HIRDLS

SW-HIR-2007

Originator: Charles Cavanaugh

Date: 10 Dec. 2012

Subject/Title:

The Design of the HIRDLS Level 1 Correction Processor

Description/Summary/Contents:

The purpose of this document is to create a design specification for the High Resolution Dynamics Limb Sounder (HIRDLS) Level 1 Correction Processor, hereby known as L1C Processor. This design specification will extend the architecture of L1C Processor 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 L1C Processor.

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 1 Correction Processor

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 1 Correction Processor, hereby known as L1C Processor. This design specification will extend the architecture of L1C Processor 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 L1C Processor.

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 L1C Processor Requirements document, L1C Processor 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 L1C Processor 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 L1C Processor. 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 L1C Processor Architecture document introduced various packages from which L1C Processor 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 L1C Processor 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 L1C Processor packages introduced in the L1C Processor 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 L1C Processor. 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 L1C Processor 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 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 L1C Processor: 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 HIRDLS1C File. Data abstractions are about content, so they have "contenty" names, such as HIRDLS1 Scan or Diagnostic Data. All abstractions, regardless of type, are to fully encapsulate all functionality needed to accomplish their specified task, and to present to L1C Processor 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.

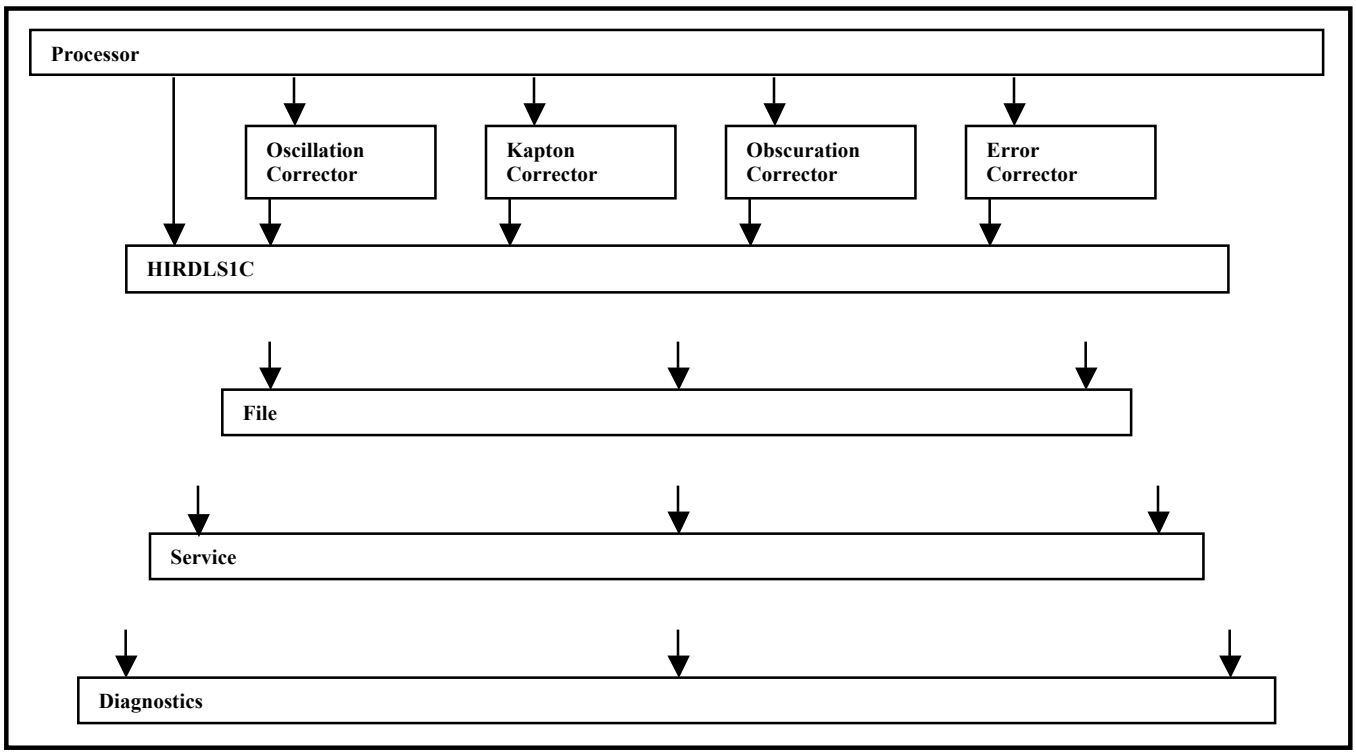

Figure 1 L1C Processor Hierarchy of Packages

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 L1C Processor. 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 L1C Processor is to maximize maintainability. The L1C Processor Architecture document shows that packages communicate with each other via data aggregations, and therefore L1C Processor 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 L1C Processor 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 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

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

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 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 L1C Processor 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 L1C Processor 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 L1C Processor. 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 L1C Processor 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 L1C Processor 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 L1C Processor, 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 L1C Processor. 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 L1C Processor has been delivered. Therefore, while the majority of the elements of L1C Processor 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

L1C Processor 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.

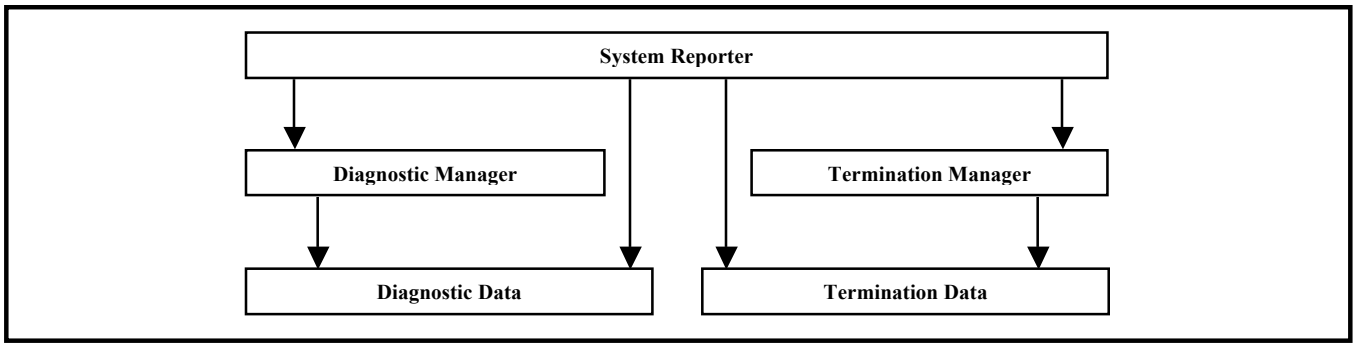

Figure 2 Diagnostics Package Hierarchy

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.

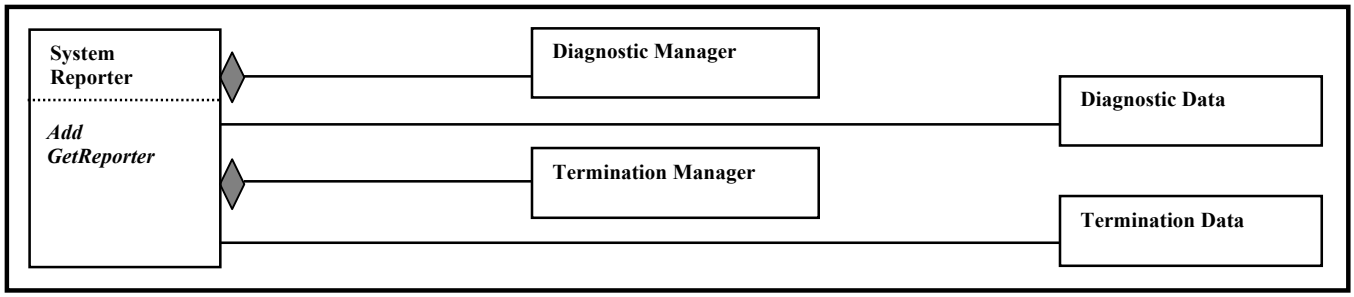

Figure 3 System Reporter Abstraction

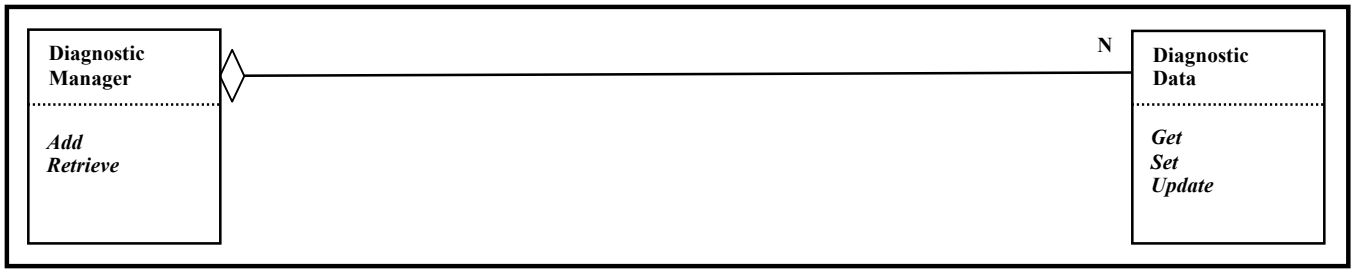

Figure 4 Diagnostic Manager and Diagnostic Data Abstractions

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.

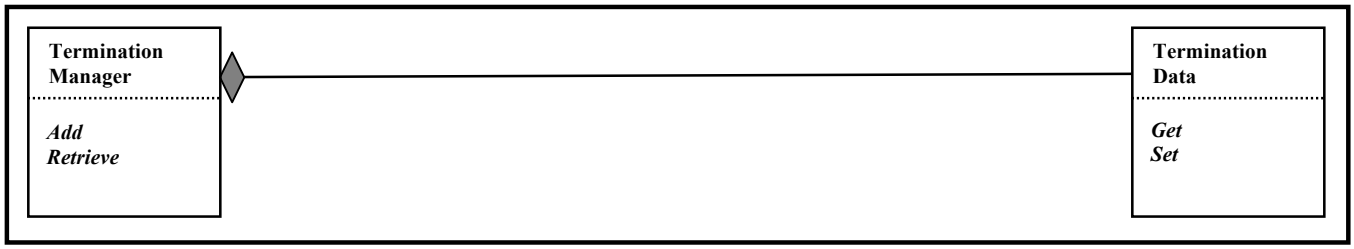

Figure 5 Termination Manager and Termination Data Abstractions

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.

