Effective Date: August 09, 2023
Expiration Date: August 09, 2038

GSFC JPSS CMO
November 08, 2023
Released

474-00448-03-03, Revision A
Joint Polar Satellite System (JPSS) Code 474

Joint Polar Satellite System Algorithm Specification Volume III: Operational Algorithm Description (OAD) for the CrIS RDR/SDR

Goddard Space Flight Center
Greenbelt, Maryland

NOAA / NASA

Joint Polar Satellite System
Algorithm Specification Volume III:
Operational Algorithm Description (OAD) for the CrIS RDR/SDR

Review/Signature/Approval Page

Prepared by:

LEO Ground Services Project SE

Approved by:

Kellyann Jeletic
LEO Ground Services Project SEIT Lead

Nicolaie Todirita
LEO Ground Services Project Manager

Electronic Approval available on-line at: https://jpssmis.gsfc.nasa.gov/frontmenu_dsp.cfm

Preface

This document is under JPSS Ground configuration control. Once this document is approved, JPSS approved changes are handled in accordance with Class I and Class II change control requirements as described in the JPSS Configuration Management Procedures, and changes to this document shall be made by complete revision.

Any questions should be addressed to:

JPSS Configuration Management Office
NASA/GSFC
Code 474
Greenbelt, MD 20771

Change History Log

Revision	Effective Date	Description of Changes (Reference the CCR & CCB/ERB Approval Date)
Rev -	Aug 08, 2013	This was approved by the JPSS Ground ERB via **474-CCR-13-1110** on the effective date shown.
Rev -1	Oct 23, 2014	This version incorporates **474-CCR-14-2094** which was approved by JPSS Ground ERB on the effective date shown. This version was baselined in support of Change Order 10.
Rev -1 - Revalidated	Dec 11, 2019	The expiration date on this document has been extended five years by 474-CCR-19-4789, which was approved by the JPSS Ground AERB on Dec 11, 2019. The effective date and the revision/version on this document will remain the same, only the expiration date has been changed.
Rev A	Aug 09, 2023	This version incorporates 474-CCR-23-6516 which was approved by the Ground ERB on May 01, 2023 and the Ground CCB on May 24, 2023 and 474-CCR-23-6461 which was approved by the Ground Segment ERB on Aug 04,2023 and by the JPSS Ground Segment CCB on the effective date shown.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

1.1 Objective

The purpose of this Operational Algorithm Description (OAD) document is to express, in computer-science terms, the remote sensing algorithms that produce the Joint Polar Satellite System (JPSS) end-user data products. These products are individually known as Raw Data Records (RDRs), Temperature Data Records (TDRs), Sensor Data Records (SDRs) and Environmental Data Records (EDRs). In addition, any Intermediate Products (IPs) produced in the process. The OAD provides a software description of that science as implemented in the operational ground system.

The purpose of an OAD is two-fold:

- Provide initial implementation design guidance to the operational software developer.
- Capture the "as-built" operational implementation of the algorithm reflecting any changes needed to meet operational performance/design requirements.

An individual OAD document describes one or more algorithms used in the production of one or more data products. This particular document describes operational software implementation for the Cross-track Infrared Sounder (CrIS) Sensor Data Record (SDR).

1.2 Scope

The scope of this document is limited to the description of the core operational algorithm(s) required to create the CRIS FS SDR. The basis for the geolocation algorithm is described in this document.

2 RELATED DOCUMENTATION

The latest JPSS documents can be obtained from URL: https://jpssmis.gsfc.nasa.gov/frontmenu_dsp.cfm. JPSS Project documents have a document number starting with 470, 472 or 474 indicating the governing Configuration Control Board (CCB) (Program, Flight or Ground) that has the control authority of the document.

2.1 Parent Documents

The following reference document is the Parent Document from which this document has been derived. Any modification to a Parent Document will be reviewed to identify the impact upon this document. In the event of a conflict between a Parent Document and the content of this document, the JPSS Program CCB has the final authority for conflict resolution.

Document Number	Title
474-00448-01-03	JPSS Algorithm Specification Volume I: SRS for the CRIS RDR/SDR

2.2 Applicable Documents

The following documents are the Applicable Documents from which this document has been derived. Any modification to an Applicable Document will be reviewed to identify the impact upon this document. In the event of conflict between an Applicable Document and the content of this document, the JPSS Program CCB has the final authority for conflict resolution.

Document Number	Title
429-05-02-42	NPP Mission Data Format Control Book (MDFCB)
429-05-02-42-02	NPP MDFCB Appendix A
472-00251	Mission Data Format Control Book (MDFCB) Joint Polar Satellite System-1 (JPSS-1)
474-00448-02-03	JPSS Algorithm Specification Volume II: Data Dictionary for the CRIS RDR/SDR
474-00448-04-03	JPSS Algorithm Specification Volume IV: SRS Parameter File (SRSPF) for the CRIS RDR/SDR

3 ALGORITHM OVERVIEW

This document presents the theoretical basis of the CrIS SDR Algorithms. The functional flow of algorithms required to transform a Raw Data Record (RDR) coming from the satellite into a Sensor Data Record (SDR) is described.

This document describes the CrIS SDR Algorithms specific to processing required at the ground segment. It covers the processing needs for all data being sent to the ground when the instrument is operational, including observational and calibration data, for all measurements performed by the instrument. The algorithms for decoding and calibrating the calibration data (e.g., generation of Internal Calibration Target (ICT) radiance) are also covered here.

Figure 3-1 identifies the top level processing flow of the CrIS SDR Algorithm.

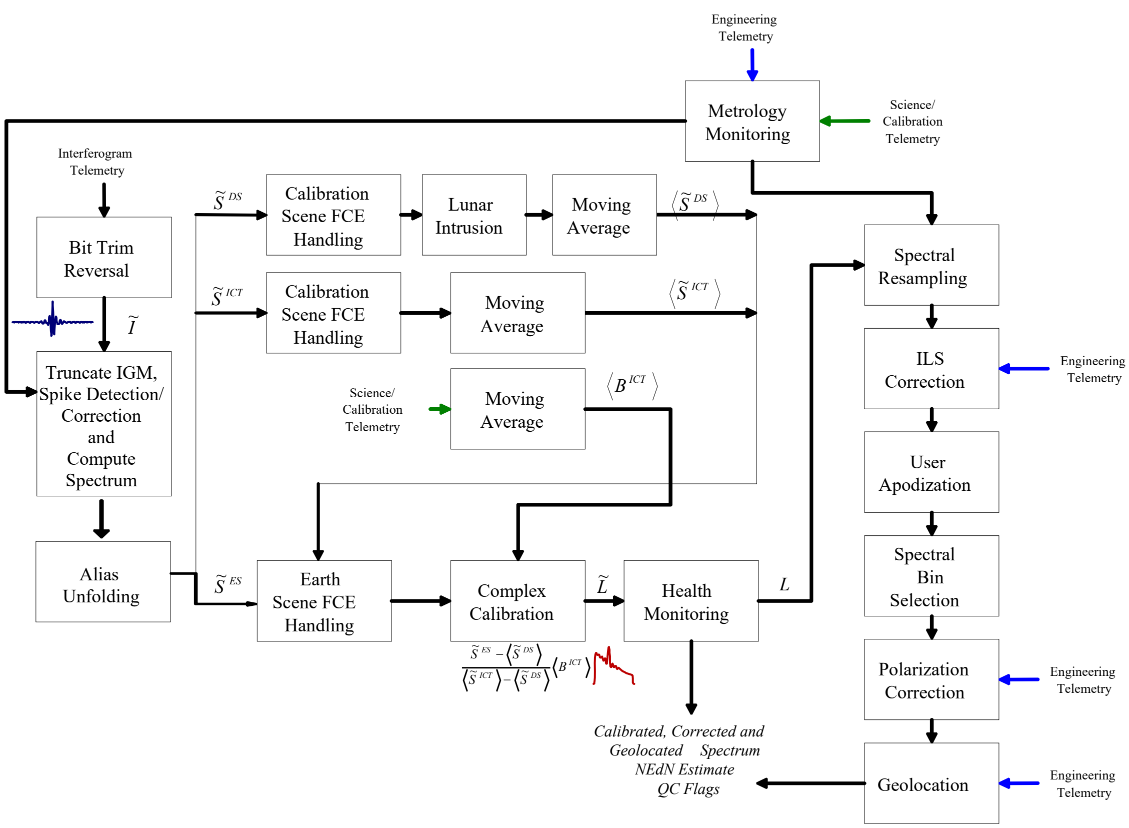

Figure 3-1. Processing Chain

(Note: Figure 3-1 needs to be updated to include nonlinearity correction.)

3.1 CrIS SDR Description

3.1.1 Interfaces

To begin data processing, the Infrastructure (INF) Subsystem Software Item (SI) initiates the CrIS SDR algorithm. The INF SI provides tasking information to the algorithm indicating which granule to process. The Data Management Subsystem (DMS) SI provides data storage and retrieval capability. CrIS SDR processing is retaskable, so instead of shutting down after processing it requests additional tasking information from INF and continues processing with this information. A library of C++ classes implements the SI interfaces, as depicted in Figure 3.1.1-1.

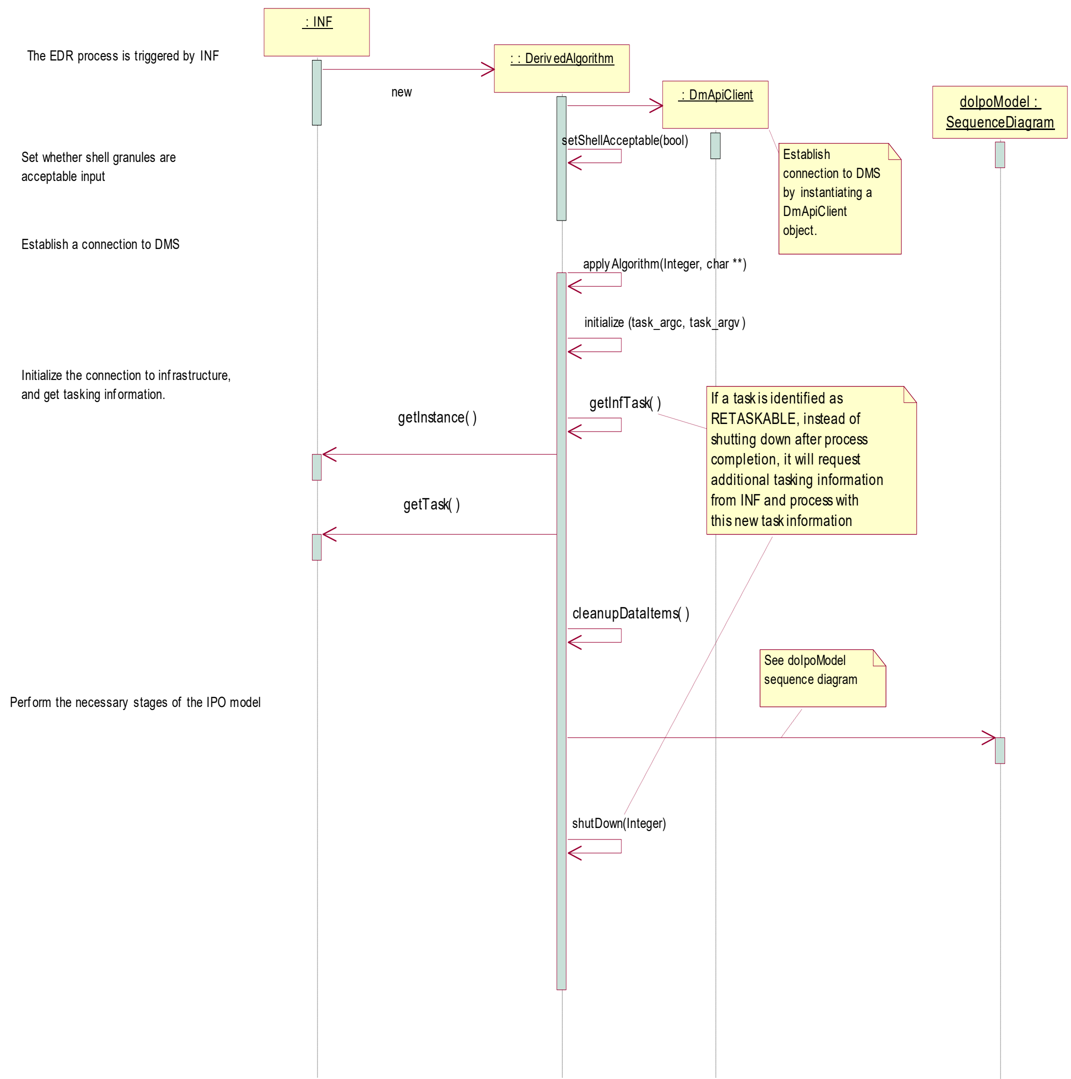

Figure 3.1.1-1. IPO Model Interface to INF and DMS

3.1.1.1 Inputs

Implementation of CrIS SDR requires CrIS RDRs, Space Craft Diary RDR, engineering calibration record, the correction matrix, the fill packet look-up table, and the tunable parameters. See Tables 3.1.1.1-1 and 3.1.1.1-2.

Table 3.1.1.1-1 CrIS SDR Truncated Spectral Resolution Inputs

Input	Description	Reference Document
CrIS Science RDR	The RDRs processed by the CrIS SDR algorithm are science RDRs of three types: interferogram packets, 8 s science/calibration packets (SciCalP) and 4min engineering packets (EP). An interferogram packet contains Earth Scene (ES), Deep Space (DS), or Internal Calibration Target (ICT) interferogram measurements. An 8s SciCalP is created every 8 seconds for each scan. It contains calibration data such as ICT and scan baffle temperature measurements. A 4 min EP is created every 4 minutes, or after every 30 scan measurements.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
Spacecraft Diary RDR	The JPSS-1 Spacecraft produces several application packets which are related to attitude and ephemeris.	474-00448-02-08_JPSS-DD-Vol-II-Part-8
CrIS Telemetry RDR	CrIS sensor telemetry data	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Correction Matrix AUX	The Cross-track Infrared Sounder (CrIS) Correction Matrix PC is applied to spectra as they are ejected from a sliding window. The 4-minute Engineering packet is used as input to create it. It is created at least once an orbit, estimated. The neon lamp measurements provide the measured laser wavelength that is stored in the Engineering packet binary file (CrIS-SDR-ENGPKT-BACKUP-AUX) or CrIS-FS-SDR-ENGPKT-BACKUP-AUX. The corresponding updated CMO matrix is stored in the CMO binary file (CrIS-Correct-Matrix-AUX) or CrIS-FS-Correct-Matrix-AUX. The FSR contains 2736 channels whereas the TSR has 1592.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR Engineering Packet Backup AUX	The Cross-track Infrared Sounder (CrIS) SDR Engineering Packet Backup PC provides tunable processing coefficients for use by the algorithm during execution. The coefficients can be modified (tuned) through a configuration control process in response to algorithm, performance, inputs, sensitivity, etc. changes. The TSR and FSR files have the same fields. Some fields have greater array size for FSR. Other fields, such as threshold parameters, have different values.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR DQTT	Data Quality Test Table	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR Processing Coefficients	The Cross-track Infrared Sounder (CrIS) SDR Ephemeral PC provides tunable processing coefficients for use by the algorithm during execution. The coefficients can be modified (tuned) through a configuration control process in response to algorithm, performance, inputs, sensitivity, etc. changes.	474-00448-02-03_JPSS-DD-Vol-II-Part-3

Input	Description	Reference Document
CrIS Fill Packet LUT	The CrIS Fill Packet PC contains templates of each of the Earth Scene, Deep Space, and Internal Calibration Target Interferogram packets (APIDs 1315-1395). These templates are used to create "fill" packets that are used to replace packets missing from the CrIS RDR inputs, in order to minimize the effect of missing packets to the CrIS sliding window processing.	474-00448-02-03_JPSS-DD-Vol-II-Part-3

