

GSFC JPSS CMO
April 13, 2017
Released

Joint Polar Satellite System (JPSS) Ground Project
Code 474
474-00071

Joint Polar Satellite System (JPSS)
Operational Algorithm Description
(OAD)
Document for Cross-track Infrared
Sounder (CrIS) Sensor Data Record
(SDR) Software

For Public Release

The information provided herein does not contain technical data as defined in the International Traffic in Arms Regulations (ITAR) 22 CFC 120.10. This document has been approved For Public Release to the NOAA Comprehensive Large Array-data Stewardship System (CLASS).



Goddard Space Flight Center
Greenbelt, Maryland

National Aeronautics and
Space Administration

**Joint Polar Satellite System (JPSS)
Operational Algorithm Description (OAD) Document for
Cross-track Infrared Sounder (CrIS) Sensor Data Record
(SDR) Software
JPSS Electronic Signature Page**

Prepared By:

Bruce Guenther
JPSS Data Products and Algorithms SDR Lead
(Electronic Approvals available online at (https://jpssmis.gsfc.nasa.gov/mainmenu_dsp.cfm))

Approved By:

Gilberto Vicente
JPSS Ground Project Algorithm Integration Team (AIT) Manager
(Electronic Approvals available online at (https://jpssmis.gsfc.nasa.gov/mainmenu_dsp.cfm))

**Goddard Space Flight Center
Greenbelt, Maryland**

Preface

This document is under JPSS Ground Algorithm ERB 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 Approve Date)
Original	06/03/2011	This version incorporates 474-CCR-11-0087 : (This version baselines D39132, Rev B dated 07/07/2010. This is the version that was approved for NPP launch. Per NPOESS CDFCB - External, Volume V – Metadata, doc number D34862-05, this has been approved for Public Release into CLASS.) to create JPSS version, Rev-. This was approved by the JPSS Ground Algorithm ERB on June 3, 2011.
Revision A	01/18/2012	474-CCR-11-0246 : This version baselines 474-00071, Joint Polar Satellite System (JPSS) Operational Algorithm Description (OAD) Document for Cross-track Infrared Sounder (CrIS) Sensor Data Record (SDR) Software, for the Mx 6 IDPS release. This CCR was approved by the JPSS Algorithm ERB on January 18, 2012.
Revision B	10/09/2012	474-CCR-12-0627 : This version authorizes 474-00071, Joint Polar Satellite System (JPSS) Operational Algorithm Description (OAD) Document for Cross-track Infrared Sounder (CrIS) Sensor Data Record (SDR) Software, for the Mx 6.1 - 6.3 IDPS releases. Includes ECR-ALG-0035 which contains Raytheon PCR029809; ADR 4389: PRO: OAD: Update the CrIS SDR OAD - for the ADR 4389: CrIS SDR: ESZPDMagnitude data type field inconsistencies, updated Table 9. Raytheon PCR031565, OAD: Implement 474-CCR-12-0483 CrIS DR 4478, 4680, 4808 Code Change updated Table 38.
Revision C	05/14/2013	474-CCR-13-0948 : This version authorizes 474-00071, JPSS OAD Document for CrIS SDR Software, for the Mx 7.0 IDPS release. Includes ECR-ALG-0037 which contains Raytheon PCR033136; OAD updates associated with PCR032693 (CrIS SDR OAD), in section 2.1.2 and Tables 19, 33 & 34. Also includes PCR033560; DR7007 - PRO - OAD - CrIS SDR - Update description of Invalid RDR Data QF (QF4 bit 2), in Table 38. Includes Raytheon PCR032720; 474-CCR-13-0916/ECR-ALG-0037: Update applicable OAD filenames/template/Rev/etc. for Mx7 Release.
Revision D	01/29/2014	474-CCR-14-1517 : This version authorizes 474-00071, JPSS OAD Document for CrIS SDR Software, for the Mx 8.1 IDPS release. Includes Raytheon PCR035209; DR7201 - PRO OAD - CrIS SDR: Update SDR Quality QF (QF3 bits 1-2) definition to account for Short ganules (474-CCR-13-1128), in Table 37. Also includes PCR035482; PRO:

		OAD: 474-CCR-13-1127: DRs: 7204, 7239, 7247: Update to CrIS SDR ATBD DRs 7204, 7239, 7247, in Figure 3.
Revision E	01/07/2015	474-CCR-14-2212: This version authorizes 474-00071, JPSS OAD Document for CrIS SDR Software, for the Mx 8.8 IDPS release. Includes Raytheon PCR041470; PRO: OAD: 474-CCR-14-1941: CrIS Instrument Configuration Files for Full Spectrum RDR (DR 7738), in Figure 1.
Revision F	03/13/2017	474-CCR-17-3243 (ECR-CGS-0734): This version authorizes 474-00071, JPSS OAD Document for CrIS SDR Software, for the Block 2.0 IDPS release. Includes Raytheon PCRs: a). PCR045351: CO-10: CHILD: CCR: 474-CCR-15-2482: Update 474-00071 (CrIMSS EDR Removal)_OAD-CrIS-SDR NLT TTO of Block 2.0; in Sections 2.0 & 2.1.1.2. b). PCR048656-CCR-2278-CrIS SDR UPPER, OAD: CCR-15-2278 CrIS SDR Updates for Block 2 - Truncated SDR updates only, in Sections 2.1.1.2 & 2.1.2.1.4.2 and Tables 14, 18, 20 and 26. c). PCR051646, PRO OAD: 474-CCR-15-2536: Update CrIS SDR OAD for Full Spectral Resolution Values/One, in Section 2.1.1.2. d). PCR045678-CCR-2444-All_OAD_Cleanup



NATIONAL POLAR-ORBITING OPERATIONAL ENVIRONMENTAL SATELLITE SYSTEM (NPOESS)

OPERATIONAL ALGORITHM DESCRIPTION DOCUMENT FOR CROSS-TRACK INFRARED SOUNDER (CrIS) SENSOR DATA RECORD (SDR)

**SDRL No. S141
SYSTEM SPECIFICATION SS22-0096**

**RAYTHEON COMPANY
INTELLIGENCE AND INFORMATION SYSTEMS (IIS)
NPOESS PROGRAM
OMAHA, NEBRASKA**

**Copyright © 2004–2011
Raytheon Company
Unpublished Work
ALL RIGHTS RESERVED**

Portions of this work are the copyrighted work of Raytheon. However, other entities may own copyrights in this work. Therefore, the recipient should not imply that Raytheon is the only copyright owner in this work.

This data was developed pursuant to Contract Number F04701-02-C-0502 with the US Government under subcontract number 7600002744. The US Government's right in and to this copyrighted data are as specified in DFAR 252.227-7013, which was made part of the above contract.

Northrop Grumman Space & Mission Systems Corp.
Space Technology
One Space Park
Redondo Beach, CA 90278



**Engineering & Manufacturing Development (EMD) Phase
Acquisitions & Operations Contract**

CAGE NO. 11982
Operational Algorithm Description
CrIS SDR
Document Number: D39132
Revision: C5
Document Date: Nov 05, 2011

PREPARED BY:

Degui Gu *Date*
Models & Simulations Lead

Paul D. Siebels *Date*
IDPS Processing SI Software Manager

ELECTRONIC APPROVAL SIGNATURES:

Roy Tsugawa *Date*
SEIT Lead & ACCB Chair

Bob Hughes *Date*
Algorithm Implementation Thread Lead

Bob Hughes *Date*
Data Product System Engineering Lead

Stephen E. Ellefson *Date*
IDPS Processing SI Lead

Northrop Grumman Space & Mission Systems Corp.
Space Technology
One Space Park
Redondo Beach, CA 90278

Prepared for
Department of the Air Force
NPOESS Integrated Program Office
C/O SMC/CIK
2420 Vela Way, Suite 1467-A8
Los Angeles AFB, CA 90245-4659

Under
Contract No. F04701-02-C-0502

This document has been identified per the NPOESS Common Data Format Control Book – External Volume 5 Metadata, D34862-05, Appendix B as a document to be provided to the NOAA Comprehensive Large Array-data Stewardship System (CLASS) via the delivery of NPOESS Document Release Packages to CLASS.

Northrop Grumman Space & Mission Systems Corp. Space Technology One Space Park Redondo Beach, CA 90278		 	
Revision/Change Record		Document No. D39132	
Revision	Document Date	Revision/Change Description	Pages Affected
---	6-17-04	Initial version.	N/A
A1	6-30-04	Modifications to include references to Bomem documentation monitored and tunable parameters, additional detail of source code classes, specific dependence on commercial libraries.	2, 7-10, 12-14, 18-23
A2	6-30-04	Sec. 2.1.1.1 updated to reflect ATBD --- included: <ol style="list-style-type: none"> 1. CrIS Specific Calibration Equation 2. Geolocation 3. FFT Sec. 2.2.1 Table 4 updated to reflect configuration file. Sec. 2.2.2 Outputs discussing apodization and EDR inputs. Sec. 2.2.2 Table 8 updated with units. Sec. 5.1 Tables 12 and 13 updated to reflect paths in VTP.	4-6 9-11 11-12 14-17 33-37
A3	6-30-04	Invalid Data SDR QF is set if any of the other error flags (except Excess Thermal Drift SDR Quality Flag) are set.	29
A4	6-30-04	Table 13 updated to reflect list of delivered data.	36-37
A	10-26-04	Sec 4.4 Added two new SDR quality flags: SAA and Lunar Inception. Sec 2.2.2 Added guideline for converting SDR outputs into EDR inputs. Sec 2.1.1.1 Corrected an error in the implemented calibration equation: $F_{int} = 1$, was $F_{int} = F_{int} - 1 = 1$.	30-31 12-13 6

Northrop Grumman Space & Mission Systems Corp. Space Technology One Space Park Redondo Beach, CA 90278		 	
Revision/Change Record		Document No. D39132	
Revision	Document Date	Revision/Change Description	Pages Affected
B1	6-18-06	<p>21Nov05 – Reflects Raytheon-Omaha’s Science To Operational Code Conversion edits, but does not include making 19Oct05 CrIS SDR Tech Memo changes about Fringe Count Error (FCE) Handling or any Quality Flag follow-on updates, although Omaha added a “Value” column to Table 4.4-2, SDR Quality Flags.</p> <p>01Mar06 – Removed NPOESS-Omaha’s electronic signatures from title/signature page, changed copyright to 2006 on coversheet, put current dates on upper right header. Title/Signature page, and Revision Change/History page. Replaced initial Sci2Ops Code Completion 120-page Unit Test dated 10/28/05 with the follow-on 10-page Unit Test dated 02/28/06 to address Fringe Count Errors, then updated Table of Content/List of Figures/List of Tables/TBD page numbers.</p> <p>21Mar06 – Changed Section 2.4, Graceful Degradation to Section 2.7 and inserted Sections 2.4, 2.4.1, 2.4.2, 2.4.3, 2.5, and 2.6 discussing the “Sliding Window” concept.</p> <p>04Apr06 – Deleted a couple of “Algorithm Control Parameters” from Table 2.2-5. Deleted a couple of words from the table under Section 2.2.2, Outputs.</p> <p>12Apr06 – Worked off the 30Mar06 follow-on CCPR comments to include the following changes. Replaced Figure 2.2-1, IPO Model Interface to INF & DMS, with the latest version. Removed Section 2.1.2, Prototype Code. Removed page 6 paragraph discussing Fringe Count Errors. Deleted Section 2.1.1.1 words discussing “Sensor 8 Second Scan Epoch Timing” to include Figure 2.1-1 about this subject. Replaced Tables 2.2-2, 2.2-3, and 2.2-4.</p> <p>17Apr06 – Inserted current Unit Test dated 13Apr06 and removed obsolete one. Added 21Mar06, 04Apr06, and 12Apr06 comments to the Revision/change Record page to log document edits. Made spacing/format edits. Updated TBD/TBR page. Updated List of Figures to include Figures 2.4-1 and 2.6-1. Updated List of Tables’ page numbers. Added Raytheon-Omaha electronic signatures in preparation for official delivery of document to prime.</p> <p>18Apr06 – Did minor spacing/format edits. Put two character spaces between sentences. Changed a few text paragraphs from 12 pt or 10 pt to be consistent so they were all Arial 11 pt throughout the document.</p>	All
B2	6-27-07	Updated quality flags per Tech Memo NP-EMD-2007.510.0027.Rev.A . Updated logo, header, dates, etc.	All
B3	7-10-07	Delivered to NGST.	All
B4	12-17-07	ECR A-103, EDR-PR 1.8 CP 3, and CDFCB-X compliance updates – Added new output structure definitions for CrIS-SDR and CrIS-SDR-GEO.	All

