GOES-R AIT

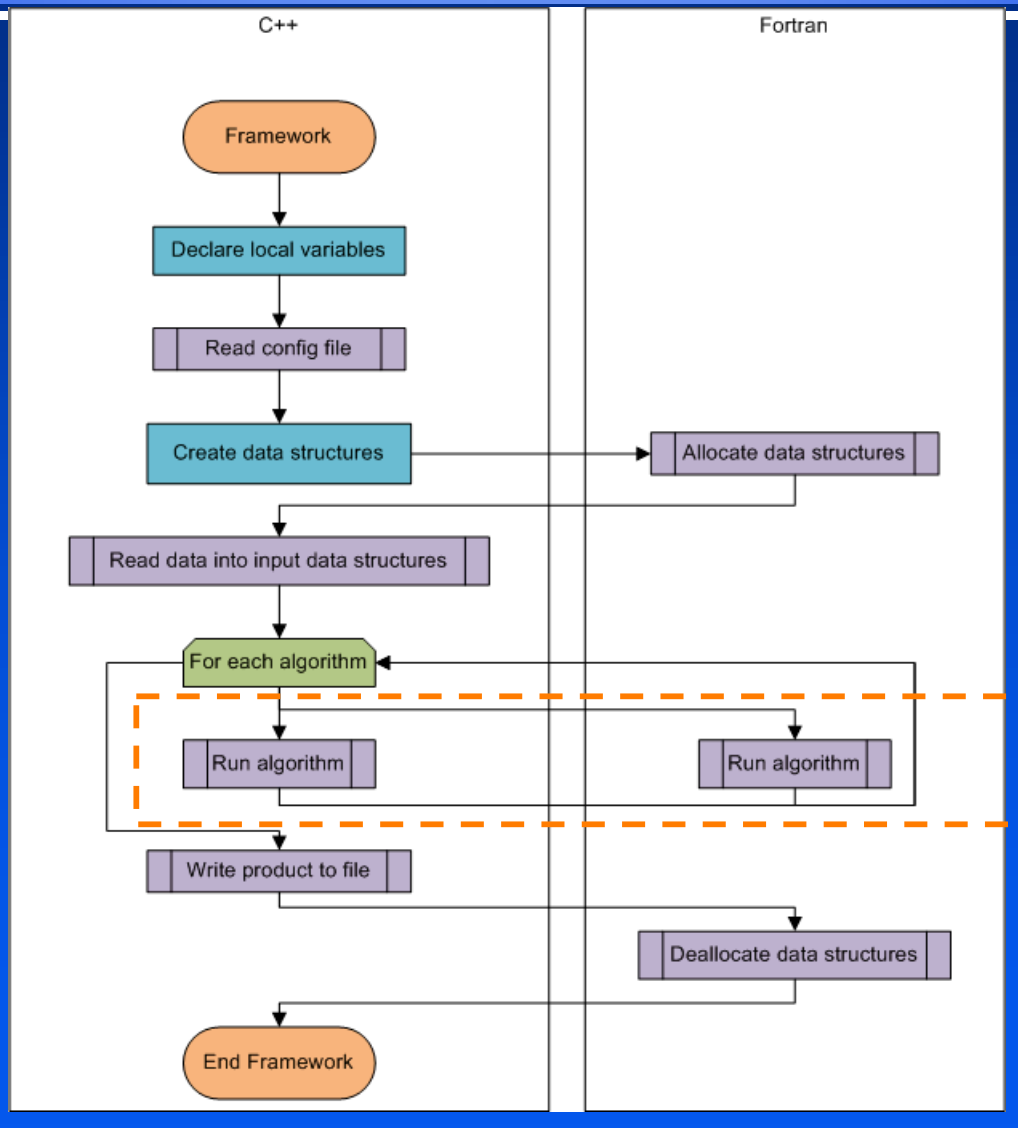
Algorithm Processing Framework

Presented by

Processing Framework Overview

- Framework written in C/C++
- Uses structures to pass information between C++ and Fortran
- C++ and Fortran data structures need to be synchronized
  - padding required in C++ structures for certain types
  - padding is compiler dependent
  - padding is rank dependent
- Calls Fortran routines to allocate/deallocate memory
- Loads input data
- Calls algorithms
- Writes results to disk