Table 3.1.1.1-2. CrIS SDR Full Spectral Resolution Inputs

Input	Description	Reference Document
CrIS Science RDR	The RDRs processed by the CrIS SDR algorithm are science RDRs of three types: interferogram packets, 8 s science/calibration packets (SciCalP) and 4min engineering packets (EP). An interferogram packet contains Earth Scene (ES), Deep Space (DS), or Internal Calibration Target (ICT) interferogram measurements. An 8s SciCalP is created every 8 seconds for each scan. It contains calibration data such as ICT and scan baffle temperature measurements. A 4 min EP is created every 4 minutes, or after every 30 scan measurements.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
Spacecraft Diary RDR	The JPSS-1 Spacecraft produces several application packets which are related to attitude and ephemeris.	474-00448-02-08_JPSS-DD-Vol-II-Part-8
CrIS Telemetry RDR	CrIS sensor telemetry data	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum Correction Matrix AUX	The Cross-track Infrared Sounder (CrIS) Correction Matrix PC is applied to spectra as they are ejected from a sliding window. The 4-minute Engineering packet is used as input to create it. It is created at least once an orbit, estimated. The neon lamp measurements provide the measured laser wavelength that is stored in the Engineering packet binary file (CrIS-SDR-ENGPKT-BACKUP-AUX) or CrIS-FS-SDR-ENGPKT-BACKUP-AUX. The corresponding updated CMO matrix is stored in the CMO binary file (CrIS-Correct-Matrix-AUX) or CrIS-FS-Correct-Matrix-AUX. The FSR contains 2736 channels whereas the TSR has 1592.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum SDR Engineering Packet Backup AUX	The Cross-track Infrared Sounder (CrIS) SDR Engineering Packet Backup PC provides tunable processing coefficients for use by the algorithm during execution. The coefficients can be modified (tuned) through a configuration control process in response to algorithm, performance, inputs, sensitivity, etc. changes. The TSR and FSR files have the same fields. Some fields have greater array size for FSR. Other fields, such as threshold parameters, have different values.	474-00448-02-03_JPSS-DD-Vol-II-Part-3

Input	Description	Reference Document
CrIS Full Spectrum SDR DQTT	Data Quality Test Table	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum SDR Processing Coefficients	The Cross-track Infrared Sounder (CrIS) SDR Ephemeral PC provides tunable processing coefficients for use by the algorithm during execution. The coefficients can be modified (tuned) through a configuration control process in response to algorithm, performance, inputs, sensitivity, etc. changes.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum Fill Packet LUT	The CrIS Fill Packet PC contains templates of each of the Earth Scene, Deep Space, and Internal Calibration Target Interferogram packets (APIDs 1315-1395). These templates are used to create "fill" packets that are used to replace packets missing from the CrIS RDR inputs, in order to minimize the effect of missing packets to the CrIS sliding window processing.	474-00448-02-03_JPSS-DD-Vol-II-Part-3

The inputs to the CrIS SDR Algorithm via the CrIS RDR input consist of three types of CCSDS packets produced by the CrIS sensor. These include:

1. Interferogram Packet
    a. Internal Calibration Target (ICT) Interferogram Packet
    b. Deep Space (DS) Interferogram Packet
    c. Earth Scene (ES) Interferogram Packet
2. Four Minute Engineering Telemetry Packet
3. Eight Second Science/Calibration Telemetry Packet

The contents and formats of these packets are defined in the CrIS Command and Data Dictionary.

Tables 3.1.1.1-3, Table 3.1.1.1-4, and Table 3.1.1.1-5 identify the parameters within the Eight Second Science/Calibration and Four Minute Engineering Telemetry Packets that are monitored throughout processing.

Table 3.1.1.1-3. Parameters Monitored via the Eight Second Science/Calibration Telemetry Packet

Parameter	Comments	Units
IE CCA Calibration Resistor Temperature (epoch 1..40) - (refers to 200 msec epoch)	40 Readings	°C
Low Range Calibration Resistor (epoch 1..40)	40 Readings	°C
High Range Calibration Resistor (epoch 1..40)	40 Readings	°C
ICT Temperature #1 (epoch 1..40)	40 Readings	°C
ICT Temperature #2 (epoch 1..40)	40 Readings	°C
Cross Track Servo Error (sample 21, epoch 4..33)	30 Readings	microradians
In Track Servo Error (sample 21, epoch 4..33)	30 Readings	microradians
Laser Diode Current (epoch 39)	Single Reading	mAmps
Laser Diode Temperature (epoch 39)	Single Reading	°C
Beamsplitter Temperature #1 (epoch 39)	Single Reading	°C
OMA Structure Input Temperature #1 (epoch 39)	Single Reading	°C
OMA Structure Input Temperature #2 (epoch 39)	Single Reading	°C

Parameter	Comments	Units
SSM Scan Mirror Temperature (epoch 39)	Single Reading	°C
SSM Scan Mirror Baffle Temperature (epoch 39)	Single Reading	°K
Stage 2 Cooler Temperature (epoch 39)	Single Reading	°K
Stage 4 Cooler Temperature (epoch 39)	Single Reading	°K
Stage 1 Cooler Temperature (epoch 39)	Single Reading	°K
Stage 3 Cooler Temperature (epoch 39)	Single Reading	°K
Telescope Temperature #1 (epoch 39)	Single Reading	°C

Table 3.1.1.1-4. Parameters Monitored via the Four Minute Engineering Telemetry Packet

Parameter	Comments	Units
CRC of Four Minute Engineering Telemetry RDR	Used to trigger check of tunable parameters	NA
Spectral Calibration Parameters	Used to calculate metrology laser frequency	NA
Laser Pulses Counted Per Sweep	Used to calculate metrology laser frequency	Unitless
Number Of Neon Calibration Sweeps	Used to calculate metrology laser frequency	Unitless
Effective Neon Wavelength	Used to calculate metrology laser frequency	nm
Neon Calibration Time Stamp	Used to calculate metrology laser frequency	ms
Repeat Neon Calibration Interval	Used to calculate metrology laser frequency	ppm
Neon Calibration Data	One sample per Number Of Neon Calibration Sweeps	NA
Starting Count (Sample 1..128)	One sample per Number Of Neon Calibration Sweeps	unitless
Starting Partial Count (Sample 1..128)	One sample per Number Of Neon Calibration Sweeps	unitless
Fringe Count (Sample 1..128)	One sample per Number Of Neon Calibration Sweeps	unitless
Ending Partial Count (Sample 1..128)	One sample per Number Of Neon Calibration Sweeps	unitless
Ending Count (Sample 1..128)	One sample per Number Of Neon Calibration Sweeps	unitless

Table 3.1.1.1-5.. Tunable Parameters Provided via the Four Minute Engineering Telemetry RDR

Parameter	Comments	Units
Effective Neon Bulb Wavelength	Used in the computation of the laser frequency using Neon calibration data.	nm
Metrology laser wavelength offset	MW and SW offset from the LW metrology laser wavelength	ppm
ILS curve fit parameters	Used to correct for modulation efficiency variation with OPD	unitless
ILS FOV offset and size parameters	Use to correct off-axis pixel self apodization effects	micro radians
ICT Emissivity	Geometric factor to convert surface emissivity to effective emissivity	unitless
LW ICT Emissivity table	Wavenumber specific effective emissivities	unitless
MW ICT Emissivity table	Wavenumber specific effective emissivities	unitless
SW ICT Emissivity table	Wavenumber specific effective emissivities	unitless
Polarization Calibration	% Polarization difference relative to ICT at a specific wavenumber	unitless
Polarization Wave Numbers	Wavenumbers at which polarization calibration information is provided	cm-1
ICT environment model	Emissivities and view angles of instrument component included in the ICT radiance calculation	Temperature: K; Emissivity: unitless; View angle: degree; Planck function: mw / (m2.sr. cm-1)
ScanBaffleTemperatureBias	Scan baffle temperature correction applied at different spacecraft orbital positions (21 values)	K
Science TLM conversion coefficients	Engineering unit conversion coefficients associated with	unitless

	parameters monitored in eight second science/calibration telemetry RDR	
Science TLM Limits	Limits associated with parameters monitored in eight second science/calibration telemetry RDR	unitless
Mapping Parameters		
SSM crosstrack positions	Angles relative to SSM mounting feet	degree
SSM mirror misalignment	Mirror mount pitch and yaw errors	degree
SSM in-track position	Intrack commanded Nadir offset	degree
SSMR, SSMF, IAR , IFM boresight & SBF alignments	Alignment angles	degree
Time stamp bias	Bias added to interferogram time stamp to account for intrack motion compensation	ms
CrIS Bit Trim Mask LW	Used to reverse the bit trimming of interferogram packets	digital counts
CrIS Bit Trim Mask MW	Used to reverse the bit trimming of interferogram packets	digital counts
CrIS Bit Trim Mask SW	Used to reverse the bit trimming of interferogram packets	digital counts
LW Data Extraction Information		
Number of A/D samples	Used to reverse the bit trimming of interferogram packets	unitless
Decimation & number of filter taps	Used to reverse the bit trimming of interferogram packets	unitless
MW Data Extraction Information		
Number of A/D samples	Used to reverse the bit trimming of interferogram packets	unitless
Decimation & number of filter taps	Used to reverse the bit trimming of interferogram packets	unitless
SW Data Extraction Information		
Number of A/D samples	Used to reverse the bit trimming of interferogram packets	unitless
Decimation & number of filter taps	Used to reverse the bit trimming of interferogram packets	Unitless
Nonlinearity Correction Parameters		
linearityCorrectionA2Parameters	2nd order IR channel nonlinearity characterization parameter. There are nine linearity correction parameters for each FOV for LWIR, MWIR, and SWIR	1/Volts
linearityCorrectionVinstParameters	Detector preamp output voltage due only from instrument background radiance & detector dark current. There are nine linearity correction vinst parameters for LWIR, MWIR, and SWIR	Volts
linearityCorrectionModEffParameters	Interferometer modulation efficiency. There are nine linearity correction modeff parameters for LWIR, MWIR, and SWIR	Volts
FIR Tailoring	LWIR FIR filter coefficient gain relative to baseline FM1_3 FIR filter coefficients. Excludes gain changes due to bit trim mask changes (LW, MW and SW).	Unitless
PGA Gain Table Map	LWIR, MWIR and SWIR channel electrical gain in Volt/Volt corresponding to PGA (16x3 numbers)	Volt/Volt Config

In addition to the parameters provided through the instrument telemetry Packets, there are many configuration options that modify the processing performed by the operational code. These values are modified via the configuration files provided by DMS. The tunable parameters provided via the CrIS SDR Processing Coefficient Files for truncated and full spectral resolution. See 474-00448-02-03_JPSS-DD-Vol-II-Part-3 for more details.

3.1.1.1.1 Requirements for Input

The primary instrument output (interferogram data) is generated as follows:

- The instrument can perform a new measurement (sweep) every 200 ms: 167 ms for data collection and 33 ms for repositioning.
- A new cycle (scan) is repeated every 8 seconds:
    - 600 msec slew from ICT to Earth Scene (ES) 1
    - 200 msec x 30 Earth Scenes
    - 200 msec slew from ES 30 to Deep Space (DS)

- 200 msec x 2 samples at DS (forward and reverse sweeps)
- 400 msec slew from DS to ICT
- 200 msec x 2 samples at ICT (forward and reverse sweeps)

- Each scan is comprised of 918 interferograms: (2 DS + 2 ICT + 30 ES) × 3 bands × 9 detectors / band

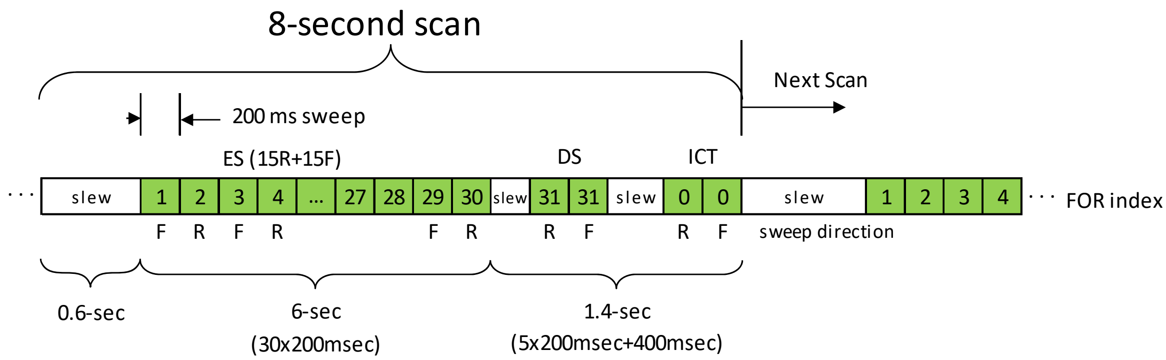

Figure 3.1.1.1.1-1. CrIS Measurement Sequence

3.1.1.1.2 Requirements for Applicable Auxiliary/Ancillary and/or Optional Input Data

In addition to the interferogram data generated by the CrIS Sensor, there are two additional packets of interest to the SDR algorithm:

1. Eight second Science/Calibration Telemetry – This data packet contains instrument temperature information required during the calibration process collected during the previous eight seconds of instrument operation. These data packets are transmitted by the instrument at the end of every eight second scan period.
2. Four Minute Engineering Telemetry – This data packet contains additional information used for interferogram processing, spectral calibration, spatial correction, and geolocation. With the exception of the Neon calibration data contained within this packet, all information is virtually static; however, it can be modified via table upload commands sent to the CrIS sensor. Neon calibration data is updated as programmed and is currently planned for once per orbit. These data packets are transmitted by the instrument at the end of every thirtieth eight second scan period.

3.1.1.2 Outputs

Tables 3.1.1.2-1 and 3.1.1.2-2 list the CrIS SDR algorithm outputs.

Table 3.1.1.2-1. CrIS Truncated Spectrum SDR Outputs

Output	Description	Reference Document
CrIS SDR	CrIS is an infrared sounder (Michelson Interferometer) designed to measure scene radiance and calculate the vertical distribution of temperature, moisture, and pressure in the Earth's atmosphere. The CrIS SDR algorithms transform the scene interferograms into fully calibrated, unapodized, spectral information. For the radiance arrays dimensioned with wavenumber, the wavenumber is increasing, and the values are most representative of the wavelength bin center. For arrays dimensioned with "band" […3], the ordering is LW (Long-wave), MW (Middle-wave), SW (Short-wave). Raw data (earth view, internal calibration and space view) are preprocessed, undergo radiometric, spectral, and geometric calibrations, and are quality checked prior to SDR creation. This output is then used in subsequent atmospheric parameter calculations.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR GEO	CrIS SDR Geolocation Data	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Correction Matrix AUX	The Cross-track Infrared Sounder (CrIS) Correction Matrix PC is applied to spectra as they are ejected from a sliding window. The 4-minute Engineering packet is used as input to create it. It is created at least once an orbit, estimated. The neon lamp measurements provide the measured laser wavelength that is stored in the Engineering packet binary file (CrIS-SDR-ENGPKT-BACKUP-AUX) or CrIS-FS-SDR-ENGPKT-BACKUP-AUX. The corresponding updated CMO matrix is stored in the CMO binary file (CrIS-Correct-Matrix-AUX) or CrIS-FS-Correct-Matrix-AUX. The FSR contains 2736 channels whereas the TSR has 1592.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR Engineering Packet Backup AUX	The Cross-track Infrared Sounder (CrIS) SDR Engineering Packet Backup PC provides tunable processing coefficients for use by the algorithm during execution. The coefficients can be modified (tuned) through a configuration control process in response to algorithm, performance, inputs, sensitivity, etc. changes. The TSR and FSR files have the same fields. Some fields have greater array size for FSR. Other fields, such as threshold parameters, have different values.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR DQN	Data Quality Notification	474-00448-02-01_JPSS-DD-Vol-II-Part-1

Table 3.1.1.2-2. CrIS Full Spectrum SDR Outputs

Output	Description	Reference Document
CrIS Full Spectrum SDR	CrIS is an infrared sounder (Michelson Interferometer) designed to measure scene radiance and calculate the vertical distribution of temperature, moisture, and pressure in the Earth's atmosphere. The CrIS SDR algorithms transform the scene interferograms into fully calibrated, unapodized, spectral information. For the radiance arrays dimensioned with wavenumber, the wavenumber is increasing, and the values are most representative of the wavelength bin center. For arrays dimensioned with "band" […3], the ordering is LW (Long-wave), MW (Middle-wave), SW (Short-wave). Raw data (earth view, internal calibration and space view) are preprocessed, undergo radiometric, spectral, and geometric calibrations, and are quality checked prior to SDR creation. This output is then used in subsequent atmospheric parameter calculations.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum SDR GEO	CrIS SDR Geolocation Data	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum Correction Matrix AUX	The Cross-track Infrared Sounder (CrIS) Correction Matrix PC is applied to spectra as they are ejected from a sliding window. The 4-minute Engineering packet is used as input to create it. It is created at least once an orbit, estimated. The neon lamp measurements provide the measured laser wavelength that is stored in the Engineering packet binary file (CrIS-SDR-ENGPKT-BACKUP-AUX) or CrIS-FS-SDR-ENGPKT-BACKUP-AUX. The corresponding updated CMO matrix is stored in the CMO binary file (CrIS-Correct-Matrix-AUX) or CrIS-FS-Correct-Matrix-AUX. The FSR contains 2736 channels whereas the TSR has 1592.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum SDR Engineering Packet Backup AUX	The Cross-track Infrared Sounder (CrIS) SDR Engineering Packet Backup PC provides tunable processing coefficients for use by the algorithm during execution. The coefficients can be modified (tuned) through a configuration control process in response to algorithm, performance, inputs, sensitivity, etc. changes. The TSR and FSR files have the same fields. Some fields have greater array size for FSR. Other fields, such as threshold parameters, have different values.	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum SDR DQN	Data Quality Notification	474-00448-02-01_JPSS-DD-Vol-II-Part-1

The CrIS SDR algorithm shall produce unapodized radiances to be compliant with CrIS SDR product requirements. The only problem is that information is lost at the two ends of each of the three CrIS bands due to convolution. This can be avoided, however, by extending the spectral grid of the unapodized radiances.