Northrop Grumman Space & Mission Systems Corp. Space Technology One Space Park Redondo Beach, CA 90278		 	
Revision/Change Record		Document No. D39132	
Revision	Document Date	Revision/Change Description	Pages Affected
B5	2-28-08	ISTN_CRIS_SDR_UNIX_NGST_1.1 science drop and TM MS_Tech_Memo_CrIS_SDR_QF_Updates_RevB , NP-EMD-2007.510.0027.Rev.B, implementation. Format updates for CDFCB-X compliance. <ol style="list-style-type: none"> 1. Table 7 – added 6 new configurable parameters 2. Table 9 – format updates 3. Tables 25 through 29 – quality flag updates 4. Paragraph 1.3.2 – added the above-mentioned TM to the NPOESS Reference Documents table; changed reference to CDFCB Vol I to ref. Vol III, instead Responses to comments provided in 070710 SDRL S141 D39132_B3_CrIS_SDR_OAD_review <ol style="list-style-type: none"> 1. Figure 4 – updated Figure with explanations 2. Paragraph 2.1.2.9 – provided clarification on the meaning of “hard-coded the sliding window size” 3. Paragraph 2.1.2.9.2 – provided explanations for the variables W and S in the two equations 4. Paragraph 2.1.2.9.1 - Updated the paragraph to use the phrase “degraded quality” instead of “invalid”. 	2, 11, 13, 15, 24-25, 31-33, 36
B6	4-11-08	Added Cris SDR Exit Angles IP to Table 8 per ECR R871. Implemented Tech Memos NP-EMD.2005.510.0118 and NP-EMD.2005.510.0130. Reformatted in accordance to new template. Addressed Peer Review comments. Delivered to NGST.	All
B7	4-17-08	Updated description of the CrIS SDR Exit Angles IP in Table 8. Updated paragraph 2.1.2.2.1 (ScienceDataProcessor Attributes) with an explanation between the ATBD and operational software’s Planck Radiation Constants.	13, 19
B8	3-03-09	NP-EMD.2008.510.0055 Rev A, PCR019301: Updates after Sensor Characterization Phase I: <ol style="list-style-type: none"> 1. Add CrIS SDR ATBD (D43773) Rev A in Table 1 Document References. 2. Add CrIS SDR operational software in Table 2 Source Code References. 3. Change ICT Emissivity Constant to ICT Emissivity in Table 6 Tunable Parameters Provided via the Four Minute Engineering Telemetry RDR. 4. Add a function generateUSN in Table 18 Spectra Operations. 	2,3,8,22
B9	3-06-09	Prepared for SDRL Delivery (for Sensor Characterization)	All
B10	3-10-09	Redelivered to NGAS with Template corrections	1-37
B11	12-01-09	Updated for RFA No.551 and updated subcontract number	Title pg & Table 9

Northrop Grumman Space & Mission Systems Corp. Space Technology One Space Park Redondo Beach, CA 90278		 	
Revision/Change Record		Document No. D39132	
Revision	Document Date	Revision/Change Description	Pages Affected
		lwFirFilterScale mwFirFilterScale swFirFilterScale lwPassBandStar mwPassBandStart swPassBandStart lwPassBandStop mwPassBandStop swPassBandStop firFilterResponse Longwave Real Imag Midwave Real Imag Shortwave Real Imag GainMappingLW GainMappingMW GainMappingSW Add the following tunable parameters in Table 7: ShiftFactorArray AppShiftFactorFlag performLWLinearityCorrectionControl performMMLinearityCorrectionControl performSWLinearityCorrectionControl lwLinearityCorrectionControlParam lwLinearityCorrectionControlParam lwLinearityCorrectionControlParam	
B13	2-23-10	Incorporated CCPR comments and prepared for SDRL delivery	All
B14	3-24-10	Implemented TM 2009.510.0046	All
B15	5-20-2010	Updated Table 3 with Two Line Element (TLE) listed as a required algorithm input and Table 8 with CrIS Correction Matrix internal auxiliary item listed as an optional output of the algorithm.	7,18
B	7-07-10	Prepared for TIM/ARB/ACCB	All
C1	9-14-10	Updated for PCR018349	Table 3. pg 13/14
C2	10-13-10	Updated due to document convergence	All
C3	29-06-11	Updated due to Drop ISTN_CRIS_SDR_NGST_SC7.1_NG1 (ECR A-0017) and TM 2011.510.0006	All
C4	8-10-11	Updated due to PCR027378; Min/Max Radiance Range update (Table 9)	Page 16

Northrop Grumman Space & Mission Systems Corp. Space Technology One Space Park Redondo Beach, CA 90278		 	
Revision/Change Record		Document No. D39132	
Revision	Document Date	Revision/Change Description	Pages Affected
C5	11-05-211	Updated due to PCR028276 (2.1.2.20-new section), PCR027467 (ADL 3.0), PCR028340 (DR4373 – Laser monitoring issue), PCR028501 (474-CCR-11-0204 Mounting coefficients update), and PCR026561 (BOOST throwing exception)	All

Table of Contents

1.0 INTRODUCTION..... 1

 1.1 Objective..... 1

 1.2 Scope 1

 1.3 References 1

 1.3.1 Document References 1

 1.3.2 Source Code References 2

2.0 ALGORITHM OVERVIEW 4

 2.1 CrIS SDR Description..... 5

 2.1.1 Interfaces 5

 2.1.2 Algorithm Processing 16

 2.1.3 Graceful Degradation..... 45

 2.1.4 Exception Handling 45

 2.1.5 Data Quality Monitoring 45

 2.1.6 Computational Precision Requirements 46

 2.1.7 Algorithm Support Considerations 46

 2.1.8 Assumptions and Limitations 46

3.0 GLOSSARY/ACRONYM LIST 47

 3.1 Glossary 47

 3.2 Acronyms..... 49

4.0 OPEN ISSUES..... 50

List of Figures

Figure 1. Processing Chain..... 5
 Figure 2. IPO Model Interface to INF and DMS 6
 Figure 3. CrIS Measurement Sequence 12
 Figure 4. Sliding Window Concept..... 40
 Figure 5. Sliding Window Optimization Concept..... 42
 Figure 6. Sequential Tasking of a Correction Matrix Build..... 43

List of Tables

Table 1. Document References 2
 Table 2. Source Code References..... 2
 Table 3. CrIS SDR Truncated Spectral Resolution Inputs 7
 Table 4. CrIS SDR Full Spectral Resolution Inputs 8
 Table 5. Parameters Monitored via the Eight Second Science/Calibration Telemetry Packet..... 9
 Table 6. Parameters Monitored via the Four Minute Engineering Telemetry Packet 10
 Table 7. Tunable Parameters Provided via the Four Minute Engineering Telemetry RDR..... 10
 Table 8. CrIS Truncated Spectrum SDR Outputs 13
 Table 9. CrIS Full Spectrum SDR Outputs 14
 Table 10. Some Important Classes Used During Processing..... 16
 Table 11. CalibratedSpectra Attributes 17
 Table 12. CalibratedSpectra Operations..... 17
 Table 13. CCSDSFormat Operations 18
 Table 14. CCSDSFormat Operations 18
 Table 15. CorrectionMatrix Attributes 18
 Table 16. CorrectionMatrix Operations 19
 Table 17. EngineeringCalibrationRecord Attributes..... 20
 Table 18. EngineeringCalibrationRecord Operations 20
 Table 19. Interferogram Attributes 21
 Table 20. Interferogram Operations..... 22
 Table 21. TemperatureHistory Attributes 22
 Table 22. TemperatureHistory Operations..... 23
 Table 23. ScienceDataProcessor Attributes 24
 Table 24. ScienceDataProcessor Operations..... 25
 Table 25. SpectraManager Attributes 28
 Table 26. SpectraManager Operations..... 29
 Table 27. Spectra Attributes 31
 Table 28. Spectra Operations 31
 Table 29. TelemetryProcessor Attributes..... 32
 Table 30. TelemetryProcessor Operations 33
 Table 31. VideoData Attributes 34
 Table 32. VideoData Operations..... 34
 Table 33. ScienceCalibrationRecord Attributes 36
 Table 34. ScienceCalibration Operations 38
 Table 35. Glossary 47
 Table 35. Acronyms 49
 Table 37. TBXs 50

1.0 INTRODUCTION

1.1 Objective

The purpose of the 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 are also described in the OAD.

The science basis of an algorithm is described in a corresponding Algorithm Theoretical Basis Document (ATBD). The OAD provides a software description of that science as implemented in the operational ground system.

The purpose of an OAD is two-fold:

1. Provide initial implementation design guidance to the operational software developer.
2. 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. There is a general, but not strict, one-to-one correspondence between OAD and ATBD documents. 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 required to create the CrIS SDR. The theoretical basis for this algorithm is described in the Cross Track Infrared Sounder (CrIS) Sensor Data Records (SDR) Algorithm Theoretical Basis Document ATBD, D0001-M01-S01-002.

1.3 References

1.3.1 Document References

The science and system engineering documents relevant to the algorithms described in this OAD are listed in Table 1.