Processing Framework Flowchart

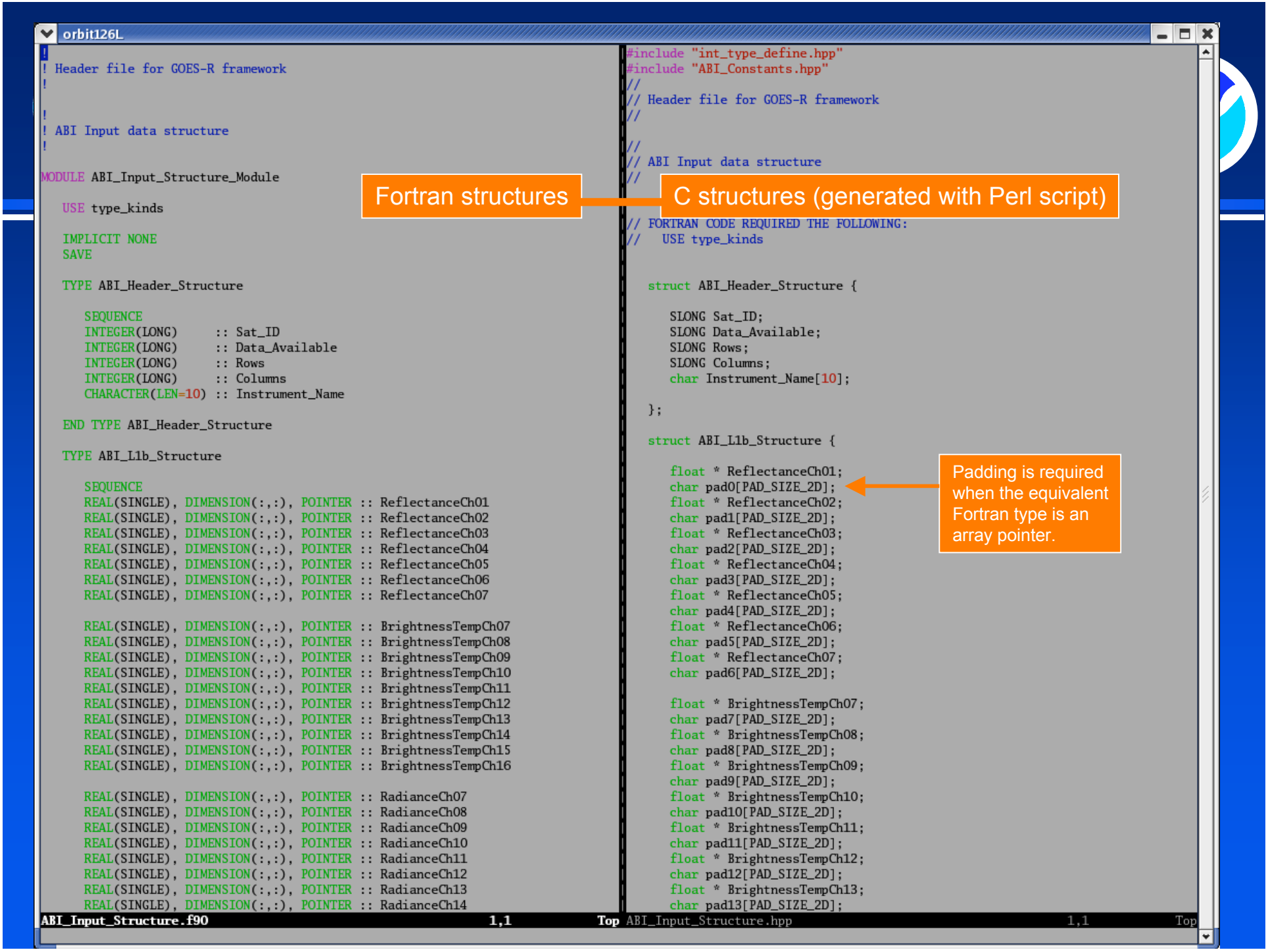


Algorithms can be written in C or Fortran

Processing Framework Tools

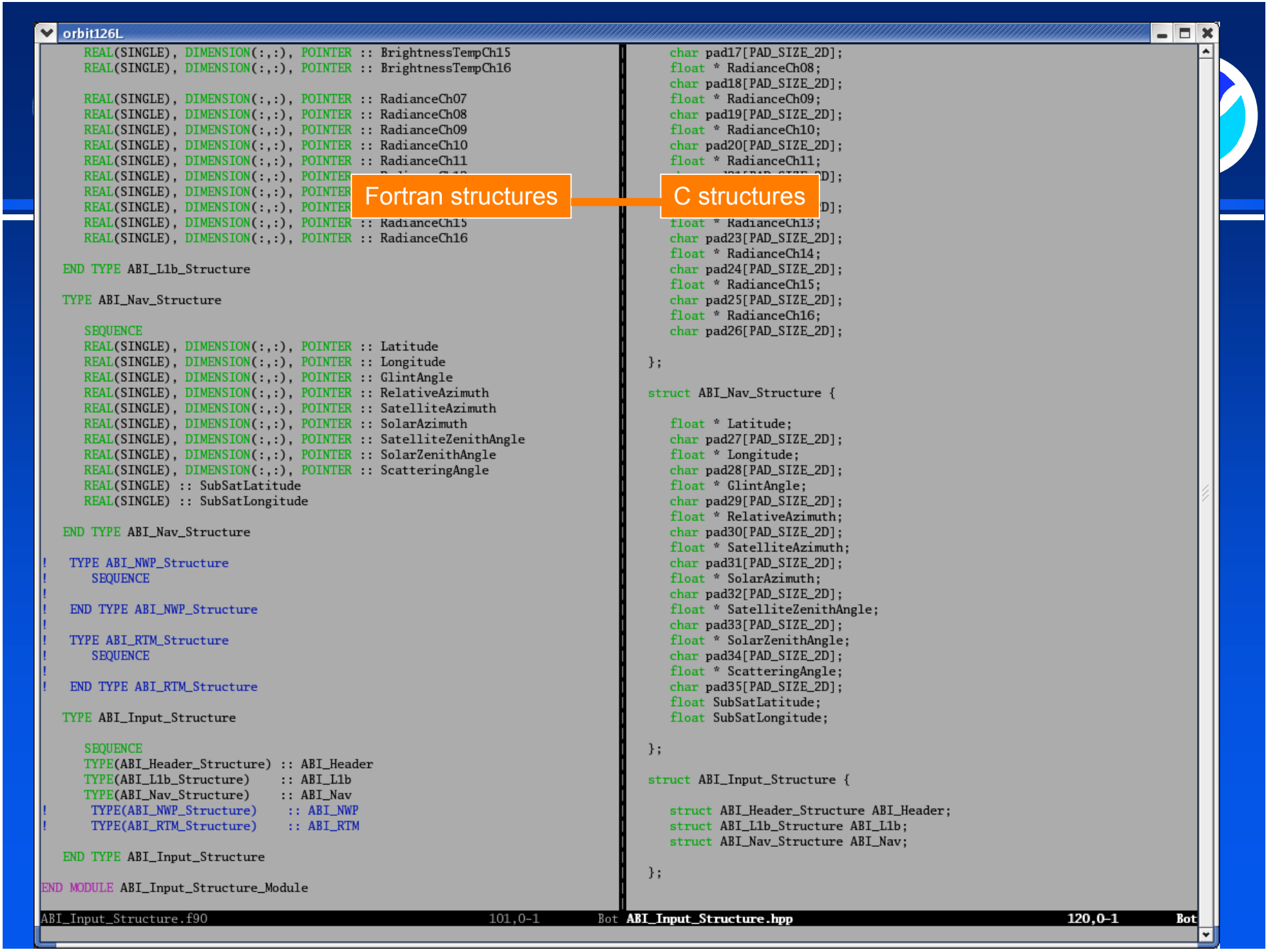
- Perl Scripts
  - Generate equivalent C structures from the Fortran structures
  - Generate Fortran allocate and deallocate routines from the information contained in the structures