For CrIS Truncated Spectral Resolution, use the following values:

<edrLwDeltaSigma Default_"0.625000"/>
<edrLwMaximumWavenumber Default="1096.250000"/>
<edrLwMinimumWavenumber Default="648.750000"/>
<edrLwNumberOfPoints Default="717"/>
<edrMwDeltaSigma Default="1.250000"/>
<edrMwMaximumWavenumber Default="1752.500000"/>
<edrMwMinimumWavenumber Default="1207.500000"/>
<edrMwNumberOfPoints Default="437"/>
<edrSwDeltaSigma Default="2.500000"/>
<edrSwMaximumWavenumber Default="2555.000000"/>
<edrSwminimumWavenumber Default="2150.000000"/>
<edrSwNumberOfPoints Default="163"/>

For Truncated Spectral Resolution, the CrIS SDR algorithm outputs unapodized radiances for the following spectral grid:

LWIR Band: a total of 717 bins from 648.75 cm-1 to 1096.25 cm-1 with a spacing of 0.625 cm-1;
MWIR Band: a total of 437 bins from 1207.5 cm-1 to 1752.5 cm-1 with a spacing of 1.25 cm-1;
SWIR Band: a total of 163 bins from 2150 cm-1 to 2555 cm-1 with a spacing of 2.5 cm-1.

For CrIS Full Spectral Resolution, use the following values:

<edrLwDeltaSigma Default_"0.625000"/>
<edrLwMaximumWavenumber Default="1096.250000"/>
<edrLwMinimumWavenumber Default="648.750000"/>
<edrLwNumberOfPoints Default="717"/>
<edrMwDeltaSigma Default="1.250000"/>
<edrMwMaximumWavenumber Default="1751.2500000"/>
<edrMwMinimumWavenumber Default="1208.7500000"/>
<edrMwNumberOfPoints Default="869"/>
<edrSwDeltaSigma Default="2.500000"/>
<edrSwMaximumWavenumber Default="2551.250000"/>
<edrSwminimumWavenumber Default="2153.750000"/>
<edrSwNumberOfPoints Default="637"/>

For Full Spectral Resolution, the CrIS SDR algorithm outputs unapodized radiances for the following spectral grid:

LWIR Band: a total of 717 bins from 648.75 cm-1 to 1096.25 cm-1 with a spacing of 0.625 cm-1;
MWIR Band: a total of 869 bins from 1208.75 cm-1 to 1751.25 cm-1 with a spacing of 0.625 cm-1;
SWIR Band: a total of 637 bins from 2153.75 cm-1 to 2551.25 cm-1 with a spacing of 0.625 cm-1.

Comparing to the original CrIS SDR spectral grid, the new one added two more grid points to the ends of each of the three CrIS bands. With the 12 newly added bins, the total number of bins of the CrIS SDR output radiances has increased from 1305 to 1317.

3.1.2 Algorithm Processing

CrIS SDR processing retrieves the ingest Raw Data Records (RDRs) from DMS. CrIS RDR data in DMS is in CCSDS packet format. The CrIS SDR algorithm is tasked to process a granule of RDR data. In order to do that, it needs to retrieve 15 scans prior to the beginning of the granule and 15 scans after the end of the granule. The first step is to determine the beginning scan in the current granule and then determine the granule IDs for the RDR data on both sides of the granule. Once the RDR files have been retrieved and the time stamp for the beginning scan determined, a structure of packet pointers is populated with pointers to the actual packets in the RDR files based on scans and time. This structure of packet pointers is passed into the CrIS SDR algorithm which parses and processes the packets. The RDR data are created as heap items to allow the RDR data in heap memory to be modified if needed. When CrIS full resolution data is processed, in Truncated Spectral Resolution (TSR) SDR processing mode, a truncation module is executed before any processing is done to truncate the interferograms for processing by the algorithm, otherwise, i.e., in Full Spectrum Resolution (FSR) SDR processing mode. Engineering packet values are then updated to reflect the truncated trim table values. Common geolocation routines are used to get the location by providing the exit vector for that sample so that geolocation information can be updated. The specific common geolocation routines used, here, are satPosAtt() to locate the ephemeris and attitude for the specific observation time, and ellipIntersect() to calculate the geodetic latitude and longitude and terrain height for the given sensor exit vector and ephemeris and attitude. The exit vectors are calculated with

$$E_x = -\sin(P_{LOS})$$
$$E_y = \cos(P_{LOS}) * \sin(R_{LOS})$$
$$E_z = \cos(P_{LOS}) * \cos(R_{LOS})$$

Where and are the line-of-sight roll and pitch and is the exit vector.

3.1.2.1 Main Module - SDR Generator Application

This paragraph outlines the major software contributors used to process CrIS data. Detailed design relationships of these classes can be obtained from the design model. Detailed implementation details are documented in the source with in-line comments. Table 3.1.2.1-1 lists some important classes used during processing.

Table 3.1.2.1-1. Some Important Classes Used During Processing

Name	Type	Initialization	Description
CalibratedSpectra	Class	Instantiation	Specialization of Spectra which implements the complex calibration interface
CCSDSFormat	Class	Singleton	Utility for packaging and extraction of CCSDS primary and secondary headers
CorrectionMatrix	Class	Instantiation	Container for the spatial corrections that comprise the CMO
EngineeringCalibrationRecord	Class	Specialization	Implements the abstract CalibrationRecord for Algorithm specific 4min values
Interferogram	Class	Instantiation	Container for organizing video data
ScienceCalibrationRecord	Class	Specialization	Implements the abstract CalibrationRecord for Algorithm specific 8sec values

Name	Type	Initialization	Description
ScienceDataProcessor	Class	Instantiation	Central engine for Algorithm behavior
Spectra	Class	Abstract	Interface container
SpectraManager	Class	Instantiation	Collection of Spectra used in window average
TelemetryProcessor	Class	Singleton	Extracts binary instrument data defined by config file
TemperatureHistory	Class	Instantiation	Collection responsible for min, max, avg temperature
VideoData	Class	Aggregation	Implements TelemetryProcessor's bit-trim behavior

3.1.2.2 Class CalibratedSpectra

This class provides complex calibration and applies corrections to the instance spectra.

Table 3.1.2.2.1-1 shows the CalibratedSpectra attributes and Table 3-10 shows the CalibratedSpectra operations.

3.1.2.2.1 CalibratedSpectra Attributes

Table 3.1.2.2.1-1. CalibratedSpectra Attributes

Name	Type	Description
VERSION_NUMBER	primitive unsigned short int UInt16	*Collection of version number.*

3.1.2.2.2 CalibratedSpectra Operations

Table 3.1.2.2.2-1. CalibratedSpectra Operations

Name	Return	Parameters	
Applies a complex calibration to the instance spectra. The cold contribution is added only when coldTarget Temp is !=NULL			
calibrate	primitive bool	Class Spectra*	*hotReference*
		Class Spectra*	*coldReference*
		Class Spectra*	*hotTargetTemp*
		Class Spectra*	*coldTragetTemp*
		BOOST::vector<Float64>*	*calibrationMatrix*
Applies a complex calibration to the instance spectra. The cold contribution is added only when coldTarget Temp is !=NULL (Overloaded function)			
calibrate	primitive bool	Class Spectra*	*hotReference*
		Class Spectra*	*coldReference*
		Class Spectra*	*hotTargetTemp*
		Class Spectra*	*coldTragetTemp*
		BOOST::vector<Float64>&	*theFirGainRealBin*
		BOOST::matrix<Float64>*	*calibrationMatrix*
Applies a bin by bin correction			
correct	primitive bool	Class Spectra*	*correctionSpectra*
Return linear phase shift when fringe count error exist.			
correctFringeCountError	Class BOOST::vector<std::complex<Float64>&	Class BOOST::vector<std::complex<Float64>>&	*linearPhaseShift*
Fringe count errors are computed and returned			
detectReferenceSpectraFringeCountError	Float64	Class Spectra*	*hotReference*
		Class Spectra*	*coldReference*
Fringe count errors are computed and returned			
detectESFringeCountError	Int32	Class Spectra*	*hotReference*

Name	Return	Parameters	
		Class Spectra*	*coldReference*
		Class Spectra*	*rawEarthSpectra*
		Class Spectra*	*hotTargetTemp*
		Class Spectra*	*coldTargetTemp*
Sync ICT and DS validate tests			
FCE_ICT_DS_sync_validateTests	bool	BOOST::vector<Float64>&	*fitIndex*
		UInt32	*binSize*
		Float64	linearPhaseSlope
		Float64	linearPhaseOrdinate
		BOOST::vector<Float64>&	fitSigma
		Float64	fringeCount
Prepare Phase Linear Regression			
preparePhase_LinearRegression	void	BOOST::vector<Float64>&	linearPhase
		enum Band::Wavelength	*theBand*
		FCE_LIMIT	threshold_type
		BOOST::vector<Float64>&	fitIndex
		BOOST::vector<Float64>&	fitSigma
Estimate NEdN			
estimateNEdN	Bool	Class Spectra*	*theSlope*
		Class Spectra*	*theIntercept*
		typedef unsigned int UInt32	*userNumberOfPoints*
		typedef double Float64	*userMinWavenumber*
		typedef double Float64	*userDeltaSigma*
Shifts the imaginary component of phase to align the real component with the reference spectra.			
adjustPhase	primitive void	Class BOOST::vector<std::complex<Float64>>&	*linearPhaseShift*
Estimate NEdN			
estimateNEdN	Bool	Class Spectra*	*theSlope*
		Class Spectra*	*theIntercept*
		typedef unsigned int UInt32	*userNumberOfPoints*
		typedef double Float64	*userMinWavenumber*
		typedef double Float64	*userDeltaSigma*

3.1.2.3 CCSDSFormat

This class implements the CommonMessageFormat for CCSDS packetization. Table 3.1.2.3.1-1 shows the CCSDSFormat attributes and Table 3.1.2.3.2-1 shows the CCSDSFormat operations.

3.1.2.3.1 CCSDSFormat Attributes

Table 3.1.2.3.1-1. CCSDSFormat Operations

Name	Type	Description
thePrimaryHeader	Class CcsdsPrimaryHeader	*Collection of routing a packet and its data.*
theSecondaryHeader	Class CcsdsSecondaryHeader	*Collection of routing the timestamp portion of a CCSDS header*

3.1.2.3.2 CCSDSFormat Operations

Table 3.1.2.3.2-1. CCSDSFormat Operations

Name	Return	Parameters	
Populates the primary and secondary headers based on the data at the offset of startingLocation.			
unpack	typedef int	Class Octets	*&theData*

Name	Return	Parameters	
	Int32		
		typedef unsigned int UInt32	*&startingLocation*

3.1.2.4 Class CorrectionMatrix

This class is responsible for implementing a Correction Matrix that is applied to each spectra. Table 3.1.2.4.1-1 shows the CorrectionMatrix attributes and Table 3.1.2.4.2-1 shows the CorrectionMatrix operations.

3.1.2.4.1 CorrectionMatrix Attributes

Table 3.1.2.4.1-1. CorrectionMatrix Attributes

Name	Type	Description
size	typedef unsigned int UInt32	*Size of Correction Matrix*
childTable[ ]	Class CorrectionTable*	*Collection of child tables*
Invsa_calibrationMatrix	Class BOOST::matrix<Float64>	*Collection of inverse self-apodization calibration matrix*
lastUpdateTime	typedef struct SYSTEMTIME	*Collection of last update time*
AlgorithmParameter	Class SDR_AlgorithmnParameter	*Collection of SDR algorithm parameters*
CorrectionParameter	Class SDR_CorrectionParam	*Collection of SDR correction parameters*
InstrumentCharacteristic	Class CrIS_InstrumentCharacteristics	*Collection of CrIS instrument parameters*
maxPathDifferential[ ]	typedef double Float64	*Collection of maximum path difference*
correctionTableDeltaSigma[]	typedef double Float64	*Collection of table delta sigma*
correctionTableLowestWavenumber[]	typedef double Float64	*Collection of table lowest wavenumber*
instrumentDeltaSigma[]	typedef double Float64	*Collection of instrument delta sigma*
instrumentLowestWavenumber[]		*Collection of instrument lowest wavenumber*
userDeltaSigma[ ]	typedef double Float64	*Collection of user delta sigma*
userLowestWavenumber[ ]	typedef double Float64	*Collection of user lowest wavenumber*

3.1.2.4.2 CorrectionMatrix Operations

Table 3.1.2.4.2-1. CorrectionMatrix Operations

Name	Return	Parameters	
Rebuilds child tables for calibration matrix construction			
buildILSCorrectionTable	typedef int Int32	Class EngineeringCalibrationRecord&	*theEngCalRec*
		enum SceneElement	*theFOV*
		enum Band::Wavelength	*theBand*
Builds self-apodization table			
buildSelfApodization Table	primitive bool	Class EngineeringCalibrationRecord&	*theEngCalRec*
		enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
Builds Residual ILS Table			
buildResidual_ILS_Table	primitive bool	Class EngineeringCalibrationRecord&	*theEngCalRec*
		enum SceneElement	*theFOV*

Name	Return	Parameters	
		enum Band::Wavelength	*theBand*
Calculates and return the inverse SA calibration matrix.			
matrixProduct	boost::matrix <Float64>	boost::matrix<Float64>	*&A*
		boost::matrix<Float64>	*&B*
		boost::matrix<Float64>	*&C*
Resizes the correction and child tables to new dimension.			
setSize	primitive void	typedef unsigned int UInt32	*newDimension*
Creates a comma separated string containing values from the calibrationMatrix			
toString	primitive string	n/a	*n/a*
Serializes the correction matrix with user deltaSigma.			
serializeCMO	primitive void	Class cris_oarchive&	*ar*
		enum Band::Wavelength	*theBand*
Serializes the correction matrix with user deltaSigma.			
serializeCMO	primitive void	Class cris_iarchive&	*ar*
		enum Band::Wavelength	*theBand*

3.1.2.5 Class EngineeringCalibrationRecord

This class is a parent class to each of the Engineering Calibration Record subcategories. Table 3.1.2.5.1-1 shows the EngineeringCalibrationRecord attributes and Table 3.1.2.5.2-1 shows the EngineeringCalibrationRecord operations.