Table 1. Document References

Document Title	Document Number/Revision	Revision Date
CrIS Sensor Calibration and Validation Master Plan (ITT Document)	8180003 Rev. D	23 Feb 2005
SDR Generator Validation Test Procedure	VTP8189758 Rev. D	12 Sep 2005
JPSS-1 CrIS Mission Data Packet Structures	472-00333	6/17/2015
SDR Generator Software User Manual	UM8189758	Latest Revision Date
SDR Generator Rose Model	RM8189758	Latest Revision Date
CrIS_Science_Data_Packet_Format_Oct2003_v3.xls	Rev. V3	23 Oct 2003
CrIS Command and Data Packet Dictionary	8196185 Rev. H	14 Nov 2008
Joint Polar Satellite System (JPSS) Program Lexicon	470-00041	Latest
MS_Tech_Memo_CrIS_SDR_QF_Updates_RevB	NP-EMD-2007.510.0027 Rev. B	10 Oct 2007
MS Tech Memo CrIS SDR QF Updates_0027_Rev_A	NP-EMD-2007.510.0027 Rev. A	07 Oct 2007
Memo_CrIS_SDR_OAD_Update-v2	NP-EMD.2005.510.0118	09 Sep 2005
Memo_CrIS_SDR_OAD_Revision	NP-EMD.2005.510.0130	19 Oct 2005
Operational Algorithm Description Document for Common Geolocation	474-00091	Latest
Cross Track Infrared Sounder (CrIS) Sensor Data Records (SDR) Algorithm Theoretical Basis Document ATBD	D0001-M01-S01-002	Latest
A&DP Tech memo CrIS_SDR_OAD_Update_for_CrIS_SDR_UNIX_1.1	NP-EMD.2007.510.0061	10 Oct 2007
A&DP Tech memo CrIS_SDR_OAD_Updates_for_CrIS_SDR_UNIX_2.1	NP-EMD.2008.510.0055_RevA	20 Jan 2009
A&DP Tech memo CrIS_SDR_OAD_Updates_for_CrIS-RevA_SDR_UNIX_2.2.doc	NP-EMD.2009.510.0045.Rev.A	22 Sep 2009
A&DP Tech memo CrIS_Scan Baffle Temperature Correction	NP-EMD.2009.510.0046	16 Sep 2009
A&DP Tech memo CrIS_SDR_Code_Updates	NP-EMD.2010.510.0006.RevA	26 Apr 2010
A&DP Tech memo CrIS_inst2sc_techMemo	NP-EMD.2011.510.0006	21 Feb 2011

1.3.2 Source Code References

The science and operational code and associated documentation relevant to the algorithms described in this OAD are listed in Table 2.

Table 2. Source Code References

Reference Title	Reference Tag/Revision	Reference Date
SASS_CRIS_SDR_v 3.0.0 QL		
ISTN_CRIS_SDR_2.1	(OAD Rev ---)	16 Aug 2004
CrIS SDR operational software	B1.3	28 Apr 2004
ISTN_CRIS_SDR_NGST_2.1.1	ECR-A40 (OAD Rev 6)	04 Nov 2004
ISTN_CRIS_SDR_NGST_2.1.2	ECR-A065 (OAD Rev A)	06 Oct 2005
Memo_CrIS_SDR_OAD_Update-v2	NP-EMD.2005.510.0118	09 Sep 2005
Memo_CrIS_SDR_OAD_Revision	NP-EMD.2005.510.0130	19 Oct 2005

Reference Title	Reference Tag/Revision	Reference Date
CrIS SDR science-grade software	ISTN_CRIS_SDR_NGST_UNIX_1.1 ECR-A135	18 Dec 2007
MS_Tech_Memo_CrIS_SDR_QF_Updates_RevB	NP-EMD-2007.510.0027 Rev. B	10 Oct 2007
OAD Update	NP-EMD-2007.510.0061	
CrIS SDR operational software	B1.5 (OAD Rev B4)	17 Dec 2007
CrIS SDR science-grade software	ISTN_CRIS_SDR_NGST_UNIX_2.1 ECR-A175A	17 Dec 2008
CrIS SDR operational software	B1.5.x.1 (OAD Rev B6)	01 Apl 2008
CrIS SDR operational software (PCR019301)	IDPS Post-X, Build A (OAD Rev B9)	19 Feb 2009
CrIS SDR science-grade software	ISTN_CRIS_SDR_NGST_UNIX_2.2 ECR-A255 (OAD Rev B11)	09 Nov 2009
Tech Memo: CrIS SDR OAD Updates for CrIS-RevA__SDR_UNIX_2.2	NP-EMD.2009.510.0045.Rev.A	22 Sep 2009
CrIS SDR operational software	IDPS Sensor Characterization Build SC-7 (OAD Rev B12)	15 Feb 2010
OAD SDRL (no code changes)	(OAD Rev B13)	23 Feb 2010
CrIS SDR Algorithms Unix 2.2.1 Delta Delivery	ISTN_CRIS_SDR_UNIX_NGST_2.2. 1 (ECR-A275)	22 Mar 2010
CrIS SDR operational software(PCR21464) NP-EMD.20909.510.0049	IDPS Sensor Characterization Build SC-9 (OAD Rev B14)	24 Mar 2010
A&DP Tech memo TM_CrIS_SDR_Code_Updates NP-EMD.2010.510.0006.RevA	ISTN_CRIS_SDR_UNIX_NGST_2.2. 2	29 Apr 2010
CrIS SDR operational software	IDPS Sensor Characterization Build SC-10 (OAD Rev B15)	25 May 2010
ACCB	OAD Rev B	07 Jul 2010
PCR018349	IDPS Sensor Characterization Build SC-14 (OAD Rev C1)	14 Sep 2010
Convergence Update (No code updates)	(OAD Rev C2)	13 Oct 2010
CrIS SDR science-grade software Includes TM 2011.510.0006 (PCRs 026134 & 026135)	ISTN_CRIS_SDR_NGST_SC7.1_NG 1 (ECR-A0017) Maintenance Build 1.5.5.00 (OAD Rev C3)	10 Mar 2011 & 29 Jun 2011 (OAD)
PCR027378	Build Maintenance 1.5.06_F (OAD Rev C4)	10 Aug 2011
PCR027467 (ADL)	Build Maintenance 1.5.05 E	05 Nov 2011
PCR026561	Build Maintenance 1.5.06 F	05 Nov 2011
PCR021927	Build Maintenance 1.5.06_J	05 Nov 2011
PCR028340	Build Maintenance 1.5.06_J	05 Nov 2011
PCR028500 (474-CCR-11-0204) (DQM)	Build Maintenance 1.5.06 J	05 Nov 2011
PCR028276 & PCR028578 (OAD update for PCR027467, PCR026561, PCR021927, PCR028340, PCR028500)	(OAD Rev C5)	05 Nov 2011
OAD transitioned to JPSS Program – this table is no longer updated.		

2.0 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 1 identifies the top level processing flow of the CrIS SDR Algorithm.

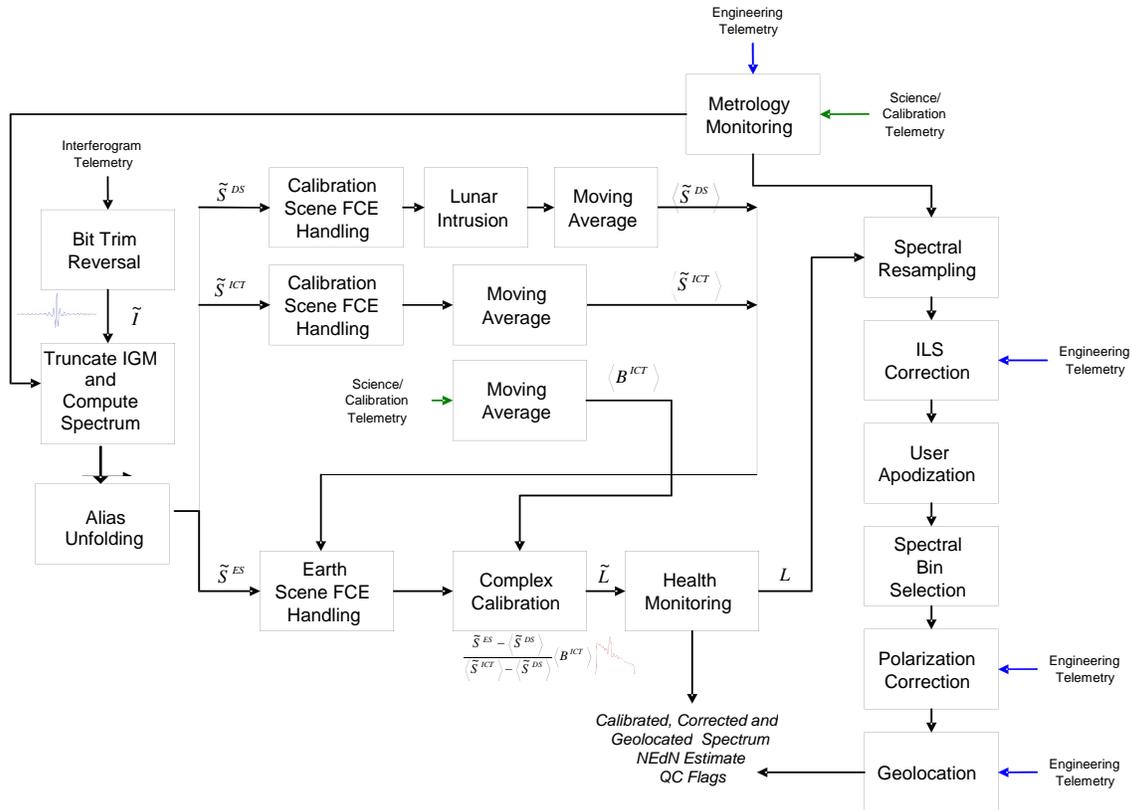


Figure 1. Processing Chain

2.1 CrIS SDR Description

2.1.1 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 2.

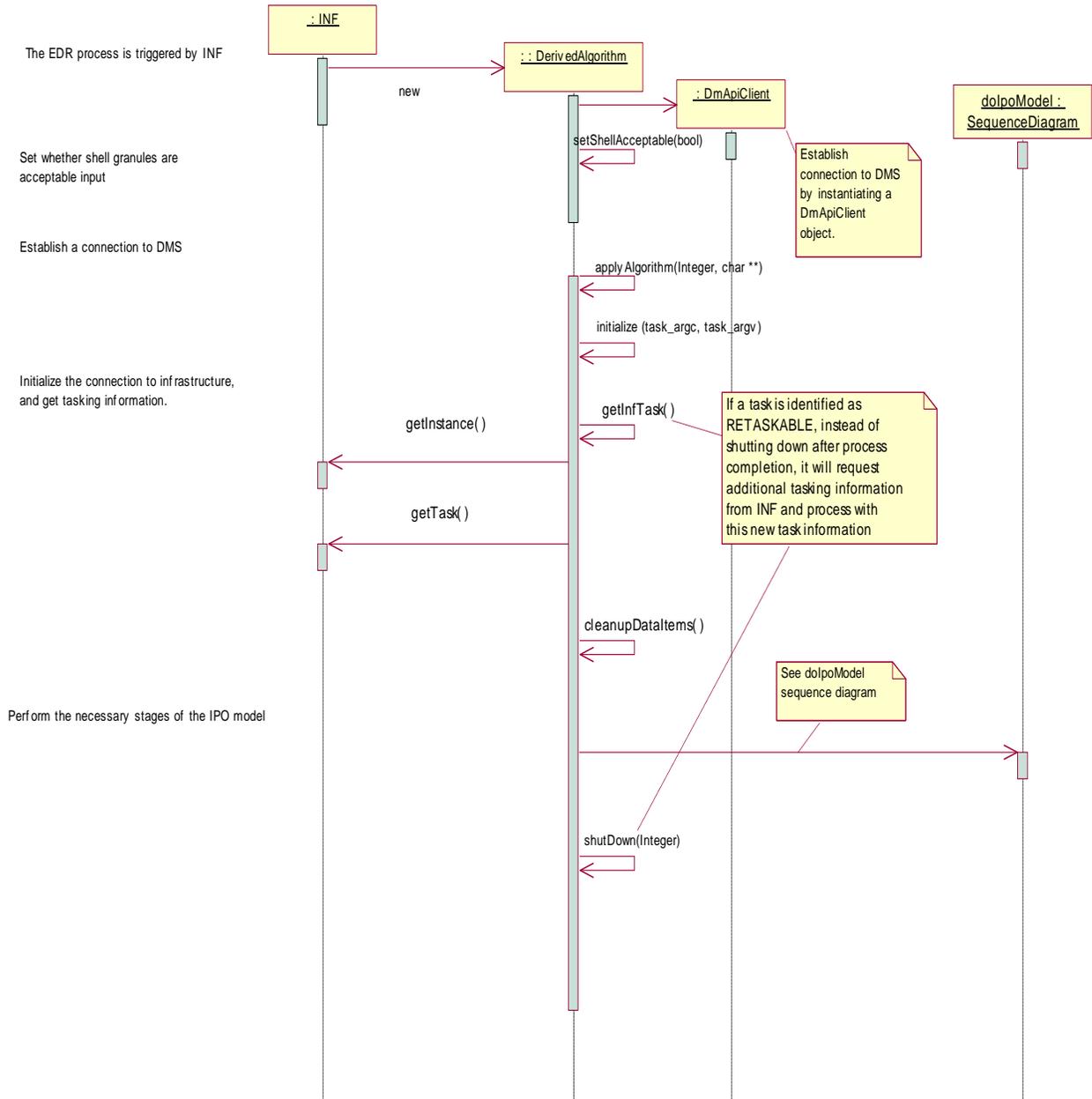


Figure 2. IPO Model Interface to INF and DMS

2.1.1.1 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 and 4.

