAXSIZE]) const
void Get (bool& isupscan, int& scanlength, float elshaftangles[SCANREVS_MAXSIZE],
          float radiances[][SCANREVS_MAXSIZE]) const
void Get (bool& isupscan, int& scanlength, int& scannumber, int boresightaltitudes[],
          int flags[], int orbitnumbers[], int scanmodeids[], int scantables[],
          int spacecraftaltitudes[], int ecirays[][3], int ecisurfaces[][3],
          int spacecraftpositions[][3], int altitudes[][SCANREVS_MAXSIZE],
          float azlosangles[], float ellosangles[], float elshaftangles[],
          float coldfiltertemps[], float ifctemps[], float warmfiltertemps[],
          float latitudes[], float longitudes[], float localsolartimes[],
          float solarzenithangles[], float viewdirections[], float spacecraftlatitudes[],
          float spacecraftlongitudes[], float raderrors[][SCANREVS_MAXSIZE],
          float radiances[][SCANREVS_MAXSIZE], double times[]) const
void Set (int revcount, int mifcount, int mafcount, int scannumber,
          const float scalefactors[], const float scaleoffsets[],
          const short ch01altoffsets[], const short ch02altoffsets[],
          const short ch03altoffsets[], const short ch04altoffsets[],
          const short ch05altoffsets[], const short ch06altoffsets[],
          const short ch07altoffsets[], const short ch08altoffsets[],
          const short ch09altoffsets[], const short ch10altoffsets[],
          const short ch11altoffsets[], const short ch12altoffsets[],
          const short ch13altoffsets[], const short ch14altoffsets[],
          const short ch15altoffsets[], const short ch16altoffsets[],
          const short ch17altoffsets[], const short ch18altoffsets[],
          const short ch19altoffsets[], const short ch20altoffsets[],
          const short ch21altoffsets[], const short ch01raderrors[],
          const short ch02raderrors[], const short ch03raderrors[],
          const short ch04raderrors[], const short ch05raderrors[],
          const short ch06raderrors[], const short ch07raderrors[],
          const short ch08raderrors[], const short ch09raderrors[],
          const short ch10raderrors[], const short ch11raderrors[],
          const short ch12raderrors[], const short ch13raderrors[],
          const short ch14raderrors[], const short ch15raderrors[],
          const short ch16raderrors[], const short ch17raderrors[],
          const short ch18raderrors[], const short ch19raderrors[],
          const short ch20raderrors[], const short ch21raderrors[],
          const unsigned short ch01rads[], const unsigned short ch02rads[],
          const unsigned short ch03rads[], const unsigned short ch04rads[],
          const unsigned short ch05rads[], const unsigned short ch06rads[],
          const unsigned short ch07rads[], const unsigned short ch08rads[],
          const unsigned short ch09rads[], const unsigned short ch10rads[],
          const unsigned short ch11rads[], const unsigned short ch12rads[],
          const unsigned short ch13rads[], const unsigned short ch14rads[],
          const unsigned short ch15rads[], const unsigned short ch16rads[],
          const unsigned short ch17rads[], const unsigned short ch18rads[],
          const unsigned short ch19rads[], const unsigned short ch20rads[],
          const unsigned short ch21rads[], const short scantables[],
          const short solarzenithangles[], const short viewdirections[],
          const short azlosangles[], const int scanmodeids[],
          const int orbitnumbers[], const int flags[], const int ellosangles[],
          const int localsolartimes[], const int altitudes[],
```

```
          const int spacecraftaltitudes[], const int ecirays[],
          const int ecisurfaces[], const int spacecraftpositions[],
          const float coldfiltertemps[], const float ifctemps[],
          const float warmfiltertemps[], const float elevations[],
          const float latitudes[], const float longitudes[],
          const float spacecraftlatitudes[], const float spacecraftlongitudes[],
          const double times[])
void SetRadiances (const float radiances[][SCANREVS_MAXSIZE])
```

A.25 Radiance Filterer

```
radiance_filterer ()

~radiance_filterer ()

bool Filter (hirdsl1c_scan& scan)
```

A.26 Config File

```
static config_file* GetFile ()

~config_file ()

bool Read (config_data& data)
```

A.27 Config Data

```
config_data ()
config_data (const config_data& data)
config_data& operator= (const config_data& data)