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.

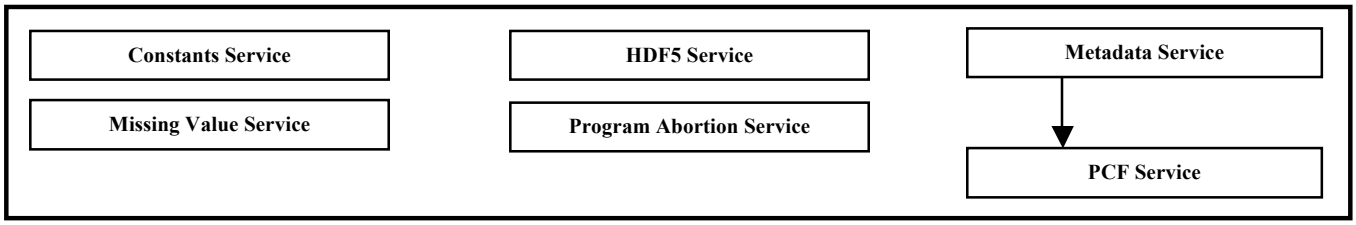

Figure 6 Service Package Hierarchy

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 size constants, 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 OpenFile contract opens an existing HDF5 file. The CloseFile contract closes an HDF5 file. The OpenSwath contract opens an existing swath. The CloseSwath contract closes access to a swath. The GetDimensionSize contract returns a defined dimension size, the GetFieldDimensions retrives the dimensions of a given field, 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.

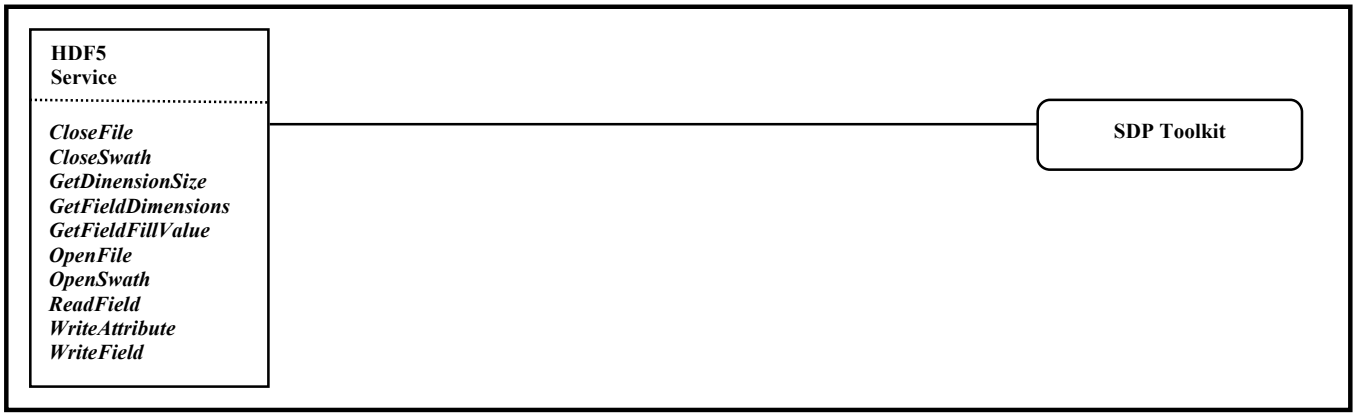

Figure 8 HDF5 Service Abstraction

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.

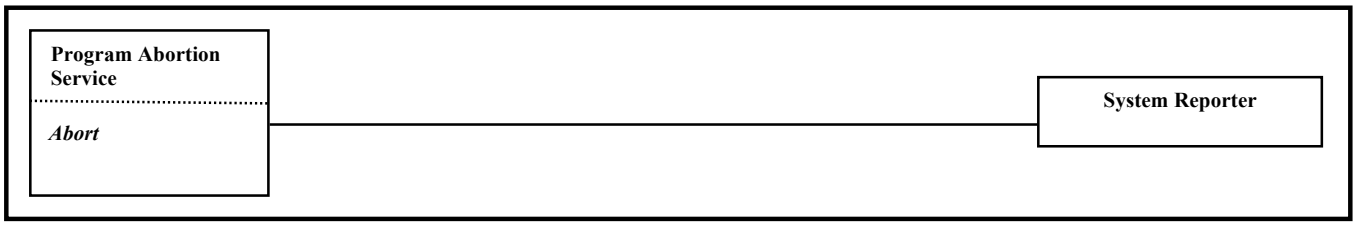

Figure 10 Program Abortion Service Abstraction

8.5 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 11. 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.

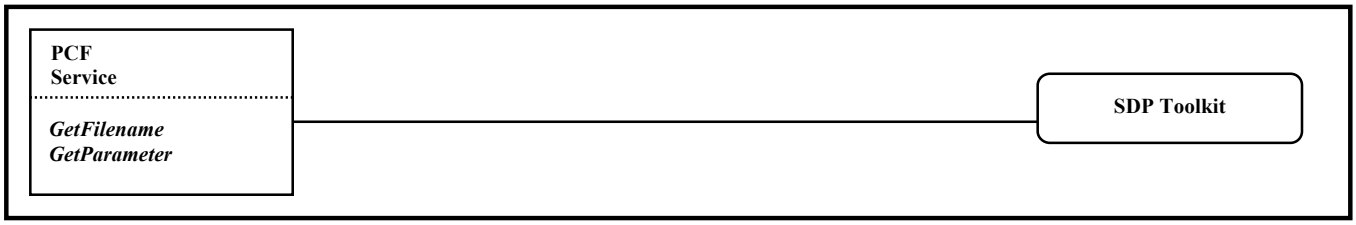

Figure 11 PCF Service Abstraction

8.6 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 12. 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.

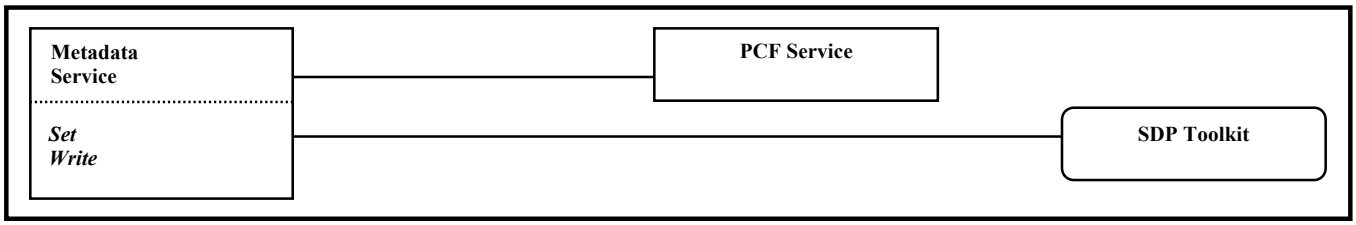

Figure 12 Metadata Service Abstraction

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 13 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

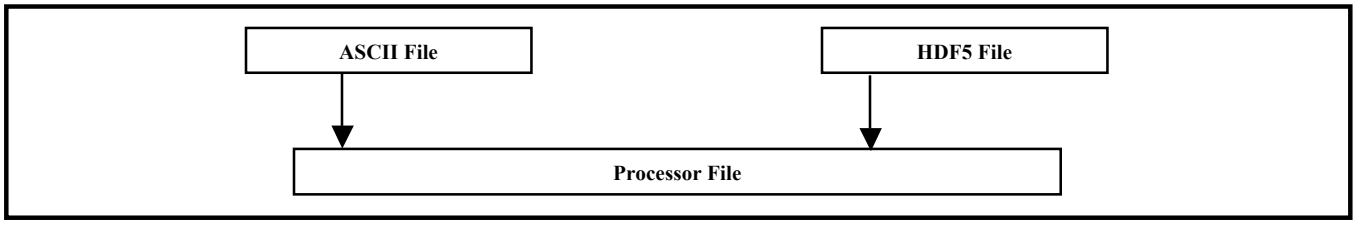

Figure 13 File Package Hierarchy

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 14. 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 14 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 15. 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 15 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 opening, closing, reading and writing contracts, as shown in Figure 16. The Open contract opens an HDF5 file, and the Close contract closes an HDF5 file. The GetDimensionSize contract retrieves a given field dimension size. The ReadField contracts read data from an HDF5 file, the WriteField contract writes a field's data to an 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.

10 HIRDLS1C Package

The HIRDLS1C package has the responsibility to encapsulate all abstractions necessary to provide the system a means to extract scans from and write scans to the HIRDLS L1C file. As shown in Figure 1, this package is used by the Processor, Oscillation Corrector, Kapton Corrector, Obscuration Corrector and Error Corrector packages, and has access to the File, Service and Diagnostics packages. Figure 17 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

Figure 16 HDF5 File Abstraction

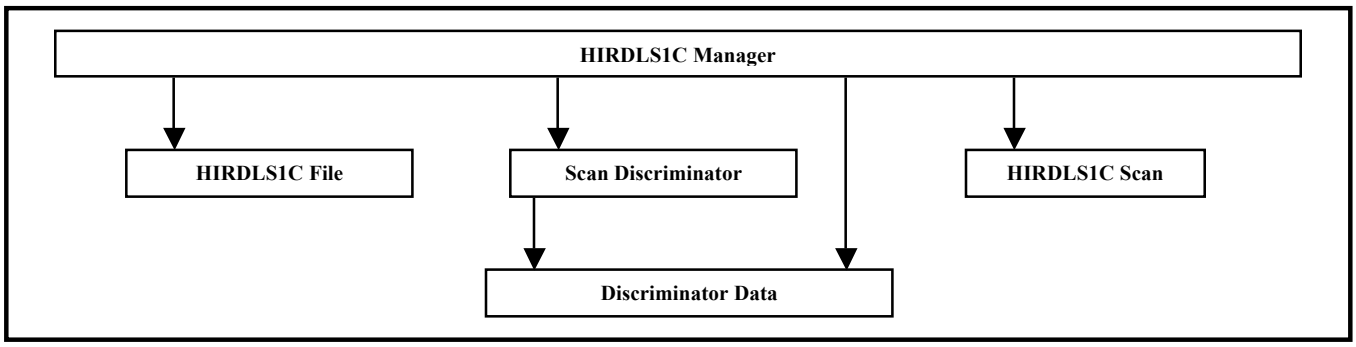

Figure 17 HIRDLS1C Package Hierarchy

10.1 HIRDLS1C Manager Abstraction

Scan Extractor is a control abstraction, and has the responsibility to manage extracting scans from and writing scans to a HIRDLS1C File To fulfill this responsibility, this abstraction collaborates with HIRDLS1C File, Scan Discriminator, Discrimination Data and HIRDLS1C Scan, and presents an interface of one ExtractScan contract and one WriteScan contract, as shown in Figure 18. The ExtractScan contract is to return the *next* scan in the file, in the form of a HIRDLS1C Scan. 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. The WriteScan contract writes a scan into the *next* scan spot in the file. Also specified here is that HIRDLS1C Manager 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.