3.1.2.5.1 EngineeringCalibrationRecord Attributes

Table 3.1.2.5.1-1. EngineeringCalibrationRecord Attributes

Name	Type	Description
ietTime	typedef long long Int64	*IET time*
computedWavelengthRejectionThreshold	typedef unsigned int UInt32	*Computed wavelength rejection threshold*
rejectedWavelengthsRatio	typedef double Float64	*Rejected wavelength ratio*
averageMetrologyWavelength	typedef double Float64	*Averaged metrology wavelength*
engLaserDiodeTemperature	typedef double Float64	*Laser diode temperature*
engLaserDiodeCurrent	typedef double Float64	*Laser diode current*
laserDiodeWavelengthOrigin	enum LaserWavelengthSource	*Original laser diode wavelength*
ilsParams	typeded struct ILS_Parameters	*Collection of ILS parameters*
ilsOrigin	enum IlsOriginSource	*Collection of ILS original source*
theEngCalRec_ICTEmissivityParameters	Class EngCalRec_ICTEmissivityParameters	*Collection of engineering ICT emissivity parameters*
theEngCalRec_ILSCurveFitParameters	Class EngCalRec_ILSCurveFitParameters	*Collection of ILS curve fit parameters*
theEngCalRec_ILSFOVParameters	Class EngCalRec_ILSFOVParameters	*Collection of ILS fov parameters*
theEngCalRec_ICTEnvironmentalModel	Class EngCalRec_ICTEnvironmentalModel	*Collection of ICT environmental model*

Name	Type	Description
theEngCalRec_ScienceTelemetryConversionCoefficients	Class EngCalRec_ScienceTelemetryConversionCoefficients	*Collection of science telemetry conversion coefficients*
theEngCalRec_PolarizationCalibrationWavenumbers	Class EngCalRec_PolarizationCalibrationWavenumbers	*Collection of polarization calibration wavenumbers*
theEngCalRec_LaserMetrologyInfo	Class EngCalRec_LaserMetrologyInfo	*Collection of laser metrology information*
theEngCalRec_NeonCalInfo;	Class EngCalRec_NeonCalInfo	*Collection of neon calibration information*
theEngCalRec_ScienceTelemetryLimits	Class EngCalRec_ScienceTelemetryLimits	*Collection of science telemetry limits*
theEngCalRec_PitchRollYawInfo	Class EngCalRec_PitchRollYawInfo	*Collection of pitch, roll and yaw information*
theEngCalRec_LinearityErrorParameters	Class EngCalRec_LinearityErrorParameters	*Collection of linearity error parameters*
theEngCalRec_ExtractionRecord	Class EngCalRec_ExtractionRecord	*Trim table and extraction record values*
losInSSMF[ ]	Class BOOST::vector<Float64>	*Collection of pitch and yaw misalignments between interferometer and scan mechanism*
normalRMF	Class BOOST::vector<Float64>	*Collection of mirror normal to SSM mirror in RMF coordinate system*
sbfToSSMFMatrix	Class BOOST::vector<Float64>	*Collection of transformation from SBF to SSMF*
lastCheckSumValue	typedef int Int32	*Last check sum value*
packetVersionNumber	typedef int Int32	*Packet version number*

3.1.2.5.2 EngineeringCalibrationRecord Operations

Table 3.1.2.5.2-1. EngineeringCalibrationRecord Operations

Name	Return	Parameter	
Refreshes data			
refreshData	primitive bool	typedef unsigned short int UInt16	*theApid*
		class CCSDSFormat*	*the TelemetryFormat*
Refreshes data (Overloaded function)			
refreshData	primitive bool	typedef unsigned short int UInt16	*theApid*
		class CCSDSFormat*	*the TelemetryFormat*
		CalibrationOrder	*theOrder*
calculates metrology wavelength			
calculateMetrologyWavelength	primitive void	n/a	*n/a*
Loads the values from the engineering calibration record			
serialize	primitive void	Class cris_iarchive	*&ar*
		Class std::ostringstream stringstream	*&oss*
Initializes the scene selection module frame (SSMF) to spacecraft body frame (SBF) transformation operator.			
initializeGeometricCalibration	primitive void	n/a	*n/a*
Computes rotation matrix around Y axis			

Name	Return	Parameter	
rotationMatrixY	Class BOOST::matrix<Float64>	typedef double Float64	*thePitch*
Compute rotation matrix around X axis			
rotationMatrixX	Class BOOST::matrix<Float64>	typedef double Float64	*theRoll*
Computes rotation matrix around Z axis.			
rotationMatrixZ	Class BOOST::matrix<Float64>	typedef double Float64	*theYaw*

3.1.2.6 Class Interferogram

This class is responsible for extracting interferogram data from the telemetry processor. Each interferogram consists of a vector of real parts and a vector of imaginary parts. Table 3.1.2.6.1-1 shows the Interferogram attributes and Table 3.1.2.6.2-1 shows the Interferogram operations.

3.1.2.6.1 Interferogram Attributes

Table 3.1.2.6.1-1. Interferogram Attributes

Name	Type	Description
realSample	Class BOOST::vector<Float64>*	*Collection of real parts*
imaginarySample	Class BOOST::vector<Float64>*	*Collection of imaginary parts*
RDR_Status	typedef struct RawDataRecordStatusRegister	*Collection of raw data status*
status	typedef unsigned int UInt32	*Status*
channel	typedef unsigned int UInt32	*Channel number*
theScene	enum FieldOfRegard	*Actual field of regard reported in interferogram*
porchSwingDirection	enum SweepDirection	*Actual porch swing direction reported in interferogram*
spectralBand	enum Band::Wavelength	*Actual band reported in interferogram*
fieldOfView	enum SceneElement	*Actual field of view reported in interferogram*

3.1.2.6.2 Interferogram Operations

Table 3.1.2.6.2-1. Interferogram Operations

Name	Return	Parameters	
Adds an interferogram to the current interferogram.			
operator+	primitive void	Class Interferogram&	*rightOperand*
subtracts an interferogram from the current interferogram.			
operator-	primitive void	Class Interferogram&	*rightOperand*
Divides each element of this interferogram by a specified number.			
operator/	primitive void	typedef double Float64	*factor*
Copies all data from source interferogram.			
operator=	Class Interferogram&	Class Interferogram&	*source*
Outputs information of the current interferogram.			
operator<<	Class ostream&	Class ostream&	*outStream*
		Class Interferogram&	*i*
returns data of the specified type from the interferogram.			

Name	Return	Parameters	
getSamples	Class BOOST::vector<Float64>&	enum Complex	*Real or imaginary samples*
Computes deep space symmetry.			
computeDeepSpaceSymmetry	typedef double Float64	Class Interferogram&	*forwardDS*
		Class Interferogram&	*reverseDS*
References to the root of the complex squares.			
getMagnitude	Class BOOST::vector <Float64>&	n/a	*n/a*
Return the state f the missing data flag			
getMissingData	Bool	n/a	*n/a*
Set the state of the missing data flag			
setMissingData	Void	n/a	*n/a*
Apply apodization function to apodize the interferogram			
apodize_interferogram	Void	BOOST::vector<Float64>&	*apodization*

3.1.2.7 Class TemperatureHistory

This class is a container for temperatures gathered from the data. Table 3.1.2.7.1-1 shows the temperatureHistory attributes and Table 3.1.2.7.2-1 shows the temperatureHistory operations.

3.1.2.7.1 TemperatureHistory Attributes

Table 3.1.2.7.1-1. TemperatureHistory Attributes

Name	Type	Description
windowSize_	typedef int Int32	*Moving average window size*
historicalWindow_	Class vector<CalRecord*>	*A vector that contains average temperatures*
minTemperature_	typedef double Float64	*Minimum temperature*
maxTemperature	typedef double Float64	*Maximum temperature*
median_b	typedef double Float64	*Median B*
median_e	typedef double Float64	*Median E*
median_m	typedef double Float64	*Median M*

3.1.2.7.2 TemperatureHistory Operations

Table 3.1.2.7.2-1. TemperatureHistory Operations

Name	Return	Parameters	
Removes all records in the temperature history window.			
clearHistory	primitive void	n/a	*n/a*
Sets the window size for a temperature window.			
setWindowSize	primitive void	typedef int Int32	*newSize*
Returns a temperature at any given position.			
getTemperature	typedef double Float64	typedef int Int32	*position*
Adds the current average temperature to the window if a temperature is not supplied.			
addTemperature	typedef double Float64	n/a	*n/a*
Adds temperature to a temperature window.			
addTemperature	typedef double	typedef double Float64	*newTemp*

Name	Return	Parameters	
	Float64		
		typedef struct CrisSdrHeapDataType	**algDataPtr*
		typedef unsigned int UInt32	*scanIdx*
		typedef int Int32	*ictTempType*
Calculates an average of a temperature window.			
calculateAverageTemperature	typedef double Float64	n/a	*n/a*
returns the number of valid temperatures in temp history window			
getValidCount	typedef int Int32	n/a	*n/a*
Removes a temperature record from the temperature window.			
removeTemperature	primitive bool	n/a	*n/a*
Update ICT			
updateICT	void	CrisSdrAlgDataType	*algDataPtr*
		typedef int Int32	*scanIdx*
		typedef int Int32	*ictTempType*
Calculate Median Thermal Temp Drift			
medianThermalTempDrift	void	n/a	*n/a*

3.1.2.8 Class ScienceDataProcessor

This class is responsible for processing the CrIS SDR science data. Table 3.1.2.8.1-1 shows the ScienceDataProcessor attributes and Table 3.1.2.8.2-1 shows the ScienceDataprocessor operations.

3.1.2.8.1 ScienceDataProcessor Attributes

Table 3.1.2.8.1-1 shows the science data processor attributes. This class instantiates the Planck radiation constants for the Planck Function. C_1 requires at least five extra points of precision. This extra amount of precision also ensures better accuracy in Planck function calculations.

Table 3.1.2.8.1-1. ScienceDataProcessor Attributes

Name	Type	Description
correctionTable[ ][ ]	Class CorrectionMatrix	Collection of correction matrix tables
resamplingMatrix[]	BOOST::matrix<Float64>	Resampled Matrix
resampling_laserwavelength	Float64	Sampled Laser Wavelength
resamplingMatrix[]	BOOST::matrix<Float64>	Resampled Matrix
postFilter[]	boost::numeric::ublas::banded_matrix<Float64 , boost::numeric::ublas::column_major>	Post filter Matrix
apodizationMatrix[]	BOOST::matrix<Float64>	Apodization Matrix
calibrationmatrix [][]	BOOST::matrix<Float64>	Calibration Matrix
engPacketCount	typedef int Int32	Engineering packet count
*DMS_Data_Inputs	typedef struct CrisSdrAlgDataType	Collection of DMS input parameters
windowSize	typedef int Int32	Window size
processingPeriod	Class SpectraManager	Collection of spectra currently used in the moving window average.
polarizationCorrectionCurve	Class Spectra*	Collection of polarization correction curves
CalibrationParameter	typedef struct CalibrationParam	Collection of calibration parameters
CorrectionParameter	typedef struct SDR_CorrectionParam	Collection of SDR correction parameters

InstrumentCharacteristic	Class CrIS_InstrumentCharacteristics	Collection of CrIS instrument characteristics
TestEnvironmentParameter	typedef struct CrIS_TestEnvironmentParameters	Collection of CrIS environment parameters
GeneralParameter	typedef struct SDR_GeneralParameters	Collection of CrIS general parameters
AlgorithmParameter	struct SDR_AlgorithmnParameter	Collection of SDR algorithm parameters
maxPathDifferential[ ]	typedef double Float64	Max path differential
deltaSigma[ ]	typedef double Float64	Delta sigma
lowestWavenumber[ ]	typedef double Float64	Lowest wavenumber
validCalTargetDuration	Class TimeRange	Collection of valid calibration target time
validCalTargetTolerence	typedef struct SYSTEMTIME	Collection of valid calibration time tolerance
validColdTargetDuration	Class TimeRange	Collection of valid cold target time
validColdTargetTolerence	typedef struct SYSTEMTIME	Collection of valid cold target tolerance
validHotTargetDuration	Class TimeRange	Collection of valid hot target time
validHotTargetTolerence	typedef struct SYSTEMTIME	Collection of valid hot target tolerance
sequenceCount[ ][ ][ ][ ]	typedef int Int32	Collection of spectra sequence counts
scienceCount[ ][ ][ ][ ]	typedef int Int32	Collection of spectra science counts
targetCount[ ][ ][ ][ ]	typedef int Int32	Collection of spectra target counts
invalidInterferogramCount[ ][ ]	typedef int Int32	Collection of invalid interferogram counts
invalidSpectraCount[ ][ ]	typedef int Int32	Collection of invalid spectra counts
currentExternalCalFileLine	primitive string	Current external cal file line
theEngineeringCalibrationRecord	Class EngineeringCalibrationRecord	Collection of engineering calibration record
theILSParametersRecord	Class ILSParametersRecord	Collection of ILS Parameters record
theTargetTempCalibrationRecord	Class TargetTempCalibrationRecord	Collection of target temperature calibration record
theScienceCalibrationRecord	Class ScienceCalibrationRecord	Collection of science calibration record
plankConstC1	typedef double Float64	Planck function constant value
plankConstC2	typedef double Float64	Planck function constant value
appDirectory	primitive string	Directory name
calculateDesiredBandCenter	primitive void	Calculate desired band center
SCIENCE_CALIBRATION_FRAME_TYPE	typedef unsigned int UInt32	Science calibration frame type
ENGINEERING_CALIBRATION_FRAME_TYPE	typedef unsigned int UInt32	Engineering calibration frame type
requiresPhaseSync[ ][ ][ ]	primitive bool	Collection of required phase synchronization
syncAttemptCount[ ][ ][ ]	typedef int Int32	Collection of attempt synchronization count
totalActiveThreads	typedef int Int32	Total active threads
threadFOVmap[ ]	typedef int Int32	Thread FOV
earthScenefceDetectionComplete[ ]	primitive bool	Collection of earth scene FCE detection completion
interferogramApodizationTable[]	BOOST::vector < Float64>	Interferogram Apodization Table.
earthScenefceDetectionComplete[]	primitive bool	Flag indicating earth scene FCE detection complete.
referenceHotSceneReceived	Int32	Count of hot reference received after psd == 1.
referenceSceneReceived[ ][ ]	primitive bool	Collection of reference scene received
referenceSceneSync[ ][ ][ ]	primitive bool	Collection of reference scene synchronization

initialCorrectionReady	primitive bool	Initial correction ready flag
FIR_GAIN[ ]	Class UncalibratedSpectra	Collection of FIR gain
shiftFactor[]	typedef double Float64	Array of shift factors
appShiftFactorFlag	primitive bool	Flag indicating to apply the shift factor.
scanIdx_	typedef unsigned int UInt32	Scan index
numCMOBuilds_	typedef unsigned short UInt16	Number of CMO builds
numEngBuilds_	typedef unsigned short UInt16	Number of Engineer Packet builds
taskedSequentially_	primitive bool	Task sequential flag
scanWhichCausedCMORebuild_	typedef signed short Int16	Scan number that causes to rebuild CMO
scanWhichCausedEngRebuild_	typedef signed short Int16	Scan number that causes to rebuild Engineer Packet
minNpCrossingTime_	SYSTEMTIME	Min North Pole Crossing time.
maxNpCrossingTime_	SYSTEMTIME	Max North Pole Crossing time.
orbitMsgThrottle_	std::map<Int32, bool>	Map of scan index to debug message reported of Polar crossing.
currentDSfwd	Class Interferogram	Current DS forward interferogram
currentDSrev	Class Interferogram	Current DS reverse interferogram
currentDSfwd_p	primitive bool	Current DS forward flag
currentDSrev_p	primitive bool	Current DS reverse flag
engPkt_Loaded	primitive bool	EngPkt loaded flag
ilsCorrection_Loaded	primitive bool	ILS Correction Matrix loaded flag
firstEngPkt	primitive book	First EngPkt flag
engPktOutputTimeStep	typedef int Int64	Step to output EngPkt
forDS_reals[ ][ ][ ]	Class BOOST::vector < Float64>	Collection of forward DS real parts
forDS_imags[ ][ ][ ]	Class BOOST::vector < Float64>	Collection of forward DS imaginary parts
revDS_reals[ ][ ][ ]	Class BOOST::vector < Float64>	Collection of reverse DS real parts
revDS_imags[ ][ ][ ]	Class BOOST::vector < Float64>	Collection of reverse DS imaginary parts

3.1.2.8.2 ScienceDataProcessor Operations

Table 3.1.2.8.2-1 shows the science data processor operations.

Table 3.1.2.8.2-1. ScienceDataProcessor Operations

Name	Return	Parameter	
Cleans up of ScienceDataProcessor data for next dispatch.			
cleanup	primitive void	n/a	n/a
Populates high access frequency domain data.			
calculateWavenumberBinSpacing	primitive void	enum Band::Wavelength	theBand
Calculates the post calibration table. This calculation is only band specific.			
buildPostCalibrationTable	primitive bool	enum Band::Wavelength	theBand
Calculates the post calibration table. This calculation is only band specific.			
buildPostCalibrationTable_raisedCos	primitive bool	enum Band::Wavelength	theBand
Calculates Resampling table. This calculation is only band specific.			
buildResamplingTable	primitive bool	Float64	lowestWavenumberFrom
		Float64	deltaSigmaFrom
		Float64	lowestWavenumberTo
		Float64	deltaSigmaTo
		UInt32	bigNto
		BOOST::matrix<Float64>	rMatrix
Calculates Resampling table. This calculation is only band specific.			