Table 3. CrIS SDR Truncated Spectral Resolution Inputs

Input	Description	Reference Document
CrIS Science RDR	The CrIS sensor provides cross-track measurements of scene radiance to permit the calculation of the vertical distribution of temperature and moisture in the Earth's atmosphere. It also provides supporting measurements for a variety of other geophysical parameters.	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 2711 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 4. CrIS SDR Full Spectral Resolution Inputs

Input	Description	Reference Document
CrIS Science RDR	The CrIS sensor provides cross-track measurements of scene radiance to permit the calculation of the vertical distribution of temperature and moisture in the Earth's atmosphere. It also provides supporting measurements for a variety of other geophysical parameters.	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 2711 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 DQTT	Data Quality Test Table	474-00448-02-03_JPSS-DD-Vol-II-Part-3

Input	Description	Reference Document
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 5, 6, and 7 identify the parameters within the Eight Second Science/Calibration and Four Minute Engineering Telemetry Packets that are monitored throughout processing.

Table 5. 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	N/A
In Track Servo Error (sample 21, epoch 4..33)	30 Readings	N/A
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	N/A
OMA Structure Input Temperature #2 (epoch 39)	Single Reading	°C
SSM Scan Mirror Temperature (epoch 39)	Single Reading	N/A
SSM Scan Mirror Baffle Temperature (epoch 39)	Single Reading	°K
Stage 2 Cooler Temperature (epoch 39)	Single Reading	°K

Parameter	Comments	Units
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 6. 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 7. 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 parameters monitored in eight second science/calibration telemetry RDR	unitless

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.

2.1.1.1.1 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:

- o 600 msec slew from ICT to Earth Scene (ES) 1
- o 200 msec x 30 Earth Scenes
- o 200 msec slew from ES 30 to Deep Space (DS)
- o 200 msec x 2 samples at DS (forward and reverse sweeps)
- o 400 msec slew from DS to ICT
- o 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

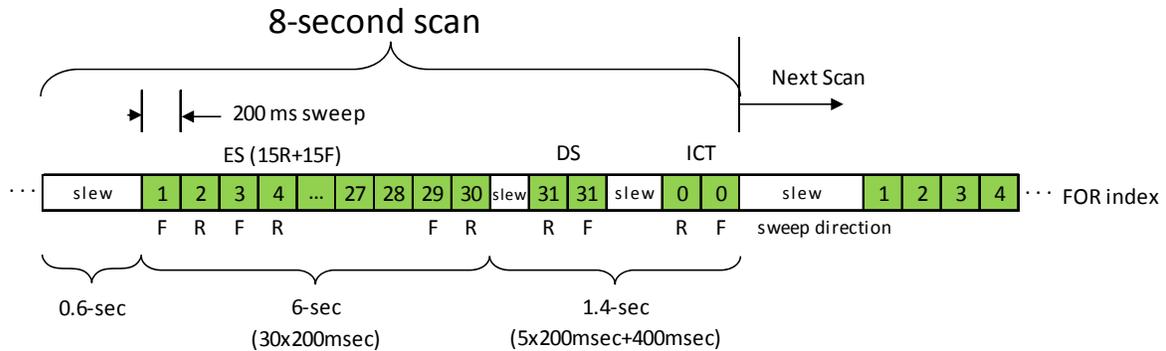


Figure 3. CrIS Measurement Sequence

2.1.1.1.2 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.

2.1.1.2 Outputs

Tables 8 and 9 list the CrIS SDR algorithm outputs.

Table 8. CrIS Truncated Spectrum SDR Outputs

Output	Description	Reference Document
CrIS SDR	<p>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).</p> <p>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.</p>	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	<p>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 2711 channels whereas the TSR has 1592.</p>	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS SDR Engineering Packet Backup AUX	<p>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.</p>	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 9. CrIS Full Spectrum SDR Outputs

Output	Description	Reference Document
CrIS Full Spectrum SDR	<p>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).</p> <p>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.</p>	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	<p>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 2711 channels whereas the TSR has 1592.</p>	474-00448-02-03_JPSS-DD-Vol-II-Part-3
CrIS Full Spectrum SDR Engineering Packet Backup AUX	<p>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.</p>	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="1751.250000"/>
<edrMwMinimumWavenumber Default="1208.750000"/>
<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.

2.1.2 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. For information regarding common geolocation refer to 474-00091 (current revision)_ Operational Algorithm Description Document for Common Geolocation. 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 P_{Los} and R_{Los} are the line-of-sight roll and pitch and \vec{E} is the exit vector.

2.1.2.1 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 10 lists some important classes used during processing.

Table 10. 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
ScienceDataProcessor	Class	Instantiation	Central engine for Algorithm behavior
Spectra	Class	Abstract	Interface container

Name	Type	Initialization	Description
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

2.1.2.2 Class CalibratedSpectra

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

Table 11 shows the CalibratedSpectra attributes and Table 12 shows the CalibratedSpectra operations.

2.1.2.2.1 CalibratedSpectra Attributes

Table 11. CalibratedSpectra Attributes

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

2.1.2.2.2 CalibratedSpectra Operations

Table 12. CalibratedSpectra Operations

Name	Return	Parameters
<i>Applies a complex calibration to the instance spectra. The cold contribution is added only when coldTarget Temp is !=NULL</i>		
calibrate	primitive bool	Class Spectra* hotReference
		Class Spectra* coldReference
		Class Spectra* hotTargetTemp
		Class Spectra* coldTragetTemp
<i>Applies a complex calibration to the instance spectra. The cold contribution is added only when coldTarget Temp is !=NULL (Overloaded function)</i>		
calibrate	primitive bool	Class Spectra* hotReference
		Class Spectra* coldReference
		Class Spectra* hotTargetTemp
		Class Spectra* coldTragetTemp
		BOOST::vector<Float64>& theFirGainRealBin
		BOOST::matrix<Float64>& calibrationMatrix
<i>Applies a bin by bin correction</i>		
correct	primitive bool	Class Spectra* correctionSpectra
<i>Return linear phase shift when fringe count error exist.</i>		
correctFringeCountError	Class BOOST::vector<std::complex<Float64>&	Class BOOST::vector<std::complex<Float64>>& linearPhaseShift
<i>Fringe count errors are computed and returned</i>		
detectFringeCountError	typedef double Float64	Class Spectra* hotReference
		Class Spectra* coldReference
<i>Estimate NEdN</i>		
estimateNEdN	Bool	Class Spectra* theSlope
		Class Spectra* theIntercept
		typedef unsigned int UInt32 userNumberOfPoints
		typedef double Float64 userMinWavenumber
		typedef double Float64 userDeltaSigma
<i>Shifts the imaginary component of phase to align the real component with the reference spectra.</i>		

Name	Return	Parameters	
adjustPhase	primitive void	Class BOOST::vector<std::complex<Float64>>&	<i>linearPhaseShift</i>
<i>Estimate NEdN</i>			
estimateNEdN	Bool	Class Spectra*	<i>theSlope</i>
		Class Spectra*	<i>theIntercept</i>
		typedef unsigned int UInt32	<i>userNumberOfPoints</i>
		typedef double Float64	<i>userMinWavenumber</i>
		typedef double Float64	<i>userDeltaSigma</i>

2.1.2.3 CCSDSFormat

This class implements the CommonMessageFormat for CCSDS packetization. Table 13 shows the CCSDSFormat attributes and Table 14 shows the CCSDSFormat operations

2.1.2.3.1 CCSDSFormat Attributes

Table 13. CCSDSFormat Operations

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

2.1.2.3.2 CCSDSFormat Operations

Table 14. CCSDSFormat Operations

Name	Return	Parameters	
<i>Populates the primary and secondary headers based on the data at the offset of startingLocation.</i>			
unpack	typedef int Int32	Class Octets	<i>&theData</i>
		typedef unsigned int UInt32	<i>&startingLocation</i>

2.1.2.4 Class CorrectionMatrix

This class is responsible for implementing a Correction Matrix that is applied to each spectra. Table 15 shows the CorrectionMatrix attributes and Table 16 shows the CorrectionMatrix operations.

2.1.2.4.1 CorrectionMatrix Attributes

Table 15. CorrectionMatrix Attributes

Name	Type	Description
size	typedef unsigned int UInt32	<i>Size of Correction Matrix</i>
childTable[]	Class CorrectionTable*	<i>Collection of child tables</i>
calibrationMatrix	Class BOOST::matrix<Float32>	<i>Collection of calibration matrix</i>
lastUpdateTime	typedef struct SYSTEMTIME	<i>Collection of last update time</i>
AlgorithmParameter	Class SDR_AlgorithmnParameter	<i>Collection of SDR algorithm parameters</i>
CorrectionParameter	Class SDR_CorrectionParam	<i>Collection of SDR correction parameters</i>
InstrumentCharacteristic	Class CrIS_InstrumentCharacteristics	<i>Collection of CrIS instrument parameters</i>
maxPathDifferential[]	typedef double Float64	<i>Collection of maximum path difference</i>

Name	Type	Description
userDeltaSigma[]	typedef double Float64	Collection of user delta sigma
userLowestWavenumber[]	typedef double Float64	Collection of user lowest wavenumber

2.1.2.4.2 CorrectionMatrix Operations

Table 16. CorrectionMatrix Operations

Name	Return	Parameters	
<i>Rebuilds child tables for calibration matrix construction</i>			
buildILSCorrectionTable	typedef int Int32	Class EngineeringCalibrationRecord&	<i>theEngCalRec</i>
		enum SceneElement	<i>theFOV</i>
		enum Band::Wavelength	<i>theBand</i>
<i>Builds self-apodization table</i>			
buildSelfApodization Table	primitive bool	Class EngineeringCalibrationRecord&	<i>theEngCalRec</i>
		enum SceneElement	<i>theFov</i>
		enum Band::Wavelength	<i>theBand</i>
<i>Builds Residual ILS Table</i>			
buildResidual_ILS_Table	primitive bool	Class EngineeringCalibrationRecord&	<i>theEngCalRec</i>
		enum SceneElement	<i>theFOV</i>
		enum Band::Wavelength	<i>theBand</i>
<i>Calculates and return the inverse SA calibration matrix.</i>			
matrixProduct	boost::matrix<Float64>	boost::matrix<Float64>	&A
		boost::matrix<Float64>	&B
		boost::matrix<Float64>	&C
<i>Resizes the correction and child tables to new dimension.</i>			
setSize	primitive void	typedef unsigned int UInt32	<i>newDimension</i>
<i>Creates a comma separated string containing values from the calibrationMatrix</i>			
toString	primitive string	n/a	<i>n/a</i>
<i>Serializes the correction matrix data</i>			
serializeCmo	primitive void	Class CrisDataStream&	&ss
<i>Serializes the correction matrix with user deltaSigma.</i>			
serializeCMO	primitive void	Class cris_oarchive&	<i>ar</i>
		enum Band::Wavelength	<i>theBand</i>
<i>Serializes the correction matrix with user deltaSigma.</i>			
serializeCMO	primitive void	Class cris_iarchive&	<i>ar</i>
		enum Band::Wavelength	<i>theBand</i>

2.1.2.5 Class EngineeringCalibrationRecord

This class is a parent class to each of the Engineering Calibration Record sub categories. Table 17 shows the EngineeringCalibrationRecord attributes and Table 18 shows the EngineeringCalibrationRecord operations.