Allows us to make changes to the structures in one place (Fortran)… then automatically update the code for the C structures, and Fortran allocate/deallocate.




```
!
! Header file for GOES-R framework
!


!
! ABI Input data structure
!

MODULE ABI_Input_Structure_Module

   USE type_kinds

   IMPLICIT NONE
   SAVE

   TYPE ABI_Header_Structure

      SEQUENCE
      INTEGER(LONG)      :: Sat_ID
      INTEGER(LONG)      :: Data_Available
      INTEGER(LONG)      :: Rows
      INTEGER(LONG)      :: Columns
      CHARACTER(LEN=10) :: Instrument_Name

   END TYPE ABI_Header_Structure

   TYPE ABI_L1b_Structure

      SEQUENCE
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh01
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh02
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh03
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh04
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh05
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh06
      REAL(SINGLE), DIMENSION(:,:), POINTER :: ReflectanceCh07

      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh07
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh08
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh09
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh10
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh11
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh12
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh13
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh14
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh15
      REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh16

      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh07
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh08
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh09
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh10
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh11
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh12
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh13
      REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh14
```

```
#include "int_type_define.hpp"
#include "ABI_Constants.hpp"
//
// Header file for GOES-R framework
//


//
// ABI Input data structure
//

// FORTRAN CODE REQUIRED THE FOLLOWING:
//   USE type_kinds


   struct ABI_Header_Structure {

      SLONG Sat_ID;
      SLONG Data_Available;
      SLONG Rows;
      SLONG Columns;
      char Instrument_Name[10];

   };

   struct ABI_L1b_Structure {

      float * ReflectanceCh01;
      char pad0[PAD_SIZE_2D];
      float * ReflectanceCh02;
      char pad1[PAD_SIZE_2D];
      float * ReflectanceCh03;
      char pad2[PAD_SIZE_2D];
      float * ReflectanceCh04;
      char pad3[PAD_SIZE_2D];
      float * ReflectanceCh05;
      char pad4[PAD_SIZE_2D];
      float * ReflectanceCh06;
      char pad5[PAD_SIZE_2D];
      float * ReflectanceCh07;
      char pad6[PAD_SIZE_2D];

      float * BrightnessTempCh07;
      char pad7[PAD_SIZE_2D];
      float * BrightnessTempCh08;
      char pad8[PAD_SIZE_2D];
      float * BrightnessTempCh09;
      char pad9[PAD_SIZE_2D];
      float * BrightnessTempCh10;
      char pad10[PAD_SIZE_2D];
      float * BrightnessTempCh11;
      char pad11[PAD_SIZE_2D];
      float * BrightnessTempCh12;
      char pad12[PAD_SIZE_2D];
      float * BrightnessTempCh13;
      char pad13[PAD_SIZE_2D];
```