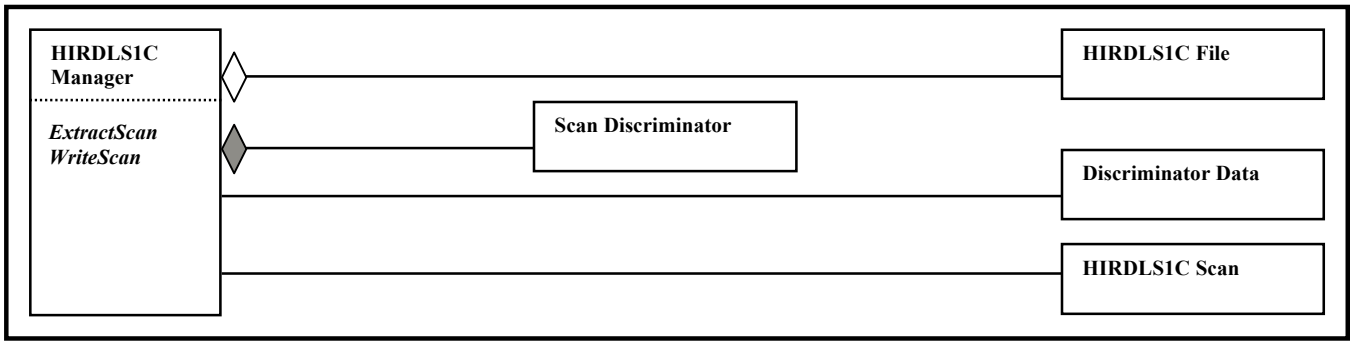

Figure 18 HIRDLS1C Manager Abstraction

10.2 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 File, and presents an interface of one GetFile contract, one Read contract and one Write contract, as shown in Figure 19. The GetFile contract returns an instance of a HIRDLS1C File abstraction. At this time, this instance is considered a Singleton[3], as there must be only one file instance in the processor. The Read contract returns the contents of a given HIRDLS1C field, and the Write contract writes data into a given HIRDLS1C field. These two contracts are to return a Boolean status to the caller, indicating if it was able to retrieve or write the field or not. This abstraction needs to be kept persistent to work correctly.

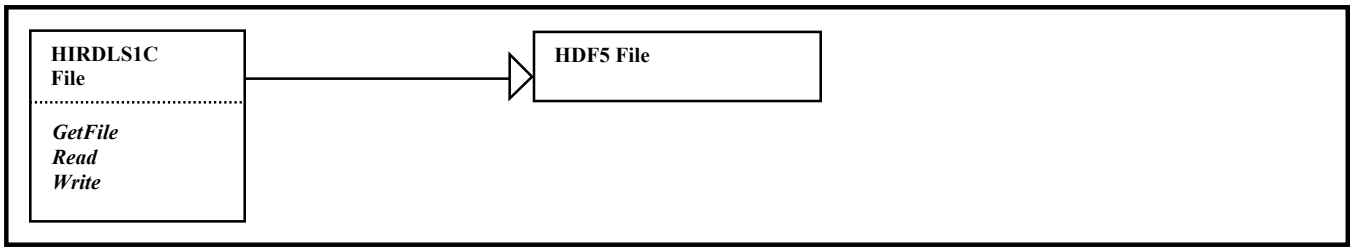

Figure 19 HIRDLS1C File Abstraction

10.3 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.

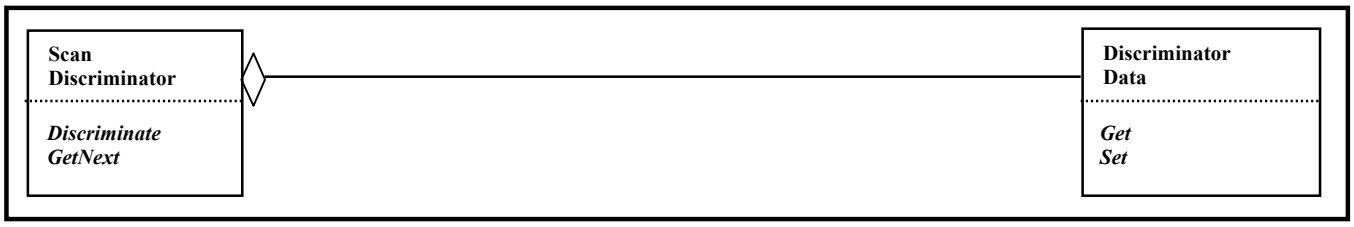

Figure 20 Scan Discriminator and Discriminator Data Abstractions

10.4 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.

[3] See Section 7.1

10.5 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 contracts to test scan aspects, one Denominalize contract, one Reset 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 Reset contract resets corrected radiances and radiance errors. The Denominalize contract denominalizes the scan. 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 21 HIRDLS1C Scan Abstraction

11 Oscillation Corrector Package

The Oscillation Corrector package has the responsibility to encapsulate all abstractions necessary to provide the system a means to correct for the oscillation effect in HIRDLS1C scans. As shown in Figure 1, this package is used by the Processor package, and has access to the HIRDLS1C, File, Service and Diagnostics packages. Figure 22 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

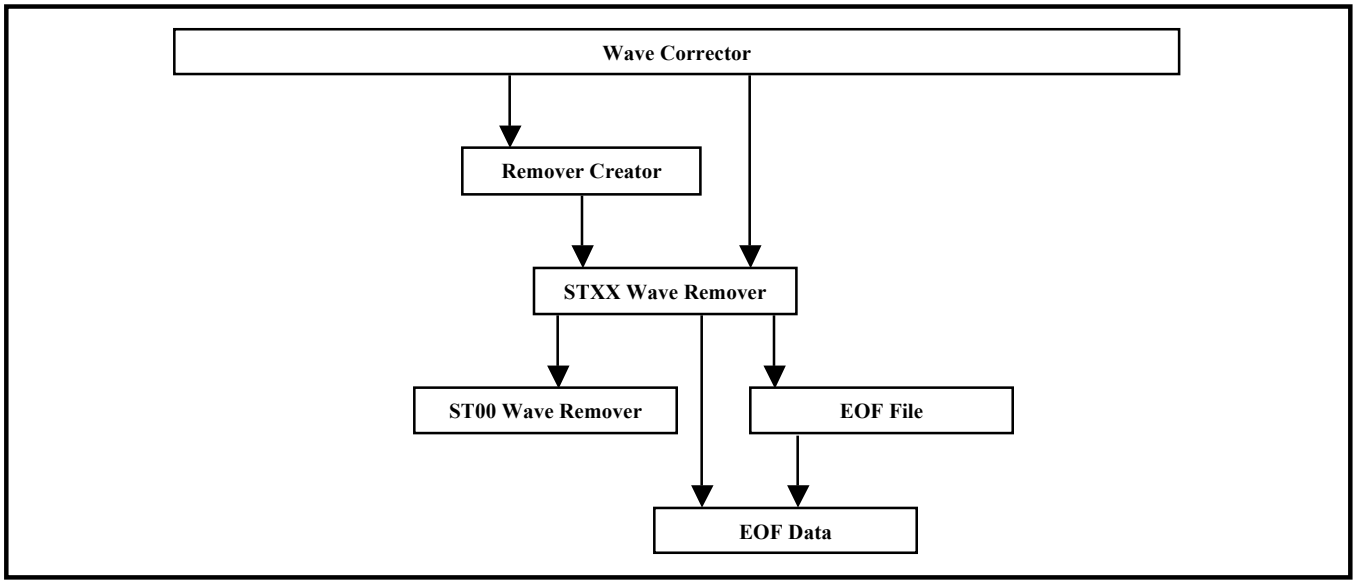

Figure 22 Oscillation Corrector Package Hierarchy

11.1 Wave Corrector Abstraction

Wave Corrector is a process abstraction, and has the responsibility to correct for the oscillation effects on all HIRDLS scans. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Manager, HIRDLS1C Scan, Remover Creator and STXX Wave Remover, and presents an interface of one Correct contract, as shown in Figure 23. The Correct contract loops over all the scans returned by HIRDLS1C Manager, first calling Remover Creator to return the appropriate ST-specific wave remover, then extracting the radiances and/or errors from HIRDLS1C Scan, passing them to the wave remover for correction, resetting the corrected radiances and/or errors in HIRDLS1C Scan, and then passing the corrected scan back to HIRDLS1C Manager for writing. Since this abstraction needs to initialize itself with EOF data before it can correct scans, it needs to be persistent to work efficiently.