2.1.2.5.1 EngineeringCalibrationRecord Attributes

Table 17. EngineeringCalibrationRecord Attributes

Name	Type	Description
ietTime	typedef long long Int64	<i>IET time</i>
computedWavelengthRejectionThreshold	typedef unsigned int UInt32	<i>Computed wavelength rejection threshold</i>
rejectedWavelengthsRatio	typedef double Float64	<i>Rejected wavelength ratio</i>
averageMetrologyWavelength	typedef double Float64	<i>Averaged metrology wavelength</i>
engLaserDiodeTemperature	typedef double Float64	<i>Laser diode temperature</i>
engLaserDiodeCurrent	typedef double Float64	<i>Laser diode current</i>
laserDiodeWavelengthOrigin	enum LaserWavelengthSource	<i>Original laser diode wavelength</i>
ilsParams	typeded struct ILS_Parameters	<i>Collection of ILS parameters</i>
ilsOrigin	enum IlsOriginSource	<i>Collection of ILS original source</i>
theEngCalRec_ICTEmissivityParameters	Class EngCalRec_ICTEmissivityParameters	<i>Collection of engineering ICT emissivity parameters</i>
theEngCalRec_ILSCurveFitParameters	Class EngCalRec_ILSCurveFitParameters	<i>Collection of ILS curve fit parameters</i>
theEngCalRec_ILSFOVParameters	Class EngCalRec_ILSFOVParameters	<i>Collection of ILS fov parameters</i>
theEngCalRec_ICTEnvironmentalModel	Class EngCalRec_ICTEnvironmentalModel	<i>Collection of ICT environmental model</i>
theEngCalRec_ScienceTelemetryConversionCoefficients	Class EngCalRec_ScienceTelemetryConversionCoefficients	<i>Collection of science telemetry conversion coefficients</i>
theEngCalRec_PolarizationCalibrationWavenumbers	Class EngCalRec_PolarizationCalibrationWavenumbers	<i>Collection of polarization calibration wavenumbers</i>
theEngCalRec_LaserMetrologyInfo	Class EngCalRec_LaserMetrologyInfo	<i>Collection of laser metrology information</i>
theEngCalRec_NeonCallInfo;	Class EngCalRec_NeonCallInfo	<i>Collection of neon calibration information</i>
theEngCalRec_ScienceTelemetryLimits	Class EngCalRec_ScienceTelemetryLimits	<i>Collection of science telemetry limits</i>
theEngCalRec_PitchRollYawInfo	Class EngCalRec_PitchRollYawInfo	<i>Collection of pitch, roll and yaw information</i>
theEngCalRec_LinearityErrorParameters	Class EngCalRec_LinearityErrorParameters	<i>Collection of linearity error parameters</i>
theEngCalRec_ExtractionRecord	Class EngCalRec_ExtractionRecord	<i>Trim table and extraction record values</i>
losInSSMF []	Class BOOST::vector<Float64>	<i>Collection of pitch and yaw misalignments between interferometer and scan mechanism</i>
normalRMF	Class BOOST::vector<Float64>	<i>Collection of mirror normal to SSM mirror in RMF coordinate system</i>
sbfToSSMFMatrix	Class BOOST::vector<Float64>	<i>Collection of transformation from SBF to SSMF</i>
lastChecksumValue	typedef int Int32	<i>Last check sum value</i>
packetVersionNumber	typedef int Int32	<i>Packet version number</i>

2.1.2.5.2 EngineeringCalibrationRecord Operations

Table 18. EngineeringCalibrationRecord Operations

Name	Return	Parameter
<i>Refreshes data</i>		

Name	Return	Parameter	
refreshData	primitive bool	typedef unsigned short int UInt16	<i>theApid</i>
		class CCSDSFormat*	<i>the TelemetryFormat</i>
<i>Refreshes data (Overloaded function)</i>			
refreshData	primitive bool	typedef unsigned short int UInt16	<i>theApid</i>
		class CCSDSFormat*	<i>the TelemetryFormat</i>
		CalibrationOrder	<i>theOrder</i>
<i>calculates metrology wavelength</i>			
calculateMetrologyWavelength	primitive void	n/a	<i>n/a</i>
<i>Loads the values from the engineering calibration record</i>			
serialize	primitive void	Class cris_iarchive	<i>&ar</i>
		Class std::ostream stringstream	<i>&oss</i>
<i>Initializes the scene selection module frame (SSMF) to spacecraft body frame (SBF) transformation operator.</i>			
initializeGeometricCalibration	primitive void	n/a	<i>n/a</i>
<i>Computes rotation matrix around Y axis</i>			
rotationMatrixY	Class BOOST::matrix<Float64>	typedef double Float64	<i>thePitch</i>
<i>Compute rotation matrix around X axis</i>			
rotationMatrixX	Class BOOST::matrix<Float64>	typedef double Float64	<i>theRoll</i>
<i>Computes rotation matrix around Z axis.</i>			
rotationMatrixZ	Class BOOST::matrix<Float64>	typedef double Float64	<i>theYaw</i>

2.1.2.6 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 19 shows the Interferogram attributes and Table 20 shows the Interferogram operations

2.1.2.6.1 Interferogram Attributes

Table 19. Interferogram Attributes

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

Name	Type	Description
fieldOfView	enum SceneElement	Actual field of view reported in interferogram

2.1.2.6.2 Interferogram Operations

Table 20. Interferogram Operations

Name	Return	Parameters
<i>Adds an interferogram to the current interferogram.</i>		
operator+	primitive void	Class Interferogram& <i>rightOperand</i>
<i>subtracts an interferogram from the current interferogram.</i>		
operator-	primitive void	Class Interferogram& <i>rightOperand</i>
<i>Divides each element of this interferogram by a specified number.</i>		
operator/	primitive void	typedef double Float64 <i>factor</i>
<i>Copies all data from source interferogram.</i>		
operator=	Class Interferogram&	Class Interferogram& <i>source</i>
<i>Outputs information of the current interferogram.</i>		
operator<<	Class ostream&	Class ostream& <i>outStream</i>
		Class Interferogram& <i>i</i>
<i>returns data of the specified type from the interferogram.</i>		
getSamples	Class BOOST::vector<Float64>&	enum Complex <i>Real or imaginary samples</i>
<i>Computes deep space symmetry.</i>		
computeDeepSpaceSymmetry	typedef double Float64	Class Interferogram& <i>forwardDS</i>
		Class Interferogram& <i>reverseDS</i>
<i>References to the root of the complex squares.</i>		
getMagnitude	Class BOOST::vector<Float64>&	n/a <i>n/a</i>

2.1.2.7 Class TemperatureHistory

This class is a container for temperatures gathered from the data. Table 21 shows the temperatureHistory attributes and Table 22 shows the temperatureHistory operations.

2.1.2.7.1 TemperatureHistory Attributes

Table 21. 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

2.1.2.7.2 TemperatureHistory Operations

Table 22. TemperatureHistory Operations

Name	Return	Parameters	
<i>Removes all records in the temperature history window.</i>			
clearHistory	primitive void	n/a	n/a
<i>Sets the window size for a temperature window.</i>			
setWindowSize	primitive void	typedef int Int32	newSize
<i>Returns a temperature at any given position.</i>			
getTemperature	typedef double double Float64	typedef int Int32	position
<i>Adds the current average temperature to the window if a temperature is not supplied.</i>			
addTemperature	typedef double double Float64	n/a	n/a
<i>Adds temperature to a temperature window.</i>			
addTemperature	typedef double double Float64	typedef double Float64	newTemp
		typedef struct CrisSdrHeapDataType	*algDataPtr
		typedef unsigned int UInt32	scanIdx
		typedef int Int32	ictTempType
<i>Calculates an average of a temperature window.</i>			
calculateAverageTemperature	typedef double double Float64	n/a	n/a
<i>returns the number of valid temperatures in temp history window</i>			
getValidCount	typedef int Int32	n/a	n/a
<i>Removes a temperature record from the temperature window.</i>			
removeTemperature	primitive bool	n/a	n/a
<i>Update ICT</i>			
updateICT	void	CrisSdrAlgDataType	algDataPtr
		typedef int Int32	scanIdx
		typedef int Int32	ictTempType
<i>Calculate Median Thermal Temp Drift</i>			
medianThermalTempDrift	void	n/a	n/a

2.1.2.8 Class ScienceDataProcessor

This class is responsible for processing the CrIS SDR science data. Table 23 shows the ScienceDataProcessor attributes and Table 24 shows the ScienceDataprocessor operations.

2.1.2.8.1 ScienceDataProcessor Attributes

Table 23 shows the science data processor attributes. This class instantiates the Planck radiation constants for the Planck Function specified in the Cross Track Infrared Sounder (CrIS) Sensor Data Records (SDR) Algorithm Theoretical Basis Document ATBD (D0001-M01-S01-002). They are of type double, as opposed to the single precision format specified in the ATBD. C_1 requires at least five extra points of precision, because the ATBD multiplies it by a factor of 10^{-5} . This extra amount of precision also ensures better accuracy in Planck function calculations.

Table 23. ScienceDataProcessor Attributes

Name	Type	Description
correctionTable[][]	Class CorrectionMatrix	Collection of correction matrix tables
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
updateTargetRadiance	primitive bool	Updates target radiance
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
validCalTargetTolerance	typedef struct SYSTEMTIME	Collection of valid calibration time tolerance
validColdTargetDuration	Class TimeRange	Collection of valid cold target time
validColdTargetTolerance	typedef struct SYSTEMTIME	Collection of valid cold target tolerance
validHotTargetDuration	Class TimeRange	Collection of valid hot target time
validHotTargetTolerance	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
theILSPParametersRecord	Class ILSPParametersRecord	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
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
numCMOBuils_	typedef unsigned short UInt16	Number of CMO builds
numEngBuils_	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
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
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

2.1.2.8.2 ScienceDataProcessor Operations

Table 24 shows the science data processor operations.

