AATSR LEVEL 1B DETAILED PROCESSING MODEL & PARAMETER DATA LIST

SCIENCE AND TECHNOLOGY FACILITIES COUNCIL

RUTHERFORD APPLETON LABORATORY

Chilton, Didcot

Oxfordshire OX11 0QX

Prepared by:	A.R Birks (RAL)	Digitally signed by Andrew R Birks DN: cn=Andrew R Birks, o=STFC, ou, email=andrew.birks@stfc.ac.uk, c=GB Date: 2011.10.21 15:01:59 +01'00'	*Date*
Approved by:	Caroline Cox (RAL)	Caroline Cox 2011.10.21 15:07:34 +01'00'	*Date:*

Amendment Record

Issue	Rev.	Date	Details
0	1	19 Aug 96	first draft issue compiled by ESYS from RAL inputs
0	2	22 Oct 96	revised working copy, under ESYS control
0	3	4 Dec 96	tables revised; basis is RAL server version DPM_L1B.DOC;5
		11 Dec 96	uploaded to RAL
0	4	27 Jan 97	Significantly revised by RAL
0	5	14 Feb 97	Revised geolocation and regridding
1	0	28 Feb 97	First formal release.
1	1	31 July 1997	Second formal release.
			Revised and corrected in the light of comments received.
			Breakpoint formats added.
			Visible Calibration revised. Spatial coherence test specification
			(cloud clearing) revised. Regridding algorithm revised.
			Longitude switch for testing added to Land/Sea flagging
			interface.
1	2	6 Nov 1997	Revised and corrected in the light of comments received.
			Developments in the geolocation and regridding algorithms
			included. Module 24 updated.
1	2	5 Dec. 1997	Third formal release (to ESA).
1	3	26 Jan. 1998	Visible Calibration (Module 8) updated.
			Geolocation equations corrected. Browse Product (Section 5.23)
			revised and expanded.
1	3	30 Jan. 1998	Fourth formal release.
1	4	17 April 1998	Section 5.6: mapping of Earth view pixels into arrays corrected.
			Sections 5.10, 5.11 updated to include use of po_interpol for
			restituted orbit files. Section 5.17 modified to clarify required
			rounding. Section 5.22: Spatial Coherence test revised and
			clarified. Sections 5.23, 24 corrected. General: miscellaneous
			minor revisions and corrections.
1	4	29 April 1998	Fifth formal release.
1	5	12 Oct. 1998	Revised to address SPRs to date.
1	5	11 March 1999	Revised to address SPRs to date. Visible channel calibration
			revised to include non-linear correction at 1.6 micron; infra-red
			channel calibration modified for AATSR black body distribution,
			and to support independent emissivity values for the two black
			bodies.
1	5	15 March 1999	Sixth formal release.
1	6	25 May 2000	Revised to address SPRs to date. Night-time Browse algorithm
			updated. Documentation of Cloud Test Points added.
1	6	4 July 2000	Seventh formal release.
1	7	25 Aug. 2000	Topographic correction (Module 16) extensively revised.
1	7	15 July 2002	Revised to address SPRs to date. Minor formatting corrections.
			Visible calibration algorithm revised to reflect actual AATSR

			instrument characteristics.
1	7	18 July 2002	Initial post-launch release.
1	8	29 Oct. 2002	Visible channel calibration algorithm further revised to
			accommodate orbit state vectors having an arbitrary time
			relationship to the VISCAL pulse (as in NRT data).
1	8	29 May 2003	Second post-launch release.
1	9	15 Nov. 2005	New and revised cloud tests over land.
			BROWSE algorithm updated to match current processing.
1	9	2 June 2006	Third post-launch release.
1	10	16th June 2011	Amendments to incorporate VISCAL Drift Correction.
			Amendment to VISCAL algorithm to allow for missing or
			invalid Instrument Source packets. Minor corrections.
		16th June 2011	Draft A issued.
1	10	14th Oct. 2011	Fourth post-launch release.

CONFIGURATION CONTROL

Document Last Updated by: **RAL** at **14:37 PM** on **14th October 2011**

(to update these data, click on the field, hold down right mouse *button* ***and use Update Field)***

Status(Draft / Controlled) : **Controlled**

TABLE OF CONTENTS

1. .... PURPOSE OF DOCUMENT 9
2. .... INTRODUCTION 10
3. .... REFERENCE DOCUMENTS AND APPLICABLE DOCUMENTS 11
4. .... DETAILED PROCESSING MODEL 13
4.1 Conventions 14
4.2 Overview of Processing Structure 15
4.2.1 General 15
4.2.2 Source packet Processing (Modules 1 - 6) 18
4.2.3 Infra-Red Channel calibration (Module 7) 19
4.2.4 Visible Channel calibration (Module 8) 19
4.2.5 Satellite Time Calibration (Module 9) 19
4.2.6 Geolocation 19
4.2.7 Signal Calibration (Module 17) 20
4.2.8 Regrid Pixels (Module 18) 21
4.2.9 Cosmetic Fill (Module 19) 21
4.2.10 Image Pixel Positions (Module 20) 21
4.2.11 Determine Land-Sea Flag (Module 21) 21
4.2.12 Cloud clearing (Module 22) 21
4.2.13 Browse Product Generation (Module 23) 21
4.2.14 Output GBTR Records (Module 24) 21
4.2.15 Breakpoints 21
4.2.16 Breakpoint Format tables 23
4.2.17 Cloud Test Points 31
4.3 Processing Considerations 38
4.3.1 Product Limits and Data Gaps 38
4.3.2 Logical Order of Processes 40
4.3.3 Data Buffering 40
4.3.4 Visible Channel Calibration Coefficients 40
4.3.5 Overview of Auxiliary Data Processing 41

5. .... MODULE DETAILS 43
5.1 Module Definition: Source Packet Quality Checks 43
5.1.1 Functional Description 43
5.1.2 Interface Definition 45
5.1.3 Algorithm definition 45
5.2 Unpacking the auxiliary data. 49
5.2.1 Functional description 49
5.2.2 Interface definition. 50
5.2.3 Algorithm definition. 51
5.3 Validation of unpacked auxiliary data. 51
5.3.1 Functional Description. 51
5.3.2 Interface definition. 52
5.3.3 Algorithm Definition: 53
5.4 Converting the unpacked auxiliary data to engineering units. 54
5.4.1 Functional description. 54
5.4.2 Interface Definition 56
5.4.3 Algorithm definition. 57
5.5 Validation of the converted auxiliary data. 63
5.5.1 Functional Description. 63
5.5.2 Interface Definition 65
5.5.3 Algorithm definition. 65
5.6 Module Definition: Science Data Processing. 72
5.6.1 Functional description. 72
5.6.2 Interface definition 73
5.6.3 Algorithm definition. 74
5.7 Module Definition: Infra-Red Channel Calibration. 77
5.7.1 Functional Description 77
5.7.2 Interface definition 79
5.7.3 Algorithm Definition 80
5.8 Module Definition: Visible Channel Calibration 93
5.8.1 Functional Description 93
5.8.2 Interface definition 93

5.8.3 Algorithm Description 95
5.9 Module Definition: Satellite Time Calibration 104
5.9.1 Functional Description 104
5.9.2 Interface Definition 104
5.9.3 Algorithm Definition 105
5.10 Module Definition: Generate Geolocation Grid 106
5.10.1 Functional Description 106
5.10.2 Interface Definition 106
5.10.3 Algorithm Definition 109
5.11 Module Definition: Predict ENVISAT Orbit 115
5.11.1 Functional Description 115
5.11.2 Interface Definition 116
5.11.3 Algorithm Definition 117
5.12 Module Definition: Geolocate Pixels 120
5.12.1 Functional Description 120
5.12.2 Interface Definition 120
5.12.3 Algorithm Definition 121
5.13 Module Definition: Calculate Pixel x and y coordinates 124
5.13.1 Functional Description 124
5.13.2 Interface Definition 126
5.13.3 Algorithm Definition 127
5.14 Module Definition: Interpolate Pixel Positions 133
5.14.1 Functional Description 133
5.14.2 Interface Definition 133
5.14.3 Algorithm Definition 134
5.15 Module Definition: Calculate Solar Angles 136
5.15.1 Functional Description 136
5.15.2 Interface Definition 136
5.15.3 Algorithm Definition 138
5.16 Module Definition: Topographic Correction 141
5.16.1 Functional Description 141
5.16.2 Interface Definition 142

5.16.3 Algorithm Definition 142
5.17 Module Definition: Signal Calibration 145
5.17.1 Functional Description 145
5.17.2 Interface Definition 145
5.17.3 Algorithm Definition 147
5.18 Module Definition: Regrid Pixels 149
5.18.1 Functional Description 149
5.18.2 Interface Definition 149
5.18.3 Algorithm Definition 151
5.19 Module Definition: Cosmetic Fill 162
5.19.1 Functional Description 162
5.19.2 Interface Definition 162
5.19.3 Algorithm Definition 163
5.20 Module Definition: Image Pixel Positions 168
5.20.1 Functional Description 168
5.20.2 Interface Definition 168
5.20.3 Algorithm Definition 168
5.21 Module Definition: Determine Land/Sea Flag 169
5.21.1 Functional Description 169
5.21.2 Interface Definition 170
5.21.3 Algorithm Definition 171
5.22 Module Definition: Cloud Clearing 172
5.22.1 Functional Description 172
5.22.2 Interface Definition 173
5.22.3 Algorithm Definition 178
5.23 Module Definition: Generate Browse Product 232
5.23.1 Functional Description 232
5.23.2 Interface definition 233
5.23.3 Algorithm Definition 234
5.24 Module Definition: Output GBTR Records 240
5.24.1 Functional Description 240
5.24.2 Interface Definition 240

5.24.3 Algorithm Definition 240

6. .... INTERNAL PARAMETER LIST 250

1. Purpose of Document

The current document defines the Level 1B Data Processing Model and the Parameter Data List for ENVISAT AATSR product GBTR, together with the BROWSE product.

2. INTRODUCTION

AATSR is an imaging radiometer whose design is derived from that of ATSR and ATSR-2 instruments on the ERS satellites. It is functionally equivalent to ATSR-2. It images the surface in thermal radiation in seven visible, near-infra-red and infra-red channels, and the scanning geometry is such that each arc of the swath is viewed twice at different angles of incidence; this permits a more accurate atmospheric correction to be made.

The optical system (common to all channels) defines an instantaneous field of view whose angular width is such that the instrument pixel at nadir is 1 km by 1 km. The line of sight executes a conical scan about an axis inclined to the vertical. The intersection of the scan on the surface is an ellipse that intersects the ground track at nadir and (at an incidence angle of about 45°) about 900 km forward of this. The scan duration is 150 ms, and continuous imaging permits the image to be built up from successive scans.

Each channel is sampled 2000 times around the scan. Simultaneous sampling of all channels ensures exact co-location of the channels. However not all the sampled pixels are down-linked, since many are viewing the instrument housing. Apertures define the forward and nadir visible fields. Internal calibration targets are placed in other portions of the scan so as to be visible during the scan. Of the 2000 pixels sampled, 974, representing significant portions of the scan, are down-linked in the instrument telemetry. The data from a single scan are packed into a single instrument source packet.

The Level 1b product comprises calibrated images of brightness temperature (for the three infra-red channels) or reflectance (for the near-visible and visible channels), together with cloud identification. The level 1b product will be used as the starting point of scientific processing to produce the Level 2 product. The processes required to generate the Level 1b product are as follows.

First, source packets are processed to unpack and validate the science and auxiliary data that they contain. Calibration coefficients for the instrument channels are derived using the data from the on-board calibration targets and auxiliary temperature data.

Signal calibration uses these calibration coefficients to convert the science data in each channel to units of brightness temperature or reflectance, as appropriate.

Geolocation makes use of orbit propagation software in conjunction with available satellite orbit state vectors to determine the position on the Earth's surface of each instrument pixel.

Because the scans are curved, it is necessary to re-sample the data to rectify the images. This is done by assimilating each pixel of the instrument scan onto the nearest point of a Cartesian grid, using the pixel co-ordinates derived at the Geolocation stage. The same grid is used for both the nadir and forward view images, and therefore the collocation of the two images is ensured. This process may lead to gaps in the image, particularly in the forward view, where the density of instrument pixels is lower than the density of point on the Cartesian grid. A process of cosmetic filling of these gaps is therefore applied.

Finally, land flagging and cloud clearing algorithms are applied to the images to distinguish pixels over land from those over sea, and to identify those regions of the image that contain cloud.

3. Reference Documents And Applicable Documents

Reference	Title	Document Number or Author	Source
RD 1	Test and Calibration of ATSR-2 Final Report. Issue 2.21 December 1993	ER-RP-OXF-AT-2001	Oxford University DAOPP
AD 1	ENVISAT-1 reference definitions document for mission related software	PO-TN-ESA-GS-00361	
AD 2	ENVISAT-1 Mission CFI Software: General Software User Manual	PO-IS-GMV-GS-0556	ESA
AD 3	ENVISAT-1 Mission CFI Software: PPF_LIB Software User Manual	PO-IS-GMV-GS-0557	ESA
AD 4	ENVISAT-1 Mission CFI Software: PPF_ORBIT Software User Manual	PO-IS-GMV-GS-0558	ESA
AD 5	ENVISAT-1 Mission CFI Software: PPF_POINTING Software User Manual	PO-IS-GMV-GS-0559	ESA
AD 6	AATSR Telemetry Definition. Issue 3 August 1995	PO-TN-MMB-AT-0025	
AD 7	Instrument Measurement Data Definition. Issue 1. December 1995	PO-TN-MMB-AT-38	
AD 8	AATSR Telemetry Specification	PO-IS-BAE-AT-0006	
AD 9	Numerical Recipes in C. 2nd edition 1992	N/A	Cambridge University Press
AD10	Input/Output Data Definition	PO-RS-RAL-GS-10003	
AD11	ENVISAT-1 Products Specification	PO-RS-MDA-GS-2009	ESA
AD11	AATSR Operational Processor: Test Definition and Procedures Document	PO-TN-RAL-GS-10007	RAL

4. DETAILED PROCESSING MODEL

This section describes the level 1 processing. It includes a module by module breakdown of the processing structure. First a general overview of the processing structure can be found (See also Figure 4-1-1.). This is followed by a more detailed description of each of the component modules; each module having:

- A functional description
- An interface definition
- An algorithm definition or detailed structure description

The interface definition consists of two tables, an Input table and an Internal table. The first deals principally with the interface to the IODD and to external files defined within the ENVISAT processing environment. The second defines parameters which are “internal to the processor”, that is to say parameters which are defined within a particular module, used by one or more modules. The Input table can contain parameters defined either in the IODD, or another external file - it represents the interface with the “outside world”. The Internal table contains newly defined parameters or parameters defined in a previously declared Internal table, which the module under discussion needs to access. Internal parameters can be local (to the module), or global. Global parameters are available to other modules, including the product output module. All global internal parameters are summarized in Table 6-1: Summary List of Internal Parameter.

Parameter names are defined using the following conventions:

IODD parameters: format: <ProductCode>-<Data set>-<ID number> (e.g. L0-MDS1-1)

Internal parameters format (global variables): <DPM>-INT-<SEQ> (e.g. L1B-INT-1), where SEQ is a unique number for any given DPM level.

The data tables have the following columns:

Parameter ID: refers to the ID in the IODD (for external parameters) or the internal data ID for internally generated parameters. These are also used within pseudo code and text descriptions to refer to the parameters. In the case of internal parameters, if this field is blank, or contains “(local)”, the variable is taken to be local to the module. For example, loop counters would generally be defined as local variables. If an Internal ID is defined, the variable is assumed to be global.

Variable: This is an optional entry, allowing a parameter to also be referred to within the text through a variable name, for ease of use, or to follow convention. If the column is blank, the parameter will always be referred to by its ID;

Name: A “long name” format, providing the means for a short description. This can be used for reference in text descriptions, but generally is not used within pseudo code for reasons of clarity.

Type: The parameter type, using standard ENVISAT PDS conventions;

Units: The parameter SI unit (where appropriate), and if necessary scaling factor;

Size: The space required by the parameter in bytes;

Fields: The number of fields for cases where the ID refers to an array of parameters.

4.1 Conventions

Underscore is regarded as an alphabetic character in variable names.

Type font and style are of no significance; for example, the same variable is meant whether the name appears in italic or roman type, or in a different font.

Type of brackets is of no significance; either parentheses or square brackets may be used equivalently. Parameter IDs used in equations are generally enclosed in square brackets, to enable them to be subscripted.

Pointed brackets <> are (except for a few points in the Level 2 processing) metasyntactical; they enclose strings that are to be substituted by one of a set of optional strings to give the true variable name. For example, the construction <view>_fill_state(i, j) is to represent one of the two quantities nadir_fill_state(i, j) or frwrd_fill_state(i, j), according as whether the nadir or forward view data is being processed.

Indices in equations may appear indifferently as subscripts or enclosed in brackets. Sometimes the convention of separating parenthesised indices with semicolons is used: e.g. I(ch, v; i, j). The significance of this is that the indices preceding the semicolon are regarded as subscripts that may be thought of as part of the variable name (and therefore need not correspond to variables in an implementation) while those following the semicolon are array indices.

In general this document aims to provide a mathematical model of the processing, without referring to specific details of any particular implementation. However, at various points it is necessary to incorporate the functionality of the ESA CFI software subroutines in the processing model, and in these cases it is necessary to include details of the interface in the from of specific examples of subroutine calls. The ESA CFI software may be called from either C or FORTRAN programs, and array indexing will differ in these cases. Following AD2 - AD5 we give the subroutine calls in the form appropriate for C language calls; however when referring to specific elements of array parameters we adopt the following nmore general notation. If array is an array parameter of a CFI subroutine, the notation array[i]/(i + 1) will denote that element of array which would be indexed by i in a C language implementation, but by (i + 1) in a FORTRAN implementation. This would of course be the (i + 1)th parameter of the array. If such a parameter is a part of a more complex data structure, so that it is, explicitly or implicitly, further indexed, the notation (for example)

[mjdp[0]/(1)](s)

would be used. Here s is the additional index.

This notation may appear cumbersome, but it has the additional merit that it enables the distinction between single elements of an array and the array as a whole to be shown in the tables. Thus if a parameter ID is associated in the tables with a single array element, the above notation mjdp[0]/(1) will be used, whereas if the notation mjdp[2] is used, the complete array is meant. In the latter case the figure in square brackets is the dimension of the array.

Indexing

The following indexing conventions are adopted generally:

s instrument scan number (equivalent to source packet number)

p scan pixel number (p = 1, 2000)

After regridding (Module 18) we have

i along track (image scan) index

j across track (image pixel) index (j = 0, 511)

Unless otherwise stated, indices start at zero.

Note: to avoid cluttering the notation the dependence on the scan index s of the source packet contents and of the unpacked data from the source packet is not explicitly shown in all or part of modules 1 to 8 inclusive. It is essential to recall that these modules operate separately on all the source packets that are present in memory at one time.

For the purpose of indexing and identifying the AATSR channels, the following conventional numbering scheme will be adopted.

AATSR Channel	Symbol	Index (ch)
12 micron	ir12	1
11 micron	ir11	2
3.7 micron	ir37	3
1.6 micron	v16	4
0.870 micron	v870	5
0.670 micron	v670	6
0.55 micron	v555	7

Requirements are identified by numbers of the form (Req. <id>-<sequence>) where <id> is an identifier that is unique to the module or chapter, and <sequence> is the sequence number within the series identified by <id>.

4.2 Overview of Processing Structure

4.2.1 General

Figure 4-2-1 shows an overview of the Level 1b processing.

The AATSR Level 1b Operational Processor (‘the processor’) will operate on segments of AATSR Level 0 data of duration up to slightly more than one orbit. For simplicity during the present discussion we will assume that the data presented to the processor contains no data gaps. The treatment of data gaps will be discussed elsewhere in this document.

The Level 0 data product comprises a series of records presented in chronological sequence, each of which contains a single instrument source packet, and as we saw above, each source packet represents a single instrument scan. For the most part (with one significant exception), the data processing up to but not including the re-gridding stage treats each source packet independently. Thus each processing module can be regarded as a looping over a series of source packets, performing the same operations on (or in connection with) each. It is a matter for implementation how many source packets are processed at one time, before they are regridded, and therefore how large the source packet buffers need to be.

The exception noted above refers to the derivation of the channel calibration coefficients. In the case of the infra-red channels, the calibration coefficients are determined by averaging calibration target data over a number of consecutive scans; the time interval containing these scans will be termed a calibration period or calibration interval. The recommended duration of the calibration interval is expected to be 10 scans; thus a new calibration interval will start at the end of the previous interval, and will continue for 10 scans or until a change in the pixel selection map is detected, whichever is the earlier. If the scans are indexed by an integer s, and if s1 and s2 are the first and last scans of a calibration period, then s2 = s1 + 9 unless a psm change is detected; then the calibration coefficients derived from the interval are applied to all scans in s1 to s2 inclusive.

For the visible channels, the channel offsets are derived by averaging over a calibration interval as above. The channel gains, however, are to be determined once per orbit, by averaging over the block of scans for which the VISCAL target is visible.

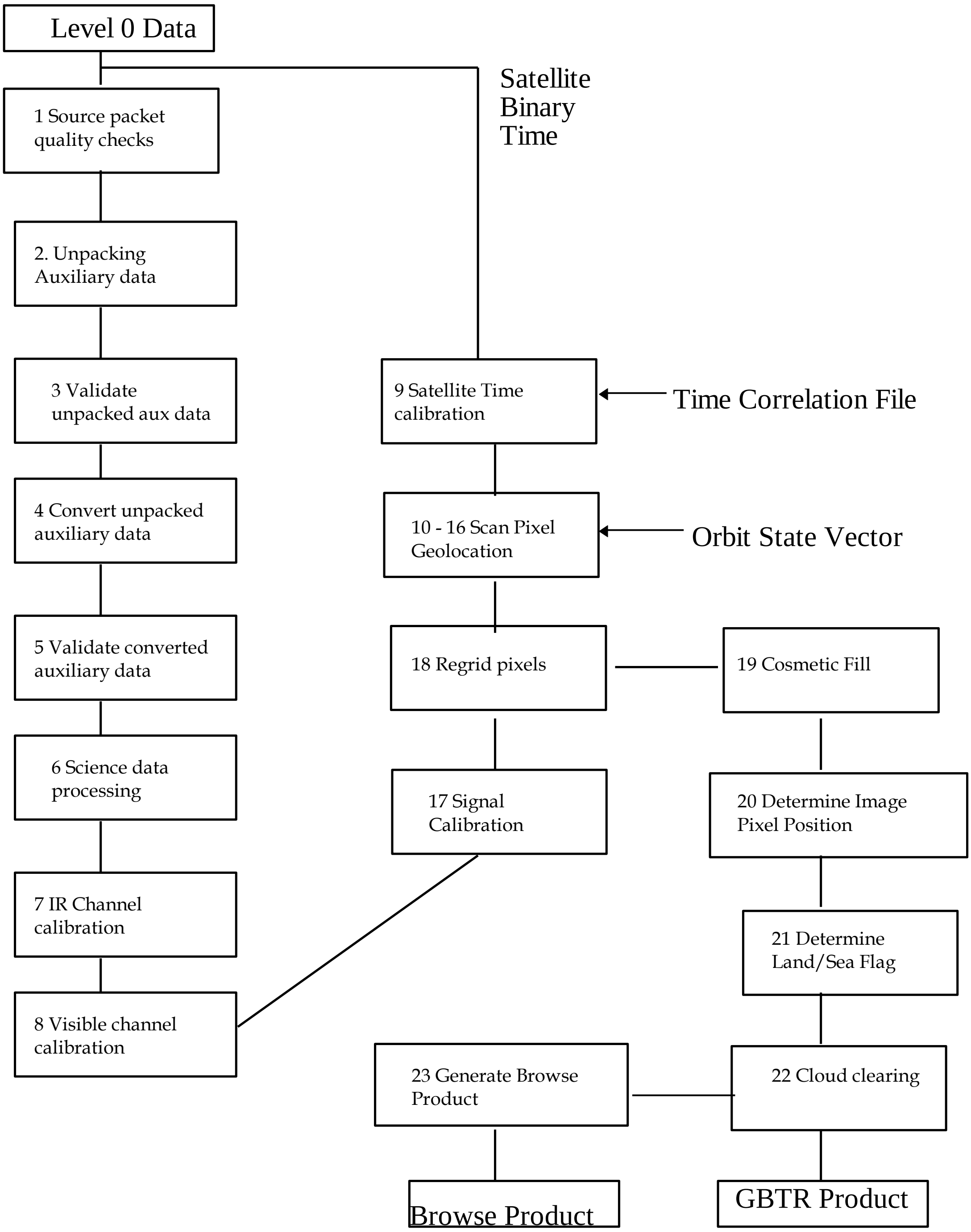


Figure 4-2-1. Level 1b processing.