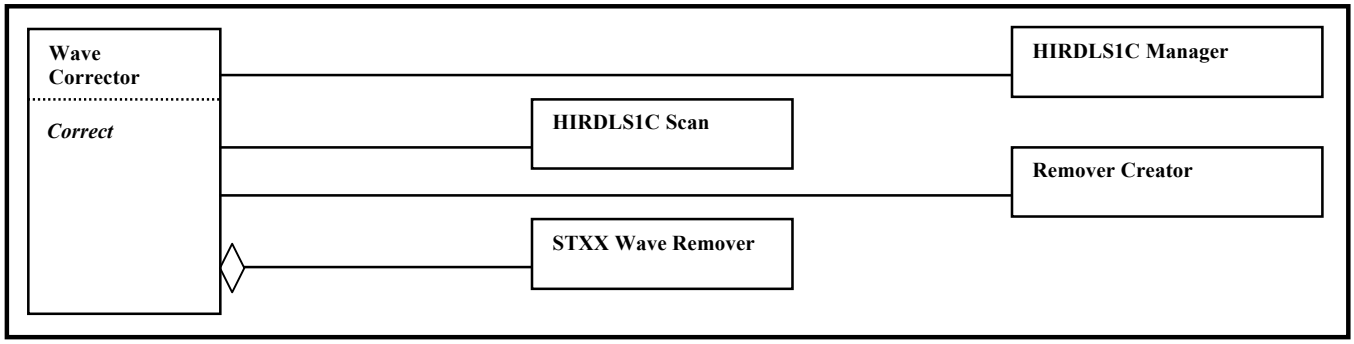

Figure 23 Wave Corrector Abstraction

11.2 Remover Creator Abstraction

Remover Creator is a process abstraction, and each has the responsibility to create the appropriate STXX Wave Remover to be used by the system. To fulfill this responsibility, this abstraction collaborates with STXX Wave Remover and ST00 Wave Remover, and presents an interface of one Create contract, as shown in Figure 24. The Create contract is specified to be an *Abstract Factory creational pattern*[4], and return an STXX Wave Remover (or ST00 Wave Remover as default). This abstraction is not meant to be instantiated, but instead provide non-dynamic data into the global space. Persistency is not an issue.

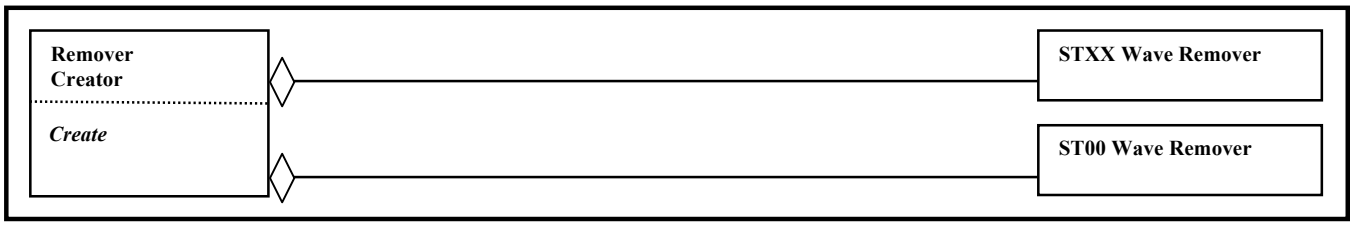

Figure 24 Remover Creator Abstraction

11.3 STXX Wave Remover Abstraction

STXX Wave Remover is a placeholder for process abstractions ST13 Wave Remover, ST22 Wave Remover, ST23 Wave Remover and ST30 Wave Remover, and each collaborates with ST00 Wave Remover, HIRDLS1C Scan, EOF File and EOF Data, and each has the responsibility to remove the oscillation effect on a HIRDLS1C scan that is of their respective scan

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

table. To fulfill this responsibility, these abstractions collaborate with ST00 Wave Remover, and present an interface of one Remove contract, as shown in Figure 25. The Remove contract fulfills the responsibility of removing the oscillation effect on a scan's radiances and/or errors. Since these abstractions load in scan table specific information, they need to be persistent to work efficiently.

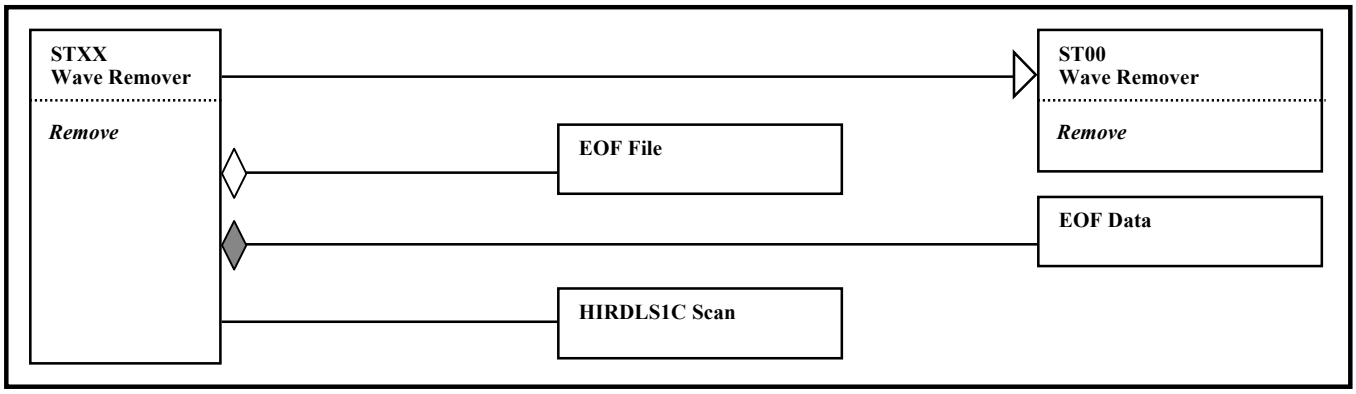

Figure 25 STXX Wave Remover and ST00 Wave Remover Abstractions

11.4 ST00 Wave Remover Abstraction

ST00 Wave Remover is a process abstraction, and each has the responsibility to provide a base abstraction for other abstractions (placeholded by STXX Wave Remover) that remove oscillation effects from a HIRDLS1C scan. To fulfill this responsibility, this abstraction presents an interface of two Remove contracts (one base, one virtual), as shown in Figure 25. The virtual Remove contract defines the virtual methodology that all STXX Wave Removers are to mimic, and returns a Boolean of "false" to the system, as its instantiation means that there is no way to remove the wave for the given scan table. The base Remove contract is to be used by the STXX Wave Removers, after they load in their respective scan table EOFs. Persistency is not an issue.

11.5 EOF File Abstraction

EOF File is a control abstraction, and has the responsibility to manage all access to a wave correction EOF file. To fulfill this responsibility, this abstraction collaborates with ASCII File and EOF Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 26. The GetFile contract returns an instance of an EOF File abstraction. At this time, this instance is not considered a Singleton[5], as EOF 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 EOF 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.

11.6 EOF Data Abstraction

EOF Data is a data abstraction, and has the responsibility to manage access to the data read from a wave correction EOF File. To fulfill this responsibility, this abstraction presents an interface of one Set contract and one Get contract, as shown in Figure 26. This abstraction also presents some constant values denoting various sizes of the data it contains. The copy

[5] See Section 7.1

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

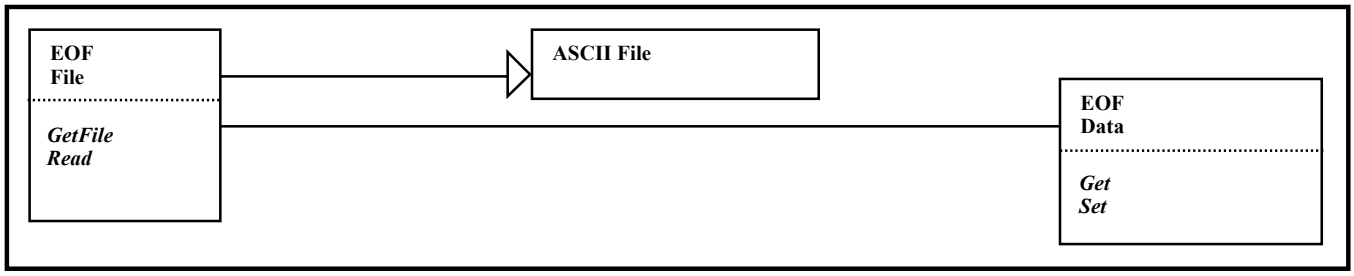

Figure 26 EOF File and EOF Data Abstractions

12 Kapton Corrector Package

The Kapton Corrector package has the responsibility to encapsulate all abstractions necessary to provide the system a means to correct for the Kapton effect in HIRDLS1C scans. As shown in Figure 1, this package is used by the Processor package, and has access to the HIRDLS1C, File, Service and Diagnostics packages. Figure 27 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

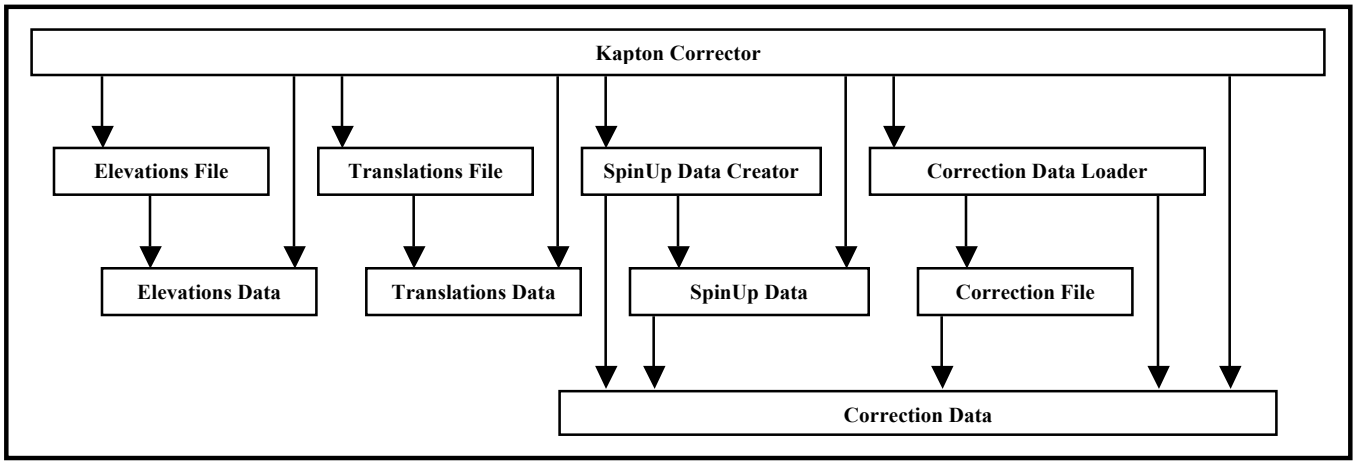

Figure 27 Kapton Corrector Package Hierarchy

12.1 Kapton Corrector Abstraction

Kapton Corrector is a process abstraction, and has the responsibility to correct for the Kapton emission effects on HIRDLS scans To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Manager, HIRDLS1C Scan, Elevations File, Elevations Data, Translations File, Translations Data, SpinUp Data Creator, SpinUpData, Correction Data Loader, Correction File and Correction Data, and presents an interface of one SpinUp contract and one Correct contract, as shown in Figure 28. To perform Kapton emission correction, it is necessary to pass through all the HIRDLS1C scans twice. The first pass calls the SpinUp contract, which uses the HIRDLS1C scans to generate the data necessary to initialize Kapton emission correction. The second pass calls the Correct contract, which applies the spin-up data, and other necessary data, to correct for the Kapton emission effect in the HIRDLS1C scans. Since this abstraction needs to initialize itself with spin-up data before it can correct scans, it needs to be persistent to work correctly.