Table 24. ScienceDataProcessor Operations

Name	Return	Parameter
<i>Cleans up of ScienceDataProcessor data for next dispatch.</i>		
cleanup	primitive void	n/a
<i>Populates high access frequency domain data.</i>		
calculateWavenumberBinSpacing	primitive void	enum Band::Wavelength theBand
<i>Calculates the post calibration table. This calculation is only band specific.</i>		
buildPostCalibrationTable	primitive bool	enum Band::Wavelength theBand
<i>Calculates Resampling table. This calculation is only band specific.</i>		
buildResamplingTable	primitive bool	enum Band::Wavelength theBand
<i>Calculates the user apodization table. This calculation is only band specific.</i>		
buildUserApodizationTable	primitive bool	enum Band::Wavelength theBand
<i>Holds the intelligence to route any RDR packet to assistant class processing.</i>		
processRawDataRecord	Class Spectra*	n/a
<i>Reads external environment data from a file.</i>		
processEnvironmentalData	primitive bool	typedef struct SYSTEMTIME& theSpectraFrameTime
<i>Builds polarization curve.</i>		

Name	Return	Parameter	
buildPolarizationCurve	primitive void	Class Spectra*	newSpectra
		enum FiledOfRegard	theScene
		enum Bans::Wavelength	theband
<i>Creates and adds a new spectrum to the spectrasliding window. Reference spectra are simply returned.</i>			
AddSpectra	Class Spectra*	Class Spectra*	newSpectra
<i>Builds each child table for every band and fov.</i>			
refreshCorrectionMatrix	primitive int	primitive bool	isloading
<i>Refreshes the self-apodization correction matrix</i>			
refresh_Invsa_CorrectionMatrix	primitive int	primitive bool	isloading
<i>Multiplies child correction table to calculate the final calibration matrix.</i>			
calculateCalibrationMatrix	primitive void	n/a	n/a
<i>Combines requested corrections.</i>			
combineRequestedCorrections	typedef unsigned int UInt32	typedef unsigned int UInt32	requestedFOV
<i>Updates correction matrix parameters from configuration file.</i>			
updateCorrectionmatrixParameters	primitive void	n/a	n/a
<i>Checks if there is a missing sciCal pkt in the last scan of the moving window.</i>			
checkICT_Status	void	typedef int Int32	scanIdx
		typedef struct CrisSdrAlgDataType*	algDataPtr
<i>Calculates the hot radiance.</i>			
calculateHotRadiance	Class Spectra*	enum SceneElement	theFOV
		enum Band::Wavelength	theBand
<i>Calculates the cold radiance.</i>			
calculateColdRadiance	Class Spectra*	enum SceneElement	theFOV
		enum Band::Wavelength	theBand
<i>Sets the window size base on the spectral calibration method.</i>			
setSpectralCalibrationMethod	primitive void	enum SpectralCalibrationMethod	newMethod
<i>Removes a spectrum from the sliding window without adding a spectrum.</i>			
flushWindow	Class Spectra*	enum SceneElement	fov
		enum Band::Wavelength	band
		enum FieldOfRegard	scene
		enum SweepDirection	psd
<i>Processes Earth Scene spectra according to configuration parameters. Calibration, correction and validity are evaluated.</i>			
processSpectra	Class Spectra*	Class Spectra*	newSpectra
<i>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.</i>			
moniterLaserDiode	primitive bool	class Spectra*	theSpectra
<i>Determines if the laser wavelength needs to be updated and updates if needed.</i>			
monitorLaserWavelength	primitive bool	Class Spectra*	theSpectra
<i>Propagates temporal window size to assistant classes.</i>			
setWindowSize	primitive void	primitive int	newSize
<i>Propagates temporal window size to assistant classes.</i>			

Name	Return	Parameter	
declareWindowSize	primitive void	typedef unsigned int UInt32	referenceSize
		typedef unsigned int UInt32	targetTempSize
		typedef unsigned int UInt32	CalibrationSize
<i>Loads each correction matrix from a DMS</i>			
LoadCMOs	primitive bool	n/a	n/a
<i>Loads EngPkt from a DMS</i>			
LoadEngPkt	primitive bool	n/a	n/a
<i>Calculates desired band center and configures spectra's desired band center.</i>			
calculateDesiredBandCenter	primitive void	n/a	n/a
<i>Applies the CMO to a spectrum.</i>			
applySpectralCorrection	primitive void	Class Spectra*	newSpectra
		enum SceneElement	theFov
		enum Band::Wavelength	theBand
<i>Saves serialized CMO. The values are from correction matrix class serialized.</i>			
saveSerializedCorrectionMatrix	primitive void	n/a	n/a
<i>Write 4-min engineering packet to DMS.</i>			
saveSerializedEngPkt	primitive void	n/a	n/a
<i>Loads serialized CMO from DMS and populates correction matrix class.</i>			
loadSerializedCorrectionmatrixFromDMS	primitive void	n/a	n/a
<i>Computes field of views line-of-sight in spacecraft body frame.</i>			
calculateGeometricCalibration	Class Spectra*	Class Spectra*	newSpectra
<i>Refreshes a spectrum time from telemetry data.</i>			
refreshTime	primitive void	Class CCSDSFormat*	theTelemetryFormat
<i>Synchronizes all windows based on reference spectra sequence counts and updates sequence counts.</i>			
syncWindows	primitive void	n/a	n/a
<i>Calls the Spectra manager's batchReferenceFCE which does fringe count detection and correction.</i>			
launchBatchreferenceDetect	primitive void	Class SpectraManager::RefernceSpectra	spectraType
		enum SweepDirection	thePSD
<i>Interpolates a subset such that individual offset values are available for each 8 seconds scan period.</i>			
buildBaffleTemperatureOffset	primitive void	Class BOOST::vector<Float32>&	source
		typedef int int32	orbitDuration
<i>Calculates the gain introduced by FIR Filter.</i>			
buildFirFilterGainTable	primitive void	enum Band::Wavelength	requestdBand
		typedef int int32	foldindex

2.1.2.9 Class SpectraManager

This class is responsible for each Spectra currently used in the moving window average. Table 25 shows the SpectraManager attributes and Table 26 shows the SpectraManager operations.

2.1.2.9.1 SpectraManager Attributes

Table 25. 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 cold 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
maintainReferenceSceneVariance	primitive bool	Reference scene variance
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
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

2.1.2.9.2 SpectraManager Operations

Table 26. SpectraManager Operations

Name	Return	Parameters	
<i>Cleans up all obsolete spectra.</i>			
Cleanup	primitive void	n/a	n/a
<i>Initializes data for re-tasking.</i>			
Init	primitive void	n/a	n/a
<i>Adds a correction spectrum to the moving window for synchronized application to calibration.</i>			
scheduleCorrection	primitive void	primitive string	newCMOFileName
		enum SceneElement	theFov
		enum Band::Wavelength	theBand
<i>Adds the newSpectra to the measure spectra container.</i>			
addSpectra	Class Spectra*	Class Spectra*	newSpectra
<i>Adds hot and cold target curves to the radiance container for the current spectra.</i>			
addTargetRadianceCurves	primitive void	Class Spectra*	hotTargetCurve
		Class Spectra*	coldTargetCurve
<i>Synchronizes the windowed data in the event the data set does not begin on an 8 sec boundary.</i>			
syncReferenceSpectra	primitive bool	Class Spectra&	targetSpectra
		primitive bool	performFringeCountErrorHandling
<i>Reconciles/Synchronizes the reference spectra's phase to the target spectra's phase.</i>			
reconcileReferenceSpectraPhase	primitive bool	SceneElement	theFOV
		typedef int Int32	maxFCETries
<i>Returns requested measured spectra.</i>			
getMeasuredSpectra	Class Spectra*	typedef int Int32	temporalItem
		enum SceneElement	fov
		enum Band::Wavelength	band
		enum FieldOfRegard	scene
		enum SweepDirection	psd
<i>Returns requested correction spectra (mathematical correction matrix).</i>			
getCorrectionFilename	primitive string	typedef int Int32	temporalItem
		enum SceneElement	theFov
		enum Band::Wavelength	theBand
<i>Gets the valid count of spectra for a given fov, band, scene, and PSD</i>			
getValidCount	typedef int Int32	enum SceneElement	fov
		enum Band::Wavelength	band
		enum FieldOfRegard	scene
		enum SweepDirection	psd
<i>Sets the internal window size. If the content of the current window exceeds the new size, the oldest contents are discarded.</i>			
setWindowSize	primitive void	typedef int Int32	numberOfSpectra
<i>Forces a Spectra out of the window without adding one.</i>			
purgeSpectra	Class Spectra*	enum SceneElement	theFov
		enum Band::Wavelength	theBand
		enum FieldOfRegard	theFOR
		enum SweepDirection	thePSD
<i>Maintains Slope and Intercept values for rapid mathematical use.</i>			
buildNoisePrediction	buildNoisePrediction	enum SceneElement	theFov
		enum Band::Wavelength	theBand
		enum SweepDirection	thePSD
<i>Propagates temporal window size to assistant classes.</i>			
declareWindowSize	typedef unsigned int UInt32	typedef unsigned int UInt32	referenceSize
		typedef unsigned int UInt32	calibrationSiz
<i>Marks the calibration window invalid for current contents.</i>			
resetCalibrationWindow	primitive void	enum Band::Wavelength	band

Name	Return	Parameters	
		typedef unsigned int UInt32	<i>newFoldIndex</i>
<i>Returns the total count of all buffering windows requested.</i>			
getBufferedSpectraCount	typedef unsigned int UInt32	primitive bool	<i>includeReferenceCount</i>
<i>Adds a polarization correction spectra to the moving window for synchronized application.</i>			
schedulePolarizationCorrection	primitive void	Class Spectra*	<i>newSpectra</i>
		enum FieldOfRegard	<i>patternScene</i>
		enum Band::Wavelength	<i>patternBand</i>
<i>Returns requested polarization correction spectra.</i>			
getPolarizationSpectra	Class Spectra*	typedef int Int32	<i>temporalItem</i>
		enum FieldOfRegard	<i>theScene</i>
		enum Band::Wavelength	<i>theBand</i>
<i>Adds a geolocation spectrum to the moving window for synchronized application to calibration.</i>			
scheduleGeolocation	primitive void	Class Spectra*	<i>newSpectra</i>
		enum FieldOfRegard	<i>theScene</i>
		enum SceneElement	<i>theFov</i>
<i>Returns requested geolocation spectra.</i>			
getGeolocationSpectra	Class Spectra*	typedef int Int32	<i>temporalItem</i>
		enum FieldOfRegard	<i>theScene</i>
		enum SceneElement	<i>theFov</i>
<i>Calibrates the newMeasurement and performs a ratio threshold check against the corresponding window average.</i>			
monitorLunarIntrusion	primitive bool	Class Spectra&	<i>newMeasurement</i>
<i>Adjusts window phase.</i>			
adjustWindowPhase	typedef int Int32	enum FieldOfRegard	<i>theFOR</i>
		enum SceneElement	<i>theFov</i>
		enum Band::Wavelength	<i>theBand</i>
		enum SweepDirection	<i>thePSD</i>
		Class BOOST::vector< std::complex<Float64> >&	<i>linearPhaseShift</i>
<i>Removes a given spectra from the rawSummationSpectra and squareSummationSpectra.</i>			
removeInvalidSpectra	primitive void	Class Spectra*	<i>oldSpectra</i>
		enum SpectralCalibrationMethod	<i>calType</i>
<i>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.</i>			
batchReferenceFCE	primitive void	enum ReferenceSpectra	<i>spectraType</i>
		enum SweepDirection	<i>psd</i>
<i>Adds new reference spectra to holdingReferenceSpectra container.</i>			
addHoldingReferenceSpectra	primitive void	Class Spectra*	<i>newSpectra</i>
<i>ICT and DS Synchronization batch job that is done when the first ES pops out of the window.</i>			
batchICTDSSynchronization	primitive bool	Class Spectra*	<i>oldSpectra</i>
		primitive int	<i>maxFCETries</i>
<i>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.</i>			
batchEarthSceneFCE	primitive bool	Class Spectra*	<i>oldSpectra</i>
<i>Calculates the estimated detector voltage at the preamp.</i>			
calculateVdc	primitive bool	Class Spectra*	<i>sceneSpectra</i>

2.1.2.10 Class Spectra

This class is the container for the processed video data. Table 27 shows the Spectra attributes and Table 28 shows the Spectra operations.