orbit126L
REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh15
REAL(SINGLE), DIMENSION(:,:), POINTER :: BrightnessTempCh16
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh07
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh08
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh09
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh10
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh11
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh15
REAL(SINGLE), DIMENSION(:,:), POINTER :: RadianceCh16
END TYPE ABI_L1b_Structure
TYPE ABI_Nav_Structure
SEQUENCE
REAL(SINGLE), DIMENSION(:,:), POINTER :: Latitude
REAL(SINGLE), DIMENSION(:,:), POINTER :: Longitude
REAL(SINGLE), DIMENSION(:,:), POINTER :: GlintAngle
REAL(SINGLE), DIMENSION(:,:), POINTER :: RelativeAzimuth
REAL(SINGLE), DIMENSION(:,:), POINTER :: SatelliteAzimuth
REAL(SINGLE), DIMENSION(:,:), POINTER :: SolarAzimuth
REAL(SINGLE), DIMENSION(:,:), POINTER :: SatelliteZenithAngle
REAL(SINGLE), DIMENSION(:,:), POINTER :: SolarZenithAngle
REAL(SINGLE), DIMENSION(:,:), POINTER :: ScatteringAngle
REAL(SINGLE) :: SubSatLatitude
REAL(SINGLE) :: SubSatLongitude
END TYPE ABI_Nav_Structure
! TYPE ABI_NWP_Structure
! SEQUENCE
! END TYPE ABI_NWP_Structure
! TYPE ABI_RTM_Structure
! SEQUENCE
! END TYPE ABI_RTM_Structure
TYPE ABI_Input_Structure
SEQUENCE
TYPE(ABI_Header_Structure) :: ABI_Header
TYPE(ABI_L1b_Structure) :: ABI_L1b
TYPE(ABI_Nav_Structure) :: ABI_Nav
! TYPE(ABI_NWP_Structure) :: ABI_NWP
! TYPE(ABI_RTM_Structure) :: ABI_RTM
END TYPE ABI_Input_Structure
END MODULE ABI_Input_Structure_Module
Fortran structures
C structures
char pad17[PAD_SIZE_2D];
float * RadianceCh08;
char pad18[PAD_SIZE_2D];
float * RadianceCh09;
char pad19[PAD_SIZE_2D];
float * RadianceCh10;
char pad20[PAD_SIZE_2D];
float * RadianceCh11;
float * RadianceCh13;
char pad23[PAD_SIZE_2D];
float * RadianceCh14;
char pad24[PAD_SIZE_2D];
float * RadianceCh15;
char pad25[PAD_SIZE_2D];
float * RadianceCh16;
char pad26[PAD_SIZE_2D];
};
struct ABI_Nav_Structure {
float * Latitude;
char pad27[PAD_SIZE_2D];
float * Longitude;
char pad28[PAD_SIZE_2D];
float * GlintAngle;
char pad29[PAD_SIZE_2D];
float * RelativeAzimuth;
char pad30[PAD_SIZE_2D];
float * SatelliteAzimuth;
char pad31[PAD_SIZE_2D];
float * SolarAzimuth;
char pad32[PAD_SIZE_2D];
float * SatelliteZenithAngle;
char pad33[PAD_SIZE_2D];
float * SolarZenithAngle;
char pad34[PAD_SIZE_2D];
float * ScatteringAngle;
char pad35[PAD_SIZE_2D];
float SubSatLatitude;
float SubSatLongitude;
};
struct ABI_Input_Structure {
struct ABI_Header_Structure ABI_Header;
struct ABI_L1b_Structure ABI_L1b;
struct ABI_Nav_Structure ABI_Nav;
};
ABI_Input_Structure.f90 101,0-1 Bot
ABI_Input_Structure.hpp 120,0-1 Bot

Interface between Framework and Algorithms

- Data structures for input data within the framework.
- Data structures within the algorithm code to move the data around.
- Interface subroutine the match the data in the two data structures.
- Data matching can be instrument specific; case or if statements within the interface subroutine.

Delivered Algorithm Package (DAP)

✓ **Source Code**

This includes all the source code necessary to build your algorithm, as well as any scripts or makefiles used in the build process. Code to test your algorithm should also be included.

✓ **Tools required for compilation**

This includes things like NetCDF and HDF libraries and compilers that were used to compile the code.