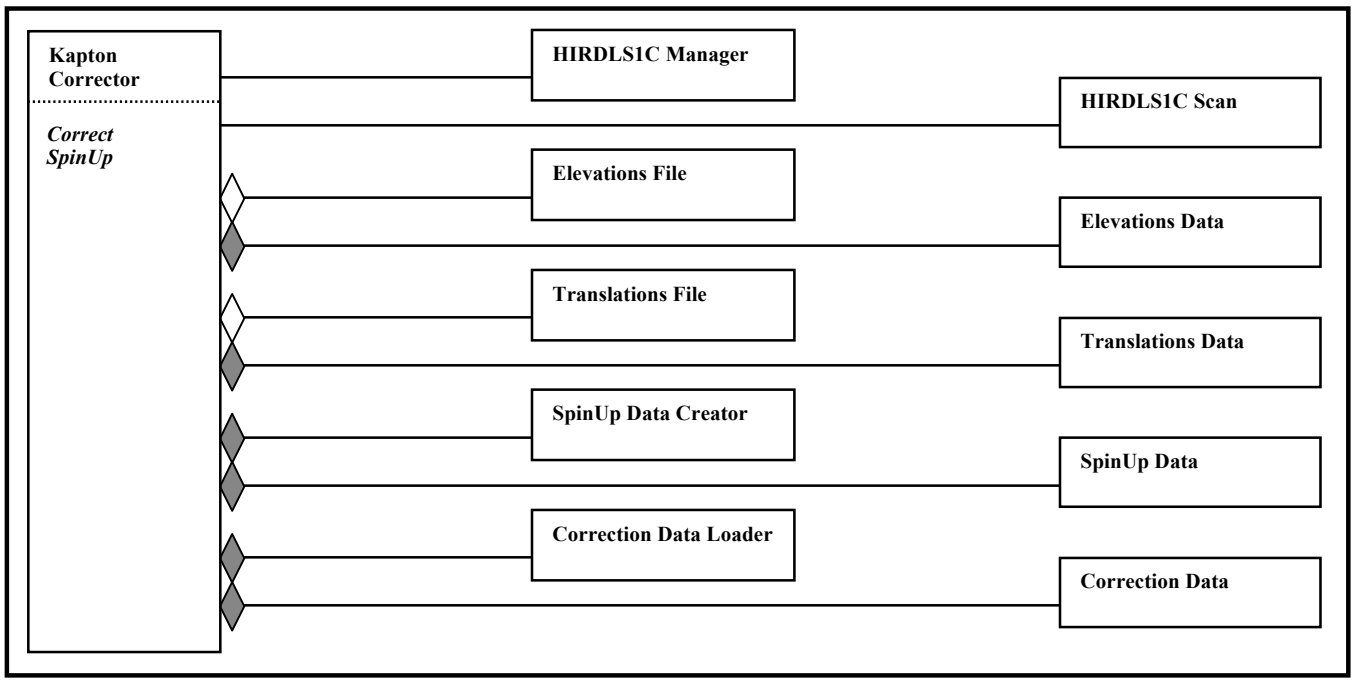

Figure 28 Kapton Corrector Abstraction

12.2 Elevations File Abstraction

Elevations File is a control abstraction, and has the responsibility to manage all access to a Kapton correction elevations file. To fulfill this responsibility, this abstraction collaborates with ASCII File and Elevations Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 29. The GetFile contract returns an instance of an Elevations File abstraction. At this time, this instance is not considered a Singleton[6], as elevation 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 Elevations 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.

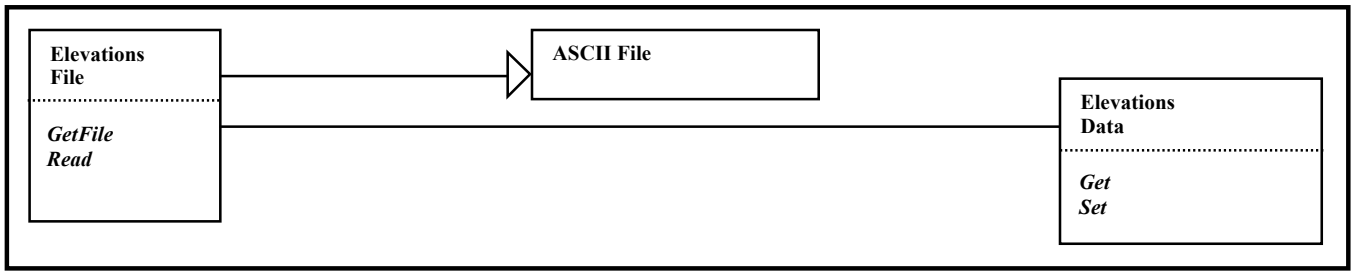

Figure 29 Elevations File and Elevations Data Abstractions

[6] See Section 7.1

12.3 Elevations Data Abstraction

Elevations Data is a data abstraction, and has the responsibility to manage access to the data read from a Kapton correction Elevations File. To fulfill this responsibility, this abstraction presents an interface of one Set contract and one Get contract, as shown in Figure 29. The copy constructor, assignment operator, or Set contract can be used by Elevations File to initialize the abstraction. The Get contract allows retrieval of all the data.

12.4 Translations File Abstraction

Translations File is a control abstraction, and has the responsibility to manage all access to a Kapton correction translations file. To fulfill this responsibility, this abstraction collaborates with ASCII File and Translations 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 Translations File abstraction. At this time, this instance is not considered a Singleton[7], as translation 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 Translations 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.

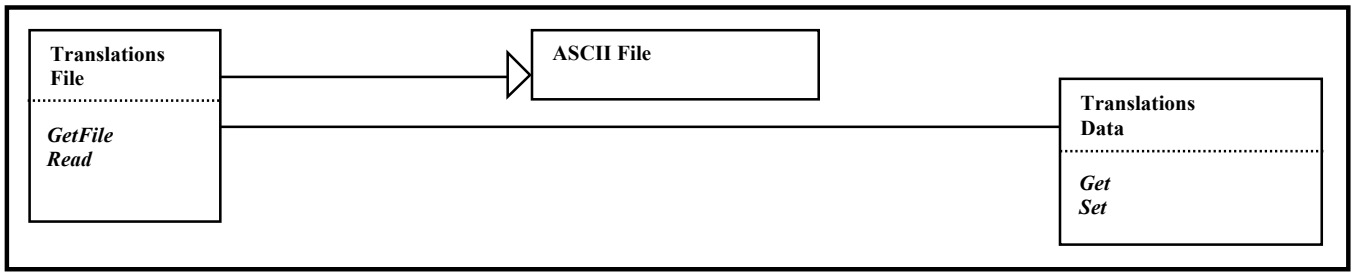

Figure 30 Translations File and Translations Data Abstractions

12.5 Translations Data Abstraction

Translations Data is a data abstraction, and has the responsibility to manage access to the data read from a Kapton correction Translations 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 Translations File to initialize the abstraction. The Get contract allows retrieval of all the data.

12.6 SpinUp Data Creator Abstraction

SpinUp Data Creator is a control abstraction, and has the responsibility to manage creation of the Kapton correction spin-up data. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Scan, Correction Data and SpinUp Data, and presents an interface of one Add contract and one Retrieve contract, as shown in Figure 31. The Add contract adds HIRDLS1C Scan data to the spin-up data collection. The Retrieve contract retrieves the spin-up data. This abstraction needs to be persistent to work correctly.

[7] See Section 7.1

12.7 SpinUp Data Abstraction

SpinUp Data is a data abstraction, and has the responsibility to manage access to the data created by SpinUp Data Creator. To fulfill this responsibility, this abstraction presents an interface of one Set contract and one Get contract, as shown in Figure 31. The copy constructor, assignment operator, or Set contract can be used by SpinUp Data Creator to initialize the abstraction. The Get contract allows retrieval of all the data.

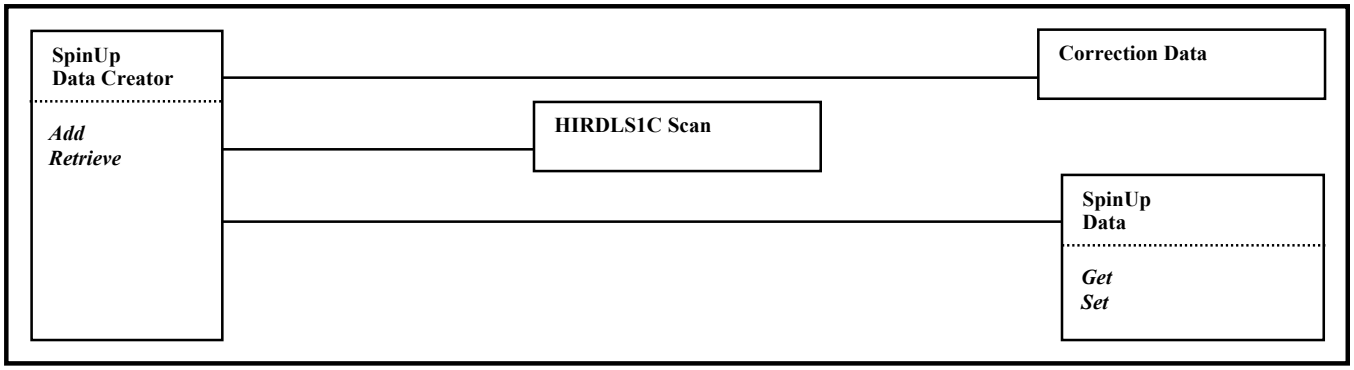

Figure 31 SpinUp Data Creator and SpinUp Data Abstractions

12.8 Correction Data Loader Abstraction

Correction Data Loader is a process abstraction, and each has the responsibility to load the appropriate correction data into the system. To fulfill this responsibility, this abstraction collaborates with Correction File and Correction Data, and presents an interface of one Load contract, as shown in Figure 32. The Load contract returns the appropriate correction data to be used for correction. Since this abstraction reads in data from file(s), it needs to be persistent to work efficiently.