~config_data ()

void Get (int channelnumber, int& spectra, int& deconv, int& filtertype,
          float& frequencycutoff, float& maxboost) const
void Get (int spectra[CHANNEL_SIZE], int deconv[CHANNEL_SIZE], int filtertype[CHANNEL_SIZE],
          float frequencycutoff[CHANNEL_SIZE], float maxboost[CHANNEL_SIZE]) const
void Set (const int spectra[CHANNEL_SIZE], const int deconv[CHANNEL_SIZE],
          const int filtertype[CHANNEL_SIZE], const float frequencycutoff[CHANNEL_SIZE],
          const float maxboost[CHANNEL_SIZE])
```

A.28 FOV File

```
static fov_file* GetFile ()

~fov_file ()

bool Read (fov_data& data)
```

A.28 FOV Data

```
fov_data ()
fov_data (const fov_data& data)
fov_data& operator= (const fov_data& data)

~fov_data ()

static int const RESPONSE_SIZE
```

```
void Get (float grid[RESPONSE_SIZE], float response[CHANNEL_SIZE][RESPONSE_SIZE]) const
void Set (const float grid[RESPONSE_SIZE], const float response[CHANNEL_SIZE][RESPONSE_SIZE])
```

A.29 Data Gridder

```
data_gridder ()

~data_gridder ()

static int const GRID_MAXSIZE

void Grid (int scanlength, const float elshaftangles[hirdls1c_scan::SCANREVS_MAXSIZE],
           const float radiances[CHANNEL_SIZE][hirdls1c_scan::SCANREVS_MAXSIZE],
           int& gridlength, double gridsubset[GRID_MAXSIZE],
           float griddedradiances[CHANNEL_SIZE][GRID_MAXSIZE]) const
void Ungrid (int gridlength, const double gridsubset[GRID_MAXSIZE],
             const float griddedradiances[CHANNEL_SIZE][GRID_MAXSIZE], int scanlength,
             const float elshaftangles[hirdls1c_scan::SCANREVS_MAXSIZE],
             float radiances[CHANNEL_SIZE][hirdls1c_scan::SCANREVS_MAXSIZE]) const
```

A.30 Signal Deconvolver

```
signal_deconvolver ()

~signal_deconvolver ()

void Initialize (int fovcount, double anglespacing, const int spectra[CHANNEL_SIZE],
                 const int deconv[CHANNEL_SIZE], const int filtertype[CHANNEL_SIZE],
                 const float frequencycutoff[CHANNEL_SIZE],
                 const float maxboost[CHANNEL_SIZE],
                 double fovresponse[CHANNEL_SIZE][data_gridder::GRID_MAXSIZE])
void Deconvolve (int gridlength,
                 const float radiances[CHANNEL_SIZE][data_gridder::GRID_MAXSIZE],
                 float griddedradiances[CHANNEL_SIZE][data_gridder::GRID_MAXSIZE])
```

A.31 Scan Transformer

```
scan_transformer ()

~scan_transformer ()

bool Transform (const hirdls1c_scan& hirdsl1cscan, hirdls1r_scan& hirdls1rscan) const
```

A.32 HIRDLS1R Scan

```
hirdls1r_scan ()

~hirdls1r_scan ()

static int const ALTITUDE_SIZE
static int const LOS_SIZE

inline int GetOrbitNumber () const;
inline int GetScanMode () const
inline int GetScanNumber () const
inline int GetScanTable () const
inline int GetScanUpFlag () const
inline int GetSpacecraftDayFlag () const
inline float GetAzimuth () const
inline float GetColdFilterTemperature () const
```