Name	Return	Parameter	
buildResamplingTable	primitive bool	enum Band::Wavelength	theBand
Compute the Max Path Difference of the sampled interferogams.			
computeMaxPathDifference	Float64	enum Band::Wavelength	theBand
		UInt32	interferogramSize
Calculates the user apodization table. This calculation is only band specific.			
buildUserApodizationTable	primitive bool	enum Band::Wavelength	theBand
Holds the intelligence to route any RDR packet to assistant class processing.			
processRawDataRecord	Class Spectra*	n/a	n/a
Reads external environment data from a file.			
processEnvironmentalData	primitive bool	typedef struct SYSTEMTIME&	theSpectraFrameTime
Builds polarization curve.			
buildPolarizationCurve	primitive void	Class Spectra*	newSpectra
		enum FiledOfRegard	theScene
		enum Bans::Wavelength	theband
Creates and adds a new spectrum to the spectrasliding window. Reference spectra are simply returned.			
AddSpectra	Class Spectra*	Class Spectra*	newSpectra
Builds each child table for every band and fov.			
refreshCorrectionMatrix	primitive int	primitive bool	isloading
Refreshes the self-apodization correction matrix			
refresh_Invsa_CorrectionMatrix	primitive int	primitive bool	isloading
Multiplies child correction table to calculate the final calibration matrix.			
calculateCalibrationMatrix	primitive void	n/a	*n/a*
Combines requested corrections.			
combineRequestedCorrections	typedef unsigned int UInt32	typedef unsigned int UInt32	requestedFOV
Updates correction matrix parameters from configuration file.			
updateCorrectionmatrixParameters	primitive void	n/a	n/a
Checks if there is a missing sciCal pkt in the last scan of the moving window.			
checkICT_Status	void	typedef int Int32	scanIdx
		typedef struct CrisSdrAlgDataType*	algDataPtr
Calculates the hot radiance.			
calculateHotRadiance	Class Spectra*	enum SceneElement	theFOV
		enum Band::Wavelength	theBand
Calculates the cold radiance.			
calculateColdRadiance	Class Spectra*	enum SceneElement	theFOV
		enum Band::Wavelength	theBand
Sets the window size base on the spectral calibration method.			
setSpectralCalibrationMethod	primitive void	enum SpectralCalibrationMethod	newMethod
Removes a spectrum from the sliding window without adding a spectrum.			
flushWindow	Class Spectra*	enum SceneElement	fov

Name	Return	Parameter	
		enum Band::Wavelength	band
		enum FieldOfRegard	scene
		enum SweepDirection	psd
Processes Earth Scene spectra according to configuration parameters. Calibration, correction and validity are evaluated.			
processSpectra	Class Spectra*	Class Spectra*	newSpectra
Batch Earth Scene Fringe Count Error.			
batchEarthSceneFCE	bool	Class Spectra*	newSpectra
		Class Spectra*	rawSpectra
		CrisSdrCoefficients*	cfgParmsPtr
Calibrate			
calibrate	void	Class Spectra*	newSpectra
		Class Spectra*	hotRefSpectra
		Class Spectra*	coldRefSpectra
		SceneElement*	oldFOV
		Enum Band::Wavelength	oldBand
Monitors the laser diode to determine if the CMO needs to be updated. This method also update the SDR Monitored Laser Wavelength and SDR Spectral Resampling laser Wavelength.			
moniterLaserDiode	primitive bool	class Spectra*	theSpectra
Determines if the laser wavelength needs to be updated and updates if needed.			
monitorLaserWavelength	primitive bool	Class Spectra*	theSpectra
Propagates temporal window size to assistant classes.			
setWindowSize	primitive void	primitive int	newSize
Propagates temporal window size to assistant classes.			
declareWindowSize	primitive void	typedef unsigned int UInt32	referenceSize
		typedef unsigned int UInt32	targetTempSize
		typedef unsigned int UInt32	CalibrationSize
Loads each correction matrix from a DMS			
LoadCMOs	primitive bool	n/a	n/a
Loads EngPkt from a DMS			
LoadEngPkt	primitive bool	n/a	n/a
Calculates desired band center and configures spectra's desired band center.			
calculateDesiredBandCenter	primitive void	n/a	n/a
Applies the CMO to a spectrum.			
applySpectralCorrection	primitive void	Class Spectra*	newSpectra
		enum SceneElement	theFov
		enum Band::Wavelength	theBand
Saves serialized CMO. The values are from correction matrix class serialized.			
saveSerializedCorrectionMatrix	primitive void	n/a	*n/a*
Write 4-min engineering packet to DMS.			
saveSerializedEngPkt	primitive void	n/a	*n/a*
Loads serialized CMO from DMS and populates correction matrix class.			
loadSerializedCorrectionmatrixFromDMS	primitive void	n/a	*n/a*
Computes field of views line-of-sight in spacecraft body frame.			

Name	Return	Parameter	
calculateGeometricCalibration	Class Spectra*	Class Spectra*	*newSpectra*
Refreshes a spectrum time from telemetry data.			
refreshTime	primitive void	Class CCSDSFormat*	*theTelemetryFormat*
Synchronizes all windows based on reference spectra sequence counts and updates sequence counts.			
syncWindows	primitive void	n/a	*n/a*
Calls the Spectra manager's batchReferenceFCE which does fringe count detection and correction.			
launchBatchreferenceDetect	primitive void	Class SpectraManager::RefernceSpectra	*spectraType*
		enum SweepDirection	*thePSD*
Interpolates a subset such that individual offset values areavailable for each 8 seconds scan period.			
buildBaffleTemperatureOffset	primitive void	Class BOOST::vector<Float32>&	*source*
		typedef int int32	*orbitDuration*
Calculates the gain introduced by FIR Filter.			
buildFirFilterGainTable	primitive void	enum Band::Wavelength	*requestdBand*
		typedef int int32	*foldindex*

3.1.2.9 Class SpectraManager

This class is responsible for each Spectra currently used in the moving window average. Table 3.1.2.9.1-1 shows the SpectraManager attributes and Table 3.1.2.9.2-1 shows the SpectraManager operations.

3.1.2.9.1 SpectraManager Attributes

Table 3.1.2.9.1-1. SpectraManager Attributes

Name	Type	Description
earthSceneTemp_Q[ ][ ][ ]	Class BOOST::vector<std::complex<Float64> >	*Collection of deep space spectra*
hotColdDiff[ ][ ][ ]	Class BOOST::vector<std::complex<Float64> >	*Collection of differences between hot and col spectra*
windowSize	typedef int Int32	*Window size*
monitorWindowTime	primitive bool	*Monitor window time*
measuredSpectra[ ][ ][ ][ ]	typedef list<Spectra*> SpectraCollection	*Collection of measured spectra*
streamReference	typedef list<Spectra*> SpectraCollection	*Stream reference*
referenceSpectra[ ][ ][ ][ ]	Class Spectra*	*Collection of reference scenes*
windowSize	primitive int	*Depth of reference average*
SpectraCollection	typedef SpectraCollection	*Collection of measured scenes*
referenceSpectra[ ][ ][ ][ ]	Class Spectra*	*Collection of calculated average scenes*
hotReferenceFieldOfRegard	enum FieldOfRegard	*Hot reference field of regard*
coldReferenceFieldOfRegard	enum FieldOfRegard	*Cold reference field of regard*
refWindowSize	typedef int Int32	*Reference window size*
calWindowSize	typedef int Int32	*Calibration window size*
valWindowSize	typedef int Int32	*Valid window size*
maintainReferenceSceneVarianc	primitive bool	*Reference scene variance*

Name	Type	Description
e		
correctionFilename[ ][ ]	typedef list<string> CMOCollection	*Collection matrix file names*
polarizationSpectra[ ][ ]	typedef SpectraCollection	*Collection of polarization spectra*
windowTimeSpan[ ][ ][ ][ ]	typedef struct SYSTEMTIME	*Collection of window time*
radiance[ ][ ][ ]	Class Spectra*	*Collection of radiance spectra*
rawSummationSpectra[ ][ ][ ][ ]	Class Spectra*	*Collection of raw spectra summation*
lastSpectra[ ][ ][ ][ ]	Class Spectra*	*Collection of last spectra*
currentCorrectionFilename[ ][ ]	primitive string	*Collection of current file name*
currentPolarizationSpectra[ ][ ]	Class Spectra*	*Collection of current polarization spectra*
geolocationSpectra[ ][ ]]	typedef list<Spectra*> SpectraCollection	*Collection of current geolocation spectra*
rawSummationSpectraNoCal[ ][ ][ ][ ]	Class Spectra*	*Collection of no calibrated spectra summation*
squareSummationSpectraNoCal[ ][ ][ ][ ]	Class Spectra*	*Collection of no calibrated spectra square summation*
trackLunarIntrusion	primitive bool	*Lunar intrusion flag*
maxLunarIntrusionRatio[]	typedef float Float32	*Maximum lunar intrusion ratio*
minFreqMicroWindow[]	typedef double Float64	*Minimum wavenumber to detect lunar intrusion*
maxFreqMicroWindow[]	typedef double Float64	*Max wavenumber to detect lunar intrusion*
edrMinimumWavenumber[ ]	typedef double Float64	*EDR minimum wavenumber*
edrMaximumWavenumber[ ]	typedef double Float64	*EDR maximum wavenumber*
lunarIntrusionCount[ ][ ][ ]	typedef unsigned int UInt32	*Collection of lunar intrusion counts*
requiresPhaseSync[ ][ ][ ]	primitive bool	*Collection of spectra phase sync*
lastReferenceSpectra[ ][ ][ ][ ]	Class Spectra*	*Collection of last reference spectra*
holdingReferenceSpectra[ ][ ][ ][ ]	Class Spectra*	*Collection of holding reference spectra*
LinearityCorrectionParameter[ ]	typedef struct LinearityCorrectionParameters	*Collection of linearity correction parameters*
FIR_GAIN[ ]	Class Spectra*	*Collection of FIR gain*
performLinearityCorrectionControl[ ]	primitive bool	*Collection of information for performing linearity correction*
boxcarAverageWidth	typedef int Int32	*smoothing width for NEdN Estimate*
calibrationMatrix[][]	BOOST::matrix<Float64>*	*Calibration Matrix*
resamplingMatrix[]	BOOST::matrix<Float64>*	*Resampling Matrix*
CorrectionParameter	SDR_CorrectionParam*	*Reference to SDR correction parameters*
CalibrationParameter	CalibrationParam*	*Reference to* Calibration parameters

3.1.2.9.2 SpectraManager Operations

Table 3.1.2.9.2-1. SpectraManager Operations

Name	Return	Parameters	
Cleans up all obsolete spectra.			
Cleanup	primitive void	n/a	*n/a*
Initializes data for re-tasking.			
Init	primitive void	n/a	*n/a*
Adds a correction spectrum to the moving window for synchronized application to calibration.			
scheduleCorrection	primitive void	primitive string	*newCMOFileName*
		enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
Adds the newSpectra to the measure spectra container.			
addSpectra	Class Spectra*	Class Spectra*	*newSpectra*
Adds hot and cold target curves to the radiance container for the current spectra.			

Name	Return	Parameters	
addTargetRadianceCurves	primitive void	Class Spectra*	*hotTargetCurve*
		Class Spectra*	*coldTargetCurve*
Synchronizes the windowed data in the event the data set does not begin on an 8 sec boundary.			
syncReferenceSpectra	primitive bool	Class Spectra&	*targetSpectra*
		primitive bool	*performFringeCountErrorHandling*
Reconciles/Synchronizes the reference spectra's phase to the target spectra's phase.			
reconcileReferenceSpectraPhase	primitive bool	SceneElement	*theFOV*
		enum Band::Wavelength	*theBand*
		enum SweepDirection	*thePSD*
		typedef int Int32	*maxFCETries*
Returns requested measured spectra.			
getMeasuredSpectra	Class Spectra*	typedef int Int32	*temporalItem*
		enum SceneElement	*fov*
		enum Band::Wavelength	*band*
		enum FieldOfRegard	*scene*
		enum SweepDirection	*psd*
Returns requested correction spectra (mathematical correction matrix).			
getCorrectionFilename	primitive string	typedef int Int32	*temporalItem*
		enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
Gets the valid count of spectra for a given fov, band, scene, and PSD			
getValidCount	typedef int Int32	enum SceneElement	*fov*
		enum Band::Wavelength	*band*
		enum FieldOfRegard	*scene*
		enum SweepDirection	*psd*
Sets the internal window size. If the content of the current window exceeds the new size, the oldest contents are discarded.			
setWindowSize	primitive void	typedef int Int32	*numberOfSpectra*
Forces a Spectra out of the window without adding one.			
purgeSpectra	Class Spectra*	enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
		enum FieldOfRegard	*theFOR*
		enum SweepDirection	*thePSD*
Maintains Slope and Intercept values for rapid mathematical use.			
buildNoisePrediction	buildNoisePrediction	enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
		enum SweepDirection	*thePSD*
Propagates temporal window size to assistant classes.			
declareWindowSize	typedef unsigned int UInt32	typedef unsigned int UInt32	*referenceSize*
		typedef unsigned int UInt32	*calibrationSiz*
Marks the calibration window invalid for current contents.			
resetCalibrationWindow	primitive void	enum Band::Wavelength	*band*
		typedef unsigned int UInt32	*newFoldIndex*
Returns the total count of all buffering windows requested.			
getBufferedSpectraCount	typedef unsigned int UInt32	primitive bool	*includeReferenceCount*
Adds a polarization correction spectra to the moving window for synchronized application.			
schedulePolarizationCorrection	primitive void	Class Spectra*	*newSpectra*
		enum FieldOfRegard	*patternScene*
		enum Band::Wavelength	*patternBand*
Returns requested polarization correction spectra.			
getPolarizationSpectra	Class Spectra*	typedef int Int32	*temporalItem*

Name	Return	Parameters	
		enum FieldOfRegard	*theScene*
		enum Band::Wavelength	*theBand*
Adds a geolocation spectrum to the moving window for synchronized application to calibration.			
scheduleGeolocation	primitive void	Class Spectra*	*newSpectra*
		enum FieldOfRegard	*theScene*
		enum SceneElement	*theFov*
Returns requested geolocation spectra.			
getGeolocationSpectra	Class Spectra*	typedef int Int32	*temporalItem*
		enum FieldOfRegard	*theScene*
		enum SceneElement	*theFov*
Calibrates the newMeasurement and performs a ratio threshold check against the corresponding window average.			
monitorLunarIntrusion	primitive bool	Class Spectra&	*newMeasurement*
Adjusts window phase.			
adjustWindowPhase	typedef int Int32	enum FieldOfRegard	*theFOR*
		enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
		enum SweepDirection	*thePSD*
		Class BOOST::vector< std::complex<Float64> >*	*linearPhaseShift*
Removes a given spectra from the rawSummationSpectra and squareSummationSpectra.			
removeInvalidSpectra	primitive void	Class Spectra*	*oldSpectra*
		enum SpectralCalibrationMethod	*calType*
Starts a batch processing job on holdingReferenceSpectra. Detects FCE on each Ref Scene starting with LW1. If LW1 does not detect we move on to LW2, and so on and so forth. When a detection passes that fringe count is applied to each fov, band, and PDS of that FOR. The ref scene is added to the window.			
batchReferenceFCE	primitive void	enum ReferenceSpectra	*spectraType*
		enum SweepDirection	*psd*
Adds new reference spectra to holdingReferenceSpectra container.			
addHoldingReferenceSpectra	primitive void	Class Spectra*	*newSpectra*
ICT and DS Synchronization batch job that is done when the first ES pops out of the window.			
batchICTDSSynchronization	primitive bool	Class Spectra*	*oldSpectra*
		primitive int	*maxFCETries*
Starts a batch processing job of Earth Scene Spectra. The argument oldSpectra is the spectra that just popped out of a window for a certain FOR and PSD. Detection is started on LW1, and continues for each FOV and Band until a fringe count passes or all FOVs and Bands are exhausted.			
batchEarthSceneFCE	primitive bool	Class Spectra*	*oldSpectra*
Calculates the estimated detector voltage at the preamp.			
calculateVdc	primitive bool	Class Spectra*	*sceneSpectra*
Calculates the variance of DS or ICT in the sliding window.			
calculateVariance	primitive void	enum ReferenceSpectra	*spectraType*
		enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
		enum SweepDirection	*thePSD*
A combination of spectra calibration calls			
calibrate	primitive void	Class Spectra*	*newSpectra*
		Class Spectra*	*hotRefSpectra*
		Class Spectra*	*coldRefSpectra*
		enum SceneElement	*theFov*
		enum Band::Wavelength	*theBand*
Apply polarization correction on user grid after calibation			
polarizationCorrection	primitive void	Class Spectra*	*newSpectra*
		Class Spectra*	*hotRefSpectra*

3.1.2.10 Class Spectra

This class is the container for the processed video data. Table 3.1.2.10.1-1 shows the Spectra attributes and Table 3.1.2.10.2-1 shows the Spectra operations.