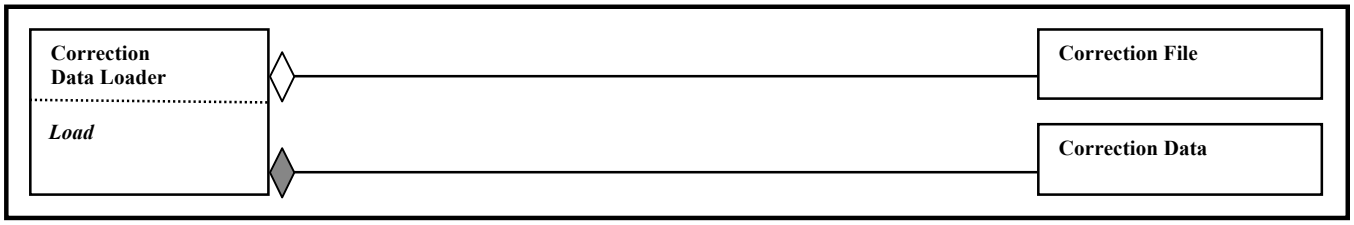

Figure 32 Correction Data Loader Abstraction

12.9 Correction File Abstraction

Correction File is a control abstraction, and has the responsibility to manage all access to a Kapton correction file. To fulfill this responsibility, this abstraction collaborates with HDF5 File and Correction Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 33. The GetFile contract returns an instance of a Correction File abstraction. At this time, this instance is not considered a Singleton[8], as translation 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 Correction 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.

[8] See Section 7.1

12.10 Correction Data Abstraction

Correction Data is a data abstraction, and has the responsibility to manage access to the data read from a Kapton Correction File. To fulfill this responsibility, this abstraction presents an interface of one size constant, one Set contract and one Get contract, as shown in Figure 33. The copy constructor, assignment operator, or Set contract can be used by Correction File to initialize the abstraction. The Get contract allows retrieval of all the data.

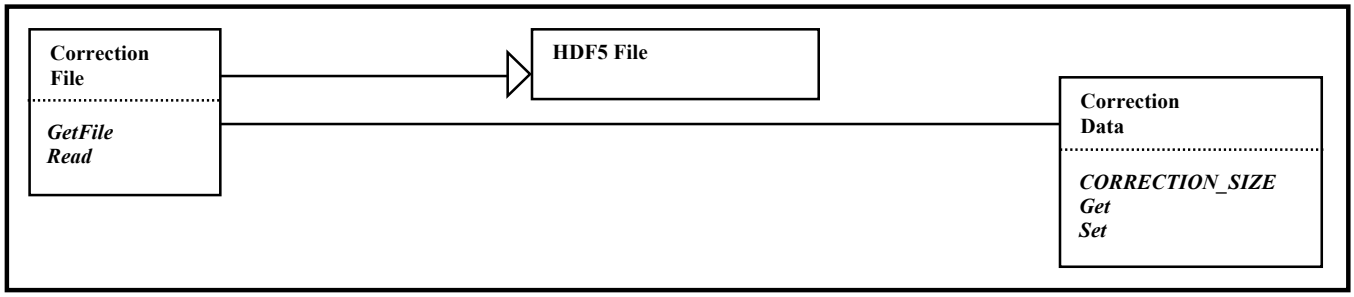

Figure 33 Correction File and Correction Data Abstractions

13 Obscuration Corrector Package

The Obscuration Corrector package has the responsibility to encapsulate all abstractions necessary to provide the system a means to correct for the field-of-view obscuration effect in HIRDLS1C scans. As shown in Figure 1, this package is used by the Processor package, and has access to the HIRDLS1C, File, 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.

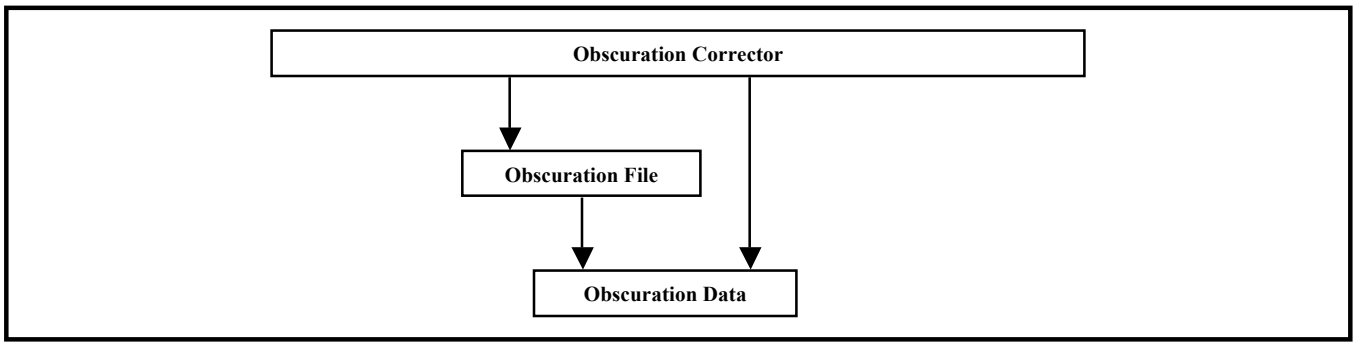

Figure 34 Oscillation Corrector Package Hierarchy

13.1 Obscuration Corrector Abstraction

Obscuration Corrector is a process abstraction, and has the responsibility to correct for the field-of-view obscuration effects on a HIRDLS scan. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Manager, HIRDLS1C Scan, Obscuration File and Obscuration Data, and presents an interface of one Correct contract, as shown in Figure 35. The Correct contract applies the obscuration data (read in on instantiation) to all HIRDLS1C scan radiances and/or errors. Since this abstraction needs to initialize itself with obscuration data before it can correct scans, it needs to be persistent to work efficiently.

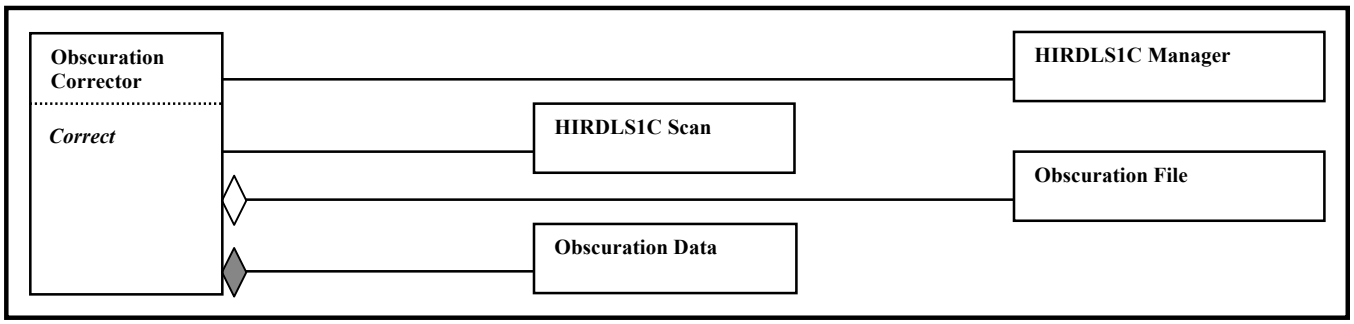

Figure 35 Obscuration Corrector Abstraction

13.2 Obscuration File Abstraction

Obscuration File is a control abstraction, and has the responsibility to manage all access to an obscuration file. To fulfill this responsibility, this abstraction collaborates with HDF5 File and Obscuration Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 36. The GetFile contract returns an instance of an Obscuration File abstraction. At this time, this instance is not considered a Singleton[9], as obscuration 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 Obscuration 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.

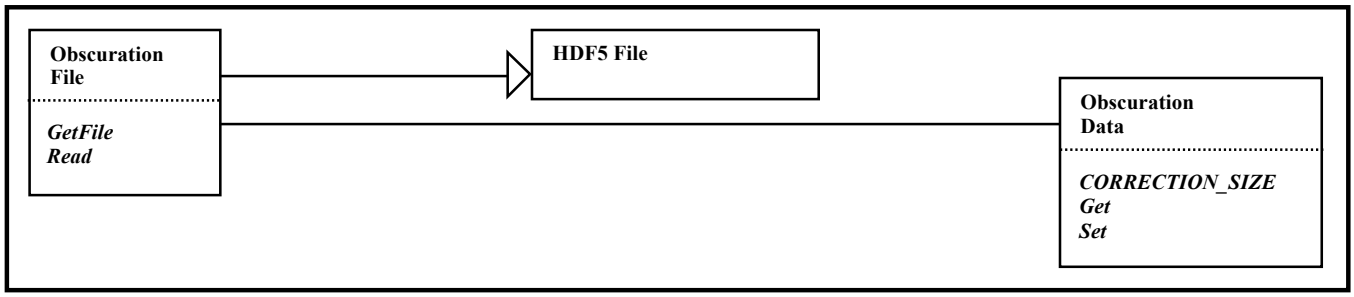

Figure 36 Obscuration File and Obscuration Data Abstractions

13.3 Obscuration Data Abstraction

Obscuration Data is a data abstraction, and has the responsibility to manage access to the data read from an Obscuration File. To fulfill this responsibility, this abstraction presents an interface of one size constant, one Set contract and one Get contract, as shown in Figure 36. The copy constructor, assignment operator, or Set contract can be used by Obscuration File to initialize the abstraction. The Get contract allows retrieval of all the data.