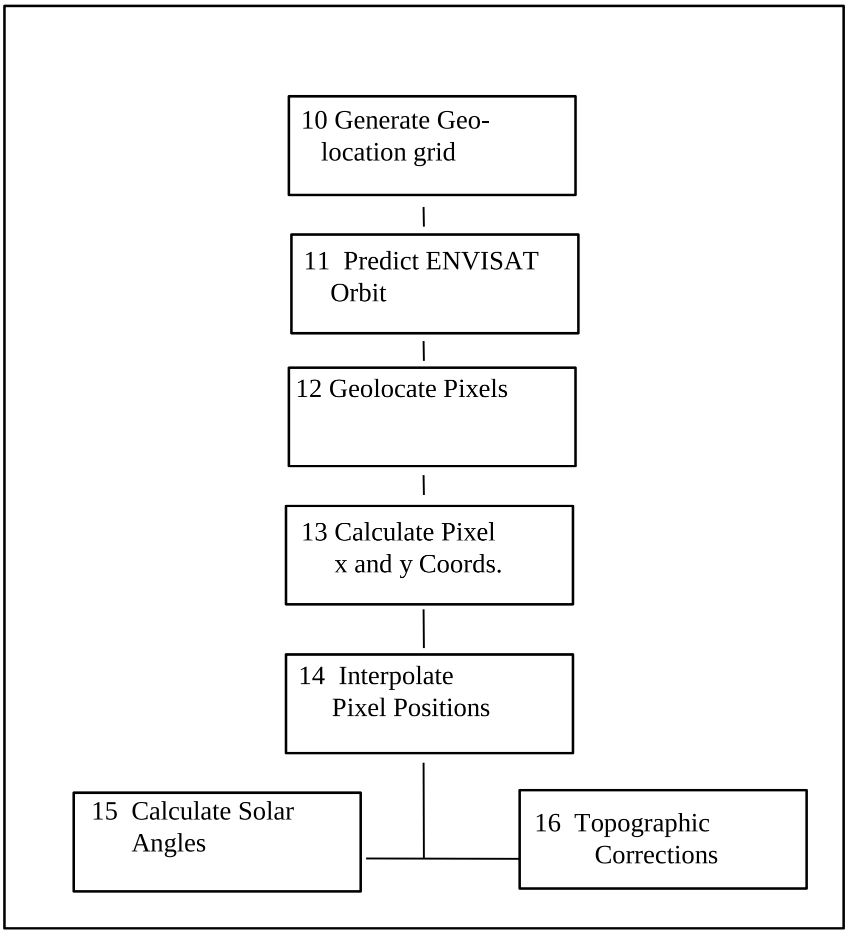


Figure 4-2-2. Scan Pixel Geolocation processing

4.2.2 Source packet Processing (Modules 1 - 6)

The purpose of the initial modules is to convert the data into the form most useful for higher processing levels which generate the products and geolocate the data. The main functions of these modules can be summarised as follows:

Module 1. To perform basic quality checks on each raw packet, ensuring that only those that pass the checks continue on to the calibration process.

Module 2. To unpack all the auxiliary and housekeeping data containing the temperatures of the on-board black bodies and instrument health and status information.

Module 3. To validate the unpacked auxiliary and housekeeping data items.

Module 4. To convert those items of auxiliary data containing the temperatures of the on-board black bodies and any other temperatures and data items that can be converted to engineering units.

Module 5. To extensively validate the converted auxiliary data items, especially those which are vital to the calibration.

Module 6. To routinely unpack and validate the science data containing the earth view and black body view pixel counts for all available channels from each packet.

The inputs, outputs, and relationship between each module are summarised in figures 4-2-1 and 4-2-2.

4.2.3 Infra-Red Channel calibration (Module 7)

This module calculates the calibration parameters (offset and slope) that describe the linear relationship between pixel count and radiance for the three IR channels, and for odd and even pixels, (odd and even pixels use different integrators). The parameters are determined from the black body pixels counts and the black body temperatures. This process makes use of look-up tables for the conversion of temperature to radiance.

4.2.4 Visible Channel calibration (Module 8)

To unpack the viscal data once per orbit (if present), when it is detected that the viscal unit is in sunlight. (The viscal data is used as a reference for the calculation of the calibration parameters for the visible channels.) In addition, to calculate calibration parameters for all visible channels. These calibration parameters are not used to calibrate the science data in the visible channels for the current orbit, but are are written to the Visible Calibration Coefficients ADS in the GBTR product.

4.2.5 Satellite Time Calibration (Module 9)

The satellite binary times associated with each AATSR source packet are calibrated using the ESA CFI software; input satellite clock times are converted to UTC. The conversion makes use of input parameters from the SBT calibration file.

4.2.6 Geolocation

Modules 10 to 16 relate to geolocation and related matters.

4.2.6.1 Generate Geolocation Grid (Module 10)

This module, which is called only once, generates look-up tables for use in the subsequent geolocation and image pixel co-ordinate determination modules. The tables comprise:

(1) Tables of the latitude, longitude, and y co-ordinate of a series of sub-satellite points; these points are equally spaced in time, at an interval corresponding to 1 product granule, and extending for sufficient time to cover the whole of the data to be processed. These points define the satellite ground track, for use in the calculation of the pixel x and y co-ordinates. This table is derived with the aid of the ENVISAT CFI orbit propagation subroutine ‘ppforb’.

(2) Tables of the latitudes and longitudes of a rectangular grid of points covering the satellite swath. These points are spaced by 25 km across-track, and at the granule interval defined above in the along-track direction. Thus the points in the centre of the column coincide with those described at (1) above, while there are 23 points in the across-track direction. Thus in the across-track direction the points extend 275 km on either side of the ground track. The co-ordinates of the image pixels for land flagging are derived by linear interpolation in this

table, and the table forms the basis of the grid pixel latitude and longitude ADS described in the I/O DD (AD 10).

4.2.6.2 Predict ENVISAT Orbit (Module 11)

This module uses the ENVISAT CFI orbit propagation subroutine ‘ppforb’ in conjunction with a supplied orbit state vector, to determine the position and orientation of the ERS platform at the time of each scan.

4.2.6.3 Geolocate Pixels (Module 12)

Geolocation is performed on selected tie point pixels of each scan. The direction of the line of sight to each tie point pixel is calculated, and used as input to the ENVISAT CFI ‘target’ subroutine, along with the platform parameters derived in Module 11. The subroutine ‘target’ then determines the geodetic latitude and the longitude, on the reference ellipsoid, of the pixel.

4.2.6.4 Calculate Pixel x and y co-ordinates (Module 13)

The x-y (across-track and along-track) coordinates of each tie point pixel are derived from the pixel latitude and longitude.

4.2.6.5 Interpolate Pixel Positions (Module 14)

Positions of other instrument pixels will be determined by linear interpolation between the tie points.

4.2.6.6 Calculate Solar Angles (Module 15)

Solar and viewing angles required for cloud clearing are also determined at this stage. The angles are calculated at a series of tie points around the scan at increments of 50 km in x.

4.2.6.7 Calculate Topographic Corrections (Module 16)

For those scan pixels that coincide with tie points for which topographic corrections are required, and that are over land, the topographic height is determined from a digital terrain model and topographic corrections to the latitude and longitude are calculated.

4.2.7 Signal Calibration (Module 17)

Uncalibrated scan pixels are calibrated using calibration coefficients derived earlier. Pixel calibration uses these calibration coefficients to convert the pixel data to brightness temperature, in the case of the infra-red channels, or to reflectance in the case of the visible and near-visible channels.

In the case of the infra-red channels, this process makes use of look-up tables for the conversion of radiance to brightness temperature. Thus for each pixel and channel, the detector count from the source packet is converted to a radiance using the linear calibration law, and the resulting radiance is used to enter a look-up table from which the brightness temperature of the pixel is derived.

In the case of the visible channels, the linear calibration law gives a calibrated reflectance directly. Calibration parameters to be used by the current module for the visible channels are read from an input file.

4.2.8 Regrid Pixels (Module 18)

Calibrated AATSR pixels are regridded into co-located forward and nadir images, onto a 1 km grid (modified to allow for equal time sampling along-track), using the pixel positions derived above.

4.2.9 Cosmetic Fill (Module 19)

Cosmetic filling of nadir/forward view images is performed, to fill remaining image pixels.

4.2.10 Image Pixel Positions (Module 20)

Linear interpolation is performed to determine the latitude and longitude coordinates of the grid pixels for use in the subsequent stages.

4.2.11 Determine Land-Sea Flag (Module 21)

Given the image pixel latitude and longitude, the surface type for each image pixel is derived using the land/sea flagging algorithm.

4.2.12 Cloud clearing (Module 22)

The cloud-clearing algorithms are used to identify all image pixels as cloudy or cloud-free. Up to nine independent tests, depending on whether the image is in day or night time and using different channel combinations, are applied to each pixel.

4.2.13 Browse Product Generation (Module 23)

The image channels will also be combined to derive the BROWSE product, at 4 by 4 km resolution. The generation of the BROWSE product may logically be regarded as part of the Level 1B processing, or as a separate stage in which the Browse product is derived from the Level 1B product.

4.2.14 Output GBTR Records (Module 24)

All data required for the Level 1B (GBTR) product is now available, and is formatted into the products described in the IODD.

4.2.15 Breakpoints

The following data shall be used as breakpoints in the testing of the Level 1b process. The final column indicates the accuracy with which the data should be verified against the output of the reference processor.

Parameter ID	Name	Verification Accuracy
From Modules 1 to 5:		
L1B-INT-003	auxilary_data_validation_result[i]	exact
L1B-INT-004	converted_auxiliary_data[i]	dependent upon datum type
L1B-INT-005	unpacked_auxiliary_data[i]	exact
From Module 6 (Science Data Processing)		
L1B-INT-080	unpacked.pixels.nadir[MAX_NADIR_PIXELS]	exact

L1B-INT-081	unpacked.pixels.forward[MAX_FORWARD_PIXELS]	exact
L1B-INT-082	unpacked.pixels.plus_bb[MAX_PXBB_PIXELS]	exact
L1B-INT-083	unpacked.pixels.minus_bb[MAX_MXBB_PIXELS]	exact
L1B-INT-084	unpacked.pixels.viscal[MAX_VISCAL_PIXELS]	exact
L1B-INT-091	unpacked_blanking_nadir[MAX_NADIR_PIXELS]	exact
L1B-INT-092	unpacked_blanking_forward[MAX_FORWARD_PIXELS]	exact
L1B-INT-093	unpacked_blanking_plus_bb[MAX_PXBB_PIXELS]	exact
L1B-INT-094	unpacked[32].blanking.minus_bb[MAX_MXBB_PIXELS]	exact
L1B-INT-095	unpacked[32].blanking.viscal[MAX_VISCAL_PIXELS]	exact
From Module 7 (IR Channel Calibration)		
L1B-INT-006	calibration_invalid[channel]	exact
L1B-INT-010	gain[parity][channel] (channel[.] slope)	1 part in 1e6
L1B-INT-011	offset[parity][channel] (channel[.]intercept)	1 part in 1e6
From Module 8 (Visible Channel Calibration)		
L1B-INT-410 - 438	Visible Calibration Product Parameters	dependent upon datum type
From module 9 (Satellite Time Calibration)		
L1B-INT-400	source_packet_ut_time(s)	1e-12 days (1 microsecond)
From Modules 12 (Geolocate Pixels), 14 (Interpolate Pixel Positions)		
L1B-INT-060	nadir scan pixel latitude	1 part in 1e6
L1B-INT-061	nadir scan pixel longitude	1 part in 1e6
L1B-INT-062	forward scan pixel latitude	1 part in 1e6
L1B-INT-063	forward scan pixel longitude	1 part in 1e6
From Modules 13 (Calculate Pixel x and y co-ordinates), 14 (Interpolate Pixel Positions)		
L1B-INT-064	nadir scan x coordinate (source packet nadir pixel x coords)	1 m
L1B-INT-065	nadir scan y coordinate (source packet nadir pixel y coords)	10 m
L1B-INT-066	forward scan x coordinate (source packet forward pixel x coords)	1 m
L1B-INT-067	forward scan y coordinate (source packet forward pixel y coords)	10 m
From Module 17 (Signal Calibration)		
L1B-INT-087	calibrated.pixels.nadir[MAX_NADIR_PIXELS], infra-red channels	0.01K
L1B-INT-088	calibrated.pixels.forward[MAX_FORWARD_PIXELS], infra-red channels	0.01K
L1B-INT-089	calibrated.pixels.nadir[MAX_NADIR_PIXELS], visible channels	0.01%
L1B-INT-090	calibrated.pixels.forward[MAX_FORWARD_PIXELS], visible channels	0.01%
From Module 15 (Solar and Viewing Angles)		
L1B-INT-120 - 127	nadir_band_edge_<solar and viewing angles>	1 part in 1e6
L1B-INT-140 - 147	frwrd_band_edge_<solar and viewing angles>	1 part in 1e6
From Module 18 (Regrid Pixels).		
L1B-INT-101	regridded nadir ir12 Brightness Temp.	0.01K
L1B-INT-102	regridded nadir ir11 Brightness Temp.	0.01K
L1B-INT-103	regridded nadir ir37 Brightness Temp.	0.01K
L1B-INT-104	regridded nadir v16 Reflectance	0.01%
L1B-INT-105	regridded nadir v870 Reflectance	0.01%
L1B-INT-106	regridded nadir v670 Reflectance	0.01%
L1B-INT-107	regridded nadir v555 Reflectance	0.01%
L1B-INT-108	nadir_x_offset(i, j)	5 m
L1B-INT-109	nadir_y_offset(i, j)	5 m
L1B-INT-111	regridded forward ir12 Brightness Temp.	0.01K
L1B-INT-112	regridded forward ir11 Brightness Temp.	0.01K
L1B-INT-113	regridded forward ir37 Brightness Temp.	0.01K
L1B-INT-114	regridded forward v16 Reflectance	0.01%
L1B-INT-115	regridded forward v870 Reflectance	0.01%
L1B-INT-116	regridded forward v670 Reflectance	0.01%
L1B-INT-117	regridded forward v555 Reflectance	0.01%
L1B-INT-118	frwrd_x_offset(i, j)	5 m
L1B-INT-119	frwrd_y_offset(i, j)	5 m
L1B-INT-134	nadir view instrument scan number	Generally exact (See note)
L1B-INT-135	nadir view instrument pixel number	Generally exact (See note)
L1B-INT-154	forward view instrument scan number	Generally exact (See note)
L1B-INT-155	forward view instrument pixel number	Generally exact (See note)
From module 19 (Cosmetic Fill)		
L1B-INT-100	nadir_fill_state(i, j)	Generally exact (See note)

L1B-INT-110	frwrd_fill_state(i, j)	Generally exact (See note)
From module 20 (Determine Image Pixel Positions)		
L1B-INT-160	image_latitude(i, j)	1 part in 1e6
L1B-INT-161	image_longitude(i, j)	1 part in 1e6
From Module 21 (Determine Land/Sea Flag)		
L1B-INT-232	.nadir land flag	Generally exact (See note)
L1B-INT-248	.frwrd land flag	Generally exact (See note)
From module 22 (Cloud Clearing)		
L1B-INT-233 - 244	nadir cloud flags	Generally exact See note
L1B-INT-249 - 260	frwrd cloud flags	Generally exact See note

Table 4-2-0: Breakpoints

Note: In the table above, ‘Generally exact’ relates to flags or quantities of type integer, and indicates that test results should agree exactly with the reference processor in the majority of cases, but that a small number (TBD) of discepancies may acceptable owing to differences in machine precision.

Similar verification accuracies apply to the GBTR Product Measurement Data sets. The relevant verification accuracies are:

Brightness temperature parameters	0.01 K
Visible Channel reflectances	0.01 %
Confidence and Cloud/land flags	Generally exact
GBTR ADS Quantities	As corresponding quantities in Table 4-2-0.

4.2.16 Breakpoint Format tables

The following tables describe the formats specified for the breakpoint outputs.

Table 4-2-1: Break Point #1 Record: unpacked auxiliary data

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX25-1	12	13	aux_tot	int	n/a	2	1
L1B-INT-005	14	13+(2*aux_tot)	unpacked_auxiliary_data[i]	ss	n/a	2	aux_tot

Table 4-2-2: Break Point #2 Record: converted auxiliary data

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX25-1	12	13	aux_tot	int	n/a	2	1
L1B-INT-004	14	13+(4*aux_tot)	converted_auxiliary_data[i]	fl	n/a	4	aux_tot

Table 4-2-3: Break Point #3 Record: auxiliary data validation results

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX25-1	12	13	aux_tot	int	n/a	2	1
L1B-INT-003	14	13+(2*aux_tot)	auxiliary_data_validation_result[i]	us	n/a	2	aux_tot

Table 4-2-4: Break Point #4 Record: unpacked nadir pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-5	12	13	MAX_NADIR_PIXELS	const	n/a	2	1
L1B-INT-080	14	1163	unpacked.pixels.nadir[i] (channel 1)	ss	counts	2	575
L1B-INT-080	1164	2313	unpacked.pixels.nadir[i] (channel 2)	ss	counts	2	575
L1B-INT-080	2314	3463	unpacked.pixels.nadir[i] (channel 3)	ss	counts	2	575
L1B-INT-080	3464	4614	unpacked.pixels.nadir[i] (channel 4)	ss	counts	2	575
L1B-INT-080	4613	5763	unpacked.pixels.nadir[i] (channel 5)	ss	counts	2	575
L1B-INT-080	5764	6913	unpacked.pixels.nadir[i] (channel 6)	ss	counts	2	575
L1B-INT-080	6914	8063	unpacked.pixels.nadir[i] (channel 7)	ss	counts	2	575

Table 4-2-5: Break Point #5 Record: unpacked forward pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-7	12	13	MAX_FORWARD_PIXELS	const	n/a	2	1
L1B-INT-081	14	795	unpacked.pixels.forward[i] (channel 1)	ss	counts	2	391
L1B-INT-081	796	1577	unpacked.pixels.forward[i] (channel 2)	ss	counts	2	391
L1B-INT-081	1578	2359	unpacked.pixels.forward[i] (channel 3)	ss	counts	2	391
L1B-INT-081	2360	3141	unpacked pixels.forward[i] (channel 4)	ss	counts	2	391
L1B-INT-081	3142	3923	unpacked.pixels.forward[i] (channel 5)	ss	counts	2	391
L1B-INT-081	3924	4705	unpacked.pixels.forward[i] (channel 6)	ss	counts	2	391
L1B-INT-081	4706	5487	unpacked.pixels.forward[i] (channel 7)	ss	counts	2	391

Table 4-2-6: Break Point #6 Record: unpacked plus BB pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-6	12	13	MAX_PXBB_PIXELS	const	n/a	2	1
L1B-INT-082	14	85	unpacked.pixels.plus_bb[i] (channel 1)	ss	counts	2	36
L1B-INT-082	86	157	unpacked.pixels.plus_bb[i] (channel 2)	ss	counts	2	36
L1B-INT-082	158	229	unpacked.pixels.plus_bb[i] (channel 3)	ss	counts	2	36
L1B-INT-082	230	301	unpacked.pixels.plus_bb[i] (channel 4)	ss	counts	2	36
L1B-INT-082	302	373	unpacked.pixels.plus_bb[i] (channel 5)	ss	counts	2	36
L1B-INT-082	374	445	unpacked.pixels.plus_bb[i] (channel 6)	ss	counts	2	36
L1B-INT-082	446	517	unpacked.pixels.plus_bb[i] (channel 7)	ss	counts	2	36

Table 4-2-7: Break Point #7 Record: unpacked minus BB pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-8	12	13	MAX_MXBB_PIXELS	const	n/a	2	1
L1B-INT-083	14	85	unpacked.pixels.minus_bb[i] (channel 1)	ss	counts	2	36
L1B-INT-083	86	157	unpacked.pixels.minus_bb[i] (channel 2)	ss	counts	2	36
L1B-INT-083	158	229	unpacked.pixels.minus_bb[i] (channel 3)	ss	counts	2	36
L1B-INT-083	230	301	unpacked.pixels.minus_bb[i] (channel 4)	ss	counts	2	36
L1B-INT-083	302	373	unpacked.pixels.minus_bb[i] (channel 5)	ss	counts	2	36

L1B-INT-083	374	445	unpacked.pixels.minus_bb[i] (channel 6)	ss	counts	2	36
L1B-INT-083	446	517	unpacked.pixels.minus_bb[i] (channel 7)	ss	counts	2	36

Table 4-2-8: Break Point #8 Record: unpacked viscal pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-4	12	13	VISCAL_PIXELS	const	n/a	2	1
L1B-INT-084	14	85	unpacked.pixels.viscal[i] (Channel 1)	ss	counts	2	36
L1B-INT-084	86	157	unpacked.pixels.viscal[i] (Channel 2)	ss	counts	2	36
L1B-INT-084	158	229	unpacked.pixels.viscal[i] (Channel 3)	ss	counts	2	36
L1B-INT-084	230	301	unpacked.pixels.viscal[i] (Channel 4)	ss	counts	2	36
L1B-INT-084	302	373	unpacked.pixels.viscal[i] (Channel 5)	ss	counts	2	36
L1B-INT-084	374	445	unpacked.pixels.viscal[i] (Channel 6)	ss	counts	2	36
L1B-INT-084	446	517	unpacked.pixels.viscal[i] (Channel 7)	ss	counts	2	36

Table 4-2-9: Break Point #9 Record: unpacked nadir blanking pulse flag

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-5	12	13	MAX_NADIR_PIXELS	const	n/a	2	1
L1B-INT-091	14	1163	unpacked_blanking_nadir[i]	ss	counts	2	575

Table 4-2-10: Break Point #10 Record: unpacked forward blanking pulse flag

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-7	12	13	MAX_FORWARD_PIXELS	const	n/a	2	1
L1B-INT-092	14	795	unpacked_blanking_forward[i]	ss	counts	2	391

Table 4-2-11: Break Point #11 Record: unpacked plus BB blanking pulse flag

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-6	12	13	MAX_PXBB_PIXELS	const	n/a	2	1
L1B-INT-093	14	85	unpacked_blanking_plus_bb[i]	ss	counts	2	36

Table 4-2-12: Break Point #12 Record: unpacked minus BB blanking pulse flag

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-8	12	13	MAX_MXBB_PIXELS	const	n/a	2	1
L1B-INT-094	14	85	unpacked_blanking_minus_bb[i]	ss	counts	2	36

Table 4-2-13: Break Point #13 Record: unpacked viscal blanking pulse flag

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-4	12	13	MAX_VISCAL_PIXELS	const	n/a	2	1
L1B-INT-095	14	85	unpacked_blanking_viscal[i]	ss	counts	2	36

Table 4-2-14: Break Point #14 Record: IR calibration parameters

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1B-INT-006	12	13	calibration_invalid(channel 1)	ss	n/a	2	1
L1B-INT-006	14	15	calibration_invalid(channel 2)	ss	n/a	2	1
L1B-INT-006	16	17	calibration_invalid(channel 3)	ss	n/a	2	1
L1B-INT-010	18	25	slope[parity] (channel 1)	float	radiance/ count	4	2
L1B-INT-010	26	33	slope[parity] (channel 2)	float	radiance/ count	4	2
L1B-INT-010	34	41	slope[parity] (channel 3)	float	radiance/ count	4	2
L1B-INT-011	42	49	intercept[parity] (channel 1)	float	counts	4	2
L1B-INT-011	50	57	intercept[parity] (channel 2)	float	counts	4	2
L1B-INT-011	58	65	intercept[parity] (channel 3)	float	counts	4	2

Table 4-2-15: Break Point #15 Record: Visible calibration parameters

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1B-INT-006	12	13	calibration_invalid(channel 4)	ss	n/a	2	1
L1B-INT-006	14	15	calibration_invalid(channel 5)	ss	n/a	2	1
L1B-INT-006	16	17	calibration_invalid(channel 6)	ss	n/a	2	1
L1B-INT-006	18	19	calibration_invalid(channel 7)	ss	n/a	2	1
L1B-INT-010	20	27	slope[parity] (channel 4)	float	radiance/ count	4	2
L1B-INT-010	28	35	slope[parity] (channel 5)	float	radiance/ count	4	2
L1B-INT-010	36	43	slope[parity] (channel 6)	float	radiance/ count	4	2
L1B-INT-010	44	51	slope[parity] (channel 7)	float	radiance/ count	4	2
L1B-INT-011	52	59	intercept[parity] (channel 4)	float	counts	4	2
L1B-INT-011	60	67	intercept[parity] (channel 5)	float	counts	4	2
L1B-INT-011	68	75	intercept[parity] (channel 6)	float	counts	4	2
L1B-INT-011	76	83	intercept[parity] (channel 7)	float	counts	4	2

Table 4-2-16: Break Point #16 Record: visible calibration product parameters

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-401	0	7	average_UT_time	double	days	8	1
L1B-INT-070	8	11	relative scan number (currently not defined; the contents of this field are arbitrary	sl	n/a	4	1

L1B-INT-410	12	23	Time of cal in MJD format	sl,2*ul	MJD	12	1
L1B-INT-411	24	27	1.6 micron slope	float	n/a	4	1
L1B-INT-412	28	31	0.870 micron slope	float	n/a	4	1
L1B-INT-413	32	35	0.670 micron slope	float	n/a	4	1
L1B-INT-414	36	39	0.555 micron slope	float	n/a	4	1
L1B-INT-415	40	51	UTC at ANX, in MJD format	sl, 2*ul	MJD	12	1
L1B-INT-416	52	55	Average monitor count	float	n/a	4	1
L1B-INT-417	56	59	Standard deviation of Monitor count	float	n/a	4	1
L1B-INT-418	60	63	Solar irradiance (1.6)	float	n/a	4	1
L1B-INT-419	64	67	Solar irradiance (0.870)	float	n/a	4	1
L1B-INT-420	68	71	Solar irradiance (0.670)	float	n/a	4	1
L1B-INT-421	72	75	Solar irradiance (0.555)	float	n/a	4	1
L1B-INT-422	76	79	Average VISCAL Pixel counts (1.6)	float	n/a	4	1
L1B-INT-423	80	83	Average VISCAL Pixel counts (0.87)	float	n/a	4	1
L1B-INT-424	84	87	Average VISCAL Pixel counts (0.67)	float	n/a	4	1
L1B-INT-425	88	91	Average VISCAL Pixel counts (0.555)	float	n/a	4	1
L1B-INT-426	92	95	VISCAL Pixel noise (1.6)	float	n/a	4	1
L1B-INT-427	96	99	VISCAL Pixel noise (0.87)	float	n/a	4	1
L1B-INT-428	100	103	VISCAL Pixel noise (0.67)	float	n/a	4	1
L1B-INT-429	104	107	VISCAL Pixel noise (0.555)	float	n/a	4	1
L1B-INT-430	108	111	Average -XBB pixel counts (1.6)	float	n/a	4	1
L1B-INT-431	112	115	Average -XBB pixel counts (0.87)	float	n/a	4	1
L1B-INT-432	116	119	Average -XBB pixel counts (0.67)	float	n/a	4	1
L1B-INT-433	120	123	Average -XBB pixel counts (0.555)	float	n/a	4	1
L1B-INT-434	124	127	-XBB pixel noise (1.6)	float	n/a	4	1
L1B-INT-435	128	131	-XBB pixel noise (0.87)	float	n/a	4	1
L1B-INT-436	132	135	-XBB pixel noise (0.67)	float	n/a	4	1
L1B-INT-437	136	139	-XBB pixel noise (0.555)	float	n/a	4	1
L1B-INT-438	140	141	(Reserved for parity indicator)	ss	n/a	2	1