3.1.2.10.1 Spectra Attributes

Table 3.1.2.10.1-1. Spectra Attributes

Name	Type	Description
response	typedef double Float64	Response flag
realWavenumberBin	Class BOOST::vector<double>	Collection of real component of complex spectra
imaginaryWavenumberBin	Class BOOST::vector<double>	Collection of imaginary component of complex spectra
NEdN_Estimate	Class BOOST::vector<Float64>	Collection of NEdN estimation
firstWavenumber	typedef double Float64	Frequency of amplitude in bin zero
wavenumberBinSize	typedef double Float64	Frequency offset per bin (deltaSigma)
getVersionNumber	typedef unsigned short int	Version number
foldIndex	typedef unsigned int UInt32	Index
fceParameters[ ]	typedef struct FCEParam	Collection of FCE parameters
spectralBand	enum Band::Wavelength	Actual band reported in interferogram
fieldOfView	enum SceneElement	Actual field of view reported in interferogram
fieldOfRegard	enum FieldOfRegard	Actual field of regard reported in interferogram
sweepDirection	enum SweepDirection	Actual porch swing direction reported in interferogram
userLocks	typedef int Int32	User lock flag
scalingEnabled	primitive bool	Scaling enabled flag
Channel	typedef int Int32	Channel number
RDR_Status	typedef struct RawDataRecordStatusRegister	Collection of interferogram status information
saveVersion400	primitive void	Version number

3.1.2.10.2 Spectra Operations

Table 3.1.2.10.2-1. Spectra Operations

Name	Return	Parameters	
References to the root of the complex squares.			
getMagnitude	Class BOOST::vector<Float64 >&	n/a	*n/a*
Resizes the bin count.			
setSize	primitive void	typedef unsigned int UInt32	*newSize*
Clears internal data vectors.			
clear	primitive void	n/a	*n/a*
Adds complex squares.			
addSquare	primitive void	Class Spectra&	*source*
		primitive bool	*complex*

Name	Return	Parameters	
Removes complex squares.			
removeSquare	primitive void	Class Spectra&	*source*
		primitive bool	*complex*
Divides each element of the spectra by a specified number.			
operator/	primitive void	typedef double Float64	*factor*
Multiplies each element of the spectra by a specified number.			
operator*	primitive void	typedef double Float64	*factor*
Multiplies each element of the spectra bin by bin with the factor.			
operator*	primitive void	Class Spectra&	*factor*
Divide the spectra by a specified spectra.			
operator/	primitive void	Class Spectra&	*divisor*
Makes a copy of source spectra.			
operator=	Class Spectra&	Class Spectra&	*source*
Adds a spectrum to the current spectra.			
operator+	primitive void	Class Spectra&	*rightOperand*
Subtracts a spectrum from the current spectra.			
operator-	primitive void	Class Spectra&	*rightOperand*
Returns packed content of the class.			
save	primitive string	n/a	*n/a*
Clips and re-samples it's self to new resolution.			
resample	primitive bool	typedef double Float64	*startWavenumber*
		typedef double Float64	*stopWavenumber*
		Typedef int Int32	*binSize*
Clips and re-samples it's self to new resolution.			
clipGuardBands	primitive bool	typedef double Float64	*startWavenumber*
		typedef double Float64	*stopWavenumber*
Returns packed content of the class.			
saveVersion400	primitive void	primitive string&	*binaryImage*
Generates NEdN estimations.			
generateUSN	primitive void	Class Spectra*	*rawSum*
		Class Spectra*	*squaredSum*
		typedef int Int32	*windowSize*
Smoothes the spectra.			
Smooth	primitive void	typedef int Int32	*windowSize*
Applies linearity error correction to the spectra.			
applyLinearityErrorCorrection	primitive void	n/a	*n/a*
Updates CrIS SDR geolocation.			
updateCrisSdrGeolocation	primitive void	typedef struct CrisSdrAlgDataType*	*algDataPtr*
		typedef int Int32	*scanIdx*
Updates CrIS SDR data.			
updateCrisSdrData	primitive void	typedef struct CrisSdrAlgDataType	**sdrPtr*
		typedef struct CrisSdrHdrDataType	**hdrPtr*
		typedef unsigned int UInt32	*scanIdx*
		std::auto_ptr<ScienceDataProcessor>&	*ScienceDataProcessorPtr*
After spectra phase shifted re-generate interferograms.			
reGenerateInterferogram	void		

3.1.2.11 Class TelemetryProcessor

This class provides the interface to retrieve science data from raw instrument telemetry. It encapsulates the extraction of housekeeping, diagnostic and normal mode interferograms. Table 3.1.2.11.1-1 shows the TelemetryProcessor attributes and Table 3.1.2.11.2-1 shows the TelemetryProcessor operations.

3.1.2.11.1 TelemetryProcessor Attributes

Table 3.1.2.11.1-1. TelemetryProcessor Attributes

Name	Type	Description
LRV	typedef TelemetryRecord	*Last received value*
apidInfoMap	typedef std::map<UInt16, ApidInfoStruct*> ApidInfoMap	*Collection of apid information*
apidCollection	typedef std::vector<UInt16> ApidCollection	*Collection of apid information*
ValueToChannelMap	typedef std::map<Int32, UInt32> ValueToChannelMap	*Collection of value to channel information*
channelToDetector[ ][ ]	typedef unsigned int UInt32	*Collection of channel to detector*
detectorToChannel[ ][ ]	typedef unsigned int UInt32	*Collection of detector to channel*
lastFrameTime	typedef struct SYSTEMTIME	*Last frame time*
lastFrameMiliTime	typedef unsigned int UInt32	*Last frame mili time*
lastFrameMicroTime	typedef unsigned short int UInt16	*Last frame micro time*
lastFrameDayTime	typedef unsigned short int UInt16	*Last frame day time*
telemetryLookup	Class map<string, Int32>	*Collection of telemetry lookup data*
fourMinPktUpdateIndex	typedef int Int32	*Four minutes package update index*
theCurrentVideoSource	enum BurstModes	*Collection of burst modes*
theVideoMode	enum BurstModes	*The video mode*
theEndOfEightSecEpoch	primitive bool	*The end of eight seconds epoch*
purgeCount	typedef int Int32	*The purge count*
theTelemetryVersion	enum TelemetryVersions	*Collection of telemetry versions*
fullVideoMode	primitive bool	*Full video mode flag*

3.1.2.11.2 TelemetryProcessor Operations

Table 3.1.2.11.2-1. TelemetryProcessor Operations

Name	Return	Parameters	
Returns index of first telemetry point with the specified APID.			
findRecord	typedef int Int32	typedef unsigned short int UInt16	*targetFrameType*
findRecord	typedef int Int32	primitive string	*targetName*
makePoint	primitive void	primitive string&	*pointName*
		primitive string&	*pointDescription*
		primitive string&	*subsystem*
		primitive string&	*frameGroup*
		typedef unsigned short int UInt16	*frameType*
		Class TelemetryPoint::TelemType	*telemType*
		typedef unsigned int UInt32	*bitLocation*
		typedef unsigned int UInt32	*pointSize*
		primitive string&	*pointUnits*
		typedef unsigned int UInt32	*telemetryPointID*
		Class	*coefficientContexts*

Name	Return	Parameters	
		TelemetryPoint::CoefContextCollection*	
		Class TelemetryPoint::LimitContextCollection*	*limitContexts*
Returns a constant reference to the telemetry point at the given index.			
getRecord	Class TelemetryPoint&	typedef int Int32	*vectorIndex*
Clears and performs cleanup operations on the telemetry record.			
purgeTelemetryRecord	primitive void	n/a	*n/a*
Returns the name of the frame group to which the specified APID belongs.			
identifyFrameType	primitive string	typedef unsigned short int UInt16	*keyAPID*
Parses a single telemetry frame.			
parseDisplayData	primitive void	typedef unsigned short UInt16	*theAPID*
		typedef struct SYSTEMTIME	*&theTime*
		typedef unsigned int UInt32	*theMiliTime*
		typedef unsigned short UInt16	*theMicroTime*
		typedef unsigned short UInt16	*theDayTime*
		typedef unsigned char UInt8	**newData*
Extracts the video data from the supplied data.			
extractVideoData	primitive void	typedef unsigned int UInt32	*detector*
		typedef unsigned int UInt32	*wavelength*
		typedef unsigned char UInt8	**trimmedData*
Extracts the diagnostic data from the supplied data.			
extractDiagnosticData	primitive void	typedef unsigned short UInt16	*theAPID*
		typedef unsigned int UInt32	*detector*
		typedef unsigned int UInt32	*wavelength*
		typedef unsigned char UInt8	**detData*
		typedef int Int32	*dataOffset*
Matches the specified APID with a detector and a wavelength.			
rdrLookup	primitive void	typedef unsigned short UInt16	*theAPID*
		typedef unsigned int UInt32	**detector*
		typedef unsigned int UInt32	**wavelength*
		typedef unsigned int UInt32	**scene*
Resets LastBurstApidDetection by forgetting that any of the APIDs were in the last burst.			
resetLastBurstApidDetection	primitive void	n/a	*n/a*
Updates TelemetryPoint stale value.			
updateStaleValues	primitive void	n/a	*n/a*
Checks whether the specified APID is recent.			
isApidRecent	primitive bool	typedef unsigned short UInt16	*theApid*
Finds and returns the info associated with the specified apid.			
findApidInf	typedef struct ApidInfoStruct*	typedef unsigned short UInt16	*apid*
Reads telemetry points data.			
readTelemetryPoints	primitive void	n/a	*n/a*
Checks if telemetry record defined.			
enableLimitChecking	primitive void	primitive bool	*limitCheck*

3.1.2.12 Class VideoData

This class is responsible for internal bit-trim reversal behavior of the TelemetryProcessor class. Table 3.1.2.12.1-1 shows the VideoData attributes and Table 3.1.2.12.2-1 shows the VideoData operations.

3.1.2.12.1 VideoData Attributes

Table 3.1.2.12.1-1. VideoData Attributes

Name	Type	Description
theExtractionRecord[ ]	struct EXTRACTION_RECORD	Collection of band specific bit trim values
theTlmLookupTable[ ]	struct TELEMETRY_LOOKUP	Collection of complex samples
frStartBit[] frStopBit[] frTrimTable[]	UInt32[][]	Contain original trim table values from the engineering packet (before truncation if using full resolution data)

3.1.2.12.2 VideoData Operations

Table 3.1.2.12.2-1. VideoData Operations

Name	Return	Parameters	
Populates the internal symbol table used to unpack and extract the bit-trimmed data set.			
initSymbolTable	primitive void	primitive bool	*useDefaultTable*
Coordinates the real and imaginary extraction of the complex samples into the real and imaginary portions.			
Extract	primitive void	typedef unsigned int UInt32	*detector*
		typedef unsigned int UInt32	*wavelength*
		typedef unsigned char UInt8	**trimmedData*
Retrieves the correct trim value from the symbol table for a given wavelength sample.			
getTrimSize	typedef unsigned short UInt16	typedef unsigned int UInt32	*wavelength*
		typedef int Int32	*sample_index*
Established the 4 minute bit trim tables for all three bands.			
buildExtractionTables	primitive void	primitive bool	*useDefaultTable*
Extracts the original sample from the pack data stream by using the bit-trimming information configured in the symbol table.			
restoreSamples	primitive void	typedef unsigned int UInt32	*detector*
		typedef unsigned int UInt32	*wavelength*
		typedef unsigned short int UInt16	*dataElement*
		primitive bool	*realComponent*
Updates data values in the extradction record			
setExtractionRecord	EXTRACTION RECORD struct	EXTRACTION_RECORD	*record*
		UInt32	*band*

3.1.2.13 Class ScienceCalibrationRecord

This class is responsible for processing the eight second science calibration packet (Refer to the Equation (79e) in Section 5.5 Temperature Computation, BOM-CrIS-0067 Revision E. The constants 1, 1, 2 and 5 are the coefficients of the Taylor series expansion (Note: ITT modifies these coefficients in V2.14.0)). It maintains the sliding window average ICT temperature. Table

3.1.2.13.1-1 shows the ScienceCalibrationRecord attributes and Table 3.1.2.13.2-1 shows the ScienceCalibration operations.

3.1.2.13.1 ScienceCalibrationRecord Attributes

Table 3.1.2.13.1-1. ScienceCalibrationRecord Attributes

Name	Type	Description
CalibrationParameter	typedef struct CalibrationParam	Collection of calibration configuration values
TestEnvironmentParameter	typedef struct CrIS_TestEnvironmentParameters	Collection of instrument configuration values
GeneralParameter	typedef struc SDR_GeneralParameters	Collection of processing configuration values
AlgorithmParameter	typedef struct SDR_AlgorithmnParameter	Collection of algorithm configuration values
processingIctPeriod	Class TemperatureHistory	Collection of temperature measurements
processingIctSensor1	Class TemperatureHistory	Collection of temperature measurements
processingIctSensor2;	Class TemperatureHistory	Collection of temperature measurements
processingBeamsplitterPeriod	Class TemperatureHistory	Collection of temperature measurements
processingScanMirrorPeriod	Class TemperatureHistory	Collection of temperature measurements
processingOmaPeriod	Class TemperatureHistory	Collection of temperature measurements
processingScanBafflePeriod	Class TemperatureHistory	Collection of temperature measurements
processingTelescopePeriod	Class TemperatureHistory	Collection of temperature measurements
processingCoolerStage1Period	Class TemperatureHistory	Collection of temperature measurements
processingCoolerStage2Period	Class TemperatureHistory	Collection of temperature measurements
processingCoolerStage3Period	Class TemperatureHistory	Collection of temperature measurements
processingCoolerStage4Period	Class TemperatureHistory	Collection of temperature measurements
processingLaserDiodeCurrentPeriod	Class TemperatureHistory	Collection of temperature measurements
processingLaserDiodeTempPeriod	Class TemperatureHistory	Collection of temperature measurements
processingPeriod	Class TemperatureHistory	Collection of temperature measurements
ieCCACalibResistorTemp[ ]	typedef int Int32	Collection of resistor temperature measurements
lowRangeCalibResistor[ ]	typedef int Int32	Collection of low range resistor measurements
highRangeCalibResistor[ ]	typedef int Int32	Collection of high range resistor measurements
ictTemp[ ][ ]	typedef float Float32	Collection of ICT temperature measurements
ictRaw[ ][ ]	typedef int Int32	Collection of ICT temperature measurements
crossTrackServoErr[ ]	typedef int Int32	Collection of cross track servo error measurements
inTrackServoErr[ ]	typedef int Int32	Collection of in track servo error measurements
laserDiodeCurrent	typedef int Int32	Laser diode current
laserDiodeTemp	typedef int Int32	Laser diode temperature
beamsplitterTemp	typedef int Int32	Beam splitter temperature
ssmScanMirrorTemp;	typedef int Int32	Scan mirror temperature
ssmScanMirrorBaffleTemp	typedef int Int32	Scan mirror baffle temperature
coolerStage1Temp	typedef int Int32	Cooler stage1 temperature
coolerStage2Temp	typedef int Int32	Cooler stage2 temperature
coolerStage3Temp	typedef int Int32	Cooler stage3 temperature
coolerStage4Temp	typedef int Int32	Cooler stage4 temperature
telescopeTemp;	typedef int Int32	Telescope temperature
SciTeleLimits	typedef struct Limits	Structure to hold a variety of DriftLimits
EngConvCoeff	typedef struct Coefficients	Structure to hold a variety of Coefficients
captureNextLaserInfo	primitive bool	Next laser information
capturedLaserDiodeCurrent	typedef double Float64	Laser diode current

capturedLaserDiodeTemperature	typedef double Float64	Laser diode temperature
previousAveragedLaserDiodeTemp	typedef double Float64	Previous averaged diode temperature
previousAveragedLaserDiodeCurrent	typedef double Float64	Previous averaged diode current
servoValuesChanged	primitive bool	Servo value change flag
convertedCrossTrackServoErr[ ]	typedef double Float64	Converted cross track servo error
convertedInTrackServoErr[ ]	typedef double Float64	Converted in track servo error
sciCalMissing	Primitive bool	Flag indicating science calibration missing.