```
inline float GetElevation () const
inline float GetIFCTemperature () const
inline float GetLatitude () const
inline float GetLocalSolarTime () const
inline float GetLongitude () const
inline float GetSolarZenithAngle () const
inline float GetSpacecraftAltitude () const
inline float GetSpacecraftLatitude () const
inline float GetSpacecraftLongitude () const
inline float GetTangentHeight () const
inline float GetViewDirection () const
inline float GetWarmFilterTemperature () const
inline double GetRadiusOfCurvature () const
inline double GetRTopOfTheAtmosphere () const
inline double GetTime () const
void GetAltitude (int[ALTITUDE_SIZE]) const
void GetAltitude (float[ALTITUDE_SIZE]) const
void GetJitterError (float[CHANNEL_SIZE][ALTITUDE_SIZE]) const
void GetLOSAzimuth (float[ALTITUDE_SIZE]) const
void GetLOSElevation (float[ALTITUDE_SIZE]) const
void GetLOSTopOfAtmosphere (double[ALTITUDE_SIZE][ LOS_SIZE]) const
void GetPsiTopOfAtmosphere (double[ALTITUDE_SIZE]) const
void GetRadiance (float[CHANNEL_SIZE][ALTITUDE_SIZE]) const
void GetRadianceError (float[CHANNEL_SIZE][ALTITUDE_SIZE]) const
void GetShaftElevation (float[CHANNEL_SIZE][ALTITUDE_SIZE]) const
void GetXTopOfAtmosphere (double[ALTITUDE_SIZE]) const
void GetYTopOfAtmosphere (double[ALTITUDE_SIZE]) const
void GetZTopOfAtmosphere (double[ALTITUDE_SIZE]) const
void Set (const hirdls1c_scan&);
void SetRadiance (const float[CHANNEL_SIZE][ALTITUDE_SIZE]);
```

A.33 Radiance Adjuster

```
radiance_adjuster ()

~radiance_adjuster ()

bool Adjust (hirdls1r_scan& scan) const;
```

A.34 Adjustment File

```
static adjustment_file* GetFile ()

~adjustment_file ()

bool Read (adjustement_data& data)
```

A.35 Adjustment Data

```
adjustment_data ()
adjustment_data (const adjustment_data& data)
adjustment_data& operator= (const adjustment_data& data)

~adjustment_data ()

static int const ADJUSTMENTHEIGHT_SIZE

void Get (float heights[ADJUSTMENTHEIGHT_SIZE],
          double upadjustments[CHANNEL_SIZE][ADJUSTMENT_SIZE],
          double downadjustments[CHANNEL_SIZE][ADJUSTMENT_SIZE]) const
```

```
void Set (const float heights[ADJUSTMENTHEIGHT_SIZE],
          const double upadjustments[CHANNEL_SIZE][ADJUSTMENT_SIZE],
          const double downadjustments[CHANNEL_SIZE][ADJUSTMENT_SIZE])
```

A.36 File Writer

```
file_writer ()

~file_writer ()

bool Write (const hirdls1r_scan& scan)
```

A.37 Metadata Manager

```
metadata_manager ()

~metadata_manager ()

void Add (const hirdsl1r_scan& scan)
void Retrieve (metadata_data& data)
```

A.38 Metadata Data

```
metadata_data ()
metadata_data (const metadata_data& data)
metadata_data& operator= (const metadata_data& data)

~metadata_data ()

void Get (int& qapercentmissingdata, int& qapercentoutofboundsdata,
          string& automaticqualityflag, string& automaticqualityflagexplanation,
          string& daynightflag, string& doi, string& inputpointer, string& localgranuleid,
          string& localversionid, string& operationalqualityflag,
          string& operationalqualityflagexplanation, string& parametername,
          string& pgeversion, string& rangebeginningdate, string& rangebeginningtime,
          string& rangeendingdate, string& rangeendingtime, string& shortname) const
void Set (int qapercentmissingdata, int qapercentoutofboundsdata,
          const string& automaticqualityflag, const string& automaticqualityflagexplanation,
          const string& daynightflag, const string& doi, const string& inputpointer,
          const string& localgranuleid, const string& localversionid,
          const string& operationalqualityflag,
          const string& operationalqualityflagexplanation, const string& parametername,
          const string& pgeversion, const string& rangebeginningdate,
          const string& rangebeginningtime, const string& rangeendingdate,
          const string& rangeendingtime, const string& shortname)
void Update (const string& rangeendingdate, const string& rangeendingtime)
```

A.39 HIRDLS1R File

```
static hirdls1r_file* GetFile ()

~hirdls1r_file ()

bool Write (const hirdls1r_scan& scan)
bool WriteMetadata (const metadata_data& data)
```

A.40 L2 Preprocessor

```
L2_preprocessor ()

~l2_preprocessor ()

void Process ()
```