Table 4-2-17: Break Point #17 Record: nadir pixel geolocation parameters

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1B-INT-060	12	2311	nadir scan pixel latitude[i]	float	degree	4	575
L1B-INT-061	2312	4611	nadir scan pixel longitude[i]	float	degree	4	575

Table 4-2-18: Break Point #18 Record: forward pixel geolocation parameters

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1B-INT-062	12	1575	forward scan pixel latitude[i]	float	degree	4	391
L1B-INT-063	1576	3139	forward scan pixel longitude[i]	float	degree	4	391

Table 4-2-19: Break Point #19 Record: Nadir scan pixel coordinates

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1B-INT-064	12	2311	nadir scan x coordinate[i]	float	km	4	575
L1B-INT-065	2312	4611	nadir scan y coordinate[i]	float	km	4	575

Table 4-2-20: Break Point #20 Record: Forward scan pixel coordinates

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1B-INT-066	12	1575	forward scan x coordinate[i]	float	km	4	391
L1B-INT-067	1576	3139	forward scan y coordinate[i]	float	km	4	391

Table 4-2-21: Break Point #21 Record: calibrated ir nadir pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-5	12	13	MAX_NADIR_PIXELS	const	n/a	2	1
L1B-INT-087	14	1163	calibrated.pixels.nadir[pixel] (channel 1)	ss	K/100	2	575
L1B-INT-087	1164	2313	calibrated.pixels.nadir[pixel] (channel 2)	ss	K/100	2	575
L1B-INT-087	2314	3463	calibrated.pixels.nadir[pixel] (channel 3)	ss	K/100	2	575

Table 4-2-22: Break Point #22 Record: calibrated ir forward pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-7	12	13	MAX_FRWRD_PIXELS	const	n/a	2	1
L1B-INT-088	14	795	calibrated.pixels.forward[pixel] (channel 1)	ss	K/100	2	391
L1B-INT-088	796	1577	calibrated.pixels.forward[pixel] (channel 2)	ss	K/100	2	391
L1B-INT-088	1578	2360	calibrated.pixels.forward[pixel] (channel 3)	ss	K/100	2	391

Table 4-2-23: Break Point #23 Record: calibrated vis nadir pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-5	12	13	MAX_NADIR_PIXELS	const	n/a	2	1
L1B-INT-089	14	1163	calibrated.pixels.nadir[pixel] (channel 4)	ss	0.01%	2	575
L1B-INT-089	1164	2313	calibrated.pixels.nadir[pixel] (channel 5)	ss	0.01%	2	575
L1B-INT-089	2314	3463	calibrated.pixels.nadir[pixel] (channel 6)	ss	0.01%	2	575
L1B-INT-089	3464	4613	calibrated.pixels.nadir[pixel] (channel 7)	ss	0.01%	2	575

Table 4-2-24: Break Point #24 Record: calibrated vis forward pixels

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
L1B-INT-400	0	7	source_packet_ut_time(s)	double	days	8	1
L1B-INT-070	8	11	relative scan number	sl	n/a	4	1
L1-AUX18-7	12	13	MAX_FRWRD_PIXELS	const	n/a	2	1
L1B-INT-090	14	795	calibrated.pixels.forward[pixel] (channel 4)	ss	0.01%	2	391
L1B-INT-090	796	1577	calibrated.pixels.forward[pixel] (channel 5)	ss	0.01%	2	391
L1B-INT-090	1578	2359	calibrated.pixels.forward[pixel] (channel 6)	ss	0.01%	2	391
L1B-INT-090	2360	3141	calibrated.pixels.forward[pixel] (channel 7)	ss	0.01%	2	391

Table 4-2-25: Break Point #25 Record: nadir solar and viewing angles

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-120	4	47	nadir_band_edge_solar_elevation [k]	float	degrees	4	11
L1B-INT-121	48	95	nadir_band_edge_satellite_elevation [k]	float	degrees	4	11
L1B-INT-122	96	139	nadir_band_edge_solar_azimuth [k]	float	degrees	4	11
L1B-INT-123	140	183	nadir_band_edge_satellite azimuth [k]	float	degrees	4	11
L1B-INT-124	184	223	nadir_band_centre_solar_elevation [k']	float	degrees	4	10
L1B-INT-125	224	263	nadir_band_centre_satellite_elevation [k']	float	degrees	4	10
L1B-INT-126	264	303	nadir_band_centre_solar_azimuth[k']	float	degrees	4	10
L1B-INT-127	304	343	nadir_band_centre_satellite_azimuth[k']	float	degrees	4	10

Table 4-2-26: Break Point #26 Record: forward solar and viewing angles

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-140	4	47	frwrd_band_edge_solar_elevation [i, k]	float	degrees	4	11
L1B-INT-141	48	95	frwrd_band_edge_satellite_elevation [i, k]	float	degrees	4	11
L1B-INT-142	96	139	frwrd_band_edge_solar_azimuth [i, k]	float	degrees	4	11
L1B-INT-143	140	183	frwrd_band_edge_satellite azimuth [i, k]	float	degrees	4	11
L1B-INT-144	184	223	frwrd_band_centre_solar_elevation [i, k']	float	degrees	4	10
L1B-INT-145	224	263	frwrd_band_centre_satellite_elevation[i, k']	float	degrees	4	10
L1B-INT-146	264	303	frwrd_band_centre_solar_azimuth[i, k']	float	degrees	4	10
L1B-INT-147	304	343	frwrd_band_centre_satellite_azimuth[i, k']	float	degrees	4	10

Table 4-2-27: Break Point #27 Record: nadir view instrument pixel numbers

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i = NGRANULE * ig)	sl	n/a	4	1
L1B-INT-134	4	1027	nadir view instrument scan number (ig, j)	us	n/a	2	512
L1B-INT-135	1028	2051	nadir view instrument pixel number(ig, j)	us	n/a	2	512

Table 4-2-28: Break Point #28 Record: forward view instrument pixel numbers

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i = NGRANULE * ig)	sl	n/a	4	1
L1B-INT-154	4	785	forward view instrument scan number(ig, j)	us	n/a	2	391
L1B-INT-155	786	1567	forward view instrument pixel number(ig, j)	us	n/a	2	391

Table 4-2-29: Break Point #29 Record: regridded ir 12 brightness temperature

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-101	4	1027	regridded nadir ir12 Brightness Temp(i, j)	ss	K/100	2	512
L1B-INT-111	1028	2051	regridded forward ir12 Brightness Temp(i, j)	ss	K/100	2	512

Table 4-2-30: Break Point #30 Record: regridded ir 11 brightness temperature

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-102	4	1027	regridded nadir ir11 Brightness Temp(i, j)	ss	K/100	2	512
L1B-INT-112	1028	2051	regridded forward ir11 Brightness Temp(i, j)	ss	K/100	2	512

Table 4-2-31: Break Point #31 Record: regridded ir 37 brightness temperature

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-103	4	1027	regridded nadir ir37 Brightness Temp(i, j)	ss	K/100	2	512
L1B-INT-113	1028	2051	regridded forward ir37 Brightness Temp(i, j)	ss	K/100	2	512

Table 4-2-32: Break Point #32 Record: regridded v16 reflectance

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-104	4	1027	regridded nadir v16 Reflectance(i, j)	ss	0.01%	2	512
L1B-INT-114	1028	2051	regridded forward_v16 Reflectance(i, j)	ss	0.01%	2	512

Table 4-2-33: Break Point #33 Record: regridded v870 reflectance

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-105	4	1027	regridded nadir v870 Reflectance(i, j)	ss	0.01%	2	512
L1B-INT-115	1028	2051	regridded forward_v870 Reflectance(i, j)	ss	0.01%	2	512

Table 4-2-34: Break Point #34 Record: regridded v670 reflectance

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-106	4	1027	regridded nadir v670 Reflectance(i, j)	ss	0.01%	2	512
L1B-INT-116	1028	2051	regridded forward_v670 Reflectance(i, j)	ss	0.01%	2	512

Table 4-2-35: Break Point #35 Record: regridded v555 reflectance

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-107	4	1027	regridded nadir v555 Reflectance(i, j)	ss	0.01%	2	512
L1B-INT-117	1028	2051	regridded forward_v555 Reflectance(i, j)	ss	0.01%	2	512

Table 4-2-36: Break Point #36 Record: fill state

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields

none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-100	4	515	nadir_fill_state(i, j)	uc	n/a	1	512
L1B-INT-110	516	1027	frwrd_fill_state(i, j)	uc	n/a	1	512

Table 4-2-37: Break Point #37 Record: nadir xy offset

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-108	4	515	nadir_x_offset(i, j)	uc	m	1	512
L1B-INT-109	516	1027	nadir_y_offset(i, j)	uc	n/a	1	512

Table 4-2-38: Break Point #38 Record: forward xy offset

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-118	4	515	frwrd_x_offset(i, j)	uc	m	1	512
L1B-INT-119	516	1027	frwrd_y_offset(i, j)	uc	n/a	1	512

Table 4-2-39: Break Point #39 Record: image latitude/longitude

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-160	4	2051	image_latitude(i, j)	float	deg	4	512
L1B-INT-161	2052	4099	image_longitude(i, j)	float	deg	4	512

Table 4-2-40: Break Point #40 Record: land flags

Parameter ID	Start byte	End byte	Field Description	Type	Units	Field size	Fields
none	0	3	image row index (i)	sl	n/a	4	1
L1B-INT-232	4	1027	nadir land flag(i, j)	ss	n/a	2	512
L1B-INT-248	1028	2051	frwrd land flag(i, j)	ss	n/a	2	512

4.2.17 Cloud Test Points

In order to support the testing and verification of the complex cloud tests in the AATSR Operational Processor, additional test data are provided by the Reference Processor (RP). These test data are additional to the break points described in the preceding sections; they comprise intermediate cloud test points output from the RP, and relate to the following tests:

- 11 micron spatial coherence test
- 1.6 micron histogram / spatial coherence test
- infra-red histogram test

Test points are provided for these tests because they represent the more complex cloud clearing tests; the other tests, which examine individual pixels singly, are expected to present fewer difficulties of implementation. Detailed specifications of the tests appear in Section 5.22. This section specifies the content of the cloud test points.

Note that all three tests, the spatial coherence test and the infrared and 1.6 micron histogram tests, operate on image segments of 512 rows, and each file of cloud test points refers to a specific 512 row image segment. In order to limit the volume of data, the test points are only provided for a limited number of image segments. Further details will be found in the Test Definition and Procedures Document (Reference AD12).

The tables below show the values printed, with cross-referencing to the variable names and parameter identifiers used elsewhere in this document. Note that the order in which the test points are printed is not necessarily the same as that in which they appear in the tables. The test point data are output in an ASCII format, and values are annotated in such a way that the output should be self-explanatory.

It is not a requirement that the Operational Processor produce these outputs, only that the corresponding variables can be inspected with the aid of a debug utility, and compared with the test data, if need be.

4.2.17.1 Spatial Coherence Test (11 micron)

This test is applied to a complete image segment of 512 by 512 pixels. The test comprises three stages as follows.

1. The basic spatial coherence test is applied to groups of 3 by 3 pixels.
2. An algorithm to unset group cloud flags that may have been set in error at stage 1 is applied.
3. A large-scale coherence test is applied to sub-areas of 128 by 128 pixels.

Suitable test point data are generated during each stage. These are listed below.

4.2.17.1.1 Small scale spatial coherence test (to end of Step 5.22.3-5)

ASCII header: RESULTS OF STEPS 5.22.3-1 to 5.22.3-5

The contents of Table 4-2-41 are repeated 171 x 171 times, once for each group. (The group indices x_index, y_index both range from 0 to 170.)

Table 4-2-41. Test point data for small scale spatial coherence test.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-263		solar_elevation	float	deg	4	1
L1B-INT-264		ir11[3][3]	int	K/100	2	9
L1B-INT-265		group_land_flag[x_index][y_index]	flag	n/a	2	1
L1B-INT-266		average_11	double	K/100	8	1
L1B-INT-267		sigma_11	double	K/100	8	1
L1B-INT-269		n	int	n/a	2	1
L1B-INT-270		average_11_array[x_index][y_index]	int	K/100	2	1
L1B-INT-271		threshold_sd	int	K/100	2	1
L1B-INT-272		group_cloud_flag[x_index][y_index]	flag	n/a	2	1

4.2.17.1.2 Unflag Pixels (Steps 5.22.3-6, 7)

ASCII Header: RESULTS OF STEPS 5.22.3-6 to 5.22.3-7:

The contents of Table 4-2-42 are repeated for each unflagged group.

Table 4-2-42. Test point data for the Unflag Pixels stage: Completion of Steps 5.22.3-6, 7.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-273		average_11_12_dif_cloudy	double	K/100	8	
L1B-INT-274		average_11_12_dif_clear	double	K/100	8	
L1B-INT-272		group_cloud_flag[x_index][y_index]	flag	n/a	2	

4.2.17.1.3 Completion of Step 5.22.3-8 (set extended land flag)

ASCII Header: RESULTS OF STEP 5.22.3-8

The extended land flag is repeated for each group.

Table 4-2-43.Test Point Data: Extended land Flag.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-279		extended_land_flag[171][171]	flag	n/a	2	29241

4.2.17.1.4 Large scale spatial coherence test

ASCII Headers: See Table 4-2-44.

The test points shown in Table 4-2-44 are repeated for each of the 16 sub-areas indexed by x_index_1 = 0, 3, y_index_1 = 0, 3)

Table 4-2-44. Test point data for large scale spatial coherence test.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
	Test point: Completion of Step 5.22.3-10.					
ASCII Header	RESULTS OF STEP 5.22.3-10					
L1B-INT-280		sub_area_n	int	Km	2	1
L1B-INT-281		sub_area_max_11[x_index_1][y_index_1]	int	K/a	2	1
L1B-INT-283		sub_area_dif[x_index_1][y_index_1]	int	K/100	2	1
		Test point: Completion of Steps 5.22.3-11, 12.				
ASCII Header		RESULTS OF STEPS 5.22.3-11 and 5.22.3-12				
L1B-INT-284		land_sub_area[x_index_1][y_index_1]	flag	n/a	2	1
L1B-INT-298		valid_sub_area_flag[x_index_1][y_index_1]	flag	n/a	2	1
	Test point: Completion of Steps 5.22.3-13 (m.1 to m.7 inclusive).					
ASCII Header	RESULTS OF STEP 5.22.3-13					
L1B-INT-291		threshold_11	int	K/100	2	1
L1B-INT-293		land_in_areas	flag	n/a	2	1
L1B-INT-294		bt_dif_max	int	K/100	2	1
L1B-INT-295		difference_threshold	int	K/100	2	1
L1B-INT-297		lowest_max_bt	int	K/100	2	1

4.2.17.2 Histogram Test (1.6 micron)

ASCII Header (Complete test): HISTOGRAM TEST (1.6 MICRON):

This test is done only on day-time data, therefore validation should use a day-time image. The test operates separately on 32 by 32 pixel sub-areas, and involves essentially three cases as follows:

1. Sunglint. If sunglint is detected, a spatial coherence test is applied to the 1.6 micron data in the sub-area.

2. Near-glint. If the sub-area is close to a sunglint region but is not actually affected by sunglint, a 'detrending' process is applied to the data before the histogram test is applied.
3. Default. In the absence of sunglint or near-glint conditions, the basic histogram test is applied without any detrending of the data.

In addition, if certain conditions arise during the histogram tests (detrended or not), control passes to the spatial coherence test and it is applied instead. A more detailed breakdown of the non-sunglint cases is given in the table below.

Table 4-2-45. Alternative paths through the 1.6 micron cloud clearing algorithm.

Case:	Not Near-Glint	Near-Glint
The three conditions of Step 5.22.3-27, c.4) are all valid	Derive threshold from simple histogram	Detrend data and attempt to derive threshold from detrended histogram
The three conditions of Step 5.22.3-27, c.4) are not all valid	Set all pixels in area cloudy	Detrend data and attempt to derive threshold from detrended histogram

4.2.17.2.1 Tilt angle calculations and sunglint test

ASCII Headers: See Table 4-2-46.

Test point data from the tilt angle calculations is repeated for each row index = 16, 496, 32 and for across-track position = 0, 15. Sunglint flagging is repeated for each subgroup of 32 by 32 pixels (16 x 16 times). The test point data printed is listed in Table 4-2-46.

Table 4-2-46. Test point data at the completion of the tilt angle calculations.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
	Completion of Step 5.22.3-30					
ASCII Header	RESULTS OF STEPS 5.22.3-30 (TILT ANGLE CALCULATIONS)					
L1B-INT-305	SolarElev	sol_elev	float	degree	4	1
L1B-INT-306	SatelElev	sat_elev	float	degree	4	1
L1B-INT-307	AzimuthDifference	azim_dif	float	degree	4	1
L1B-INT-308	i_vec	v_x/2	float	n/a	4	1
L1B-INT-309	j_vec	v_y/2	float	n/a	4	1
L1B-INT-310	k_vec	v_z/2	float	n/a	4	1
L1B-INT-311	TiltAtBands[11], TiltAngle[32][32]	tilt	float	degree	4	1
L1B-INT-312	magnitude	magnitude/2	float	n/a	4	1
	Test point: Completion of Step 5.22.3-32 (sunglint result)					
ASCII Header	TILT ANGLES AT 32KM INTERVALS AND RESULTS OF STEP 5.22.3-32					
L1B-INT-313	SunGlintResult	glint_present	flag	n/a	2	1

4.2.17.2.2 Creation of array of scaled reflectances and subsequent steps

ASCII Header: RESULTS OF STEP 5.22.3-33,5.22.3-36 and Req. 5.22-106:

Table 4-2-47 shows the test point data printed following the creation of array of scaled reflectances. This and the following sets of test points (if applicable) are repeated for each 32 by 32 image subarea (16 x 16 times). Note that parameters [L1B-INT-324] and [L1B-INT-330] will not be present if the histogram has too few valid pixels.

Table 4-2-47. Test point data at the completion of Step 5.22.3-33, Step 5.22.3-36 and Req. 5.22-106

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-301	Reflectance[32][32]	ir16[32][32]	float	%/100	2	1024
	InfraRed[32][32]					1024
L1B-INT-304	PixelCount	total_valid	int	n/a	2	1
L1B-INT-324	SpreadAdjusted	spread_adjusted	float	%	4	1
L1B-INT-330		n	float	%	4	1

4.2.17.2.3 Generation of histogram and derived parameters

ASCII Header (First histogram): HISTOGRAM GENERATED AT STEP 5.22.3-38:

ASCII Header (Following Detrend): HISTOGRAM GENERATED AT STEP 5.22.3-42.2:

The test point data listed in Table 4-2-48 will be printed if a histogram is generated.

Table 4-2-48. Test point data: Generation of histogram and derived parameters (Completion of Steps 5.22.3-39, 43, 52).

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-314	Histogram[1500]	histogram_16[1000]	int	n/a	2	1500
L1B-INT-315	LowInterval	low_interval	float	%	4	1
L1B-INT-316	HighInterval	high_interval	float	%	4	1
L1B-INT-317	HistRange	hist_range	float	%	4	1
L1B-INT-318	(PeakInterval)	peak_box_no	int	n/a	2	1
L1B-INT-319	PeakValue	peak_value	int	n/a	2	1
L1B-INT-320	AverageValue	average_value	float	n/a	4	1

4.2.17.2.4 Refinement of histogram derived parameters

ASCII Header: REFINEMENT OF HISTOGRAM DERIVED PARAMETERS:

The test point data listed in Table 4-2-49 will be printed when these parameters are derived following histogram generation.

Table 4-2-49. Test point data: Refinement of histogram derived parameters (Completion of Steps 5.22.3-40, 44, 53).

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-321	PeakInterval	peak_interval	float	%	4	
L1B-INT-322	Denominator	D	float	n/a	4	
L1B-INT-323	PeakDelta	delta	float	n/a	4	

4.2.17.2.5 Detrend Data

ASCII Header: RESULTS OF STEP 5.22.3-42.1 (Detrend data):

The test point data listed in Table 4-2-50 will be printed if a detrend stage is completed.

Table 4-2-50. Test point data: Detrend Data (Completion of Steps 5.22.3-42.1, 5.22.3-50)

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-331	SLOPE	gradient_16	float	%/Km/100	4	
L1B-INT-333	DELTA	b	float	n/a	4	

L1B-INT-335	Reflectance[32][32]	detrended_16[32][32]	int	%/100	2	

4.2.17.2.6 Reflectance threshold check

ASCII Header: REFLECTANCE THRESHOLD CHECK AT STEP 5.22.3-45:

The test point data listed in Table 4-2-51 is printed following the execution of step 5.22.3-45.

Table 4-2-51. Test point data: Reflectance threshold check at step 5.22.3-45.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-327		spread	float	%	4	
L1B-INT-328		peak_factor	float	%	4	
L1B-INT-329	ReflectanceThreshold	reflectance_threshold	float	%	4	

4.2.17.2.7 Spatial coherence test (1.6 micron)

ASCII Header: RESULTS OF STEP 5.22.3-60:

The test point data shown in Table 4-2-52 will be present if the spatial coherence test has been applied to the sub-area. (Note that this may follow directly from the data in Table 4-2-47 if the histogram test is omitted.)

Table 4-2-52. Test point data from the spatial coherence test (generation of SD threshold): completion of Steps 5.22.3-60

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
L1B-INT-336	SDThreshold	sd_threshold	float	%/100	4	
L1B-INT-337		x_y_12um_max[2]	int	n/a	2	
L1B-INT-338		reflectance_at_12um_max	float	%	4	

4.2.17.3 Histogram Test (11 micron)

This test is applied to a complete image segment of 512 by 512 pixels. The major stages of this test are as follows.

1. Compute major histogram;
2. Compute minor histogram;
3. Identify which histogram is valid, define threshold and set flags.

Test points are appropriately set at the end of each of these stages. Suitable test points, and quantities to test at the corresponding stage, are listed below.

ASCII Header (Complete test): HISTOGRAM TEST (11 micron):

4.2.17.3.1 Generation of major histogram

ASCII Header: RESULTS OF STEP 5.22.3-69 (generation of major histogram):

Table 4-2-53 shows the test point data following the generation of the major histogram.

Table 4-2-53. Test point data associated with the generation of major histogram.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
		End of Step 5.22.3-69				
ASCII Header		RESULTS OF STEP 5.22.3-69				
L1B-INT-342		valid_pixels	int	2	1	1
L1B-INT-343		histogram[MAJOR][1000]	int	2	2000	1000
L1B-INT-351		ir12boxtotal[1000]	int	K/100	4	
		Test point: End of Step 5.22.3-71.				
ASCII Header		RESULT OF STEP 5.22.3-71:				
L1B-INT-353		peak_value[MAJOR]	int	2	2	1
L1B-INT-354		peak_interval[MAJOR]	int	K/10	2	1
		Test point: End of Step 5.22.3-74				
ASCII Header		RESULTS OF STEP 5.22.3-74				
L1B-INT-360		low_limit[MAJOR]	int	K/10	2	1
L1B-INT-361		high_limit[MAJOR]	int	K/10	2	1
		Test point: End of Steps 5.22.3-78, 79.				
ASCII Header		RESULTS OF STEPS 5.22.3-78, 5.22.3-79:				
L1B-INT-371		peak_valid[MAJOR]	flag	2	2	1
L1B-INT-362		average_bt_mode[MAJOR]	float	K/100	4	1
		Test point: Competion of Steps 5.22.3-80, 81.				
ASCII Header		RESULTS OF STEPS 5.22.3-80, 5.22.3-81				
L1B-INT-372		nightime	flag	2	1	1
L1B-INT-355		exact_peak_value[MAJOR]	float	4	2	1
L1B-INT-356		exact_peak_interval[MAJOR]	float	K/10	4	1
L1B-INT-357		ir11_ir12_diff_at_peak[MAJOR]	float	K/100	4	1
		Test point: Completion of Steps 5.22.3-82, 83.				
ASCII Header						
L1B-INT-369		upperhalfwidth_index	float	K/10	4	1
L1B-INT-370		lowerhalfwidth_index	float	K/10	4	1
L1B-INT-368		half_width[MAJOR]	float	K/100	4	1
L1B-INT-367		half_width_threshold[MAJOR]	float	K/100	4	1

4.2.17.3.2 Generation of minor histogram

ASCII Header: RESULTS TO STEP 5.22.3-85 (generation of minor histogram):

Table 4-2-54 shows the test point data following the generation of the minor histogram.