3.1.2.13.2 ScienceCalibrationRecord Operations

Table 3.1.2.13.2-1. ScienceCalibration Operations

Name	Return	Parameters	
Calls clearHistory to free memory.			
clearHistory	primitive void	n/a	*n/a*
Sets window size.			
setWindowSize	primitive void	typedef int Int32	*newSize*
Reads Values from Science Telemetry Packet into class variables.			
refreshData	primitive bool	typedef unsigned short UInt16	*apid*
		typedef struct CrisSdrAlgDataType*	*algDataPtr*
		typedef struct IngMsdCoefficients_CrisSdrStruct*	*cfgParmsPtr*
		typedef unsigned int UInt32	*scanIdx*
Calculates ICT temperature and adds to sliding window average.			
calculateIctTemperature	primitive void	Class TemperatureHistory::AvgPRTTemp	**avgIctTempPtr*
		typedef struct IngMsdCoefficients_CrisSdrStruct*	*cfgParmsPtr*
Calculates Beamsplitter temperature and adds to sliding window average.			
calculateBeamsplitterTemperature	typedef double Float64	n/a	n/a
Calculates Scan Mirror temperature and adds to sliding window average.			
calculateScanBaffleTemperature	typedef double Float64	n/a	n/a
Calculates OMA temperature and adds to sliding window average.			
calculateOmaTemperature	typedef double Float64	n/a	n/a
Adds a missing temperature record to the temperature history window.			
markMissingCal	primitive void	n/a	n/a
Removes last temperature record from the temperature history window.			
removeLastCal	primitive void	n/a	n/a
Finds the max and min temperatures in the temperature history window and determines if the difference exceeds the Science Telemetry Drift Limits.			
hasExcessThermalDrift	primitive bool	n/a	n/a

3.1.2.14 SDR Generator Application COTS Components

3.1.2.14.1 uBLAS (BOOST)

BOOST, uBLAS provides templated C++ classes for elementary linear algebra, and access into vectors and matrices by way of matrix and vector adapters.

3.1.2.14.2 LAPACK

CrIS SDR uses the LAPACK dgetrf and dgetri functions when inverting matrices.

3.1.2.14.3 FFTW

CrIS SDR uses three one-dimensional discrete Fourier transform routines from FFTW; one which performs DFT on real data returning only the real part of the result, another which performs DFT on real data and returns a complex result, and finally, a function that performs DFT on complex data and returns a complex result.

3.1.2.15 The Sliding Window

The C++, CrIS SDR code, converted from MATLAB by ITT, was designed to work with a continuous stream of data. The processing model used by JPSS sends data to the CrIS SDR processes in discrete (not necessarily contiguous) chunks called granules. These granules are stored in data structures called RDRs.

CrIS granules contain one or more scans. As shown in Figure 3.1.2.15-1, each scan consists of four reference FORs and 30 earth view FORs. Each FOR is composed of nine FOVs which, in turn, are comprised of three wavelength bands (LW, MW, and SW). Each FOR/FOV/wavelength combination is processed independently and stored in separate sliding windows.

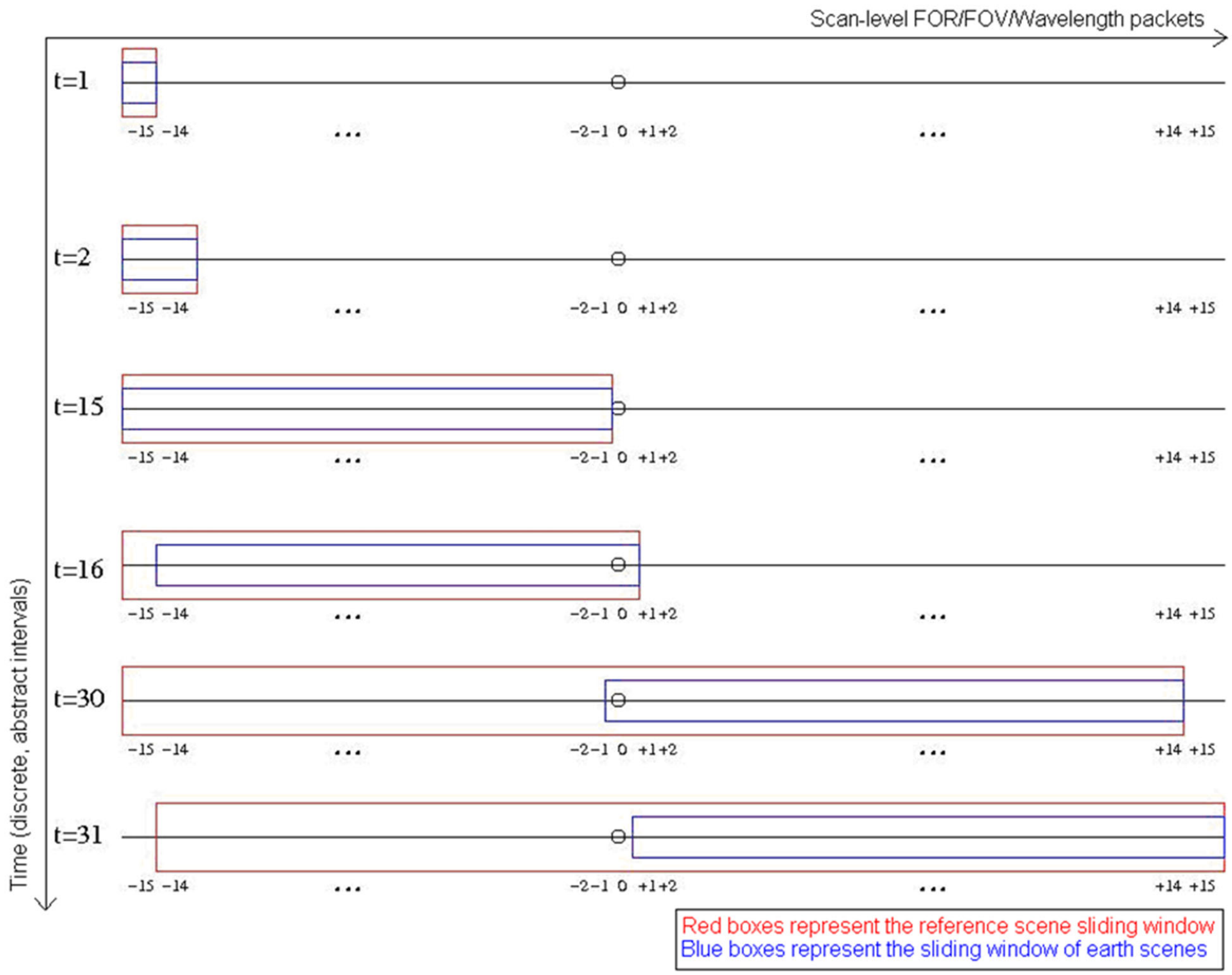

Figure 3.1.2.15-1. Sliding Window Concept

Here, the red box is the reference scene sliding window and the blue box is the sliding window of earth scenes. The figure shows the process of building the sliding windows from nothing and then processing one scan.

Because only the earth-scene “ES” FORs are calibrated, there are 9 “FOVs” x 3 “IR Bands x 2 “mirror sweep directions” separate ICT and DS sliding windows for a given scan. A single sliding window will contain a nominal 30 ICT or DS individual raw spectra. The index 30 does not refer to the number of FOR per scan but rather an average of the ICT or DS over 30 scans centered about the Earth scene spectrum.

Earth view spectra are also stored in a smaller sliding window. However, this sliding window has no scientific value and is used as a matter of convenience to assist tracking when an Earth scene spectrum should be processed. Because these spectra are not stored for processing other Earth view scans, only future Earth views need to be saved in this smaller window for future processing. Earth view spectra are processed upon being ejected from their sliding window. In the following sections, sliding window refers to the window of reference scenes.

The CrIS SDR implementation has hard-coded the sliding window size at 30, although this parameter is configurable via a global header file. (Consequently, however, this does imply that reconfiguration of the parameter requires a recompilation of the CrIS SDR library.) The earth view sliding window is one-half that size (current scan plus 14 future scans). The process of adding a single scan to a sliding window is shown in Figure 3.1.2.15-1.

3.1.2.15.1 One Additional Spectra is Needed to Move the Window

All Earth view spectra that are calibrated using a partial sliding window are marked as being of degraded quality. Earth view spectra are processed when ejected from the sliding window. Therefore, not only must enough scenes be retrieved to keep the sliding window full during processing, enough scans must be retrieved to eject the tasked spectra from the window. For example, when processing with granules consisting of a single scan, 31 scans must be retrieved (15 prior, plus 14 after, plus the scan contained within the granule that has been tasked to be processed to fill the window, plus one additional scan to eject the desired processed Earth views). If granules containing four scans were being processing, then 34 scans would be required.

3.1.2.15.2 Unaligned Scans Require Retrieval of Yet Another Scan

CrIS SDR scans are not guaranteed to align with granules boundaries. In other words, the time associated with the beginning of a scan is not equal to the time associated with the beginning of a granule. This implies that the RDR data structure need not begin with the first spectra of the first scan of the granule that has been tasked to process.

This CrIS SDR implementation ensures that the last scan is retrieved in its entirety by requesting enough granules to obtain yet another scan. This ensures success if at least a part of an extra scan has been retrieved; there must be complete scans for all scans before the extra scan. CrIS SDR uses the ceilings of the following formulae to liberally ensure that the entirety of each scan retrieved associated with the granule that has been tasked to process, where W represents the sliding window size and S represents the number of scans per granule:

Number of past granules to retrieve: $\left\lceil \dfrac{\frac{W}{2}}{S} \right\rceil$

Number of future granules to retrieve: $\left\lceil \dfrac{\frac{W}{2} + S}{S} \right\rceil$.

3.1.2.15.3 Granule Version Ids

The granules containing science data have a version id as a part of their meta-data. Occasionally, an RDR needs to be updated; and when this occurs, its granule version ID is modified as well.

Although not all scans are processed, all granules needed for processing are retrieved. Upon receipt of the granules needed for the latest tasking, CrIS SDR checks the version ID associated with the scans already in the sliding window from the last tasking. If one of those version ids has changed, then the entire sliding window is rebuilt from scratch.

3.1.2.16 Sliding Window Optimization

This implementation has ameliorated the latency introduced when switching from a stream-based input to a granule-based input paradigm by preserving as much of the sliding window as possible between taskings.

This is done by using granule IDs to determine how many granules have passed since the last tasking. CrIS SDR uses this value to determine how many future scans are added to the sliding window. If this difference implies that the number of passed scans is greater than the number of future scans already stored in the sliding window, the sliding window is cleared of all scans and refilled.

Figure 3.1.2.16-1 illustrates this concept in its idealized form. Four CrIS SDR processes are run concurrently, each of which processes a 4-scan granule. State A represents the current 4-process tasking, while State B represents the next 4-process tasking. There are 5 sequential tasks across the 2 different states, so 4 are processed in State A, while the last is processed in State B. Assume that Process 1 in State A is the first granule tasked to the CrIS SDR algorithm. This process fills the sliding window with 30 scans worth of data (15 past cross-granules scans, 4 scans from the tasked granule, and 11 future cross-granule scans) before processing the 4 tasked scans in the middle of the window. Each time a scan is processed, one scan is purged from the beginning of the window and the next scan from the future cross-granules is added to the end of the window. Therefore, 16 scans are purged while 16 are added from State-to-State during ideal processing.

Note: There are no interactions among the Sliding Windows for the illustrated 4 concurrent SDR processes, i.e., each SDR Process Sliding Window is independent of the other concurrent SDR Processing.

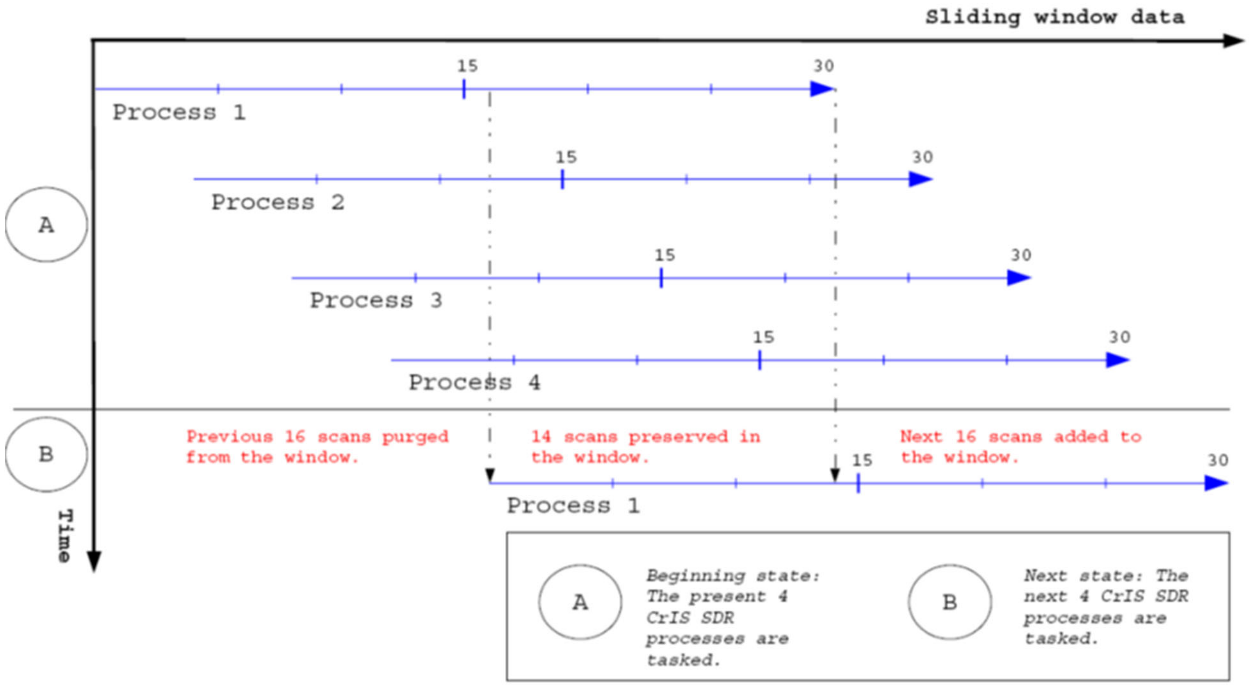

Figure 3.1.2.16-1. Sliding Window Optimization Concept

3.1.2.17 Creation and Storage of Correction Matrices

Parameters provided by the four-minute engineering packets can prompt CrIS SDR to rebuild its correction matrix. When a correction matrix rebuild is performed, both the new matrix and its corresponding four-minute packet are saved to DMS.

Adding only the scans needed to refill the sliding window limits the scans actually examined during processing. When tasked sequentially, or nearly sequentially, the four-minute engineering and eight-second packets enclosed within which the granule CrIS SDR is tasked to process, are not examined. This requires CrIS SDR to perform some correction matrix updates before they are to be used for processing.

Figure 3.1.2.17-1 shows an example of how a correction matrix build is handled in the case of sequential tasking. This figure shows us that the four-minute packet is added to the *end* of the sliding window *only once*. Processing any other granule results in that scan being placed at the +14th position. Thus CrIS SDR, when tasked sequentially, builds a correction matrix when a rebuild is detected for the +14th scan.