✓ **Test Information**

This includes descriptions, plans, and/or procedures on how to test your algorithm. Any performance testing results you may have generated should also be included.

✓ **Data Files**

This includes test input data files, ancillary data files, and the test output files for comparison to our runs.

✓ **Documentation**

This includes all information on how to build, run, and verify the output of your algorithm. It also includes such things as:

- Flow Diagrams
- Production Rules
- Process Control File
- Algorithm Theoretical Basis Documents (ATBDs)
- Software Implementation Document
- List of known latency and accuracy issues, error handling/messaging codes, and resources for execution.

Deliveries

- Three deliveries with a possible Delta delivery
  - » Delivery I
    - – Working code
  - » Delivery II
    - – Working code that meets the standards (80%).
    - – Draft ATBD should be done by this point.
  - » Delivery III
    - – Working code that meets standards and accuracy
    - – All documentation included.
  - » Delta Delivery – Just science algorithms.

Algorithm Acceptance Procedure

1. **Check-out from CM**
   Check-out a working copy of the algorithm package from revision control.

2. **Makefile**
   Verify a Makefile exists to build the entire package. Modify the Makefile to use our compilers.

3. **Compile**
   Compile the source code with our compilers.
   Linux – PGI and Intel
   IBM – XL

4. **Verify output**
   Run the code as-is to verify our output matches their output

5. **Valgrind**
   Valgrind is a suite of tools for debugging and profiling Linux programs. Memcheck is one of these tools which can check for memory errors. Compile your program with -g to include debugging information so that Memcheck's error messages include exact line numbers.

Algorithm Acceptance Procedure (cont.)

6. **Profiling**
   Profiling gives an indication of how much time is spent in each function. Compile the source code with the –pg option (for gprof use) or the –Mprof=func option (for pgprof use). Run the code in its normal way and when it is done there should be a gmon.out (or pgprof.out if –Mprof=func was used) file to analyze with gprof (or pgprof if –Mprof=func was used).

7. **Forcheck** (fortran code)
   Use Forecheck on Fortran code to check for non-standard source code and other bugs that may not have been caught by the compiler.

8. **Lint** (C code)
   Use lint (or splint, which claims to be a better lint) on C code to check for common programming mistakes

9. **Code Checker**
   Use our in house code checking tools to verify the source code meets our coding standards.

10. **Commentary**
    Perform a Peer Review to make sure comments in the code are readable, concise, and informative.

11. **Check-in to CM**
    Check-in the updated algorithm package to revision control.

Flowchart Status

Item Received/Click to download Item Not Available

Flowchart Package Status							
	Top Level Flowchart	Detailed Flowchart	Subroutine Flowchart	Subroutine Table	Calling Tree	I/O Table	Directory Listing
Active							
Air Quality							
Aerosol/SO2	●	●	●	●	●	●	●
Aerosol	●	●	●	●	●	●	●
Trace Gas/Emissions	●	●	●	●	●	●	●
Total Ozone	●	●	●	●	●	●	●
Clouds	●	●	●	●	●	●	●
Land							
LST	●	●	●	●	●	●	●
NDVI	●	●	●	●	●	●	●
Fire	●	●	●	●	●	●	●
Space Weather	●	●	●	●	●	●	●
Soundings	●	●	●	●	●	●	●
SST	●	●	●	●	●	●	●
Winds							
AMV	●	●	●	●	●	●	●
Hurricane Intensity	●	●	●	●	●	●	●
Just Starting							
Aviation	●	●	●	●	●	●	●
Cryosphere	●	●	●	●	●	●	●
Hydrology	●	●	●	●	●	●	●
Ocean Dynamics	●	●	●	●	●	●	●
Radiation Budget							
OLR	●	●	●	●	●	●	●
SW-NASA	●	●	●	●	●	●	●
SW-STAR	●	●	●	●	●	●	●