Table 4-2-54. Test point data associated with the generation of minor histogram.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
		Completion of Step 5.22.3-85				
L1B-INT-342		valid_pixels	int	2	1	
L1B-INT-343		histogram[MINOR][1000]	int	2	2000	1000
L1B-INT-351		ir12boxtotal[1000]	int	K/100	4	1
L1B-INT-353		peak_value[MINOR]	int	2	2	1
L1B-INT-354		peak_interval[MINOR]	int	K/10	2	1
L1B-INT-360		low_limit[MINOR]	int	K/10	2	1
L1B-INT-361		high_limit[MINOR]	int	K/10	2	1
L1B-INT-371		peak_valid[MINOR]	flag	2	2	1
L1B-INT-362		average_bt_mode[MINOR]	float	K/100	4	1
L1B-INT-372		nightime	flag	2	1	1
L1B-INT-355		exact_peak_value[MINOR]	float	4	2	1
L1B-INT-356		exact_peak_interval[MINOR]	float	K/10	4	1
L1B-INT-357		ir11_ir12_diff_at_peak[MINOR]	float	K/100	4	1
L1B-INT-369		upperhalfwidth_index	float	K/10	4	1
L1B-INT-370		lowerhalfwidth_index	float	K/10	4	1
L1B-INT-368		half_width[MINOR]	float	K/100	4	1
L1B-INT-367		half_width_threshold[MINOR]	float	K/100	4	1

		Completion of Steps 5.22.3-86 to 98 inclusive				
L1B-INT-360		low_limit[MINOR]	int	K/10	2	1
L1B-INT-361		high_limit[MINOR]	int	K/10	2	1
L1B-INT-371		peak_valid[MINOR]	flag	2	2	1
L1B-INT-360		low_limit[MAJOR]	int	K/10	2	1
L1B-INT-361		high_limit[MAJOR]	int	K/10	2	1
L1B-INT-371		peak_valid[MAJOR]	flag	2	2	1

4.2.17.3.3 Following selection of chosen histogram

ASCII Headers: See Table 4-2-55.

The test point data following the specified steps are shown in Table 4-2-55.

Table 4-2-55. Test point data associated with the selection of the chosen histogram.

Parameter ID	Variable	Name / Description	Type	Units	Size	Fields
		Completion of Steps 5.22.3-86 to 98 inclusive				
L1B-INT-360		low_limit[MINOR]	int	K/10	2	1
L1B-INT-361		high_limit[MINOR]	int	K/10	2	1
L1B-INT-371		peak_valid[MINOR]	flag	2	2	1
L1B-INT-360		low_limit[MAJOR]	int	K/10	2	1
L1B-INT-361		high_limit[MAJOR]	int	K/10	2	1
L1B-INT-371		peak_valid[MAJOR]	flag	2	2	1
		Completion of Step 5.22.3-99				
ASCII Header		RESULT OF STEP 5.22.3-99:				
L1B-INT-347		CHOSEN	constant	2	1	1
		Completion of Step 5.22.3-106				
ASCII Header		RESULTS ON COMPLETION OF STEP 5.22.3-106				
L1B-INT-363		average_bt[1000]	float	K/100	4	1
L1B-INT-364		highest_av_bt_box[CHOSEN]	int	2	2	1
L1B-INT-365		slope_at_peak	float	4	1	1
L1B-INT-366		slope[1000]	float	4	1000	1000

4.3 Processing Considerations

4.3.1 Product Limits and Data Gaps

AATSR Level 1 processing will be controlled by a start time (start_of_product) and stop time (end_of_product) provided to the processor. Specifically, the first granule of the product will correspond to the start time, and the product will end at the end of the complete granule that includes the stop time. For consolidated products the start time is expected to correspond to an ascending node, and the product will extend for a complete orbit.

In order to provide the forward view images at the start of the product, it will be necessary to process source packets up to some 900 seconds before the start time. Thus the Level 0 Data Product supplied to the processor will (if the data is available) include source packets beginning at least K granules prior to the start time. (Note that K here refers to the parameter L1-AUX16-17 used in Module 10.)

Note that if any source packets precede in time the time t(0) = T0 - KΔT defined in Req 5.10-2 they should be ignored. Such packets fall outside the limits of the geolocation tables and so cannot be geolocated; they do not contribute to the output product.

The data presented to the AATSR processor must appear (at least logically) as a continuous stream of source packets with no data gaps; in particular this is necessary for the correct operation of the geolocation process and treatment of missing data in the regridding module.

Therefore any data gaps in the Level 0 Product must be padded with null source packets.

(Req.4.3-1)

A data gap is signalled if the the absolute difference between expected and actual source packet times exceeds 0.5 * expected_clock_increment (where expected source packet time = last source packet time + expected_clock_increment), and if the difference (actual - expected) is positive.

(Req.4.3-1.1)

A null source packet is characterised solely by its source packet time, which must increment continuously from the time of the last valid source packet until the data gap is filled. The nominal time interval between source packets is given by (ΔT/NGRANULE).

(Req.4.3-2)

A null packet must be recognisable as such. A null packet may be flagged by resetting the Packet ID word [L0-MDS1-3] to a special value, but this is a matter for implementation.

(Req.4.3-2.1)

If start_of_product precedes the first valid source packet in the supplied Level 0 data product, the gap must also be padded with null packets. In this case, to ensure product continuity, the nominal time of the first (null) source packet shall precede the first valid source packet by an integer multiple of the nominal time interval between source packets defined above.

(Req.4.3-3)

As each source packet is processed an integer sequence count of source packets shall be maintained. In the following this is denoted by relative_scan_number(s) having Parameter ID L1B-INT-070. Note that this nomenclature is to emphasise that although the relative scan number must be a linear function of instrument scan index s the two need not (as a matter of implementation) be numerically equal, and to emphasize that they are quantities of different kinds; the relative scan number is a product parameter, while s is an index.

Furthermore, in order to ensure that the scan numbers generated and output by the revised regridding algorithm (Module 18) are positive definite (they are unsigned short), the relative scan number corresponding to the first scan following T0 should not be less than 32. Thus assuming that for near real time (NRT) data, the time of the first scan (s = 0) corresponds to start_of_product, the value relative_scan_number(0) = 32 is recommended. (Note the first scan may be a null scan).

(Req.4.3-4)

It is regarded as a matter for implementation whether null packet padding is accomplished in a preprocessing stage or continuously during processing.

4.3.2 Logical Order of Processes

The process of signal calibration (Module 17) is independent of the geolocation; the pixel calibration operates on the pixel counts, while the geolocation depends only on the time of the scan in question. Thus the order in which they are applied is not important. Otherwise processes should be carried out in the order shown.

4.3.3 Data Buffering

Individual processes up to and including geolocation and signal calibration are (with the exception of the determination of calibration parameters) carried out independently for each scan (indeed, for each pixel); the number of scans held in memory at any one time is largely arbitrary. The choice of this number determines the size of the input buffers, the raw packet buffer and the processed packet buffer.

To produce gridded products, it is necessary to build up the gridded images one instrument scan (source packet) at a time in a set of image buffers, and these must be dimensioned to accommodate the complete scan.

It is necessary to build up the collocated forward and nadir views of a given image area simultaneously. However the forward and nadir sections of a given scan are separated by some 900 km along track; this has the consequence the forward view buffer is larger than that for the nadir view. Moreover, owing to the curvature of the AATSR scan, a single instrument scan in either the nadir or the forward view contributes to image rows extending over 100 km along track. Thus, the first scan to contribute to a image line is one whose forward scan just contributes to the forward image. Subsequent scans build up the forward image: still later scans will begin to contribute to the nadir image. In the case of a nadir scan, pixels at the edges of the scan will have considerably greater along-track co-ordinate than the central pixels.

Thus regridded image buffers must be capable of accommodating the full range of along-track (y) co-ordinates represented by an instrument scan, where minimum allowances are:

variation of y co-ordinate around nadir scan: 116 km (116 rows);
variation of y co-ordinate around forward scan: 116 km (116 rows);
y co-ordinate difference between scan centres (forward - nadir): 960 km.

(Req.4.3-5)

4.3.4 Visible Channel Calibration Coefficients

Data from the VISCAL unit are only available from a short sequence of source packets within an orbit, corresponding to the time when the on-board visible calibration (VISCAL) unit is illuminated by the sun. This sequence of data is the only source of data for the determination of the slope parameters for calibration of the visible channels within the orbit; thus unless parameters derived from a previous orbit are used, it is necessary to unpack and process those source packets corresponding to the VISCAL illumination period before the data from source packets that precede them in the orbit can be calibrated.

Note: One solution to this problem would be to introduce a preprocessing stage to locate, unpack and process the VISCAL packets. It is an open question whether this approach is appropriate for the operational AATSR processor, and the issue is not addressed further in this document.

4.3.5 Overview of Auxiliary Data Processing

The auxiliary data is processed by performing the following tasks in order:

Module 2. Unpacking the data.

Module 3. Validating the unpacked items.

Module 4. Converting those unpacked items which can be converted to engineering units (temperatures etc.).

Module 5. Validating the converted items.

Processing the auxiliary data uses three output arrays: one for storing the unpacked values, one for storing the converted values, and one for containing any error codes that may be associated with each item. It is therefore necessary to maintain an index into these arrays which is clear, simple and logical, since it makes sense that the same index value may be used to access these same three arrays for any given auxiliary data item.

The naming convention used to define the auxiliary data shows a maximum possible range in identifiers from A0010 - A9999. For example, the telemetry identifier “A4411” translates from left to right as “A” for AATSR; “4” for the subsystem TEMP; “41” for the specific subsystem telemetry point; and “1” to indicate it is filled from the least significant part of the word, and is less than 16 bits in size. For a complete explanation of the identification method of auxiliary data items see pp 9 in AD 6.

Using only the integer component of this identifier as an array index is clearly inefficient, since the array sizes required will be between one and two orders of magnitude larger than the amount of space actually needed for the total (less than one thousand (TBC)) auxiliary items actually defined. Thus some intermediate look up table (or algorithm) should be used to map between telemetry identifier and its common array index into all of the processed auxiliary data output arrays. This functionality is provided through the Master Unpacking Definition Table (MUDT), described in the section 5.2.

4.3.5.1 Some general points about error handling.

If an error occurs during any of the modules of auxiliary data processing, the process aborts with a suitable error code in packet_error. Thus on the completion of the auxiliary data processing packet_error should be tested for SUCCESS. If the value is not SUCCESS, it is necessary to set the value of all output pixel buffer elements (for all channels), to the code PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED, and set the value of calibration_invalid[i] (for i = all channels), to the value of packet_error. It is easier to perform this operation once, when the auxiliary data processing has completed, (and providing that packet_error is not equal to SUCCESS of course). Consequently, only packet_error is shown as an input parameter to each of the four components of the auxiliary data processing, rather than the inefficient method of including the test and additional parameters to each component of the auxiliary data processing. Thus error handling can be considered to be an external shell conceptually and logically structured as follows:

```
if (packet_error = SUCCESS)
   packet_error = unpack_auxiliary_data(...)
   if (packet_error = SUCCESS)
```

```
      packet_error = validate_unpacked_auxiliary_data(...)
      if (packet_error = SUCCESS)
         packet_error = convert_unpacked_auxiliary_data(...)
         if (packet_error = SUCCESS)
           packet_error = validate_converted_items(...)
         end if
      end if
   end if
end if

if (packet error NOT= SUCCESS)
  set the value of all output pixels for all channels in the
  corresponding output data buffers L1B-INT-81 to L1B-INT-84 incl.
  to PIXEL_COUNT_SCIENCE DATA_NOT_DECOMPRESSED.
  for i = all channels
    set calibration_invalid[i] to packet_error
  end for
end if
```

5. MODULE DETAILS

5.1 Module Definition: Source Packet Quality Checks

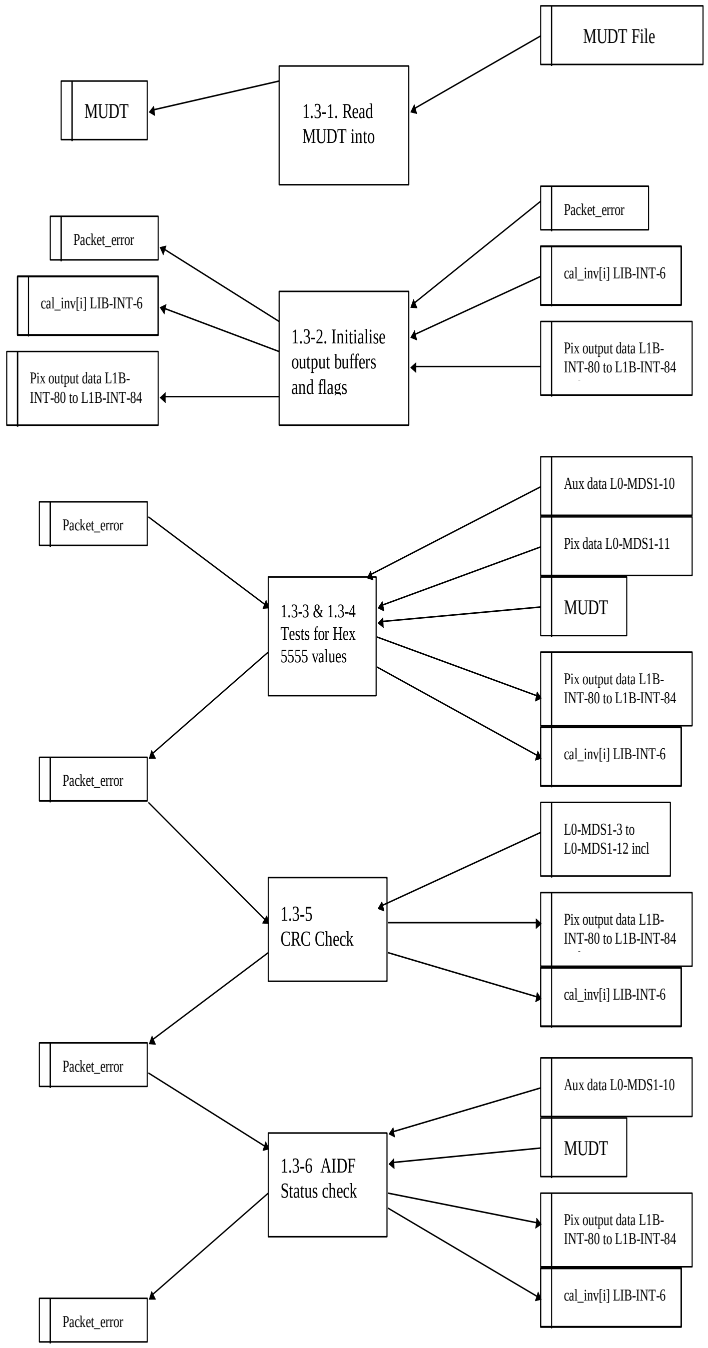
(Within this module a "word" is defined to mean 16 bits).

5.1.1 Functional Description

The purpose of performing basic validation on the packet is to detect any major problems in the data, (such as data corruption), which would make it pointless to continue processing the data in the packet. This is achieved by executing a sequence of tests; only if the packet passes one test will it go on to the next, and only if it passes all the tests will it be further processed. The remainder of this section describes the individual checks in the sequence of execution. If a packet fails basic validation, the error code packet_error is changed from its initial value of SUCCESS, to an error code which indicates the nature of the failure. In addition, all pixel counts (for all channels) in the corresponding output pixel data for that packet are set to PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED, and the calibration_invalid flag for each channel is set to the value of packet_error. These measures ensure that the data from this packet are not used in the calibration.

The first check is a test for hex 5555 values. If the packet successfully passes this test, the cyclical redundancy check (CRC) is performed, calculating the check sum for this packet independently, and comparing it with the CRC check sum in the last word of the packet. The CRC value is derived from all data in the packet. Finally, checks are made of the AIDF status word, to ensure that the IDF data buffers were correctly loaded with data before they were telemetered.

Fig. 5.1-1. Source Packet Quality Checks



5.1.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L0-MDS1-3		Packet ID	us	N/A	2	1
L0-MDS1-4		Packet sequence count	us	N/A	2	1
L0-MDS1-5		Packet length	us	N/A	2	1
L0-MDS1-6		Data field header length	us	N/A	2	1
L0-MDS1-7		Instrument mode	us	N/A	2	1
L0-MDS1-8		ICU on board time	ul	N/A	4	1
L0-MDS1-9		Redundancy vector	us	N/A	2	1
L0-MDS1-10		Auxiliary data	us	N/A	2	390
L0-MDS1-11		Science (pixel) data	us	N/A	2	5357
L0-MDS1-12		CRC Error code for packet	us	N/A	2	1
L1-AUX1-1	aux_id[i]	auxiliary_item_identifier[i]	array of array of uc	N/A	5	aux_tot
L1-AUX1-2	word[i]	word_number_in_packet_containing_auxiliary_item[i]	us array	N/A	2	aux_tot
L1-AUX1-3	mask[i]	mask_required_to_extract_auxiliary_item_from_word[i]	ss array	N/A	2	aux_tot
L1-AUX1-4	shift[i]	shift_required_to_normalise[i]	us array	N/A	2	aux_tot
L1-AUX25-1	aux_tot	total_number_of_auxiliary_data_items	us	N/A	2	1
L1-AUX18-21		RAW_PKT_FAILS_BASIC_VALIDATION_ERR	ss	N/A	2	1
L1-AUX18-22		PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED	ss	N/A	2	1
L1-AUX18-27		CRC_ERR_DETECTED_ERR	ss	N/A	2	1
L1-AUX18-28		BUFFERS_FULL_CHECK_ERR	ss	N/A	2	1
		PIXEL_COUNT_FROM_NULL_PACKET	ss	N/A	2	1
L1-AUX18-19		PIXEL_COUNT_INITIAL_VALUE	ss	N/A	2	1
constant		SUCCESS (= 0)	ss	N/A	2	1
L1-AUX18-54		NULL_PCKT_ERR	ss	N/A	2	1

Table 5-1-1: Input Data Table - Raw packet quality checks

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-84	viscal_pixels[channel][i]	unpacked.pixels.viscal	array of array of ss	N/A	2	max_viscal_pixels * 7
L1B-INT-80	nadir_pixels[channel][i]	unpacked.pixels.nadir	array of array of ss	N/A	2	max_nadir_pixels * 7
L1B-INT-82	pxbb_pixels[channel][i]	unpacked.pixels.plus_bb	array of array of ss	N/A	2	max_pxbb_pixels * 7
L1B-INT-81	frwrd_pixels[channel][i]	unpacked.pixels.forward	array of array of ss	N/A	2	max_frwrd_pixels * 7
L1B-INT-83	mxbb_pixels[channel][i]	unpacked.pixels.minus_bb	array of ss	N/A	2	max_mxbb_pixels * 7
L1B-INT-6	cal_inv(i)	calibration_invalid[channel]	ss	N/A	2	7
L1B-INT-1	packet_error	packet_error	ss	N/A	2	1
local		test_completed_ok	ss	flag	2	1
local	j	internal loop counter	ss	N/A	2	1

Table 5-1-2: Internal Data Table: raw packet quality checks

5.1.3 Algorithm definition

Step 5.1.3-1 Reading the MUDT into memory.

This initialisation process is performed once only when the system is started up. The MUDT is needed to unpack specific auxiliary data items, as well as in routine processing of the auxiliary data.

```
openfile(MUDT)
i = 0                                                  (Req.5.1-1)
while not end of file (MUDT)
      read aux_id(i),word(i),mask(i),shift(i)          (Req.5.1-2)
      i = i+1                                          (Req.5.1-3)
end while
aux_tot = number of records read
closefile(MUDT)
```

Notes.

1. A single record contains the following fields: aux_id, word, mask, shift.
2. This is performed once only, when the system is initialised**.**
3. For more details on the MUDT see sections 5.2.1.

Step 5.1.3-2 Initialise the output buffers and flags.

These initialisation steps are performed once for each source packet that is tested.

```
Initialise the values of all elements in the output pixel buffers
L1B-INT-80 to L1B-INT-84 to the value PIXEL_COUNT_INITIAL_VALUE.
                                                       (Req.5.1-4)
Initialise the values of cal_inv[i] to 0 for all channels.
                                                       (Req.5.1-5)
Initialise the value of packet_error to FALSE.         (Req.5.1-6)
```

Step 5.1.3-3 Check the auxiliary data for unwanted 5555 (Hex) values.

```
if the source packet is a null packet,
Set the value of all output pixels for all channels in the
  corresponding output data buffers L1B-INT-80 to L1B-INT-84 incl. to
  the error code value
  PIXEL_COUNT_FROM_NULL_PACKET                         (Req.5.1-7)
  set packet_error to NULL_PCKT_ERR                    (Req.5.1-8)
  for i = all channels
    set cal_inv[i] to packet_error                     (Req.5.1-9)
  end for
                                                      (Req.5.1-10)
else
test_completed_ok = FALSE                             (Req.5.1-11)
While (NOT test_completed_ok) for j=0,779,2
  if (L0-MDS1-10(byte(j)) != (Hex)55)
      or (L0-MDS1-10(byte(j+1) != (Hex)55) then
      test_completed_ok = TRUE                        (Req.5.1-12)
  end if
end while

if (NOT test_completed_ok)
```

```
    Set the value of all output pixels for all channels in the
    corresponding output data buffers L1B-INT-80 to L1B-INT-84 incl. To
    the error code value
    PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED          (Req.5.1-13)

    set packet_error to RAW_PKT_FAILS_BASIC_VALIDATION (Req.5.1-14)
    for i = all channels
      set cal_inv[i] to packet_error                   (Req.5.1-15)
    end for
  end if
  end if
```

Step 5.1.3-4 Check the science data for unwanted 5555 (Hex) values.

The following applies only if the packet passed the previous test.

Step 5.1.3-4-1 Unpack auxiliary data items: A0143, A0146, A2515:

```
for each auxiliary data item (A0143, A0146, A2515)
  Perform a binary search on auxiliary data item identifier field of
  the MUDT table (which must be ordered on this field) until a match
  is made with the input item string.                  (Req.5.1-16)
```

A binary search is required for efficiency. A suitable binary search algorithm can be found in AD 9, section 3.4, pp 117. If no match is made, the function returns an error code.

```
  Return the unpacking variables (word, mask and shift), from the
  matched item in the MUDT.                            (Req.5.1-17)

  Use the unpacking variables to unpack the required auxiliary data
  item as follows:
      unpacked_value = (L0_MDS1_10(word) AND mask)
      unpacked_value = ls_Shift(unpacked_value, shift) (Req.5.1-18)
end for
```

Check the science data for hex 5555s.

Use the unpacked auxiliary data items to check the science data for HEX 5555 values. The algorithm is defined below.

```
If (the instrument is out of standby (A0143, ICU14) and is powered
      (A0146, ICU14), and it is in nominal mode (Bit 0 [LSB] of the
      source packet word 5 = 0), and the packet contents contain
      measurement data (Bit 1 of the source packet word 5 = 0), and
      the IDF data flow inhibit flag (A2515, AIDF1) is zero,
      (indicating that data flow is enabled) then
   search all the pixel data in the raw packet.        (Req.5.1-19)
   If (more than 1 contiguous word (TBC) of science data
      contain Hex 5555 values) then
```

```
        Set packet_error to RAW_PKT_FAILS_BASIC_VALIDATION_ERR
                                                        (Req.5.1-20)
        Set the value of all output pixels for all channels in the
        corresponding output data buffers L1B-INT-80 to L1B-INT-84
        incl. to PIXEL_COUNT_SCIENCE DATA_NOT_DECOMPRESSED.
                                                        (Req.5.1-21)

        for i = all channels
          set cal_inv[i] to packet_error                (Req.5.1-22)
        end for
     end if
  end if
```

Step 5.1.3-5 Check the CRC value for this packet.

```
  if the packet passed the previous test (packet_error == SUCCESS) then

    Calculate the CRC value for all data specified by parameters L0-
  MDS1-3 to L0-MDS1-11 (in that order)                 (Req. 5.1-23)
```

Note. The CRC value loaded into the packet is an implementation of the generator polynomial CRC16-CCITT, pre-loaded with all 1's. (I.e. $G(x) = x^{16} + x^{12} + x^5 + 1$). For full details see AD 9. section 20.3, pp 896 - 901. In calculating the CRC value, the data should be passed through the CRC generator in packet order: MS bit first starting at word 1.

```
    Compare the result to the CRC Error code for packet (L0-MDS1-12).
                                                       (Req. 5.1-24)

    if the calculated CRC value does not equal the CRC value specified
           by L0-MDS1-12 then
```

Exclude the packet from further processing as follows

```
        Set the packet_error to CRC_ERR_DETECTED_ERR   (Req. 5.1-25)

        Set the value of all output pixels for all channels in the
        corresponding output data buffers L1B-INT-80 to L1B-INT-84
        incl. to the value PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED
                                                       (Req. 5.1-26)
        Set cal_inv[i] to the value of packet_error for all
        channels.                                      (Req. 5.1-27)
    end if
  end if
```

Step 5.1.3-6 Check the IDF data buffers.

Note. The details of this test will have to be re-specified for AATSR because of the significant differences between the AIDF status word definitions of ATSR-2 and AATSR. A summary description is provided here.

```
  if the packet passed the previous test (packet_error == SUCCESS) then
    Check the appropriate auxiliary data value (TBD), to ensure that
```

```
  the IDF data buffers were fully loaded with data before they were
telemetered.                                          (Req. 5.1-28)
  if the buffers were not fully loaded then
```

Exclude the packet from further processing as follows:

```
      Set packet_error = BUFFERS_FULL_CHECK_ERR     (Req. 5.1-29)

      Set the value of all output pixels for all channels in the
      corresponding output data buffers L1B-INT-80 to L1B-INT-84
      inclusive to the error code value
      PIXEL_COUNT_SCIENCE DATA_NOT_DECOMPRESSED     (Req. 5.1-30)

    Set calibration_invalid[i] to the value of packet_error for all
    channels.                                       (Req. 5.1-31)
  end if
end if
```

5.2 Unpacking the auxiliary data.

5.2.1 Functional description

Within words 9 to 398 (inclusive) of a source packet (word numbering in the AATSR packets starts from 1 - see AD 7 pp 9), AATSR auxiliary data items are packed words. The auxiliary data contains items specific to all the AATSR subsystems, (ICU, SCC, SCP, BB etc.). The subsystems and all their associated words are defined on pp 82 of AD 6.