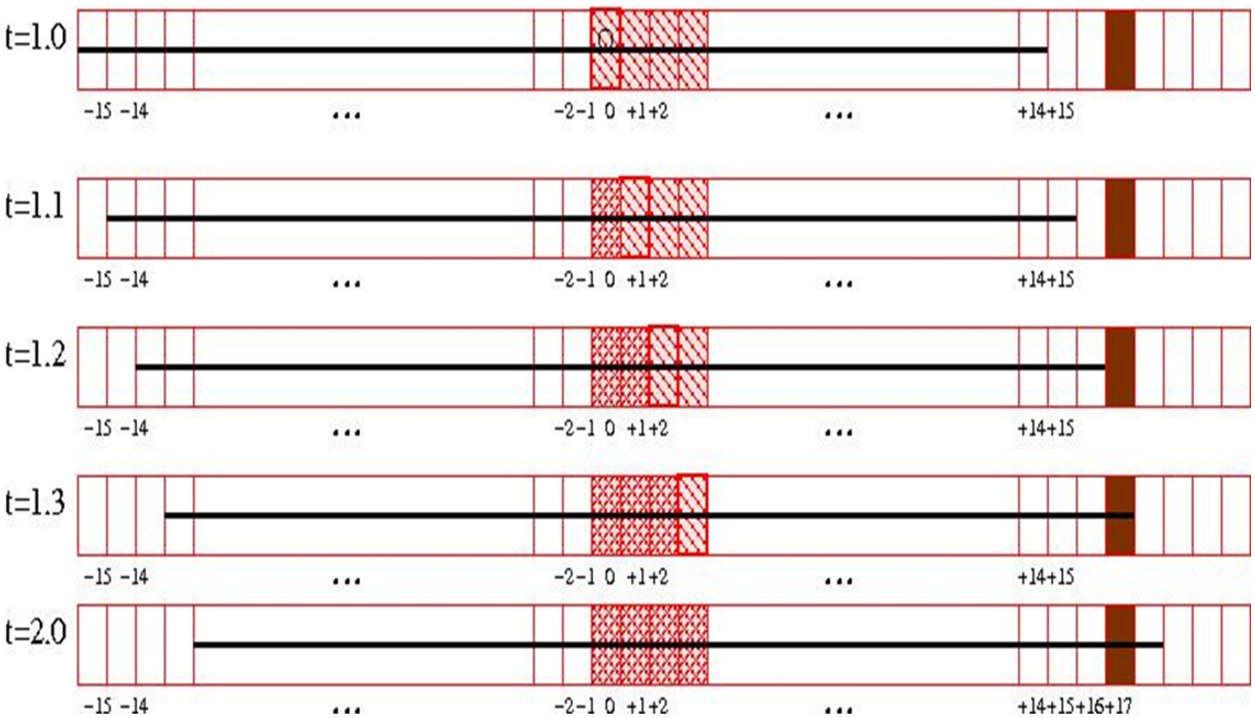

Figure 3.1.2.17-1. Sequential Tasking of a Correction Matrix Build

The boxes with single hashes all belong to the same, four-scan granule. The boxes with cross hashes are those scans after being processed. The scan currently being processed is surrounded by a bold border. The shaded box represents a scan which causes the correction matrix to be re-built. In this scenario, the correction matrix would be rebuilt at time 1.3. This scan is never again at the +14 location unless this granule is processed again.

The only other scenario where a correction matrix is rebuilt is when a rebuild is detected for the 0th (currently being processed) scan. This can only occur when the sliding window is being completely rebuilt. There is no concern about inadvertently building an already built matrix because the start time of the granule that caused the matrix rebuild is stored with the matrix as metadata. When tasked, this metadata is used to determine which correction matrix to load from DMS. When a scan that caused a rebuild during an earlier tasking (or even by another CrIS SDR process) is later than a part of the granule CrIS SDR is tasked to process, the previously built correction matrix and its accompanying four-minute engineering packet associated with this scan is the one that's returned by DMS. Therefore, parsing the four-minute engineering packet shows no change when compared to the engineering packet retrieved from DMS.

3.1.2.18 Performance

The processing of data is organized around an eight-second scan period, where the hot reference scene immediately precedes the eight-second science telemetry packet. The values derived from these packets remain constant for all measured earth scenes for the next eight-second period. The following items have been calculated at the beginning of this period.

- Hot and Cold variances for the moving window averages
- NEdN slope and intercept built from variance (optionally)
- Radiance for the current laser frequency

The sliding window average is maintained in a doubly-linked list where a sum and sum-of-squares is adjusted for each valid spectra entering or exiting the calibration period.
In the event of earth scene fringe count detection, each of the windowed Cold reference spectra are phase adjusted and the state of the average restored.

The 128 point Legendre polynomials used in the calculation of the ILS Self Apodization Matrix have been “hard-coded”.
When appropriate, the Correction matrix is loaded from DMS to address expensive recalculations that would occur at the receipt of the first four minute engineering packet.

3.1.2.19 Operational Adaptation, Deviation or Limitations

The following deviations or adaptations have been made in the C++ prototype code and are discussed in detail in this section.

1. Lunar Intrusion

Lunar Intrusion

Operationally the CrIS SDR Algorithm is required to support the detection and invalidation of space reference spectra that have experienced a lunar intrusion. The final iteration of the algorithm’s prototype code described in later sections of this document does include functionality to detect and handle lunar intrusions.

The following steps are taken with a copy of each new Deep Space spectrum to detect a lunar intrusion:

1. Perform FCE handling on the new uncalibrated DS spectrum in each band, FOV and sweep direction, $\widetilde{S}^{DS}[n]$
2. Subtract the averaged uncalibrated DS spectrum from the newest DS spectrum that was not included in the average,
   $\widetilde{R}^{DS}[n] = \widetilde{S}^{DS}[n] - \left\langle \widetilde{S}^{DS}[n] \right\rangle$
3. Subtract the averaged uncalibrated DS spectrum from the averaged uncalibrated ICT spectrum,
   $\widetilde{R}^{ICT}[n] = \left\langle \widetilde{S}^{ICT}[n] \right\rangle - \left\langle \widetilde{S}^{DS}[n] \right\rangle$
4. Compute real part of calibrated lunar spectrum in digital units averaged over the IR bands and then compare with a configurable threshold,
   $$\frac{\sum_{n=n_{\min}}^{n_{\max}} \mathrm{Re}\left\{ \frac{\widetilde{R}^{DS}[n]}{\widetilde{R}^{ICT}[n]} \right\}}{n_{\max} - n_{\min}} > \frac{LI_{\lim}}{100}$$

Where $n_{\min}$ and $n_{\max}$ define wavenumber bins corresponding to lower and upper band edge, respectively. Lunar Intrusion is assumed when that that ratio is greater than the threshold.
The configurable parameter maxLunarRadiance in the CrIS SDR PCT is a 1D array that holds 3 “threshold” values for each band (LW, MW and SW).

If Lunar Intrusion is detected, the new DS spectrum is marked as invalid and excluded from the Moving Window average. This action results in all Earth Scene spectra calibrated, while the invalid DS spectrum is in the Moving Window, to be marked as invalid with a flag identifying Lunar Intrusion. After the entire window is refreshed, i.e., the intruded spectrum falls out of the window after four minutes, Earth Scene spectra is considered valid.

3.1.2.20 Telemetry Data XML Files

The telemetry data is parsed at compile time from telemetry XML files. A source file is auto-generated (AutoGeneratedTelemetryData.cpp) and compiled into the library (libProSdrCris). Therefore, the XML are not necessary for successful run-time operation. This makes the packet parsing definitions flexible enough to be modified during development while still protecting copyright restrictions. These XML files are located in ${PRO_HOME}/cfg/CRIS.

3.1.3 Graceful Degradation

All CrIS SDR inputs are required. Therefore, there is no graceful degradation.

3.1.4 Exception Handling

Generally, exceptions are introduced by incompatible configuration setup for the data being processed. Handling of such events usually results in the offending spectra to be invalidated and processing continued according to the rules defined for missing data. The resultant log file contains details describing individual exceptions.

3.1.5 Data Quality Monitoring

Data quality tests are performed on the CrIS SDR and each test can produce a Data Quality Notification (DQN). If the thresholds are met, the algorithm stores a DQN to DMS indicating the tests that failed and the number of failures. The DQN criteria is adjustable and contained in a data quality threshold table (DQTT). If the CrIS SDR algorithm cannot obtain the DQTT, the algorithm still executes but no DQN tests are run. Appendix C in 474-00448-02-03_JPSS-DD-Vol-II-Part-3 lists the DQTT QF Mapping. DQN-able QF logic is outlined in 474-00448-04-03_JPSS-SRSPF-Vol-IV-Part-3.

3.1.5.1 Quality Flags

Both 474-00448-02-03_JPSS-DD-Vol-II-Part-3 and 474-00448-04-03_JPSS-SRSPF-Vol-IV-Part-3 provide details of the CrIS SDR quality flag descriptions and triggering logic.

3.1.6 Computational Precision Requirements

In general, all computations should be performed using double precision floating point arithmetic to maintain less than the required 1 ppm error contribution due to computation precision. It should be noted that the final output spectrum is represented with single precision floating point values.

3.1.7 Algorithm Support Considerations

Reference the list of libraries under Section 3.1.2.14.

3.1.8 Assumptions and Limitations

3.1.8.1 Assumptions

Currently this algorithm has only been tested with one-scan and four-scan granules.

3.1.8.2 Limitations

When the option to reassign the Field of Regard for the DS and/or ICT scenes to obtain calibrated output at one of these scenes, only one sweep direction of the two samples is calibrated based on which Earth Scene is selected as the hot reference.

4 GLOSSARY/ACRONYM LIST

4.1 Glossary

Below is a glossary of terms most applicable for this OAD.

Term	Description
Algorithm	A formula or set of steps for solving a particular problem. Algorithms can be expressed in any language, from natural languages like English to mathematical expressions to programming languages like FORTRAN. On JPSS, an algorithm consists of: 1. A theoretical description (i.e., science/mathematical basis) 2. A computer implementation description (i.e., method of solution) 3. A computer implementation (i.e., code)
Engineering Review Board (AERB)	Interdisciplinary board of scientific and engineering personnel responsible for the approval and disposition of algorithm acceptance, verification, development and testing transitions. Chaired by the Data Process Algorithm Lead, members include representatives from STAR, DPMS, IDPS, and Raytheon.
Algorithm Verification	Science-grade software delivered by an algorithm provider is verified for compliance with data quality and timeliness requirements by Algorithm Team science personnel. This activity is nominally performed at the IWPTB facility. Delivered code is executed on compatible IWPTB computing platforms. Minor hosting modifications may be made to allow code execution. Optionally, verification may be performed at the Algorithm Provider’s facility if warranted due to technical, schedule or cost considerations.
Ancillary Data	Any data which is not produced by the JPSS System, but which is acquired from external providers and used by the JPSS system in the production of JPSS data products.
Auxiliary Data	Auxiliary Data is defined as data, other than data included in the sensor application packets, which is produced internally by the JPSS system, and used to produce the JPSS deliverable data products.
EDR Algorithm	Scientific description and corresponding software and test data necessary to produce one or more environmental data records. The scientific computational basis for the production of each data record is described in an OAD. At a minimum, implemented software is science-grade and includes test data demonstrating data quality compliance.
Environmental Data Record (EDR)	*[IORD Definition]* Data record produced when an algorithm is used to convert Raw Data Records (RDRs) to geophysical parameters (including ancillary parameters, e.g., cloud clear radiation, etc.). *[Supplementary Definition]* An Environmental Data Record (EDR) represents the state of the environment, and the related information needed to access and understand the record. Specifically, it is a set of related data items that describe one or more related estimated environmental parameters over a limited time-space range. The parameters are located by time and Earth coordinates. EDRs may have been resampled if they are created from multiple data sources with different sampling patterns. An EDR is created from one or more JPSS SDRs or EDRs, plus ancillary environmental data provided by others. EDR metadata contains references to its processing history, spatial and temporal coverage, and quality.
Operational Code	Verified science-grade software, delivered by an algorithm provider and verified by IWPTB, is developed into operational-grade code by the IDPS IPT.
Operational-Grade Software	Code that produces data records compliant with the System Specification requirements for data quality and IDPS timeliness and operational infrastructure. The software is modular relative to the IDPS infrastructure and compliant with IDPS application programming interfaces (APIs) as specified for TDR/SDR or EDR code.

Term	Description
Raw Data Record (RDR)	*[IORD Definition]* Full resolution digital sensor data, time referenced and earth located, with absolute radiometric and geometric calibration coefficients appended, but not applied, to the data. Aggregates (sums or weighted averages) of detector samples are considered to be full resolution data if the aggregation is normally performed to meet resolution and other requirements. Sensor data shall be unprocessed with the following exceptions: time delay and integration (TDI), detector array non-uniformity correction (i.e., offset and responsivity equalization), and data compression are allowed. Lossy data compression is allowed only if the total measurement error is dominated by error sources other than the data compression algorithm. All calibration data will be retained and communicated to the ground without lossy compression. *[Supplementary Definition]* A Raw Data Record (RDR) is a logical grouping of raw data output by a sensor, and related information needed to process the record into an SDR or TDR. Specifically, it is a set of unmodified raw data (mission and housekeeping) produced by a sensor suite, one sensor, or a reasonable subset of a sensor (e.g., channel or channel group), over a specified, limited time range. Along with the sensor data, the RDR includes auxiliary data from other portions of JPSS (space or ground) needed to recreate the sensor measurement, to correct the measurement for known distortions, and to locate the measurement in time and space, through subsequent processing. Metadata is associated with the sensor and auxiliary data to permit its effective use.
Retrieval Algorithm	A science-based algorithm used to ‘retrieve’ a set of environmental/geophysical parameters (EDR) from calibrated and geolocated sensor data (SDR). Synonym for EDR processing.
Science Algorithm	The theoretical description and a corresponding software implementation needed to produce an NPP/JPSS data product (TDR, SDR or EDR). The former is described in an OAD. The latter is typically developed for a research setting and characterized as “science-grade”.
Science Algorithm Provider	Organization responsible for development and/or delivery of TDR/SDR or EDR algorithms associated with a given sensor.
Science-Grade Software	Code that produces data records in accordance with the science algorithm data quality requirements. This code, typically, has no software requirements for implementation language, targeted operating system, modularity, input and output data format or any other design discipline or assumed infrastructure.
SDR/TDR Algorithm	Scientific description and corresponding software and test data necessary to produce a Temperature Data Record and/or Sensor Data Record given a sensor’s Raw Data Record. The scientific computational basis for the production of each data record is described in an Operational Algorithm Document (OAD). At a minimum, implemented software is science-grade and includes test data demonstrating data quality compliance.

Term	Description
Sensor Data Record (SDR)	*[IORD Definition]* Data record produced when an algorithm is used to convert Raw Data Records (RDRs) to calibrated brightness temperatures with associated ephemeris data. The existence of the SDRs provides reversible data tracking back from the EDRs to the Raw data. *[Supplementary Definition]* A Sensor Data Record (SDR) is the recreated input to a sensor, and the related information needed to access and understand the record. Specifically, it is a set of incident flux estimates made by a sensor, over a limited time interval, with annotations that permit its effective use. The environmental flux estimates at the sensor aperture are corrected for sensor effects. The estimates are reported in physically meaningful units, usually in terms of an angular or spatial and temporal distribution at the sensor location, as a function of spectrum, polarization, or delay, and always at full resolution. When meaningful, the flux is also associated with the point on the Earth geoid from which it apparently originated. Also, when meaningful, the sensor flux is converted to an equivalent top-of-atmosphere (TOA) brightness. The associated metadata includes a record of the processing and sources from which the SDR was created, and other information needed to understand the data.
Temperature Data Record (TDR)	*[IORD Definition]* Temperature Data Records (TDRs) are geolocated, antenna temperatures with all relevant calibration data counts and ephemeris data to revert from T-sub-a into counts. *[Supplementary Definition]* A Temperature Data Record (TDR) is the brightness temperature value measured by a microwave sensor, and the related information needed to access and understand the record. Specifically, it is a set of the corrected radiometric measurements made by an imaging microwave sensor, over a limited time range, with annotation that permits its effective use. A TDR is a partially-processed variant of an SDR. Instead of reporting the estimated microwave flux from a specified direction, it reports the observed antenna brightness temperature in that direction.
Model Validation	The process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model. [Ref.: DoDD 5000.59-DoD Modeling and Simulation Management]
Model Verification	The process of determining that a model implementation accurately represents the developer's conceptual description and specifications. [Ref.: DoDD 5000.59-DoD Modeling and Simulation Management]

4.2 Acronyms

Below is a list of acronyms most applicable for this OAD.

Term	Description
AM&S	Algorithms, Models & Simulations
API	Application Programming Interfaces
DMS	Data Management Subsystem
DQTT	Data Quality Test Table
E&A	Ephemeris and Attitude
FPA	Focal Plane Array
HAM	Half Angle Mirror
H-S	Harvey-Shack

Term	Description
IEO	Instrument Engineering Order
IET	IDPS Epoch Time
IMG	Imagery
INF	Infrastructure
ING	Ingest
IPO	Input Processing Output
LOS	Loss of Signal
LUT	Look-Up Table
MOD	Moderate
NCSA	National Center for Supercomputing Applications
PO	Product Order
QF	Quality Flag
RTA	Rotating Telescope Assembly
SDR	Sensor Data Record
SI	International System of Units