14 Error Corrector Package

The Error Corrector package has the responsibility to encapsulate all abstractions necessary to provide the system a means to correct for the error in the corrections in HIRDLS1C scans. As shown in Figure 1, this package is used by the Processor

[9] See Section 7.1

package, and has access to the HIRDLS1C, File, Service and Diagnostics packages. Figure 37 shows the hierarchy of the abstractions in the package. Each abstraction in the package is detailed further in this section.

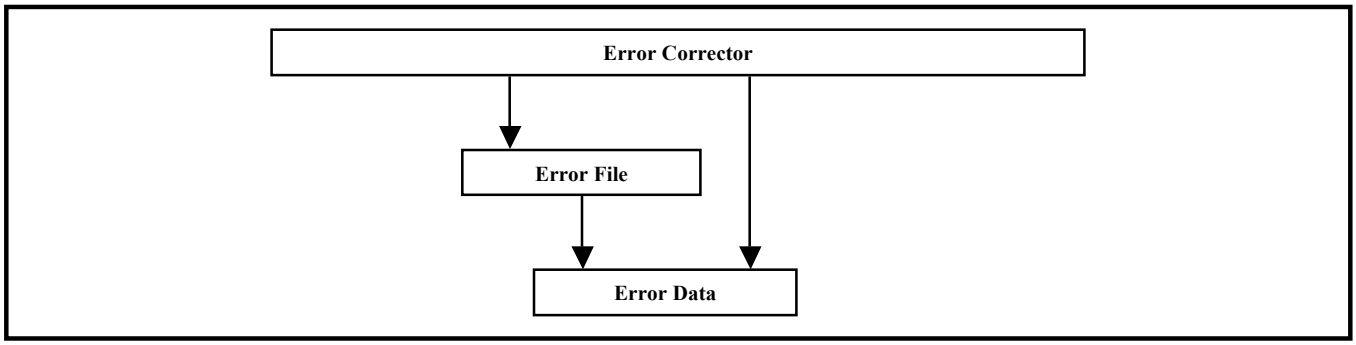

Figure 37 Error Corrector Package Hierarchy

14.1 Error Corrector Abstraction

Error Corrector is a process abstraction, and has the responsibility to correct the effect of the corrections on the radiance errors. To fulfill this responsibility, this abstraction collaborates with HIRDLS1C Scan, Error File and Error Data, and presents an interface of one Correct contract, as shown in Figure 38. The Correct contract applies the error effect data (read in on instantiation) to a HIRDLS1C scan errors. Since this abstraction needs to initialize itself with obscuration data before it can correct scans, it needs to be persistent to work efficiently.

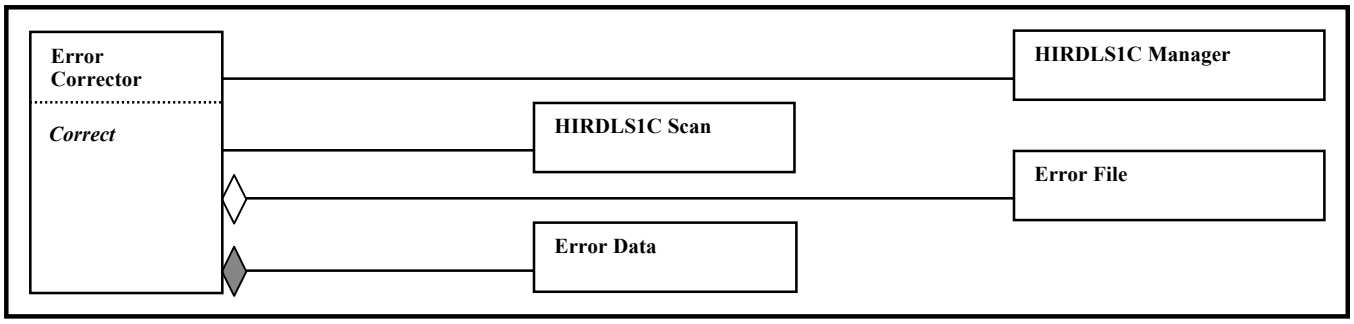

Figure 38 Error Corrector Abstraction

14.2 Error File Abstraction

Error File is a control abstraction, and has the responsibility to manage all access to an error effect file. To fulfill this responsibility, this abstraction collaborates with HDF5 File and Error Data, and presents an interface of one GetFile contract and one Read contract, as shown in Figure 39. The GetFile contract returns an instance of an Error File abstraction. At this time, this instance is not considered a Singleton[10], as obscuration 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 Error 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.

[10] See Section 7.1

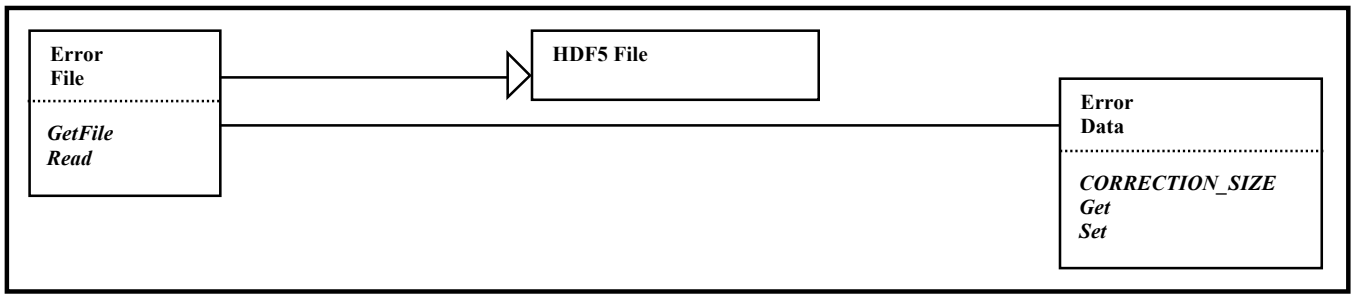

Figure 39 Error File and Error Data Abstractions

14.3 Error Data Abstraction

Error Data is a data abstraction, and has the responsibility to manage access to the data read from an Error File. To fulfill this responsibility, this abstraction presents an interface of one size constant, one Set contract and one Get contract, as shown in Figure 39. The copy constructor, assignment operator, or Set contract can be used by Error File to initialize the abstraction. The Get contract allows retrieval of all the data.

15 Processor Package

The Processor package has the responsibility to encapsulate all abstractions necessary to correct L1 data. As shown in Figure 1, this package is the highest-level package in the system, and has access to the HIRDLS1C, Oscillation Corrector, Kapton Corrector, Obscuration Corrector, Error Corrector, File, Service and Diagnostics packages. The only abstraction in this package is L1C Processor, and it is detailed further in this section.

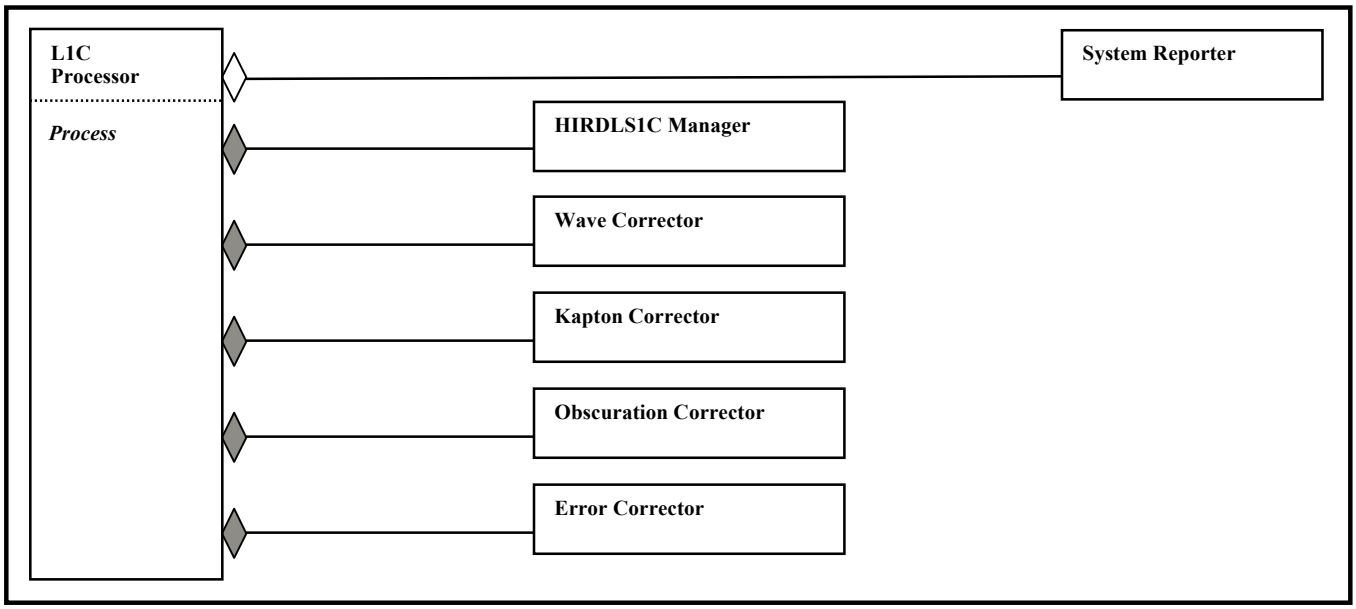

Figure 40 L1C Processor Abstraction

15.1 L1C Processor Abstraction

L1C Processor is a process abstraction, and has the responsibility to manage the correction of HIRDLS L1 scans. To fulfill this responsibility, this abstraction collaborates with System Reporter, HIRDLS1C Manager, Wave Corrector, Kapton Corrector, Obscuration Corrector and Error Corrector, and presents an interface of one Process contract, as shown in Figure 40. The Process contract first instantiates a System Reporter abstraction and a HIRDLS1C Manager abstraction, and then calls, in sequence: Wave Corrector, Kapton Corrector, Obscuration Corrector and Error Corrector, passing the HIRDLS1C Manager to each. 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 int const CHOPPERREV_SIZE
static int const MINORFRAME_SIZE
static int const SCAN_MAXIMUMSIZE
static int const MAFREVS_SIZE
static int const MIFREVS_SIZE
static int const FILEMAFS_MAXSIZE
static int const FILEMIFS_MAXSIZE
static int const FILEREVS_MAXSIZE
```

A.7 HDF5 Service