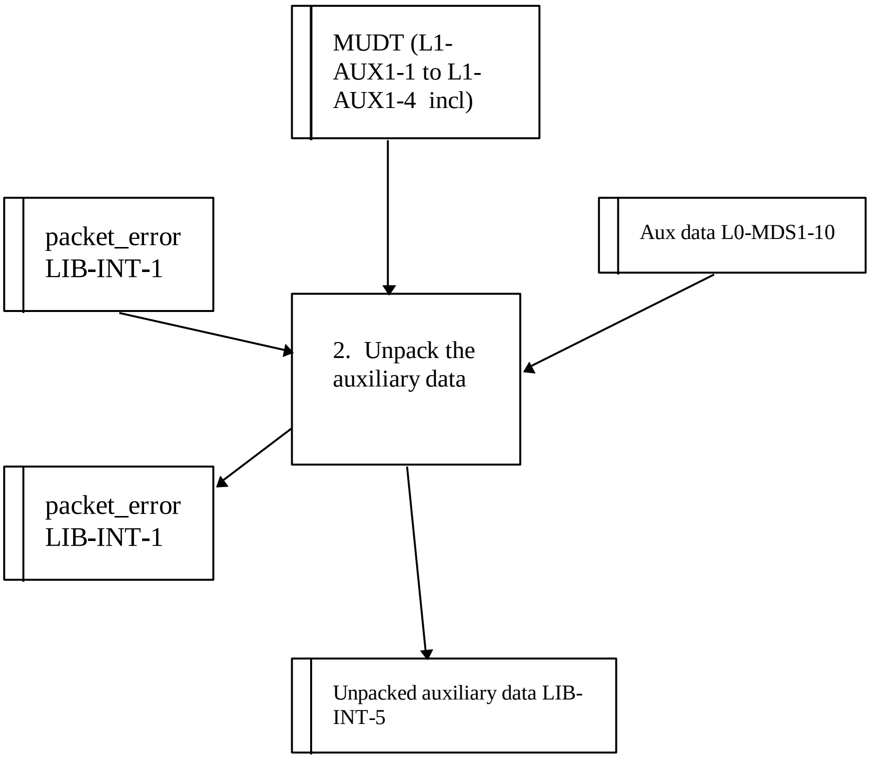
The specific bit numbers and word numbers defining each auxiliary data item, (see AD 8 for the detailed definitions), can be incorporated into a table driven unpacking algorithm. The AATSR telemetry specification will define the byte number within the packet within which the item starts, the bit number within the byte at which each item starts, and the length of each item in bits. From this information, it is possible to generate a master unpacking definition table (MUDT) which defines the word in the raw packet where each variable can be found, the mask required to extract each variable from the word, and the shift required to correctly justify each variable once it has been masked out. This table can then be read in during initialisation, and used thereafter to unpack each auxiliary data item from the raw packet, storing the resultant integers in an array.

For example, if a variable was packed into bits 4 to 8 inclusive of word 23 of the raw packet, (where bit 0 is the least significant bit), the MUDT would contain the following items (in base 10) to define the unpacking of that item:

- word number: 23
- mask: 496 $(2^4 + 2^5 + 2^6 + 2^7+2^8)$
- shift: (shift extracted item right 4 places to normalise)

Whenever the AATSR telemetry specification is updated, a new version of the MUDT will have to be generated, and a new value of aux_tot may have to be defined. Since aux_tot is used to define the size of data arrays, it will be defined as a constant. Thus a change in the MUDT causing a change in the value of aux_tot would currently necessitate a system recompilation and rebuild.

Fig. 5.2-1. Unpacking the auxiliary data.



5.2.2 Interface definition.

Parameter ID	Variable	Name	Type	Units	Field Size	Fields
L0-MDS1-10		Auxiliary data	us	N/A	2	390
L1-AUX25-1	aux_tot	total_number_of_auxiliary_data_items	us	N/A	2	1
L1-AUX1-1	aux_id[i]	auxiliary_item_identifier[i]	array of array of uc	N/A	5	aux_tot
L1-AUX1-2	word[i]	word number in packet containing auxiliary item [i]	us array	N/A	2	aux_tot
L1-AUX1-3	mask[i]	mask required to extract auxiliary item from word[i]	ss array	N/A	2	aux_tot
L1-AUX1-4	shift[i]	shift required to normalise item [i]	us array	N/A	2	aux_tot

Table 5-2-1: Input Data Table: Unpacking the auxiliary data.

The index i ranges from 0 to total_number_of_auxiliary_data_items - 1.

Parameter ID	Variable	Name	Type	Units	Field Size	Fields
L1B-INT-5	auxdata(i)	unpacked_auxiliary_data[i]	us array	N/A	2	aux_tot
local	i	internal loop counter	ss	N/A	2	1
L1B-INT-1	packet_error	packet_error	ss	N/A	2	1

Table 5-2-2: Internal Data Table: Unpacking the auxiliary data.

5.2.3 Algorithm definition.

Step 5.2.3-1 Unpacking auxiliary data from the AATSR data:

```
if packet_error == SUCCESS then
```

Unpack every item of auxiliary data from the source packet using the word, mask and shift unpacking definitions in the MUDT defined for each auxiliary data item:

```
   do for i=0 to i < aux_tot, increment=1
      auxdata(i) = L0_MDS1_10(word(i)) AND mask(i)
      auxdata(i) = ls_Shift(auxdata(i),shift(i))       (Req.5.2-1)
   end do
end if
```

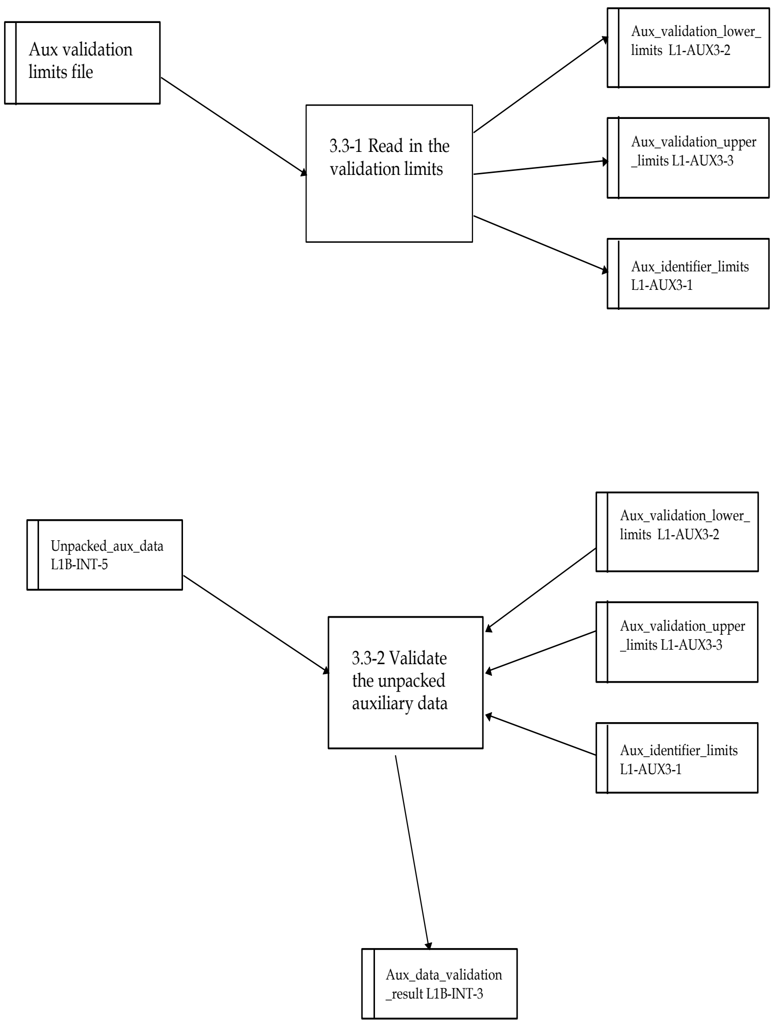
Note. ls_Shift is defined as a function which implements a bit shift to towards the least significant end of the word.

5.3 Validation of unpacked auxiliary data.

5.3.1 Functional Description.

The validation of the unpacked auxiliary data consists of limit checks on the unconverted data. Limit checking is table driven; the upper and lower limits of all specified unpacked auxiliary items are read in from a data file (along with the identifier with which they are associated). The results of each validation is loaded into the integer array auxiliary_data_validation_result (L1B-INT-3), which will be organised, such that the same index can be used to access either value from unpacked_auxiliary_data (L1B-INT-5), or the corresponding value from auxiliary_data_validation_result.

Fig. 5.3-1 Validate the unpacked auxiliary data



5.3.2 Interface definition.

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX25-1	aux_tot	total_number_of_auxiliary_data_items	us	N/A	2	1
L1-AUX25-2	aux_val_tot	number_of_aux_validation_values	us	N/A	2	1

L1-AUX1-1	aux_id[i]	auxiliary_item_identifier[i]	array of array of uc	N/A	5	aux_tot
L1-AUX3-1	aux_id_limits	auxiliary_identifier_limits[i]	array of uc	N/A	5	aux_val_tot
L1-AUX3-3	aux_val_ul[i]	aux_validation_upper_limit[i]	array of us	N/A	2	aux_val_tot
L1-AUX3-2	aux_val_ll[i]	aux_validation_lower_limit[i]	array of us	N/A	2	aux_val_tot
L1-AUX18-56		UTMZ_DOMAIN_ERROR	us	n/a	2	1

Table 5-3-1: Input Data Table - Validating the unpacked the auxiliary data

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-1	packet_error	packet_error	ss	N/A	2	1
L1B-INT-3	advr(i)	auxilary_data_validation_result[i]	array of us	N/A	2	aux_tot
L1B-INT-5	auxdata(i)	unpacked_auxiliary_data[i]	us array	N/A	2	aux_tot

Table 5-3-2: Internal Data Table - Validating the unpacked the auxiliary data

5.3.3 Algorithm Definition:

The algorithm has two parts:

- Reading in the auxiliary identifier and its associated limits data.
- Validating the unpacked auxiliary data.

Step 5.3.3-1 Read the auxiliary limits data into memory.

This operation is performed once only at system initialisation time.

```
openfile (auxiliary validation values)                    (Req.5.3-1)
i = 0                                                     (Req.5.3-2)
while not end of file (auxiliary validation values)
      read auxiliary_identifier_limits[i],
      aux_val_ul[i], aux_val_ll[i]                        (Req.5.3-3)
      i = i + 1                                           (Req.5.3-4)
end while
aux_val_tot = number of records read in                   (Req.5.3-5)
closefile (auxiliary validation values)                   (Req.5.3-6)
```

Note. A single record contains the following fields: auxiliary_identifier_limits, aux_validation_upper_limit, and aux_validation_lower_limit.

Step 5.3.3-2 Validate the unpacked auxiliary data.

```
Initialise all elements in the output array
auxiliary_data_validation_result to zero.  (The array is used to
contain any error codes for auxiliary data items from this packet).
                                                         (Req. 5.3-7)


Step through the array of identifiers in L1-AUX3-1, using each in
turn as the master item on which to search forward through the array
of all identifiers, L1-AUX1-1. When a match is made, use the index of
the item in L1-AUX1-1 to obtain access to the unpacked data item from
the array unpacked_auxiliary_data. If no match is made then an error
has occurred.                                            (Req. 5.3-8)
```

```
if the unpacked value is greater than the upper limit, or the
     unpacked value is less than the lower limit, then
   perform a logical OR between the corresponding element of the
   auxiliary_data_validation_result array and the error code
   UTMZ_DOMAIN_ERROR, to indicate that the item is out of range.
                                                    (Req. 5.3-9)
end if
```

5.4 Converting the unpacked auxiliary data to engineering units.

Not all auxiliary items can be converted to engineering units. Some items are simply status flags for which no conversion is sensible. Items that can be converted are specified in AD 6.

5.4.1 Functional description.

In the same way the auxiliary data is unpacked, the conversion of the unpacked auxiliary data is optimally performed using a look up table. In this case however, the look up table describes for each variable that can be converted, which of a series of generic conversion functions to use, and the list of parameters to use for the conversion. A series of generic functions are defined to cater for every mathematical operation required in the conversion of AATSR auxiliary data.

For example, the conversion look up table entry for the (non existent) auxiliary data item TEMP 41, (an 8 bit value defined in the 8 least significant bits of the word, and having the auxiliary item identifier A4411) has four fields:

```
A4411, F4, -128, 349.1
```

The first is the auxiliary_conversion_identifier which identifies the telemetry item to be converted. The second is the function identifier which indicates which function is to be used to perform the conversion for this item, and the remaining two are conversion parameters associated with the function.

Collectively, this conversion data indicates to the conversion algorithm that the unpacked count for auxiliary item A4411 is converted using the generic conversion function F4, with takes two parameters a and b. The values of those parameters being -128, and 349.1 respectively. The generic function F4 is defined as: $F4 = (Z+a)/b$, where Z is the value of the unconverted auxiliary item A4411, and a and b are the values of the parameters: 128 and 349.1 respectively.

It must also be possible to specify other, unpacked telemetry items in the list of function parameters, as well as constants. By convention, parameters which are other auxiliary items occur first in the parameter list, and are clearly distinguished from constants, by their identifying prefix of “A”. In the following example, the conversion look up table entry for the (non existent) auxiliary data item A4421 is:

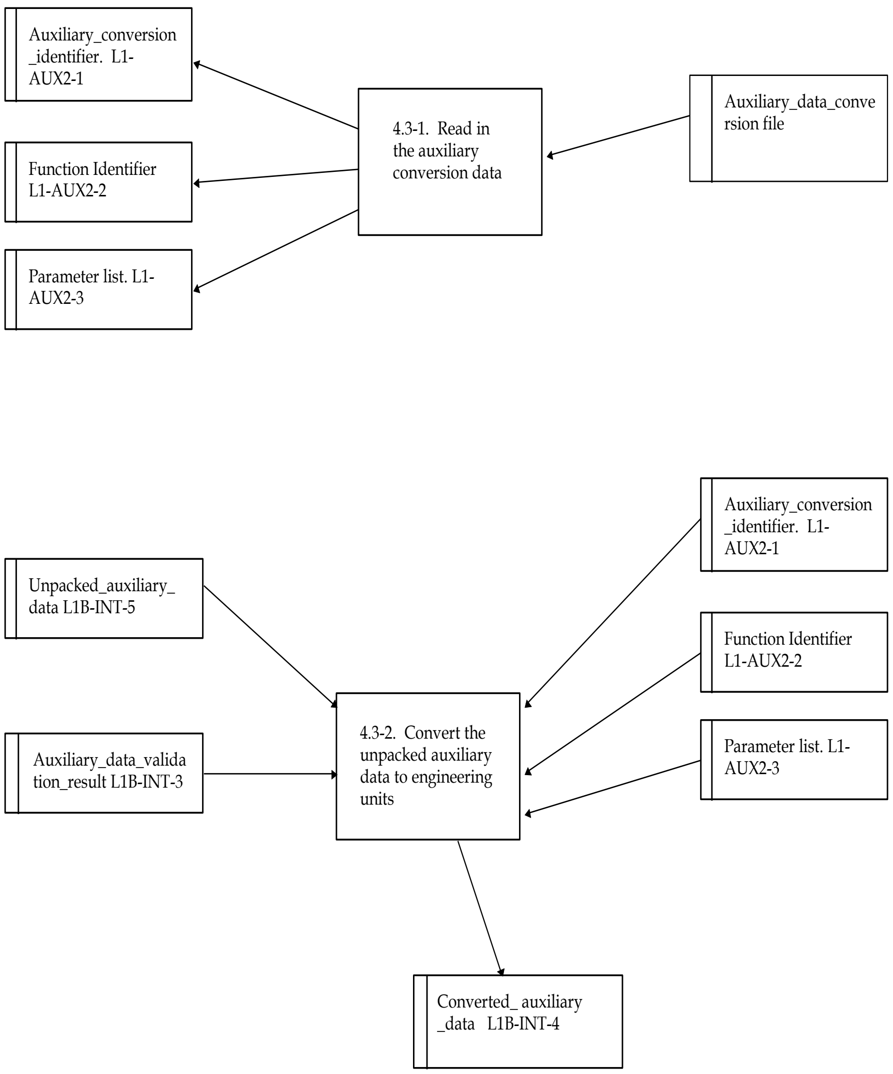
```
A4421, F21, A3282, -25.7, 45, 218.768
```

This indicates to the conversion algorithm that the unpacked count for auxiliary item A4421 is converted using the generic conversion function F21, which takes four parameters a, b, c and d. Of these parameters, the first is the unpacked auxiliary data item A3282, and the others are all constants.

The value of the converted data items are loaded into a floating point (or real) array called converted_auxiliary_data (L1B-INT-4). All auxiliary items that are not eligible for conversion shall nonetheless be copied over to that array as the float equivalents of their integer counterparts. This has the advantage that all auxiliary items are then accessible from one array, rather than two, whether converted or not.

Note: The indexing method into the array should be kept consistent with the indexing method used throughout all the auxiliary data processing, such that the same index value will return a data value for the same auxiliary data item, for all of the auxiliary data arrays which are referenced.

Fig. 5.4-1 Convert the unpacked auxiliary data to engineering units



5.4.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX1-1	aux_id[i]	auxiliary_item_identifier[i]	array of array of uc	N/A	5	aux_tot
L1-AUX2-1		auxiliary_conversion_identifier[k]	uc array	N/A	5	n_conv
L1-AUX2-2		function_identifier[k]	uc array	N/A	3	n_conv
L1-AUX2-3		parameter_list[k][j]	array of array of do	N/A	8	n_conv

						* 20
L1-AUX25-1	aux_tot	total_number_of_auxiliary_data_items	us	N/A	2	1
L1-AUX25-3	n_conv	number of items to be converted	us	N/A	2	1
L1-AUX18-56		UTMZ_DOMAIN_ERROR	us	n/a	2	1

Table 5-4-1: Input Data Table - Converting the unpacked auxiliary data

Note 1. The local counter i, indexes into the array unpacked_auxiliary_data (containing the unpacked counts which are to be converted), and also into the auxiliary_validation_result array (containing the validation results for each auxiliary item).

Note 2. The auxiliary conversion data is contained in three arrays:

auxiliary_conversion_identifier[k]: an array containing the identifiers of all auxiliary items which are to be converted. The array is indexed by local index k.

function_identifier[k]: an array containing the identifier of the function to be used in the conversion calculation for that auxiliary item. The array is indexed by local index k.

parameter_list[k][j]: a 2 dimensional array which contains up to 20 parameters for each auxiliary_identifier[k] to be converted. The parameters for each auxiliary identifier to be converted are indexed by the local counter j.

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-1	packet_error		ss	N/A	2	1
L1B-INT-3	advr(i)	auxilary_data_validation_result[i]	array of us	N/A	2	aux_tot
L1B-INT-4	conv_aux_data	converted_auxiliary_data[i]	fl array	N/A	4	aux_tot
L1B-INT-5	auxdata(i)	unpacked_auxiliary_data[i]	ss array	N/A	2	aux_tot
L1B-INT-071	aux_temp(s, jaux)	temporary auxiliary temperatures for Module 18	float	K	4	jaux = 0, 5
L1B-INT-072	aux_unconv(s, jaux)	unconverted auxiliary temperatures for Module 18	us	n/a	2	jaux = 0, 5
L1B-INT-073	pixel_map(s)	pixel map number for use by Module 18	ss	n/a	2	per s

Table 5-4-2: Internal Data Table - Converting the unpacked auxiliary data

Note. The output array converted_auxiliary_data contains all converted auxiliary data items from a single source packet, together with all unconverted items cast from integer type to floats.

5.4.3 Algorithm definition.

There are two parts to the algorithm. The first (represented by Step 5.4.3-1) requires the reading into memory of the conversion data for each (convertible) auxiliary data item. The second (represented by Steps 5.4.3-2, 5.4.3-3) is the algorithm which performs the conversions and is applied to each source packet (or instrument scan). In addition, the algorithm for every conversion function is also defined.

Step 5.4.3-1 Read in the conversion data for each (convertible) auxiliary data item.

This operation is performed once only, at system initialisation time.

```
openfile (auxiliary_data_conversion)                 (Req. 5.4-1)
k=0                                                  (Req. 5.4-2)
while not end of file (auxiliary_data_conversion)
  read auxiliary_conversion_identifier[k],
```

```
     function_identifer[k]                               (Req. 5.4-3)

  Now read in all the conversion parameters associated with the
  conversion of this auxiliary item.
  j= 0                                                   (Req. 5.4-4)
  do (while not end of parameter list)
      read parameter_list[k][j]                          (Req. 5.4-5)
      j=j+1                                              (Req. 5.4-6)
  end do

  k = k + 1                                              (Req. 5.4-7)
end while
n_conv = number of records read in                       (Req. 5.4-8)
close_file (auxiliary_data_conversion)                   (Req. 5.4-9)
```

Note. A single record consists of the following fields: auxiliary_conversion_identifier, function_identifer, and parameter_list. The records in the file are arranged in ascending order of auxiliary_conversion_identifier.

Step 5.4.3-2 Convert the unpacked auxiliary data items

```
if the packet has passed the packet quality check then
```

(For each unpacked auxiliary item)

```
  do for i = 0 to i < aux_tot, increment = 1
      current_identifer = auxiliary_item_identifier[i] (Req.5.4-10)
```

(See if the current identifier is one to be converted. If it is, then the current_identifier will have an identical entry in the auxiliary_conversion_identifier array.)

```
      Search the array auxiliary_conversion_identifer for a
      match on current_identifier using j as the search index
      into the auxiliary_conversion_identifer array.  (Req.5.4-11)

      if a match is found then
```

(The item is to be converted, so check that the unpacked item is valid)

```
        if advr[i] is not equal to UTMZ_DOMAIN_ERROR then
```

(The item is valid so convert it using convert_item; a function which takes the integer item to be converted and an index (j) which (because of the match found by the earlier search) will be used to index into the function_identifer and parameter_list arrays. (These arrays identify which conversion functions and parameters to use in the conversion of this auxiliary item. (**Note**. See also the list of generic conversion functions below.)

```
          conv_aux_data[i] =
            convert_item (unpacked_auxiliary_data[i], j)(Req.5.4-12)
        end if
      else
```

(No match found - the item cannot be converted, so simply convert the unpacked count to a floating point value and store it in the converted array.)

```
            conv_aux_data[i] =
              cast to float (unpacked_auxiliary_data[i])  (Req.5.4-13)
          end if
      end do
    end if
```

Step 5.4.3-3 Extract the aux. temps and pixel map number for regridding.

Note: the processing described in this step formed part of Module 18 in earlier revisions of this document. It has been moved here in the present revison so that processes involving the MUDT are removed from Module 18. This does not affect the logic of the processing scheme as a whole.

Extract the auxiliary temperatures to be saved from the converted auxiliary data array:

```
For jaux = 0 to 5
```

Identify the index *iaux* in the auxiliary data array corresponding to the next telemetry identifier in the following list by a search on the MUDT as in Section 5.1.3 (Req. 5.1-16).

The telemetry identifiers to be searched for are, in turn:

jaux = 0:	A5061	(SCP6	FPA BASEPLATE TEMP)
jaux = 1:	A5051	(SCP5	12 DET TEMP)
jaux = 2:	A5041	(SCP4	11 DET TEMP)
jaux = 3:	A5031	(SCP3	3.7 DET TEMP)
jaux = 4:	A5021	(SCP2	1.6 DET TEMP)
jaux = 5:	A5071	(SCP7	0.87 DETECTOR TEMP)

Set

```
aux_temp(s, jaux) = conv_aux_data(iaux)
aux_unconv(s, jaux) = auxdata(iaux)
                                                  (Req. 5.4-14)
```

(Note the implicit dependence of the quantities on the right hand side on scan number s.)

```
End for
```

Identify the index iaux in the auxiliary data array of the 8 bit pixel map identifier A2190 (IDF 19) by a search on the MUDT as in Section 5.1.3 (Req. 5.1-16).

```
pixel_map(s) = auxdata(iaux)
                                                  (Req. 5.4-15)
```

The generic conversion algorithms

For each conversion algorithm, f0 to f14, Z is the integer value of the unpacked auxiliary data item which is being converted, and a, b, c, d, e, f, g and h,...etc. are the values of parameters which are supplied specifically for each auxiliary data item which uses that function. In more complex functions, “result” is the name of a local variable in which the result of the function is returned.

```
F0:    a^Z*b

f1:    if (Z=0) then 4
else if (Z=1) then 8
else if (Z=2) then 16
else if (Z=4) then 32
else if (Z=8) then 64
else if (Z=9) then 128

f2:    aZ/b

f3:    aZ+b

f4:    (Z+a)/b

f5:    a+(Z/b)

f6:    a/(Z+b)

f7:    if (Z>=a) then b+cZ else d+eZ
```

f8: Black body temperature function.

This function operates on the BB telemetry words BB2-7, BB10-15. The structure of these words is shown in Figure 5.4-2 (below).

The function takes 7 parameters. The first (a) is an unpacked auxiliary data item which indicates if the BB is in the hot or the cold range. The remaining 6 parameters comprise two groups of three polynomial coefficients that define a third order polynomial used to derive the BB temperature: b, c and d, are the coefficients used for the hot range, and e, f, and g are for used the cold range. The sign is determined by the value of the bit B1.

dn is a local variable: it is the 12 bit temperature count; extracted from the least significant 12 bits of the 16 bit word Z. i.e. dn = (Z & 0FFF). Parameters v and result are locally declared real variables. The function returns result.

```
if Z & (Hex) 1000 = 0 (bit B1 is zero) then
      calculate v=0.001*((0.0-dn)-0.5)
else
      (bit B1 is 1)
      calculate v=0.001*(dn+0.5)
end if
if (a==HOT_RANGE)
      result=b+(c*v)+(d*v^2)
else
      result=e+(f*v)+(g*v^2)
```

```
endif
```

In the above, HOT_RANGE is a constant having the value 1. Note that each black body (MX BB and PX BB) may be selected as either the hot or the cold black body. Which is which is determined by two bits of the telemetry word BB17; these are unpacked into the two auxiliary data items A6174 and A6175. Their interpretation is as follows:

Data Item	Description	Interpretation	
A6174	PXBB Range	0 = COLD	1 = HOT
A6175	MXBB Range	0 = COLD	1 = HOT

f9: BB Electronics Unit temperature function.