2.1.2.10.1 Spectra Attributes

Table 27. 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

2.1.2.10.2 Spectra Operations

Table 28. Spectra Operations

Name	Return	Parameters	
<i>References to the root of the complex squares.</i>			
getMagnitude	Class BOOST::vector<Float64>&	n/a	n/a
<i>Resizes the bin count.</i>			
setSize	primitive void	typedef unsigned int UInt32	newSize
<i>Clears internal data vectors.</i>			
clear	primitive void	n/a	n/a
<i>Adds complex squares.</i>			
addSquare	primitive void	Class Spectra&	source
		primitive bool	complex
<i>Removes complex squares.</i>			
removeSquare	primitive void	Class Spectra&	source
		primitive bool	complex
<i>Divides each element of the spectra by a specified number.</i>			
operator/	primitive void	typedef double Float64	factor
<i>Multiplies each element of the spectra by a specified number.</i>			
operator*	primitive void	typedef double Float64	factor
<i>Multiplies each element of the spectra bin by bin with the factor.</i>			
operator*	primitive void	Class Spectra&	factor
<i>Divide the spectra by a specified spectra.</i>			
operator/	primitive void	Class Spectra&	divisor

Name	Return	Parameters	
<i>Makes a copy of source spectra.</i>			
operator=	Class Spectra&	Class Spectra&	source
<i>Adds a spectrum to the current spectra.</i>			
operator+	primitive void	Class Spectra&	rightOperand
<i>Subtracts a spectrum from the current spectra.</i>			
operator-	primitive void	Class Spectra&	rightOperand
<i>Returns packed content of the class.</i>			
save	primitive string	n/a	n/a
<i>Clips and re-samples it's self to new resolution.</i>			
resample	primitive bool	typedef double Float64	startWavenumber
		typedef double Float64	stopWavenumber
		typedef double Float64	binSize
<i>Clips and re-samples it's self to new resolution.</i>			
clipGuardBands	primitive bool	typedef double Float64	startWavenumber
		typedef double Float64	stopWavenumber
<i>Returns packed content of the class.</i>			
saveVersion400	primitive void	primitive string&	binaryImage
<i>Generates NEdN estimations.</i>			
generateUSN	primitive void	Class Spectra*	rawSum
		Class Spectra*	squaredSum
		typedef int Int32	windowSize
<i>Smooths the spectra.</i>			
Smooth	primitive void	typedef int Int32	windowSize
<i>Applies linearity error correction to the spectra.</i>			
applyLinearityErrorCorrection	primitive void	n/a	n/a
<i>Updates CrIS SDR geolocation.</i>			
updateCrisSdrGeolocation	primitive void	typedef struct CrisSdrAlgDataType*	algDataPtr
		typedef int Int32	scanIdx
<i>Updates CrIS SDR data.</i>			
updateCrisSdrData	primitive void	typedef struct CrisSdrAlgDataType	*sdrPtr
		typedef struct CrisSdrHdrDataType	*hdrPtr
		typedef unsigned int UInt32	scanIdx
		std::auto_ptr<ScienceDataProcessor>&	ScienceDataProcessorPtr

2.1.2.11 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 29 shows the TelemetryProcessor attributes and Table 30 shows the TelemetryProcessor operations.

2.1.2.11.1 TelemetryProcessor Attributes

Table 29. 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

2.1.2.11.2 TelemetryProcessor Operations

Table 30. TelemetryProcessor Operations

Name	Return	Parameters	
<i>Returns index of first telemetry point with the specified APID.</i>			
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 TelemetryPoint::CoefContextCollection*	coefficientContexts
		Class TelemetryPoint::LimitContextCollection*	limitContexts
<i>Returns a constant reference to the telemetry point at the given index.</i>			
getRecord	Class TelemetryPoint&	typedef int Int32	vectorIndex
<i>Clears and performs cleanup operations on the telemetry record.</i>			
purgeTelemetryRecord	primitive void	n/a	n/a
<i>Returns the name of the frame group to which the specified APID belongs.</i>			
identifyFrameType	primitive string	typedef unsigned short int UInt16	keyAPID
<i>Parses a single telemetry frame.</i>			
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
<i>Extracts the video data from the supplied data.</i>			
extractVideoData	primitive void	typedef unsigned int UInt32	detector
		typedef unsigned int UInt32	wavelength
		typedef unsigned char UInt8	*trimmedData

Name	Return	Parameters	
<i>Extracts the diagnostic data from the supplied data.</i>			
extractDiagnosticData	primitive void	typedef unsigned short UInt16	<i>theAPID</i>
		typedef unsigned int UInt32	<i>detector</i>
		typedef unsigned int UInt32	<i>wavelength</i>
		typedef unsigned char UInt8	<i>*detData</i>
		typedef int Int32	<i>dataOffset</i>
<i>Matches the specified APID with a detector and a wavelength.</i>			
rdrLookup	primitive void	typedef unsigned short UInt16	<i>theAPID</i>
		typedef unsigned int UInt32	<i>*detector</i>
		typedef unsigned int UInt32	<i>*wavelength</i>
		typedef unsigned int UInt32	<i>*scene</i>
<i>Resets LastBurstApidDetection by forgetting that any of the APIDs were in the last burst.</i>			
resetLastBurstApidDetection	primitive void	n/a	<i>n/a</i>
<i>Updates TelemetryPoint stale value.</i>			
updateStaleValues	primitive void	n/a	<i>n/a</i>
<i>Checks whether the specified APID is recent.</i>			
isApidRecent	primitive bool	typedef unsigned short UInt16	<i>theApid</i>
<i>Finds and returns the info associated with the specified apid.</i>			
findApidInf	typedef struct ApidInfoStruct*	typedef unsigned short UInt16	<i>apid</i>
<i>Reads telemetry points data.</i>			
readTelemetryPoints	primitive void	n/a	<i>n/a</i>
<i>Checks if telemetry record defined.</i>			
enableLimitChecking	primitive void	primitive bool	<i>limitCheck</i>

2.1.2.12 Class VideoData

This class is responsible for internal bit-trim reversal behavior of the TelemetryProcessor class. Table 31 shows the VideoData attributes and Table 32 shows the VideoData operations.

2.1.2.12.1 VideoData Attributes

Table 31. VideoData Attributes

Name	Type	Description
theExtractionRecord[]	struct EXTRACTION_RECORD	Collection of band specific bit trim values
theTImLookupTable[]	struct TELEMTRY_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)

2.1.2.12.2 VideoData Operations

Table 32. VideoData Operations

Name	Return	Parameters	
<i>Populates the internal symbol table used to unpack and extract the bit-trimmed data set.</i>			
initSymbolTable	primitive void	primitive bool	<i>useDefaultTable</i>
<i>Coordinates the real and imaginary extraction of the complex samples into the real and imaginary portions.</i>			
Extract	primitive void	typedef unsigned int UInt32	<i>detector</i>
		typedef unsigned int UInt32	<i>wavelength</i>
		typedef unsigned char UInt8	<i>*trimmedData</i>
<i>Retrieves the correct trim value from the symbol table for a given wavelength sample.</i>			

Name	Return	Parameters	
getTrimSize	typedef unsigned short UInt16	typedef unsigned int UInt32	<i>wavelength</i>
		typedef int Int32	<i>sample_index</i>
<i>Established the 4 minute bit trim tables for all three bands.</i>			
buildExtractionTables	primitive void	primitive bool	<i>useDefaultTable</i>
<i>Extracts the original sample from the pack data stream by using the bit-trimming information configured in the symbol table.</i>			
restoreSamples	primitive void	typedef unsigned int UInt32	<i>detector</i>
		typedef unsigned int UInt32	<i>wavelength</i>
		typedef unsigned short int UInt16	<i>dataElement</i>
		primitive bool	<i>realComponent</i>
<i>Updates data values in the extradction record</i>			
setExtractionRecord	EXTRACTION RECORD struct	EXTRACTION_RECORD	<i>record</i>
		UInt32	<i>band</i>

2.1.2.13 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 33 shows the ScienceCalibrationRecord attributes and Table 34 shows the ScienceCalibration operations.

2.1.2.13.1 ScienceCalibrationRecord Attributes

Table 33. 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
ieCCA CalibResistorTemp[]	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
omaStructureTemp1	typedef int Int32	OMA structure temperature
omaStructureTemp2;	typedef int Int32	OMA structure 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.

2.1.2.13.2 ScienceCalibrationRecord Operations

Table 34. ScienceCalibration Operations

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

2.1.2.14 SDR Generator Application COTS Components

2.1.2.14.1 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.

2.1.2.14.2 LAPACK

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

2.1.2.14.3 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.

2.1.2.15 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, 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.