```
hdf5_service ()

~hdf5_service ()

static int const DIMENSIONS_MAXIMUMSIZE

bool CloseFile (long fileid) const
bool CloseSwath (long swathid) const
bool GetDimensionSize (long swathid, const string& name, long& size) const
bool GetFieldDimensions (long swathid, const string& fieldname, int& rank,
                         long dimensions[DIMENSIONS_MAXIMUMSIZE]) 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
```

A.8 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.9 Program Abortion Service

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

A.10 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.11 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.12 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.13 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.14 HDF5 File

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

~hdf5_file ()

# void Close ()
# 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[],
                  int& fillvalue, int* 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[],
                  double& fillvalue, double* data) const
# bool ReadField (const string& name, int dimensions, const long start[], const long edge[],
                  unsigned short& fillvalue, unsigned short* data) const
# bool WriteAttribute (const string& name, const string& value)
# bool WriteField (const string& name, int dimensions, const long starts[],
                   const long edges[], const void* data)
```

A.15 HIRDLS1C Manager

```
hirdls1c_manager ()

~hirdls1c_manager ()

bool ExtractScan (hirdls1c_scan& scan)
bool WriteScan (const hirdls1c_scan& scan)
```

A.16 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)
bool Write (enum hirdls1c_code code, const void* data)
```

A.17 Scan Discriminator

```
scan_discriminator ()

~scan_discriminator ()

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

A.18 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.19 HIRDLS1C Scan

```
hirdls1c_scan ()

~hirdls1c_scan ()

inline bool IsCorrectable () const
inline bool IsUpScan () const
void Demoninalize ()
void Get (int& scanlength, float elevations[SCAN_MAXIMUMSIZE],
          double radiances[CHANNEL_SIZE][ SCAN_MAXIMUMSIZE]) const
void Get (int& scanlength, float elevations[SCAN_MAXIMUMSIZE],
          double errors[CHANNEL_SIZE][ SCAN_MAXIMUMSIZE]) const
void Get (int& startrev, int& scanlength, short scannumbers[SCAN_MAXIMUMSIZE],
          int flags[SCAN_MAXIMUMSIZE], double radiances[CHANNEL_SIZE][ SCAN_MAXIMUMSIZE],
          double errors[CHANNEL_SIZE][ SCAN_MAXIMUMSIZE]) const
void Set (bool iscorrectable, int startrev, int scanlength, short scannumbersfill,
          const short scannumbers[SCAN_MAXIMUMSIZE], const int flags[SCAN_MAXIMUMSIZE],
          const float elevations[SCAN_MAXIMUMSIZE],
```

```
          const double radiances[CHANNEL_SIZE][ SCAN_MAXIMUMSIZE],
          const double errors[CHANNEL_SIZE][ SCAN_MAXIMUMSIZE])
void Reset (const double radiances[CHANNEL_SIZE][SCAN_MAXIMUMSIZE]
            const double errors[CHANNEL_SIZE][SCAN_MAXIMUMSIZE])
```

A.20 Wave Corrector

```
wave_correcter ()

~wave_correcter ()

bool Correct (hirdsl1c_manager& hirdls1cmanager)
```

A.21 Remover Creator

```
static st00_waveremover* Create (short scantable, double starttime)
```

A.22 ST00 Wave Remover

```
st00_waveremover ()

~st00_waveremover ()

#bool Remove (bool isupscan, int highfrequencyfilterpoints, int oscillationfilterpoints,
              int eigenvectorsize, int scanlength,
              const float scanelevations[SCAN_MAXIMUMSIZE],
              const double scanradiances[CHANNEL_SIZE][SCAN_MAXIMUMSIZE],
              const double eofelevations[eof_data::ELEVATION_SIZE],
              const double eofscalefactors[CHANNEL_SIZE][eof_data::ELEVATION_SIZE],
              const double eofeigenvectors[eof_data::EIGENVECTOR_SIZE]
                                          [eof_data::ELEVATION_SIZE],
              double correctedradiances[CHANNEL_SIZE][SCAN_MAXIMUMSIZE]
              double correctederrors[CHANNEL_SIZE][SCAN_MAXIMUMSIZE])
bool Remove (hirdsl1c_scan& scan)
```

A.23 STXX Wave Remover

```
stXX_waveremover ()

~stXX_waveremover ()

bool Remove (hirdsl1c_scan& scan)
```

A.24 EOF File

```
static eof_file* GetFile ()

~eof_file ()

bool Read (eof_data& data)
```

A.25 EOF Data

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

```
~eof_data ()

static int const ELEVATION_SIZE
static int const EIGENVECTOR_SIZE

void Get (int& eigenvectorsize, int& highfrequencyfilterpoints, int& oscillationfilterpoints
          double elevations[ELEVATION_SIZE],
          double scalefactors[CHANNEL_SIZE][ELEVATION_SIZE],
          double eigenvectors[EIGENVECTOR_SIZE][ELEVATION_SIZE]) const
void Set (int eigenvectorsize, int highfrequencyfilterpoints, int oscillationfilterpoints
          const double elevations[ELEVATION_SIZE],
          const double scalefactors[CHANNEL_SIZE][ELEVATION_SIZE],
          const double eigenvectors[EIGENVECTOR_SIZE][ELEVATION_SIZE])
```

A.26 Kapton Corrector

```
kapton_correcter ()

~kapton_correcter ()

bool SpinUp (hirdls1c_manager& hirdls1cmanager)
bool Correct (hirdsl1c_manager& hirdls1cmanager)
```

A.27 Elevations File

```
static elevations_file* GetFile ()

~elevations_file ()

bool Read (elevations_data& data)
```

A.28 Elevations Data

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

~elevations_data ()

void Get (double elevations[CHANNEL_SIZE]) const
void Set (const double elevations[CHANNEL_SIZE])
```

A.29 Translations File

```
static translations_file* GetFile ()

~translations_file ()

bool Read (translations_data& data)
```

A.30 Translations Data

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

~translations_data ()
```

```
void Get (double translations[CHANNEL_SIZE]) const
void Set (const double translations[CHANNEL_SIZE])
```

A.31 SpinUp Data Creator

```
spinupdata_creator (const double elevations[correction_data::CORRECTION_SIZE])

~spinupdata_creator ()

void Add (hirdls1c_scan& scan)
void Retrieve (spinup_data& data) const
```

A.32 SpinUp Data

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

~spinup_data ()

void Get (int& count, double upradiances[CHANNEL_SIZE][correction_data::CORRECTION_SIZE],
          double downradiances[CHANNEL_SIZE][correction_data::CORRECTION_SIZE]) const
void Set (int count,
          const double upradiances[CHANNEL_SIZE][correction_data::CORRECTION_SIZE],
          const double downradiances[CHANNEL_SIZE][correction_data::CORRECTION_SIZE])
```

A.33 Correction Data Loader

```
correctiondata_loader ()

~correctiondata_loader ()

void Load (double time, correction_data& data)
```

A.34 Correction File

```
static correction_file* GetFile ()

~correction_file ()

bool Read (correction_data& data)
```

A.35 Correction Data

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

~correction_data ()

static int const CORRECTION_SIZE

void Get (double elevations[ELEVATION_SIZE], int up2x2d2channels[CHANNEL_SIZE],
          int down2x2d2channels[CHANNEL_SIZE], double up2x2d2scales[CHANNEL_SIZE],
          double down2x2d2scales[CHANNEL_SIZE],
          double uptimemeans[CHANNEL_SIZE][ELEVATION_SIZE],
          double downtimemeans[CHANNEL_SIZE][ELEVATION_SIZE],
          double upvectors[3][CHANNEL_SIZE][ELEVATION_SIZE],
```

```
          double downvectors[3][ CHANNEL_SIZE][ELEVATION_SIZE]) const
void Set (const double elevations[ELEVATION_SIZE], const int up2x2d2channels[CHANNEL_SIZE],
          const int down2x2d2channels[CHANNEL_SIZE],
          const double up2x2d2scales[CHANNEL_SIZE],
          const double down2x2d2scales[CHANNEL_SIZE],
          const double uptimemeans[CHANNEL_SIZE][ELEVATION_SIZE],
          const double downtimemeans[CHANNEL_SIZE][ELEVATION_SIZE],
          const double upvectors[3][CHANNEL_SIZE][ELEVATION_SIZE],
          const double downvectors[3][ CHANNEL_SIZE][ELEVATION_SIZE])
```

A.36 Obscuration Corrector

```
obscuration_correcter ()

~obscuration_correcter ()

bool Correct (hirdsl1c_manager& hirdls1cmanager)
```

A.37 Obscuration File

```
static obscuration_file* GetFile ()

~obscuration_file ()

bool Read (obscuration_data& data)
```

A.38 Obscuration Data

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

~obscuration_data ()

static int const CORRECTION_SIZE

void Get (float elevations[CORRECTION_SIZE],
          double fractions[CHANNEL_SIZE][CORRECTION_SIZE]) const
void Set (const float elevations[CORRECTION_SIZE]
          const double fractions[CHANNEL_SIZE][CORRECTION_SIZE])
```

A.39 Error Corrector

```
error_correcter ()

~error_correcter ()

bool Correct (hirdsl1c_manager& hirdls1cmanager)
```

A.40 Error File

```
static error_file* GetFile ()

~error_file ()

bool Read (error_data& data)
```

A.41 Error Data

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

~error_data ()

static int const CORRECTION_SIZE

void Get (double elevations[CORRECTION_SIZE],
          double values[CHANNEL_SIZE][CORRECTION_SIZE]) const
void Set (const double elevations[CORRECTION_SIZE]
          const double values[CHANNEL_SIZE][CORRECTION_SIZE])
```

A.42 L1C Processor

```
l1c_processor ()

~l1c_processor ()

void Process ()
```