This function operates on the BB9 telemetry word. The structure of the BB9 telemetry word is defined in [AD6] and shown in Figure 5.4-2 (below).

The function takes 6 parameters, all of which are constants: parameters to a 6th order polynomial. Initially the voltage is derived according to the value of field B1of the 16 bit word Z.

dn is a local variable: it is the 12 bit temperature count; extracted from the least significant 12 bits of the 16 bit word Z. i.e. dn = (Z & 0FFF). Parameters v and result are locally declared real variables. The function returns result.

```
if Z & (Hex) 1000 = 0 (bit B1 is zero) then
      calculate the voltage: v=0.001*((0.0-dn)-0.5)
else
      (bit B1 is 1)
      calculate the voltage: v=0.001*(dn+0.5)
endif
result=a+(b*v)+(c*v^2)+(d*v^3)+(e*v^4)+(f*v^5)
```

```
f10:  ((aZ+b)^2)/c
```

```
f11:  if (Z=0) then (Z AND FFF7DFH) else Z
```

```
f12:  aZ^2+bZ
```

```
f13:  aZ^3+bZ^2+cZ+d
```

```
f14:  if (Z<a) then
      cZ+d
else if (a<=Z<=b) then
      eZ+f
else gZ+h
```

f15: Black body temperature function (revised for AATSR data).

This function operates on the BB telemetry words BB2-7, BB10-15. The structure of these words is shown in Figure 5.4-2.

The function takes 7 parameters. The first (a) is an unpacked auxiliary data item which indicates if the BB is in the hot or the cold range. The remaining 6 parameters occur in two groups of three polynomial coefficients that define a third order polynomial used to derive the BB temperature: b, c and d, are the coefficients used for the hot range, and e, f, and g are used for the cold range. The sign is determined by the value of the bit B1.

dn is a local variable: it is the 12 bit temperature count; extracted from the least significant 12 bits (A12 - A1) of the 16 bit word Z. i.e. dn = (Z & 0FFF). Parameters v and result are locally declared real variables. The function returns the value ‘result’.

```
if Z & (Hex) 1000 = 0 (bit B1 is zero) then
      calculate v = -6.81156e-4 - 1.024916e-3 * dn
else
      (bit B1 is 1)
      calculate v=9.617964e-4 + 1.024921e-3 * dn
endif
if (a==HOT_RANGE)
      result=b+(c*v)+(d*v^2)
else
      result=e+(f*v)+(g*v^2)
endif
```

In the above, HOT_RANGE is a constant having the value 1, and the range parameter *a* is interpreted as described under function f8 above.

(This revised function is identical to function f8 except for the A/D converter coefficients used to compute the voltage v.)

f16: BB Electronics Unit temperature function (revised for AATSR data).

This function operates on the BB9 telemetry word. The structure of the BB9 telemetry word is defined in [AD6] and shown in Figure 5.4-2.

The function takes 3 parameters, a, b and c, all of which are constants. These represent an empirical odd cubic polynomial relating inverse temperature to the logarithm of the PRT resistance.

dn is a local variable: it is the 12 bit temperature count; extracted from the least significant 12 bits (A12 - A1) of the 16 bit word Z. i.e. dn = (Z & 0FFF). The parameters v and result are locally declared real variables. The function returns result.

```
if Z & (Hex) 1000 = 0 (bit B1 is zero) then
      calculate v = -6.7058e-4 - 1.025613e-3 * dn
else
      (bit B1 is 1 and the sign is positive)
      calculate v = 8.658e-4 + 1.025752e-3 * dn
endif
```

Then calculate the resistance Rth and convert to temperature.

```
R1 = 33200
R2 = 39200
VR = (v + 5)/17
Rx = R1 * VR/(1. - VR)
```

```
Rth = Rx * R2 / (R2 - Rx)
result = 1/(a + b * ln(Rth) + c * (ln (Rth))**3)
```

Figure 5.4-2. Structure of the BB telemetry words BB1-7, BB9-15.

X1	D1	C1	B1	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1

The temperature field is represented by bits A12 to A1, and the remaining bits are

X1 = don't care;
D1 = don’t care (BB1 and BB9 only);
D1 = range status (BB2 - 7 and 10 - 15: duplicates A6174 or A6175, as appropriate);
C1 = over-range bit;
B1 = sign bit; 0 = Negative, 1 = Positive.

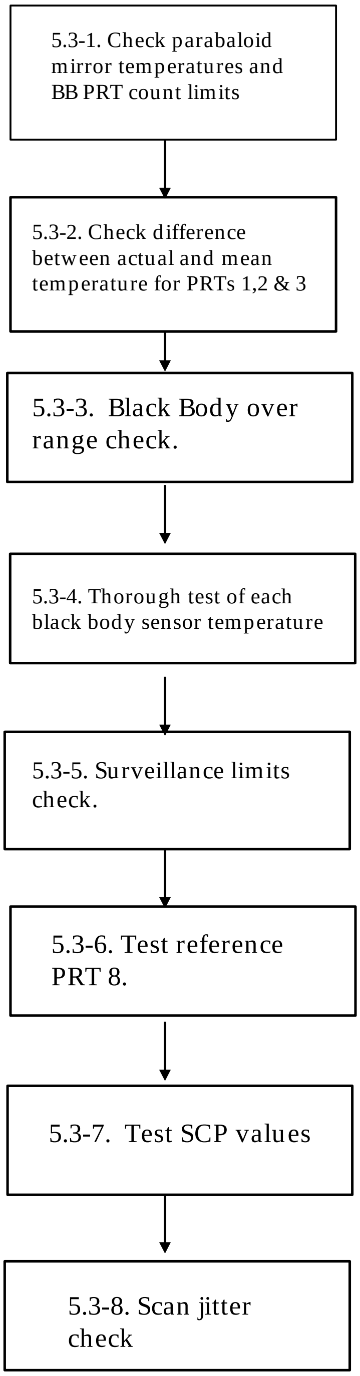
5.5 Validation of the converted auxiliary data.

Most of the following tests are performed on the converted data items, however two tests defined in this section are performed on unconverted data. Since they are also specialised tests, (rather than simple limit checks), and since it does not matter if the conversion has already been performed, it is considered appropriate to include them in this section.

5.5.1 Functional Description.

The validation of the auxiliary data consists of a sequence of specialised checks on specific items of converted data. The black body PRT temperatures receive particularly close scrutiny, since it is essential to verify that the Black body PRT temperatures are of good enough quality to be used in the calibration calculations. The results of all the validations are stored in the array ‘auxiliary_data_validation_result’ [L1B-INT-3].

Fig. 5.5-1 Summary of the main processes used to validate converted auxiliary data



5.5.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX25-4	n_surv_tot	total number of surveillance limits	us	N/A	4	1
L1-AUX26-1	aux_surv_id[j]	surveillance limits auxiliary identifiers	fl array	N/A	4	n_surv_tot
L1-AUX26-2	surv_lwb[j]	surveillance lower limits	fl array	N/A	4	n_surv_tot
L1-AUX26-3	surv_upb[j]	surveillance upper limits	fl array	N/A	4	n_surv_tot
L1-AUX4-1		PRT_MEAN_DIFFERENCE	do	K	8	1
L1-AUX4-2		MAX_HBB_PRT_MEAN_DIFF	do	K	8	1
L1-AUX4-3		MAX_CBB_PRT_MEAN_DIFF	do	K	8	1
L1-AUX4-4		SCP2_6_TEMP_DIFF	do	K	8	1
L1-AUX4-5		SCP7_10_TEMP_DIFF	do	K	8	1
L1-AUX4-6		PRT_8_FIXED_VAL	ss	N/A	4	1
L1-AUX4-7		PRT_8_VARIANCE	ss	N/A	4	1
L1-AUX18-33		PIX_SCAN_JITTER_ERR	us	N/A	2	1
L1-AUX18-34		TMZ_AT_LIMIT_ERR	us	N/A	N/A	1
L1-AUX18-35		TMZ_ROGUE_PRT_ERR	us	N/A	N/A	1
L1-AUX18-36		TMZ_CALIBRATION_ERR	us	N/A	N/A	1
L1-AUX18-37		TMZ_BB_OVERRANGE_ERR	us	N/A	N/A	1
L1-AUX18-45		TMZ_ROGUE_BB_ERR	us	N/A	N/A	1
L1-AUX18-38		TMZ_SURVEILLANCE_ERR	us	N/A	N/A	1
L1-AUX18-39		TMZ_PRT8_ERR	us	N/A	N/A	1
L1-AUX18-40		TMZ_ROGUE_SCP_ERR	us	N/A	N/A	1
L1-AUX18-41		TMZ_BB_OUT_OF_LIMIT_ERR	us	N/A	N/A	1

Table 5-5-1: Input Data Table - Validating the converted auxiliary data

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-1	packet_error		ss	N/A	2	1
L1B-INT-4	conv_aux_data	converted_auxiliary_data[i]	fl array	N/A	4	aux_tot
L1B-INT-5	auxdata(i)	unpacked_auxiliary_data[i]	ss array	N/A	2	aux_tot
L1B-INT-6	cal_inv(i)	calibration_invalid[channel]	ss	N/A	2	7
(local)	meana	(Definition TBD RAL)	fl	K	4	1
(local)	meanb	(Definition TBD RAL)	fl	K	4	1

Table 5-5-2: Internal Data Table - Validating the converted auxiliary data

5.5.3 Algorithm definition.

These specialised tests are performed principally on the converted data of each auxiliary packet which has passed the packet quality check.

Step 5.5.3-1 Check the paraboloid mirror temperatures and the black body PRT counts

This test is performed on the unconverted counts of each of the following sensors: A4511, (PRT1), A4521 (PRT2), A4531 (PRT3), A4551 (PRT5), A6024, A6034, A6044, A6054, A6064, A6074 (BB2-BB7), and A6104, A6114, A6124, A6134, A6144, A6154 (BB10-BB15) as follows.

```
for each count to be tested.
  if the value is either zero or 4095 then
    OR the error value TMZ_AT_LIMIT_ERR into the
    auxiliary_data_validation_results array in the corresponding
```

```
    index for that sensor.                                (Req. 5.5-1)
  end if
end for
```

The error code will ensure that the value is not included in the calibration calculations.

Step 5.5.3-2 Test the Paraboloid mirror temperatures: A4511 A4521 and A4531 (PRT1-PRT3)

The calculation is summarised as follows:

```
Check the appropriate entries in the
auxiliary_data_validation_results array for each of the telemetry
values A4511, A4521 and A4531                             (Req. 5.5-2)


if none of the telemetry values (A4511, A4521 and A4531) have been
   excluded from the calibration then
```

Calculate the mean of the three values

```
   mean = (A4511 + A4521 + A4531) / 3                     (Req. 5.5-3)
```

Compare the actual value of each sensor with the mean. If the absolute difference is greater than prt_mean_difference, ensure that particular BB data is not used for the calibration.

```
   for n = 1 to 3
     diff = absolute(A45n1 - mean)                        (Req. 5.5-4)
     if diff > L1-AUX4-1 then
       advr[A45n1] =
       advr[A45n1] OR TMZ_ROGUE_PRT_ERR                   (Req. 5.5-5)
     end if
   end for
end if
```

Step 5.5.3-3 Black body over range check.

```
for each of the following black body values: (A6020, A6030, A6040,
      A6050, A6060, A6070 (BB2-BB7), and A6100, A6110, A6120, A6130,
      A6140, A6150 (BB10-BB15)

  if the error code in the corresponding element of the
    auxiliary_data_validation_result (L1B-INT-3) array is less than
    TMZ_CALIBRATION_ERR then

      if the corresponding over range bit for that sensor is set then
        flag that particular black body sensor data as not to be used
        for the calibration by logically ORing the error code
        TMZ_BB_OVERRANGE_ERR into the corresponding element of the
        auxiliary_data_validation_result array.
                                                          (Req. 5.5-6)
```

Note that the over range bit referred to in Req. 5.5-6 above is to be found in the fields A6022, A6032, A6042, A6052, A6062, A6072 (for BB2 - BB7 respectively), or A6102, A6112, A6122, A6132, A6142, A6152 (for BB10 - BB15 respectively).

```
      end if
  end if
end for
```

Step 5.5.3-4 Thoroughly test each black body sensor temperature.

```
Check the value of A6177 (BB17) in the unpacked data array
[L1B-INT-005], to ensure that at least one of the BB heaters is
on.                                                 (Req. 5.5-7)
```

Note on the interpretationn of A6177: Telemetry ID A6177 represents the 'Main Heater Selected - Voltage Status' field of the BB Status telemetry word BB17. The structure of this word (omitting the interpretation for fields that are not actually used by the processor) is defined in document[AD6] as follows (Figure 5.5-2).

Figure 5.5-2. Structure of the BB Status telemetry word BB17.

J1	H1	G4	G3	G2	G1	F1	E1	D1	C2	C1	B2	B1	A3	A2	A1

where

J1 = Main Heater Current Status;
H1 = Boost Heater Current Status;
G4 - G1 = Temperature Channel Ident Field;
F1 = +XBB Range Status (0 = Cold; 1 = Hot);
E1 = -XBB Range Status (0 = Cold; 1 = Hot);
D1 = 0;
C2 - C1 = Boost Heater Selected - Voltage Status (Interpretation as B2 - B1);
B2 - B1 = Main Heater Selected - Voltage Status:
00 = Off
01 = On +XBB
10 = On -XBB
11 = Error
A3 - A1 Main Heater Power level Selection.

A6177 represents the two-bit field represented by bits B2, B1. The condition leading to the execution of Req 5.5-8 below is that the two bit field is 00 or 11. (i.e. both heaters are off or some other error condition prevents the identification of a valid hot BB).

```
if both heaters are off
```

An error condition has been detected - no data can be calibrated.

```
   for i = each of the following black body values: (A6020, A6030,
      A6040, A6050, A6060, A6070 (BB2-BB7), and A6100, A6110, A6120,
      A6130, A6140, A6150 (BB10-BB15)
      Logically OR the auxilary_data_validation_result[i] with the
      error code TMZ_ROGUE_BB_ERR                       (Req. 5.5-8)
   end for
else
Determine from A6177 which is the hot black body and which the cold.
```

```
      If the heated black body is PXBB then
        PBB_PRT_MEAN_DIFF = [L1-AUX4-2] (MAX_HBB_PRT_MEAN_DIFF)
        MBB_PRT_MEAN_DIFF = [L1-AUX4-3] (MAX_CBB_PRT_MEAN_DIFF)
      else
        PBB_PRT_MEAN_DIFF = [L1-AUX4-3]
        MBB_PRT_MEAN_DIFF = [L1-AUX4-2]
      endif                                              (Req. 5.5-9)
end if
```

```
If no error has occurred it is now possible to begin the remaining
specific checks.
```

Step 5.5.3-4-1. Calculate the weighted mean temperature of both hot and cold black bodies.

Note. The purpose of the weighted mean algorithm is to incorporate redundancy, so that the mean value can be derived, even if one (or more) sensors fail.

Calculate the values of n1, n2 and n3, where n1, n2 and n3 are the total number of **valid values** included in the sum. A valid value is one for which the error code in the corresponding element of the aux_data_validation_result array is less than TMZ_CALIBRATION_ERR (Req. 5.5-10)

Calculate the weighted mean temperature the plus X BB as follows:

```
meana = (conv_aux_data[A6020] + conv_aux_data[A6030]) / n1
                                                    (Req. 5.5-11)
meanb = (conv_aux_data[A6040] + conv_aux_data[A6050] +
         conv_aux_data[A6060]) / n2
                                                    (Req. 5.5-12)
weighted mean temperature (PXBB) = (meana + meanb) / n3
                                                    (Req. 5.5-13)
```

where n1, and n2 are the total number of **valid values** included in the respective sums, and n3 = 2 if both sums are valid (i.e. n1 > 0 and n2 > 0), or n3 = 1 if either n1 = 0 or n2 = 0. Note that if n1 = n2 = 0 no valid temperatures have been recorded and the calibration cannot proceed.

And now calculate the weighted mean temperature the minus X BB as follows:

```
meana = (conv_aux_data[A6100] + conv_aux_data[A6110]) / n1
                                                    (Req. 5.5-14)
meanb = (conv_aux_data[A6120] + conv_aux_data[A6130] +
         conv_aux_data[A6140]) / n2
                                                    (Req. 5.5-15)
weighted mean temperature (PXBB) = (meana + meanb) / n3
                                                    (Req. 5.5-16)
```

Step 5.5.3-4-2. Test each sensor value on the PXBB.

```
For each sensor on the pxbb
Calculate the absolute difference (abs_diff) between the temperature
of each sensor and the weighted mean BB temperature (derived in step
5.5.3-4-1)                                          (Req. 5.5-17)
```

```
   if abs_diff is greater than PBB_PRT_MEAN_DIFF then
```
Flag that sensor data as not to be used for the calibration, as follows:
```
      logically OR the error code TMZ_ROGUE_BB_ERR into the
      element of the auxiliary_data_validation_result array
      which corresponds to that sensor              (Req. 5.5-18)
   end if
end for
```

Step 5.5.3-4-3. Test each sensor value on the MXBB.

```
For each sensor on the MXBB
Calculate the absolute difference (abs_diff) between the temperature
of each sensor and the weighted mean BB temperature (derived in step
5.5.3-4-1)                                             (Req.5.5-19)

   if abs_diff is greater than MBB_PRT_MEAN_DIFF then
```
Flag that sensor data as not to be used for the calibration, as follows:
```
      logically OR the error code TMZ_ROGUE_BB_ERR into the
      element of the auxiliary_data_validation_result array
      which corresponds to that sensor               (Req.5.5-20)
   end if
end for
```

Step 5.5.3-5 Surveillance limits check.

Test all converted items for which maximum and minimum surveillance limits are specified (See AD 8).

Step 5.5.3-5-1. Reading in the surveillance limits.

The limit values, (together with their corresponding identifier), are read in once at system initialisation time.

```
openfile (surveillance_limits data)                    (Req.5.5-21)
i = 0                                                  (Req.5.5-22)
while not end of file (surveillance_limits data)
   read aux_surv_id[i], surv_lwb[i], surv_upb[i]       (Req.5.5-23)
   i = i + 1                                           (Req.5.5-24)
end while
n_surv_tot = number of records read in                 (Req.5.5-25)
closefile (surveillance_limits data)                   (Req.5.5-26)
```

Note. A single record contains the following fields: aux_surv_id, surv_lwb and surv_upb.

Step 5.5.3-5-2. Performing the surveillance limits check.

```
For i = 0 to n_surv_tot - 1, increment=1
```

(Determine the id of the auxiliary item to be tested.)

```
aux_id_to check = aux_surv_id[i]                (Req.5.5-27)
```

(Determine the converted value of that item)

```
value_to check = value in the converted_auxiliary_data array,
having the same auxiliary identifier as aux_id_to check
                                                (Req.5.5-28)
```

(Compare the value of the value_to_check, with the surveillance limits)

```
if value_to_check < surv_lwb[i] OR
   value_to_check > surv_upb[i] then
  if (value_to_check is derived from one of the following black
    body temperature values: A6020, A6030, A6040, A6050,
    A6060, A6070 A6100, A6110, A6120, A6130, A6140, A6150)
then
    logically OR the error code TMZ_BB_OUT_OF_LIMIT_ERR into
    the appropriate element of the
    auxiliary_data_validation_result array     (Req.5.5-29)
  else
    logically OR the error code TMZ_SURVEILLANCE_ERR into the
    appropriate element of the
    auxiliary_data_validation_result array     (Req.5.5-30)
  end if
end if
end for
```

Step 5.5.3-6 Test on A4580, the reference PRT 8. (Performed on unconverted counts.)

```
if the auxiliary_data_validation_result[A4580] (PRT8) is less than
      TMZ_CALIBRATION_ERR then
```

(Check the unconverted count)

```
if unpacked_aux_data[A4580]is outside the range PRT8_FIXED_VAL ±
  PRT8_VARIANCE, then
  for i = each of the following identifers A4551 (the FPA
        baffle temperature PRT5), A4511, A4521, and
        A4531,(parabaloid PRTs 1 to 3),
        logically OR the error code TMZ_PRT8_ERR with
        auxiliary_data_validation_result[i]      (Req.5.5-31)
     end for
end if
end if
```

Note. The error code added is not enough on its own to qualify the temperature for exclusion from the calculation of the calibration parameters, but since the error for this condition may be added to an existing error code, it is possible that the combined error may take the error value over the threshold for inclusion.

Step 5.5.3-7 Test the SCP values.

Step 5.5.3-7-1 Test the converted values of IR channel detector temperatures SCP2 -SCP6.

(Derive the maximum difference between the SCP temperatures)

```
max_diff = the maximum difference between all temperatures converted
from the auxiliary items A5021, A5031, A5041, A5051, and A5061 (SCP2
- SCP6)                                                 (Req.5.5-32)

For each of the temperatures converted from the auxiliary items
A5021, A5031, A5041, A5051, and A5061 (SCP2 - SCP6)
   if the auxiliary_data_validation_result error code value error
      corresponding to this auxiliary item is less than
      TMZ_CALIBRATION_ERR, then
      if max_diff is greater than SCP2_6_TEMP_DIFF then
         OR the error value TMZ_ROGUE_SCP_ERR into the appropriate
         error result array element in
         auxiliary_data_validation_result for this auxiliary item
                                                        (Req.5.5-33)
```

(Note. The error code added is not enough on its own to qualify the temperature for exclusion from the calculation of the calibration parameters, but since the error for this condition may be added to an existing error code, it is possible that the combined error may take the error value over the threshold for inclusion.)

```
      end if
   end if
end for
```

Step 5.5.3-7-2 Test the visible channel detector temperatures SCP7-10.

(Derive the maximum difference between the SCP temperatures)

```
max_diff = the maximum difference between all temperatures converted
from the auxiliary items A5071, A5081, A5091, and A5101 (SCP7 -
SCP10)                                                  (Req.5.5-34)
For each of the temperatures converted from the auxiliary items
A5071, A5081, A5091, and A5101 (SCP7 - SCP10)

   if the auxiliary_data_validation_result error code value error
      corresponding to this auxiliary item is less than
      TMZ_CALIBRATION_ERR, then
      if max_diff is greater than SCP7_10_TEMP_DIFF then
         OR the error value TMZ_ROGUE_SCP_ERR into the appropriate
         error result array element in
         auxiliary_data_validation_result for this auxiliary item
                                                        (Req.5.5-35)
```

(Note. The error code added is not enough on its own to qualify the temperature for exclusion from the calculation of the calibration parameters, but since the error for this condition may be added to an existing error code, it is possible that the combined error may take the error value over the threshold for inclusion.)

```
      end if
```

```
      end if
   end for
```

Step 5.5.3-8 Scan jitter check.

After the pixel data have been successfully unpacked, it is necessary to test for "scan jitter" errors. Scan jitter is a condition thought to be caused by a mechanical judder during the rotation of the mirror. The detection of scan jitter would necessitate the exclusion of the complete packet from the calibration This test is considered to be appropriate for ATSR-2. Whether or not it is also appropriate for AATSR is to be confirmed (TBC).

```
if the value of auxiliary data item A2110, (IDF11 snapshot register 2
      setting) is pointing at pixel number 2001, and the source
      sequence count for this and the previous packet are consecutive
      then

  for each channel
    compare the snapshot register 2 for this channel with that from
    the previous packet
    if they are different
       scan jitter has been detected
       packet_error = PIX_SCAN_JITTER_ERR             (Req.5.5-36)
    end if
  end for
end if
```

5.6 Module Definition: Science Data Processing.

Within this module a "word" is defined to mean 16 bits.

Processing the science data consists of unpacking the science data and validating the data. These component tasks are most effectively processed during a single pass of the data. They are discussed in the following sub sections.

5.6.1 Functional description.

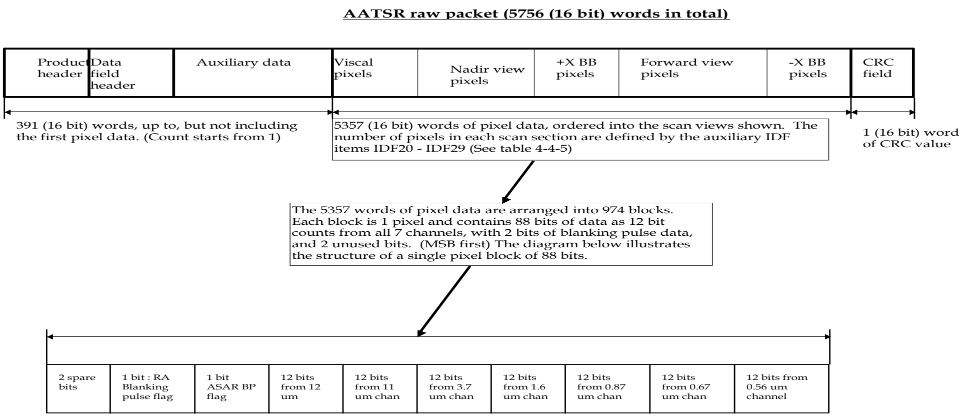
The science (or pixel) data contains all seven channels at full 12 bit resolution. Within each packets the data is arranged in blocks of 88 bits, in a fixed order view round the scan; the order being viscal, nadir view, plus X black body, along track (or forward) view, and finally minus X black body pixels. Information in the auxiliary IDF data (see Table 5-6-3) defines which 974 of the 2000 available pixels in the full scan have been selected from each component view for down linking in the AATSR packet. Within each block of 88 bits, the 12 bit radiance count from each channel is also present in a fixed order: spare bits, then blanking pulse, then 12 µm, 11µm, 3.7µm, 1.6 µm, 0.87µm. 0.67µm and 0.56 µm (AD 7, pp 19).