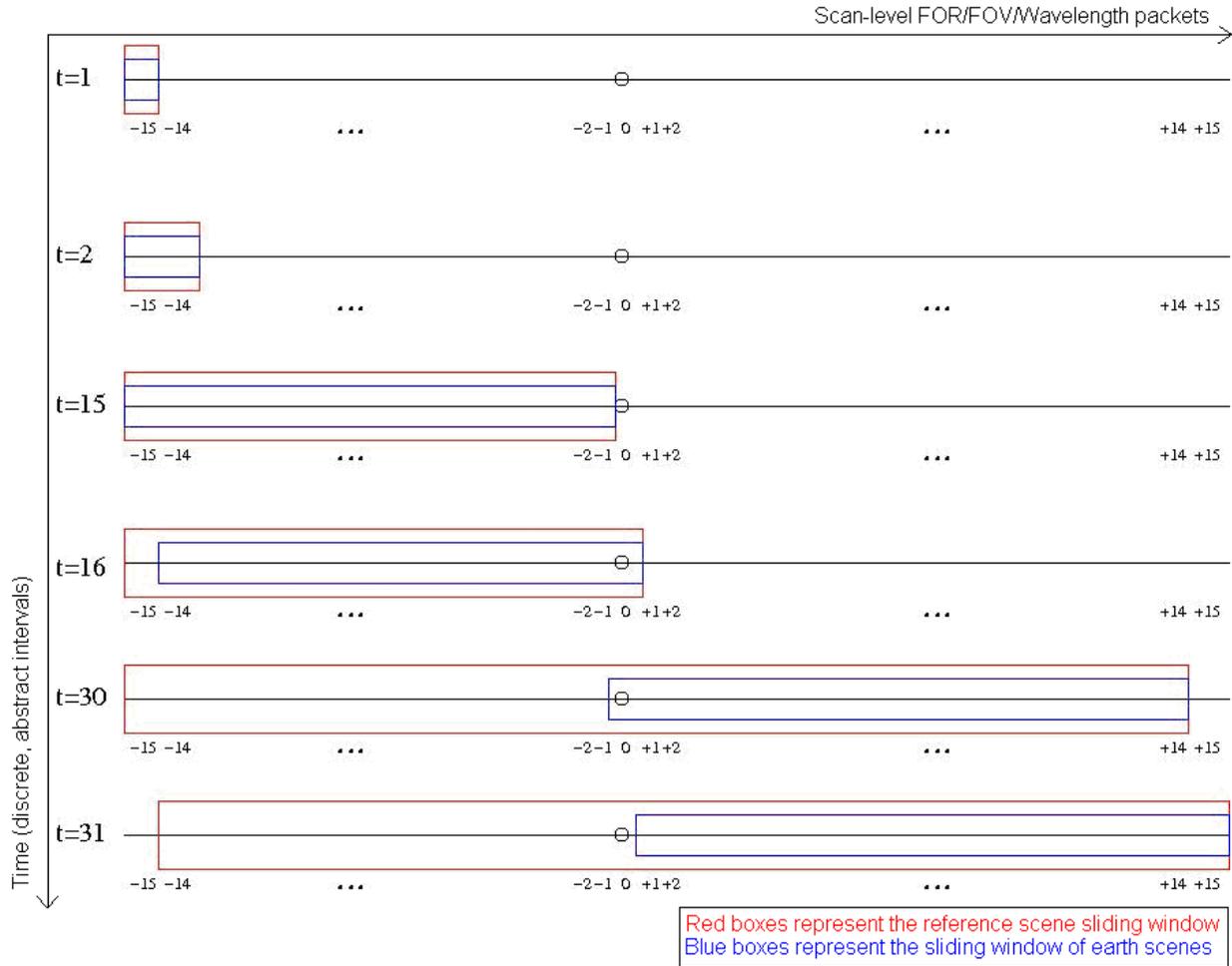


Figure 4. Sliding Window Concept

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 4.

2.1.2.15.1 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.

2.1.2.15.2 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: $\left\lceil \frac{W}{S} \right\rceil$

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

2.1.2.15.3 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.

2.1.2.16 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 5 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.

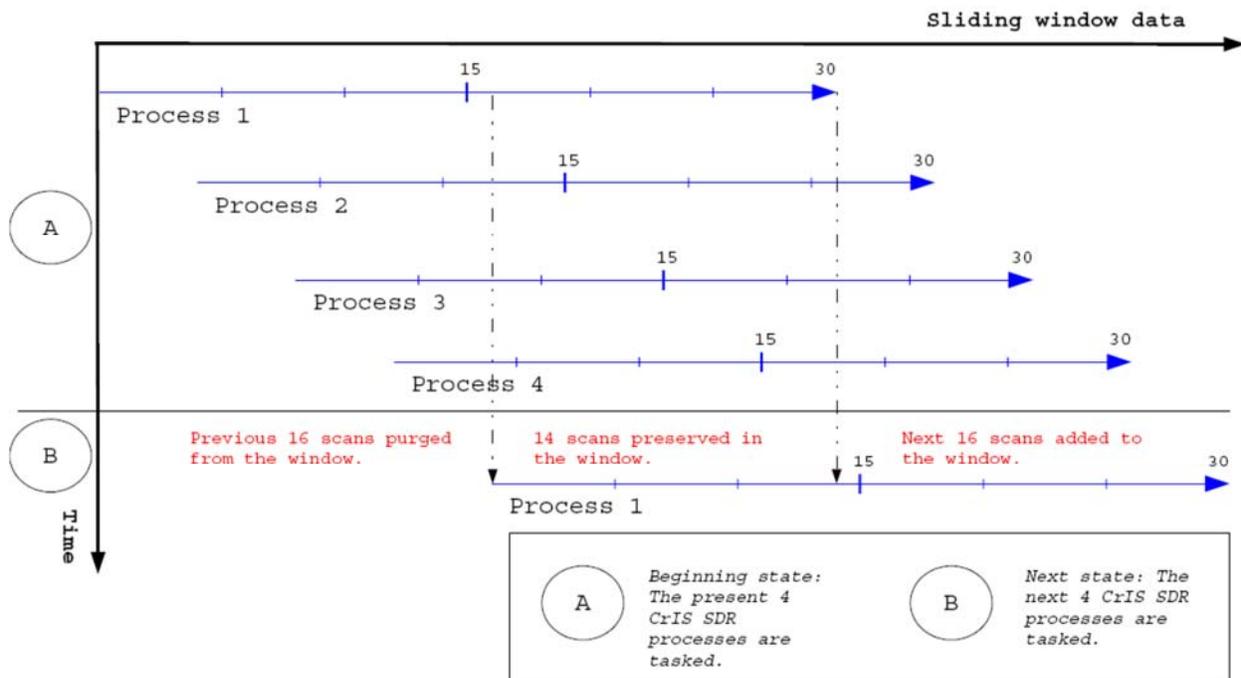


Figure 5. Sliding Window Optimization Concept

2.1.2.17 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 6 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.

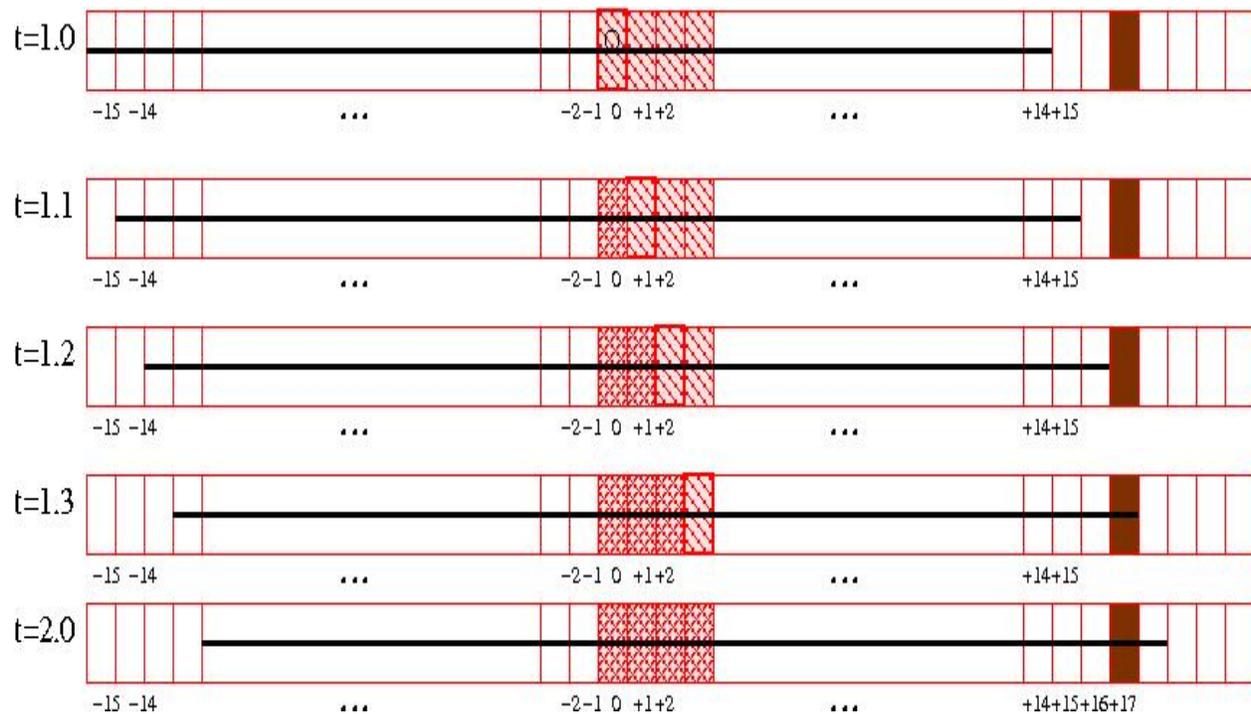


Figure 6. Sequential Tasking of a Correction Matrix Build

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.

2.1.2.18 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.

2.1.2.19 ATBD - Operational Adaptation, Deviation or Limitations

The theoretical basis for the CrIS SDR Algorithm is documented in the following document: Cross Track Infrared Sounder (CrIS) Sensor Data Records (SDR) Algorithm Theoretical Basis Document ATBD, D0001-M01-S01-002.

The following deviations or adaptations to the referenced ATBD 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 algorithm as described in the ATBD does not address this requirement, however, 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, $\tilde{S}^{DS}[n]$

2. Subtract the averaged uncalibrated DS spectrum from the newest DS spectrum that was not included in the average,

$$\tilde{R}^{DS}[n] = \tilde{S}^{DS}[n] - \langle \tilde{S}^{DS}[n] \rangle$$

3. Subtract the averaged uncalibrated DS spectrum from the averaged uncalibrated ICT spectrum,

$$\tilde{R}^{ICT}[n] = \langle \tilde{S}^{ICT}[n] \rangle - \langle \tilde{S}^{DS}[n] \rangle$$

4. Compute real part of calibrated lunar spectrum in digital units averaged over the IR bands and then compare with a configurable threshold,

$$\frac{\sum_{n=n_{\min}}^{n_{\max}} \operatorname{Re} \left\{ \begin{array}{c} \tilde{R}^{DS}[n] \\ \tilde{R}^{ICT}[n] \end{array} \right\}}{n_{\max} - n_{\min}} > \frac{LI_{\lim}}{100}$$

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 3D 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.

2.1.2.20 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 $\{\text{PRO_HOME}\}/\text{cfg}/\text{CRIS}$.

2.1.3 Graceful Degradation

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

2.1.4 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.

2.1.5 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..

2.1.5.1 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.

2.1.6 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.

2.1.7 Algorithm Support Considerations

Reference the list of libraries under Section 2.1.2.14.

2.1.8 Assumptions and Limitations

2.1.8.1 Assumptions

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

2.1.8.2 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.

3.0 GLOSSARY/ACRONYM LIST

3.1 Glossary

Table 35 contains terms most applicable for this OAD.

Table 35. Glossary

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: <ol style="list-style-type: none"> 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, DPES, 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 ATBD. At a minimum, implemented software is science-grade and includes test data demonstrating data quality compliance.
Environmental Data Record (EDR)	<p><i>[IORD Definition]</i> 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.).</p> <p><i>[Supplementary Definition]</i> 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.</p>
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)	<p><i>[IORD Definition]</i> 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.</p> <p><i>[Supplementary Definition]</i> 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.</p>
Retrieval Algorithm	<p>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.</p>
Science Algorithm	<p>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 ATBD. The latter is typically developed for a research setting and characterized as “science-grade”.</p>
Science Algorithm Provider	<p>Organization responsible for development and/or delivery of TDR/SDR or EDR algorithms associated with a given sensor.</p>
Science-Grade Software	<p>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.</p>
SDR/TDR Algorithm	<p>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 Algorithm Theoretical Basis Document (ATBD). At a minimum, implemented software is science-grade and includes test data demonstrating data quality compliance.</p>
Sensor Data Record (SDR)	<p><i>[IORD Definition]</i> 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.</p> <p><i>[Supplementary Definition]</i> 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.</p>

Term	Description
Temperature Data Record (TDR)	<p><i>[IORD Definition]</i> 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.</p> <p><i>[Supplementary Definition]</i> 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.</p>
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]

3.2 Acronyms

Table 35 contains acronyms most applicable for this OAD.

Table 35. Acronyms

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
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
TBD	To Be Determined
TBR	To Be Resolved

4.0 OPEN ISSUES

Table 37. TBXs

TBX ID	Title/Description	Resolution Date
None		