The science data also contains two blanking pulse flags for each pixel; each one requires one bit. The first is used to indicate if the Radar Altimeter is operational at the time the AATSR data was collected, and the second flags the same condition for the ASAR instrument. This information is required because the user may wish to exclude from the calibration, all data that was taken when either or both the Radar Altimeter and/or the ASAR instruments are operational.

The 12 bit data from each channel is unpacked into one of five “scan position” integer arrays for each channel, (viscal_pixels[channel][i], nadir_pixels[channel][i], pxbb_pixels[channel][i], frwrd_pixels[channel][i] or mxbb_pixels[channel][i]) depending on the pixel number of the pixel data being unpacked. The pixel number is determined from the auxiliary data items A2200, A2210, A2220….A2290 (IDF20 - IDF29) inclusive. Using these values, it is also possible to determine the parity of any pixel. Since the first pixel number in any scan section is known, its parity is known. and since all pixels in any scan section are contiguous, the pixel numbers (and thus their parity) can be derived for any pixel in any part of any scan section. The blanking pulse data is unpacked into a similar set of “scan position” integer arrays.

During the unpacking process, each count is tested for saturation or a zero value. If the actual count detected originally was saturated, the saturated 12 bit value would be 4095. If either of these conditions are detected the value of the count itself is changed to (a negative error number) indicate which of these two conditions were detected. Strictly, these conditions may not be errors, nor indeed may they be saturated values, since a value of 4095 may be perfectly valid. However, since either of these conditions **may** indicate an inaccurate reading, these two states need to be flagged. This is particularly important for the black body pixel data which is essential to the calibration. (See Algorithm definition for details)

Fig. 5.6-1 Structure of the science data in the raw packet



5.6.2 Interface definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L0-MDS1-11		Science (pixel) data	us array	N/A	2	5357
L1-AUX-18-1		blanking_pulse_calibration_flag blanking pulse calibration code	ss	N/A	2	1
L1-AUX-18-23		PIXEL_COUNT_ZERO	ss	N/A	2	1
L1-AUX-18-24		PIXEL_COUNT_SATURATED	ss	N/A	2	1
L1-AUX-18-19		PIXEL_COUNT_INITIAL_VALUE	ss	N/A	2	1
L1-AUX-18-43		BB_PIX_COUNT_OUT_OF_RANGE_ERR	ss	N/A	2	1
L1-AUX-18-44		BB_PIX_COUNT_OUT_OF_RANGE_ALL_CHA	ss	N/A	2	1

		NS				
L1-AUX18-4		max_viscal_pixels	ss	none	2	1
L1-AUX18-5		max_nadir_pixels	ss	none	2	1
L1-AUX18-6		max_pxbb_pixels	ss	none	2	1
L1-AUX18-7		max_frwrd_pixels	ss	none	2	1
L1-AUX18-8		max_mxbb_pixels	ss	none	2	1
L1-AUX18-9		Cal. bp flag value: both bp flags off	ss	n/a	2	1
L1-AUX18-10		Cal. bp flag value: ASAR only flag off	ss	n/a	2	1
L1-AUX18-11		Cal. bp flag value: RA only flag off	ss	n/a	2	1
L1-AUX18-12		Cal. bp flag value: both bp flags on	ss	n/a	2	1
L1-AUX18-13		INIT_CAL_PARAM	fl	n/a	4	1

Table 5-6-1: Input Data Table - Unpacking and validating the science data

The units, range and step size values of converted_auxiliary_data are too varied to be usefully defined. See AD 8 for details.

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-001	packet_error		ss	N/A	2	1
L1B-INT-004	conv_aux_data(i)	converted_auxiliary_data[i]	fl array	N/A	4	aux_tot
L1B-INT-005	auxdata(i)	unpacked auxiliary data	us	n/a	2	aux_tot
L1B-INT-006	cal_inv(i)	calibration_invalid[channel]	ss	N/A	2	7
L1B-INT-84	viscal_pixels[channel][i]	unpacked.pixels.viscal	array of array of ss	N/A	2	max_viscal_pixels * 7
L1B-INT-80	nadir_pixels[channel][i]	unpacked.pixels.nadir	array of array of ss	N/A	2	max_nadir_pixels * 7
L1B-INT-82	pxbb_pixels[channel][i]	unpacked.pixels.plus_bb	array of array of ss	N/A	2	max_pxbb_pixels * 7
L1B-INT-81	frwrd_pixels[channel][i]	unpacked.pixels.forward	array of array of ss	N/A	2	max_frwrd_pixels * 7
L1B-INT-83	mxbb_pixels[channel][i]	unpacked.pixels.minus_bb	array of ss	N/A	2	max_mxbb_pixels * 7
L1B-INT-95	viscal_blank[i]	unpacked.blanking.viscal	array of ss	N/A	2	max_viscal_pixels
L1B-INT-91	nadir_blank[i]	unpacked.blanking.nadir	array of ss	N/A	2	max_nadir_pixels
L1B-INT-93	pxbb_blank[i]	unpacked.blanking.plus_bb	array of ss	N/A	2	max_pxbb_pixels
L1B-INT-92	frwrd_blank[i]	unpacked.blanking.forward	array of ss	N/A	2	max_frwrd_pixels
L1B-INT-94	mixbb_blank[i]	unpacked.blanking.minus_bb	array of ss	N/A	2	max_mxbb_pixels
L1B-INT-170	IDF20	pixel_map_viscal_start_pixel	ss	N/A	2	1
L1B-INT-171	IDF21	pixel_map_viscal_end_pixel	ss	N/A	2	1
L1B-INT-172	IDF22	pixel_map_nadir_start_pixel	ss	N/A	2	1
L1B-INT-173	IDF23	pixel_map_nadirl_end_pixel	ss	N/A	2	1
L1B-INT-174	IDF24	pixel_map_pxbb_start_pixel	ss	N/A	2	1
L1B-INT-175	IDF25	pixel_map_pxbb_end_pixel	ss	N/A	2	1
L1B-INT-176	IDF26	pixel_map_along_track_start_pixel	ss	N/A	2	1
L1B-INT-177	IDF27	pixel_map_along_track_end_pixel	ss	N/A	2	1
L1B-INT-178	IDF28	pixel_map_mxbb_start_pixel	ss	N/A	2	1
L1B-INT-179	IDF29	pixel_map_mxbb_end_pixel	ss	N/A	2	1

Table 5-6-2: Internal Data Definitions - Unpacking and validating the science data

Note. The channel index array varies to allow an index into all the channels: 12 µm, 11 µm , 3.7 µm, 1.6 µm, 0.870 µm, 0.670 µm, and 0.555 µm. The unpacked pixel data is more easily handled in a 2 dimensional array, where the second dimension is the channel identifier, rather than in seven separate arrays with one for each channel. However, this is a decision for the system design/implementation.

5.6.3 Algorithm definition.

Step 5.6.3-1 Read the value of the blanking_pulse_calibration_flag

The value defines which pixel data is to be included in the calibration, according on the value of the ASAR and RA blanking pulse flags.

```
Once only (at system initialisation time), read in the value of the
blanking_pulse_calibration_flag.                          (Req.5.6-1)
```

The value may be one of the following:

[L1-AUX18-9]: INCL_PIXEL_DATA_FOR_WHICH_BOTH_ASAR_AND_RA_BP_FLAGS_ARE_OFF

[L1-AUX18-10]: INCL_PIXEL_DATA_FOR_WHICH_ASAR_BP_FLAG_ONLY_IS_OFF

[L1-AUX18-11]: INCL_PIXEL_DATA_FOR_WHICH_RA_BP_FLAG_ONLY_IS_OFF

[L1-AUX18-12]: INCL_PIXEL_DATA_FOR_WHICH_BOTH_ASAR_AND_RA_BP_FLAGS_ARE_ON

Step 5.6.3-2 Initialise the blanking pulse output arrays for each packet.

```
Initialise to zero, the set of scan component integer arrays (L1B-
INT-95 to L1B-INT-94 incl.) into which the unpacked blanking pulse
counts will be stored.                                    (Req.5.6-2)
```

Step 5.6.3-3 Unpack the pixel counts and testing for zero and saturated signal.

The following pixel map values are required in order to unpack the pixel counts. Note that these items should be taken from the array of unpacked (unconverted) auxiliary data [L1B-INT-005], since no conversion is defined for them.

Telemetry mnemonic	Telemetry identifier	Telemetry description
A2200	IDF20	Pixel Map Readout (Viscal Start pixel number)
A2210	IDF21	Pixel Map Readout (Viscal End pixel number)
A2220	IDF22	Pixel Map Readout (Nadir Start pixel number)
A2230	IDF23	Pixel Map Readout (Nadir End pixel number)
A2240	IDF24	Pixel Map Readout (+XBB Start pixel number)
A2250	IDF25	Pixel Map Readout (+XBB End pixel number)
A2260	IDF26	Pixel Map Readout (Along Track Start pixel number)
A2270	IDF27	Pixel Map Readout (Along Track End pixel number)
A2280	IDF28	Pixel Map Readout (-XBB Start pixel number)
A2290	IDF29	Pixel Map Readout (-XBB End pixel number)

Table 5-6-3. Auxiliary data items defining the pixel map

Note. By reference to the pixel map values, and since pixel numbering is contiguous in each scan section, it is always possible to determine the pixel number of any given pixel in any of the scan sections. From this information, it is also possible to determine the parity of any pixel in any of the scan sections from its pixel number. (Pixel numbering starts at 1(rather than 0))

To unpack all pixel data from a source packet, step through each of the 974 blocks of 88 bits in the raw packet in turn, maintaining an index into the 974 blocks. The pixel data in the packet are arranged in contiguous scan section order (viscal, nadir, pxbb, forward, mxbb), see figure. 5.6-1

The following algorithm defines how to unpack the pixel data and the blanking pulse data from the raw packet into the output buffers.

```
set pixel_pointer into the raw packet to point at the first bit of
the first pixel block of 88 bits
pixel_pointer = (398 * 16) + 1                          (Req.5.6-3)

n = 0 (Req.5.6-4)
for each section of the scan (viscal, nadir, pxbb, forward, mxbb)
  set start of scan pixelno = value of IDF2n            (Req.5.6-5)
  set end_of_scan_pixelno = value of IDF2(n+1)          (Req.5.6-6)

  determine the total_pixels_in_this_scan section =
      end_of_scan_pixelno - start of scan pixelno + 1  (Req.5.6-7)
```

Define the offset into the unpacked data buffer `k_offset`:

```
If nadir scan section (i.e. if n = 2) then
      k_offset = IDF22 - FIRST_NADIR_PIXEL_NUMBER
else if forward scan section (i.e. if n = 6) then
      k_offset = IDF26 - FIRST_FORWARD_PIXEL_NUMBER
else k_offset = 0
end if
```

(Now step through the pixels until all pixels in this scan section have been unpacked as follows):

```
  for i = 0; i < total_pixels_in_this_scan section; increment=1
    unpack the blanking pulse bits from this pixel into the
    element of the appropriate scan section buffer indexed by
    [i + k_offset].                                     (Req.5.6-8)

    for each channel (12, 11, 3.7, 1.6, 0.87, 0.67, 0.56)
      unpack this pixel's data into the element of the scan
      section buffer for the appropriate channel and indexed
      by [i + k_offset].                                (Req.5.6-9)
```

(If the scan section corresponds to the viscal, pxbb, or mxbb, the pixel data is unpacked into the element i of the buffer. If the scan corresponds to the nadir view, the pixel is unpacked into the element of the buffer indexed by i + IDF22 - FIRST_NADIR_PIXEL_NUMBER.

If the scan corresponds to the forward view, the pixel should be unpacked into the element of the buffr indexed by i + IDF26 - FIRST_FORWARD_PIXEL_NUMBER. These provisions ensures that in the event of a change in the pixel map, there is no change in the relative location of the earth view pixels in the buffers.)

```
      Test the unpacked values as follows:
      if the unpacked pixel count is zero
         set the pixel value to PIXEL_COUNT_ZERO       (Req.5.6-10)
      else if the unpacked count is 4095
         set the pixel value to PIXEL_COUNT_SATURATED (Req.5.6-11)
      end if
    end for
    increment the pixel pointer to point at the first bit of the
    next pixel block (of 88 bits)
```

```
    pix_pointer = pix_pointer + 88                    (Req.5.6-12)
  end for

  Increment n to point at the next scan section pair of pixel map idf
  variables
  n = n + 2                                           (Req.5.6-13)
end for
```

It is especially important to check the black body pixels:

```
for all channels
  if any black body pixels (from either black body) are either zero
      or 4095 then
     calibration_invalid[channel] =
          BB_PIX_COUNT_OUT_OF_RANGE_ERR               (Req.5.6-14)
  end if
end for


if the value of calibration_invalid[channel] is
      BB_PIX_COUNT_OUT_OF_RANGE_ERR for all channels, then
   the entire packet is excluded from the calibration by setting
   packet_error = BB_PIX_COUNT_OUT_OF_RANGE_ALL_CHANS (Req.5.6-15)
end if
```

5.7 Module Definition: Infra-Red Channel Calibration.

Derivation of the calibration parameters requires the calculation of the gain and offset values which define a linear relationship between pixel count and radiance. The gain and offset are derived for each channel, from the unpacked and validated auxiliary and pixel data averaged over a number of contiguous data packets. The actual number of packets used is defined as the “calibration period”.

5.7.1 Functional Description

The IR calibration module is a framework in which all the modules described in earlier sections are used. Up until this point all modules have been concerned with processing component data from a single source packet. The IR calibration calculations are concerned with using the unpacked, converted and validated auxiliary and pixel data generated by the modules previously described, to calculate average black body temperatures and average black body pixel counts over a series of contiguous packets. The number of packets over which these values are averaged is called the “calibration period.” This is described in more detail in section 5.7.1.1

Once the average values for both black body temperatures have been calculated from the data available in the processed source packets in a calibration period, and the corresponding black body pixel counts have been similarly averaged, it is then possible to use these values to derive the gain and offset parameters which can then be used to convert pixel count to radiance for all valid pixels in all valid packets within the calibration period. This is described in greater detail in section 5.7.1.2

5.7.1.1 Overview of a calibration period

The number of packets in a calibration period is a system parameter, (calibration_period), read in at initialisation time, and shall not be hard coded A calibration period will always contain data averaged from calibration_period packets, unless the data to be used are interrupted by a significant event. Thus a calibration period is considered complete if any of the following conditions are detected:

- All source packets in a calibration period have been processed.
- A change in the pixel map (auxiliary data item A2190) between the current and previous packet.
- A change in any of the on-board calibration parameters for any IR channel (Auxiliary data items A0040, A0050, A0060, A0070, A0081, A0091, A0101, A0111) between the current and previous packet.
- End of data.

When any of the above conditions are detected, the calibration period is considered to be complete, and the calculation of the calibration parameters is performed on the subset of packets for whom these specified data items are unchanged.

5.7.1.2 Summary of the calculations of calibration parameters

A mean value for each black body (MX and PX) temperature (calculated only from valid values from the converted_auxiliary_data) is derived for each packet in the calibration period. From all the mean values for each packet in the calibration period, a mean of means is derived; i.e. a single temperature for each black body, representative of the BB temperatures during that calibration period. Both mean black body temperatures are then converted to a radiance value for each (infra red) channel.

Similarly a mean value is derived for the odd and even black body pixel counts, for each black body, and for each channel, from all valid black body pixel counts in the calibration period. The mean value for each packet is summed, for all packets in the calibration period, until a single mean pixel count value is derived for each of the odd and even black body pixel counts, for each black body, and for each channel (Separate odd and even means are derived because the odd and even counts use separate integrators.) It is possible to exclude pixel data from the calibration which was taken when either (or both of) the Radar Altimeter and/or the ASAR instrument were operational This is determined by the value of the blanking_pulse_calibration_flag.

The look-up table for conversion from temperature to radiance contains a correction for detector non-linearity to ensure a linear relationship between radiance and detector counts. It is therefore possible to derive the calibration parameters for each IR channel from the straight line formula y = m.x + c. The mean (odd and even) black body pixel counts for the two black bodies are the two x axis values, and the two black body radiance values make up the corresponding y axis values.

The gain and offset values are derived for both odd and even pixel numbers, because AATSR uses two integrators; one for odd pixels and one for even pixels. The response time of a single integrator is too slow to use the same one for all pixels.

5.7.2 Interface definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX18-2		calibration_period	su	no of packets	4	1
L1-AUX18-1		blanking_pulse_calibration_flag	ss	N/A	2	1
L1-AUX27-1-3		mxbb_emissivity[channel]	array of do	none	8	3
L1-AUX27-21-23		pxbb_emissivity[channel]	array of do	none	8	3
L1-AUX5-1		temperature_lut[i]	array of do	Deg K	8	size_of_lut
L1-AUX5-2		radiiance_lut 12 micron[i]	array of do	$Wcm^{-2}sr^{-1}$	8	size_of_lut
L1-AUX5-3		radiiance_lut 11 micron[i]	array of do	$Wcm^{-2}sr^{-1}$	8	size_of_lut
L1-AUX5-4		radiiance_lut 3.7 micron[i]	array of do	$Wcm^{-2}sr^{-1}$	8	size_of_lut
L1-AUX27-4		size_of_lut	su	N/A	4	1
L1-AUX18-29		RAWPKT_FAILS_AUXILIARY_DATA_PROCESSING_ERR	ss	N/A	2	1
L1-AUX18-22		PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSSED	ss	N/A	2	1
L1-AUX18-30		TEMP_OUT_OF_RANGE_FOR_LUT_ERR	ss	N/A	2	1
L1-AUX18-36		TMZ_CALIBRATION_ERR	us	N/A	2	1
constant		SUCCESS	ss	N/A	2	1
L1-AUX18-4		max_viscal_pixels	ss	none	2	1
L1-AUX18-5		max_nadir_pixels	ss	none	2	1
L1-AUX18-6		max_pxbb_pixels	ss	none	2	1
L1-AUX18-7		max_frwrd_pixels	ss	none	2	1
L1-AUX18-8		max_mxbb_pixels	ss	none	2	1
L1-AUX18-9		Cal. bp flag value: both bp flags off	ss	n/a	2	1
L1-AUX18-10		Cal. bp flag value: ASAR only flag off	ss	n/a	2	1
L1-AUX18-11		Cal. bp flag value: RA only flag off	ss	n/a	2	1
L1-AUX18-12		Cal. bp flag value: both bp flags on	ss	n/a	2	1
L1-AUX18-13		INIT_CAL_PARAM	fl	n/a	4	1
L1-AUX27-5		Increment in Temperature to Radiance LUT	do	K	8	1
L1-AUX27-6		First value in Temperature to Radiance LUT	do	K	8	1
L1-AUX27-7		Last value in Temperature to Radiance LUT	do	K	8	1

Table 5-7-1: Input Data Table - Calculation of calibration parameters: infra red channels

Note. The channel index array varies to allow an index into all the IR channels: 12µm, 11µm, and 3.7 µm. The parity index array varies to allow an index for both odd and even pixel counts.

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-1	packet_error		ss	N/A	2	1
L1B-INT-4	conv_aux_data	converted_auxiliary_data[i]	fl array	N/A	4	aux_tot
L1B-INT-6	cal_inv(i)	calibration_invalid[channel]	ss	N/A	2	7
L1B-INT-82	pxbb_pixels[channel][i]	unpacked.pixels.plus_bb	array of array of ss	N/A	2	max_pxbb_pixels * 7
L1B-INT-83	mxbb_pixels[channel][i]	unpacked.pixels.minus_bb	array of ss	N/A	2	max_mxbb_pixels * 7

L1B-INT-93	pxbb_blank[i]	unpacked.blanking.plus_bb	array of ss	N/A	2	max_pxbb_pixels
L1B-INT-94	mixbb_blank[i]	unpacked.blanking.minus_bb	array of ss	N/A	2	max_mxbb_pixels
L1B-INT-10	ch_gain[i,j]	gain[parity][channel]	array of array of fl	TBD	4	7 * 2
L1B-INT-11	ch_offset[i,j]	offset[parity][channel]	array of array of fl	TBD	4	7 * 2
(local)		mean_px_bb_temperature	do	Deg K	8	1
(local)		mean_mx_bb_temperature	do	Deg K	8	1
(local)		mean_cal_period_bground_temp	do	Deg K	8	1
(local)		mean_px_bb_radiance[channel]	do array	$Wm^{-2} sr^{-1}$	8	3
(local)		mean_mx_bb_radiance[channel]	do array	$Wm^{-2} sr^{-1}$	8	3
(local)		mean_bb_pixel_count[channel][bb][parity]	array of array of array of fl	TBD	4	28
(local)		bground_radiance[channel]	do array	$Wm^{-2} sr^{-1}$	8	3
(local)		end_of_data	ss	N/A	2	1
(local)		end_calibration_period	ss	N/A	2	1
(local)		no_of_packets_processed_this_cal_period	ss	N/A	2	1
(local)		bb_pixel_count_cal_period_sum[chan][bb][parity]	do	N/A	8	28
(local)		bb_pixel_count_cal_period_tally[chan][bb][parity]	ss	N/A	4	28
(local)		mean_bb_pixel_count_cal_period[chan][bb][parity]	fl	N/A	4	28
(local)		sum_wtd_mean_px	fl	N/A	4	1
(local)		tally_of_weighted_means_px	ss	N/A	2	1
(local)		sum_wtd_mean_mx	fl	N/A	4	1
(local)		tally_of_weighted_means_mx	ss	N/A	2	1
(local)		mean_cal_period_bground_temp_sum	fl	N/A	4	1
(local)		bground_cp_tally	ss	N/A	2	1
L1B-INT-170	IDF20	pixel_map_viscal_start_pixel	ss	N/A	2	1
L1B-INT-171	IDF21	pixel_map_viscal_end_pixel	ss	N/A	2	1
L1B-INT-172	IDF22	pixel_map_nadir_start_pixel	ss	N/A	2	1
L1B-INT-173	IDF23	pixel_map_nadirl_end_pixel	ss	N/A	2	1
L1B-INT-174	IDF24	pixel_map_pxbb_start_pixel	ss	N/A	2	1
L1B-INT-175	IDF25	pixel_map_pxbb_end_pixel	ss	N/A	2	1
L1B-INT-176	IDF26	pixel_map_along_track_start_pixel	ss	N/A	2	1
L1B-INT-177	IDF27	pixel_map_along_track_end_pixel	ss	N/A	2	1
L1B-INT-178	IDF28	pixel_map_mxbb_start_pixel	ss	N/A	2	1
L1B-INT-179	IDF29	pixel_map_mxbb_end_pixel	ss	N/A	2	1

Table 5-7-2: Internal Data Table - Calculation of calibration parameters: infra red channels

5.7.3 Algorithm Definition

Up until now, all the modules defined perform operations on a single source packet or its derived products. The calibration is different however, since it requires the use of accumulated (valid) data from several packets to derive the calibration parameters. Consequently it is necessary to accumulate the appropriate data before the calibration parameters can be derived.

The algorithm below describes the overview of the calibration process, and puts the calibration calculations into context with the processing performed by the other modules described in previous sections of this document. The main steps of the algorithm are identified, and these steps are described in greater detail in the subsequent sections.

Note For the sake of clarity error handling is **not** shown in the algorithm below.

```
end_of_data = FALSE                                    (Req.5.7-1)
```

```
While not end of data

  Step 1. Initialisations
  Initialise the sums and tallies (and any other variables used in
  calculating the calibration parameters every calibration period.

  While NOT end_calibration_period

    Step 2. Test for normal completion of a cal period
    Normal completion is when all source packets in a cal_period have
    been processed.

    if NOT end_calibration_period then

      Step 3. Process the data from a single source packet
      Unpack, validate and convert the auxiliary data and unpack the
      pixel data from the current source packet

      Increment the number of packets processed.
      no_of_packets_processed_this_cal_period =
          no_of_packets_processed_this_cal_period + 1  (Req.5.7-2)

      Step 4. Test for early termination of the calibration period.
      Compare specific values between the current packet and the
      previous to determine if the calibration period should be
      terminated early.
      if NOT end_calibration_period then
        if packet_error is success then
          Step 5. Accumulate running totals of valid weighted mean BB
          temperatures for this calibration period.

          Step 6. Generate running totals of valid fore-optics
          temperatures for the calculation of the background
          radiances for this calibration period.

          Step 7. Accumulate running total of valid BB pixel counts.
        end if
      end if
    end if
  end while

  Step 8. Calculate calibration parameters for the calibration period
  End of calibration period, so calculate the calibration parameters
  for this calibration period, from accumulated data.

end while
```

Step 5.7.3-1 Initialisation

Initialise the sums and tallies (and any other variables) used in calculating the calibration parameters every calibration period.

```
no_of_packets_processed_this_cal_period = 0;           (Req.5.7-3)
end_calibration_period = FALSE                         (Req.5.7-4)
```

Initialise to zero all variables used to derive the mean BB pixel counts for this calibration period.

```
for all IR channels (3.7, 11, 12)
  for bb = PX and MX
    for parity = odd, even
      bb_pixel_count_cal_period_sum[channel][bb][parity] = 0.0
                                                       (Req.5.7-5)
      bb_pixel_count_cal_period_tally[channel][bb][parity] = 0
                                                       (Req.5.7-6)
      mean_bb_pixel_count_cal_period[channel][bb][parity] = 0.0
                                                       (Req.5.7-7)
    end for
  end for
end for
```

Initialise to zero all variables used to derive the mean BB temperatures for this calibration period:

```
sum_wtd_mean_px = 0.0                                  (Req.5.7-8)
tally_of_weighted means_px = 0                         (Req.5.7-9)
sum_wtd_mean_mx = 0.0                                 (Req.5.7-10)
tally_of_weighted means_mx = 0                        (Req.5.7-11)
mean_pxbb_temperature = 0.0                           (Req.5.7-12)
mean_mxbb_temperature = 0.0                           (Req.5.7-13)
```

Initialise to zero all variables used to derive the mean background temperatures for this calibration period.

```
mean_cal_period_bground_temp_sum = 0.0                (Req.5.7-14)
bground_cp_tally = 0                                  (Req.5.7-15)
mean_cal_period_bground_temp = 0.0                    (Req.5.7-16)
```

Initialise the output calibration parameter variables

```
for all packets in the calibration period
  for each IR channel
    for both parities (odd and even)
      gain[parity][channel] = INIT_CAL_PARAM          (Req.5.7-17)
      offset[parity][channel] = INIT_CAL_PARAM        (Req.5.7-18)
    end for
  end for
end for
```

Note. The value of INIT_CAL_PARAM used in the ATSR-2 calibration was 1E+30 for both slope and intercept, (a value which cannot be derived from calculation using valid data). This indicates to the next level of processing that these values are invalid and cannot be used.

Initialise all the black body and background radiance values for this calibration period.

```
for all IR channels (3.6, 11, 12)
   mean_mx_bb_radiance[channel] = 0.0                    (Req.5.7-19)
   mean_px_bb_radiance[channel] = 0.0                    (Req.5.7-20)
   bground_radiance[channel] = 0.0                       (Req.5.7-21)
end for
```

Step 5.7.3-2 Test for normal completion of a calibration period.

Normal completion of a calibration period occurs when all calibration_period number of source packets have been successfully processed.

```
if no_of_packets_processed_this_cal_period EQUALS
      calibration_period then
   All packets in a calibration period have been processed so
   end_calibration_period = TRUE                         (Req.5.7-22)
end if
```

Step 5.7.3-3 Process the data from a single source packet

This step uses the processing defined in the modules described in the earlier sections of this document.

```
result = success                                         (Req.5.7-23)

result = perform source packet quality checks (MODULE 1)
                                                         (Req.5.7-24)
if result NOT success then
   exclude the packet from the calibration by setting
   packet_error = result                                 (Req.5.7-25)
   for i all channels
      set calibration_invalid[i] = packet_error          (Req.5.7-26)
      set all unpacked pixel values for all parts of the scan to
       PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED         (Req.5.7-27)
   end for
else
```

The source packet has passed basic validation so unpack, validate, and convert the auxiliary data from the source packet.

```
   result = unpack the auxiliary_data (MODULE 2)         (Req.5.7-28)
   if result is success then
      validate the unpacked auxiliary data (MODULE 3) (Req.5.7-29)
      result = convert the auxiliary data to engineering units
                                        (MODULE 4)       (Req.5.7-30)
      if result is success then
         validate the converted auxiliary data (MODULE 5)
                                                         (Req.5.7-31)
      end if
```

```
      end if

      if result NOT success then
         exclude the packet from the calibration by setting
         packet_error = RAWPKT_FAILS_AUXILIARY_DATA_PROCESSING_ERR
                                                               (Req.5.7-32)
         for i all channels
           set calibration_invalid[i] = packet_error           (Req.5.7-33)
           set all unpacked pixel values for all parts of the scan to
           PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED           (Req.5.7-34)
         end for
      else
         result = process the science data (MODULE 6)          (Req.5.7-35)
      end if
   end if
```

Step 5.7.3-4 Test for early termination of the calibration period.

```
   if not first time through then
      if there is no more data to process then
         end_of_data is TRUE                                   (Req.5.7-36)
         end_calibration_period = TRUE                         (Req.5.7-37)
      else
         Compare the auto cal parameters for any of the IR channels
         from the current packet with those from the previous packet.
                                                               (Req.5.7-38)
         The auto cal parameters are obtained from items in the
         converted auxiliary data array (L1B-INT-4) which have the
         following auxiliary identifiers: A0040, A0050, A0060, A0070,
         A0081, A0091, A0101, A0111.
         if any of the autocal parameters in the current packet are
            different from those in the previous source packet then
            end_calibration_period = TRUE                      (Req.5.7-39)
         else
            compare the pixel map id (having auxiliary identifier A2190
            in the converted auxiliary data array L1B-INT-4), of the
            current source packet with the same value from the previous
            source packet.                                     (Req.5.7-40)

            if the pixel map id in the previous packet is different
             from the pixel map id in the current source packet then
               end_calibration_period = TRUE                   (Req.5.7-41)
            end if
              Requirement deleted.
                                                               (Req.5.7-42)
              Requirement deleted.
                                                               (Req.5.7-43)
         end if
      end if
   end if
```

```
Save the autocal parameters, the pixel map id, and the packet source
sequence count from this source packet to use in the end of
calibration period test for the next source packet.(Req. 5.7-43.1)
```

Step 5.7.3-5 Accumulate running totals of valid weighted mean BB temperatures for this calibration period.

Step 5.7.3-5-1 First derive the weighted mean BB temperature from the data in the current source packet.

The weighted mean BB temperatures are calculated for both hot and cold black bodies, from all valid values from one packet.

The algorithms below shows how the weighted mean is calculated for either black body.

```
For the PX BB a, b, c, d, and e are the index values for the
auxiliary telemetry items having the identifiers A6020, A6030,
A6040, A6050, and A6060 respectively.                (Req.5.7-44)

For the MX BB, a, b, c, d, and e are the index values for the
auxiliary telemetry items having the identifiers A6100, A6110,
A6120, A6130, and A6140 respectively.                (Req.5.7-45)

inner_zone_mean = 0.0                                (Req.5.7-46)
outer_zone_mean = 0.0                                (Req.5.7-47)
inner_zone_mean = (valid (conv_aux_data[a]) +
                   valid (conv_aux_data[b]))/(no of valid values)
                                                     (Req.5.7-48)
outer_zone_mean = (valid (conv_aux_data[c]) +
                   valid (conv_aux_data[d]) +
                   valid (conv_aux_data[e]))/(no of valid values)
                                                     (Req.5.7-49)
if inner_zone_mean = 0.0 OR outer_zone mean = 0.0 then
      weighted mean = inner_zone_mean + outer_zone_mean
                                                     (Req.5.7-50)
else
      weighted_mean = (inner_zone_mean + outer_zone_mean)/2.0
                                                     (Req.5.7-51)
end if
valid is a function which ensures the validity of each temperature by
comparing the element of the auxiliary_data_validation_result which
corresponds to the temperature specified, with the constant
TMZ_CALIBRATION_ERR. If the error code is less than
TMZ_CALIBRATION_ERR, the value is valid and can be included in the
sum.                                                 (Req.5.7-52)
```

Step 5.7.3-5-2 Add the weighted mean temperatures to the accumulated running totals for the calibration period.

```
For the PX BB:
if the weighted mean bb temperature is > 0.0 then
   add the weighted mean bb temperature to the sum_wtd_mean_px
                                                     (Req.5.7-53)
```

```
   tally_of_weighted_means_px = tally_of_weighted_means_px + 1
                                                      (Req.5.7-54)
end if
For the MX BB:
if the weighted mean bb temperature is > 0.0 then
   add the weighted mean bb temperature to the sum_wtd_mean_mx
                                                      (Req.5.7-55)
   tally_of_weighted_means_mx = tally_of_weighted_means_mx + 1
                                                      (Req.5.7-56)
end if
```

Step 5.7.3-6 Generate running totals of valid fore-optics temperatures

The mean background (fore-optics) temperature in a single packet is derived from valid converted data for the parabaloid mirror field stop temperatures 1 to 3 and the focal plane assembly baffle temperature, (the converted values of telemetry items A4511, A4521, A4531 and A4551 (PRT1 to PRT3 and PRT5)).

Sum the appropriate valid background temperatures in this packet:

```
bground_temp_sum = 0.0                                (Req.5.7-57)
n_items = 0                                           (Req.5.7-58)
for n = A4511, A4521, A4531 and A4551
    if auxiliary_data_validation_result[n] is less than
         TMZ_CALIBRATION_ERR then
```

Include the valid value in the sum.

```
      bground_temp_sum = bground_temp_sum +
                converted_auxiliary_data[n]           (Req.5.7-59)
```

and increment the tally of values used

```
      n_items = n_items + 1                           (Req.5.7-60)
    end if
end for
```

Calculate the mean background temperature in this packet from the sum and tally

```
if n_items > 0 then
   bground_temp = bground_temp_sum / float(n_items)   (Req.5.7-61)
```

Now add the mean value for this packet to the running total for this calibration period,

and increment the tally of values in the sum

```
   mean_cal_period_bground_temp_sum =
      mean_cal_period_bground_temp_sum + bground_temp (Req.5.7-62)
      bground_cp_tally = bground_cp_tally + 1         (Req.5.7-63)
end if
```

Note. If the data is so poor that no valid values have been used in the calculation, it is not possible to continue deriving the calibration parameters for this group of packets. In this case the calibration process exits with an error, leaving the calibration parameters at their initialised extreme value as a flag to the next level of processing that the calibration has not been possible for this data.

Step 5.7.3-7 Acumulate the running total of valid BB pixel counts in this calibration period.

The calculation of the mean counts for the current packet requires generating a mean count for each black body, for each channel, and for each parity, providing that the blanking_pulse_calibration_flag, is compatible with the state of the blanking pulse flags of the required black body pixel data, and also that the BB pixels in the calibration period are valid. For valid BB pixel data, the value of the calibration_invalid[channel] flag is 0 (the initialisation value) for each channel.

Step 5.7.3-7-1. Initialise local packet sums and tallies.

```
for all IR channels
   for bb = PX and MX
      for parity = odd and even
         bb_pkt_sum[channel][bb][parity] = 0          (Req.5.7-64)
         bb_pkt_tally[channel][bb][parity] = 0        (Req.5.7-65)
         bb_pkt_mean[channel][bb][parity] = 0.0       (Req.5.7-66)
      end for
   end for
end for
```

Step 5.7.3-7-2. Derive sums from the current packet using valid pixel data.

```
if packet_error = success then
   for all IR channels
     if calibration_invalid[channel] = 0 then
```

Point at the IDF value for the PX BB start pixel

```
        NN = 24                                         (Req.5.7-67)
        for bb = plus_bb, minus_bb
```

Determine how many pixels

```
            start_pixel_no = IDF(NN)                    (Req.5.7-68)
            end_pixel_no = IDF(NN+1)                    (Req.5.7-69)
            no_of_pixels = end_pixel_no - start_pixel_no + 1
                                                        (Req.5.7-70)
```

Determine the parity of the first pixel

```
            if start_pixel_no is ODD then
               parity is odd                            (Req.5.7-71)
            else
               parity is even                           (Req.5.7-72)
            end if
```

Sum the valid pixels of this parity

```
        for i= 0, to i < no_of_pixels, increment = 2
        if blanking_pulse_calibration_flag is compatible with
        the blanking_pulse values for this bb pixel data and
        the pixel count value is > 0 and < 4095 then
          bb_pkt_sum[channel][bb][parity] =
            bb_pkt_sum[channel][bb][parity] +
            unpacked_pixels.bb[i]                    (Req.5.7-73)
          bb_pkt_tally[channel][bb][parity] =
            bb_pkt_tally[channel][bb][parity] + 1
                                                     (Req.5.7-74)
        end if
    end for
```

Now repeat sum of valid pixels for the other parity.

First swap parity

```
    if parity is even then
           parity is odd                             (Req.5.7-75)
    else
           parity is even                            (Req.5.7-76)
    end if
```

And sum the valid pixels for the alternate parity

```
    for i= 1, to i < no_of_pixels + 1, increment= 2
      if blanking_pulse_calibration_flag is compatible with
         the blanking_pulse values for this bb pixel data and
         the pixel count value is > 0 and < 4095 then
            bb_pkt_sum[channel][bb][parity] =
               bb_pkt_sum[channel][bb][parity] +
               unpacked_pixels.bb[i]                 (Req.5.7-77)
            bb_pkt_tally[channel][bb][parity] =
                   bb_pkt_tally[channel][bb][parity] + 1
                                                     (Req.5.7-78)
      end if
    end for
```

Now point to the next Black body

```
        NN = NN + 4                                  (Req.5.7-79)
    end for
  end if
 end for
end if
```

Step 5.7.3-7-3. Calculate the mean count for the current packet, and add the value derived to the running total of bb pixel count means for this calibration period.

```
for all IR channels
   for bb = PX and MX
      for parity = odd and even
         if bb_pkt_tally[channel][bb][parity] > 0 then
```

Some valid values have been accumulated for this packet, so derive their mean

```
bb_pkt_mean[channel][bb][parity] =
   (float) bb_pkt_sum[channel][bb][parity] /
   (float) bb_pkt_tally[channel][bb][parity](Req.5.7-80)
```

And add it to the running total of means for the current calibration period.

```
bb_pixel_count_cal_period_sum[channel][bb][parity] =
 bb_pixel_count_cal_period_sum[channel][bb][parity] +
   bb_pkt_mean[channel][bb][parity]       (Req.5.7-81)
```

Increment the tally of means in the calibration period sum.

```
bb_pixel_count_cal_period_tally[channel][bb][parity] =
bb_pixel_count_cal_period_tally[channel][bb][parity] + 1
                                          (Req.5.7-82)
         end if
      end for
   end for
end for
```

Step 5.7.3-8 Calculate IR calibration parameters for the calibration period

At this point it is the end of the calibration period, so calculate the calibration parameters for the IR channels from the black body temperature data, and the black body pixel data accumulated over this calibration period.

Step 5.7.3-8-1. Derive the mean BB temperatures.

```
if tally_of_weighted_means_px > 0 then
```

There is valid temperature data to use

```
   mean_pxbb_temperature =
      (float) sum_wtd_mean_px /
      (float) tally_of_weighted_means_px             (Req.5.7-83)
end if
```

```
if tally_of_weighted_means_mx > 0 then
```

There is valid temperature data to use

```
   mean_mxbb_temperature =
      (float) sum_wtd_mean_mx /
      (float) tally_of_weighted_means_mx             (Req.5.7-84)
end if
```

Step 5.7.3-8-2. Convert the mean black body temperatures to radiances.

```
for each IR channel (3.7, 11, 12)
   if mean_mxbb_temperature > 0.0 then
```

```
      convert_temperature_to_radiance(
         mean_mxbb_temperature,
         mean_mx_bb_radiance[channel])                    (Req.5.7-85)
   end if
   if mean_pxbb_temperature > 0.0 then
      convert_temperature_to_radiance(
         mean_pxbb_temperature,
         mean_px_bb_radiance[channel])                    (Req.5.7-86)
   end if
end for
```

Here convert_temperature_to_radiance is a function using linear interpolation to perform the conversion between temperature and radiance. See 5.7.3-8-4 below for details.

Step 5.7.3-8-3. Convert the background temperature to radiance.

The mean fore-optics background temperature is similarly converted to an equivalent background radiance value for each IR channel.

First derive the mean background temperature from the summed means in the calibration period.

```
if bground_cp_tally > 0 then
```

A mean can be derived for this calibration period.

```
   mean_cal_period_bground_temp =
      mean_cal_period_bground_temp_sum /
      (float) bground_cp_tally                            (Req.5.7-87)
end if
```

Then convert the background temperature to an equivalent radiance value.

```
if mean_cal_period_bground_temp > 0.0 then
   for each IR channel
      convert temperature to radiance(
         mean_cal_period_bground_temp,
         bground_ radiance[channel])                      (Req.5.7-88)
   end for
end if
```

Here convert_temperature_to_radiance is a function using linear interpolation to perform the conversion between temperature and radiance. See 5.7.3-8-4 below for details.

Step 5.7.3-8-4. A Linear interpolation routine for the conversion of temperature to radiance.

When the system is initialised, the temperature to radiance (and radiance to temperature) conversion tables are read into memory. These are tables of values giving the equivalent radiance value in W cm^{-2} sr^{-1} for each temperature in degrees Kelvin. (and the equivalent temperature in degrees K for each radiance value in W cm-2 sr-1) The two sets of tables allow a conversion from temperature to radiance or from radiance to temperature.

All look up tables have been adjusted to ensure a linear relationship between the input and output. If the input value to be converted is outside the range of conversion values in the look up table, the calibration process exits with an error, (TEMP_OUT_OF_RANGE_FOR_LUT_ERR), leaving the calibration parameters at their initialised extreme value as a flag to the next level of processing that the calibration has not been possible for the data from this calibration period.

Conversion is efficient if the temperature (and corresponding radiance look up table data) are supplied at intervals of 0.05 degrees K (and Wcm^{-2} sr^{-1} equivalent). This will allow an algorithm for linear interpolation to be used for deriving radiances for temperatures between 0.05 degree intervals. If fewer data points are used, processor hungry polynomial interpolation is required to derive values between the supplied intervals. A linear interpolation algorithm is defined below. See AD 9., pp 113, section 3.3.

For a table of points temperature_lut[0:n], radiance_lut[0:n] the value of the radiance between 2 points temperature_lut[i] and temperature_lut[i+1] for a given temperature t is

```
radiance = A.radiance_lut[i] + B.radiance_lut[i+1]    (Req.5.7-89)
```

where

```
A = (temperature_lut[i+1] - t) /
      (temperature_lut[i+1] - temperature_lut[i])     (Req.5.7-90)
and B = 1 - A                                         (Req.5.7-91)
```

Both tables correspond and have an identical dimension. However, the radiance_lut is a 2-d array, containing all the radiance values for each IR channel. n is the size_of_lut.

Step 5.7.3-8-5. Correct the BB radiance values.

Each black body radiance value is now corrected using the mean back ground radiance, together with emissivity constants for each IR channel. This is described for ATSR-2 in RD 1, section 6.3, “The Calibration algorithm”. (pp 99). The correction is applied to each channel as follows:

At initialisation time

```
for each IR channel (12, 11, 3.7)
   read in mxbb_emissivity[channel]
   read in pxbb_emissivity[channel]                   (Req.5.7-92)
end for
```

At the end of each calibration period derive the corrected black body radiance values.

```
for each IR channel (12, 11, 3.7)
   mean_px_bb_radiance[channel] =
      pxbb_emissivity[channel] * mean_px_bb_radiance[channel] +
      ((1 - pxbb_emissivity[channel]) * bground_radiance[channel])
                                                      (Req.5.7-93)

   mean_mx_bb_radiance[channel] =
      mxbb_emissivity[channel] * mean_mx_bb_radiance[channel] +
      ((1 - mxbb_emissivity[channel]) * bground_radiance[channel])
                                                      (Req.5.7-94)
end for
```

Step 5.7.3-8-6. Calculate the gains and offsets from BB counts and BB radiances.

First derive the mean BB pixel count over the calibration period for odd and even pixels, for each BB, and for each IR channel, from the sums of means and the tallies derived in Step 5.7.3-7-3.

```
for each IR channel (12, 11, 3.7)
  for each bb (px and mx)
    for both parities (odd and even)
      if bb_pixel_count_cal_period_tally[channel][bb][parity] > 0
      then
```

Valid data, so derive a mean count for the cal period

```
        mean_bb_pixel_count_cal_period[channel][bb][parity] =
             bb_pixel_count_cal_period_sum[channel][bb][parity] /
             bb_pixel_count_cal_period_tally[channel][bb][parity]
                                                        (Req.5.7-95)
      end if
    end for
  end for
end for
```

Now derive the gains and the offsets for the calibration period, for odd and even pixels and for each IR channel using the following algorithm:

```
for each IR channel
  for both parities (odd and even)
    if mean_bb_pixel_count_cal_period[channel][px][parity] > 0.0 and
       mean_bb_pixel_count_cal_period[channel][mx][parity] > 0.0 and
       mean_px_bb_radiance[channel] > 0.0 and
       mean_mx_bb_radiance[channel] > 0.0 then
```

Valid data is available so derive the gain and offsets for each channel

```
    if ((mean_bb_pixel_count_cal_period[channel][mx][parity] -
        mean_bb_pixel_count_cal_period[channel][px][parity])
          ne 0) then
       gain[parity][channel] =
      (mean_px_bb_radiance[channel] - mean_mx_bb_radiance[channel]) /
      (mean_bb_pixel_count_cal_period[channel][px][parity] -
       mean_bb_pixel_count_cal_period[channel][mx][parity])
                                                        (Req.5.7-96)

      offset[parity][channel] =
      ((mean_px_bb_radiance[channel] *
        mean_bb_pixel_count_cal_period[channel][mx][parity]) -
       (mean_mx_bb_radiance[channel] *
        mean_bb_pixel_count_cal_period[channel][px][parity])) /
       (mean_bb_pixel_count_cal_period[channel][mx][parity] -
        mean_bb_pixel_count_cal_period[channel][px][parity])
                                                        (Req.5.7-97)
    end if
    end if
```

```
   end for
end for
```

5.8 Module Definition: Visible Channel Calibration

5.8.1 Functional Description

Determination of the calibration parameters for the AATSR visible channels involves two main stages. The offset value for each channel is derived by averaging the black body pixel counts in the channel over a calibration period in a similar way to the derivation of the mean black body counts for the infra-red channels during the infra-red channel calibration (Section 5.7). The slope value for the channel is derived from the on-board visible calibration system as described below.

This section describes the process for extracting the calibration data for the AATSR visible channels using the on-board calibration system.

Two on-board sources are used to calibrate the AATSR visible/near infra-red channels. The upper reflectance measurement is provided by the visible calibration unit, VISCAL, which gives a signal corresponding to ~ 15% spectral albedo at full solar illumination. The reflectance_factor for each channel are obtained from the pre-launch calibration of the VISCAL (Ref. PO-TN-RAL-AT-0165, “Viscal Calibration Parameters”) and are read in from L1-AUX18-63 to 66 inclusive.

The zero reflectance signal is derived from one of the on-board black bodies, used also for the thermal calibration. The data from these sources are used to derive the calibration slopes for each channel using (in the case that the MXBB is used as the reference black body)

```
Calibration_Slope[chan] = Reflectance_Factor[chan]*Gain[chan] /
                          (Average_VISCAL_Pixel_Counts[chan]-
                           Average_MXBB_Pixel_Counts[chan])
```

where the VISCAL and -XBB pixel counts are averaged only during the period when the VISCAL is at full solar illumination. The pixel counts are normalised to unit detector gain to allow for gain changes during an orbit. The Calibration_Slope is then used to convert pixel counts to top-of-atmosphere reflectance by

```
Pixel_Reflectance[chan] = Calibration_Slope[chan]*
      (Pixel_Count[chan] - Average_MXBB_Pixel_Counts[chan])/
                                                     Gain[chan]
```

where the -XBB pixel counts are taken from the same scan as the pixel counts being calibrated.

This scheme is modified in the case of the 1.6 micron channel to correct for measured detector nonlinearity, as described below.

5.8.2 Interface definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX17-2		1.6 micron slope	float	n/a	4	1
L1-AUX17-3		0.870 micron slope	float	n/a	4	1
L1-AUX17-4		0.670 micron slope	float	n/a	4	1
L1-AUX17-5		0.555 micron slope	float	n/a	4	1

L1-AUX18-3		visible calibration option code	sl	n/a	4	1
L1-AUX18-14		vis_bb_code_px	ss	n/a	2	1
L1-AUX18-15		vis_bb_code_mx	ss	n/a	2	1
L1-AUX18-57		monitor_threshold	ss	n/a	2	1
L1-AUX18-75		calibration_window_diff1	ss	n/a	2	1
L1-AUX18-76		calibration_window_diff2	ss	n/a	2	1
L1-AUX18-77	Nvis	VISCAL Threshold	ss	n/a	2	1
L1-AUX18-78		window_half_width_in_min	float	minutes	4	1
L1-AUX18-61	orbit_period	period of ENVISAT orbit	float	s	4	1
L1-AUX18-62	time_offset	interval between terminator and full VISCAL illumination	float	s	4	1
L1-AUX18-63		1.6 micron reflectance factor	float	none	4	1
L1-AUX18-64		0.870 micron reflectance factor	float	none	4	1
L1-AUX18-65		0.670 micron reflectance factor	float	none	4	1
L1-AUX18-66		0.555 micron reflectance factor	float	none	4	1
L1-AUX18-67		1.6 micron solar irradiance	float	mW/cm2/µm	4	1
L1-AUX18-68		0.870 micron solar irradiance	float	mW/cm2/µm	4	1
L1-AUX18-69		0.670 micron solar irradiance	float	mW/cm2/µm	4	1
L1-AUX18-70		0.555 micron solar irradiance	float	mW/cm2/µm	4	1
L1-AUX18-71		1.6 micron channel bandwidth	float	micron	4	1
L1-AUX18-72		0.870 micron channel bandwidth	float	micron	4	1
L1-AUX18-73		0.670 micron channel bandwidth	float	micron	4	1
L1-AUX18-74		0.555 micron channel bandwidth	float	micron	4	1
L1-AUX27-24	N(v16)	Number of entries in 1.6 micron Channel Non-linearity Correction LUT	ss	none	2	1
L1-AUX27-25	increment(v16)	Increment in 1.6 micron Channel Non-linearity Correction LUT	do	counts	8	1
L1-AUX27-26	first_value(v16)	First value in 1.6 micron nonlinear correction LUT	do	counts	8	1
L1-AUX27-27	last_value(v16)	Last value in 1.6 micron nonlinear correction LUT	do	counts	8	1
L1-AUX28-1	C0(k)	Uncorrected count entry	do	counts	8	N(v16)
L1-AUX28-2	C1(k)	Corrected count table entry	do	counts	8	N(v16)

Table 5-8-1: Input Data Table - Visible channel calibration parameters

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-410		Time of cal in MJD format	ul, 2* sl	MJD	12	1
L1B-INT-411		1.6 micron slope	float	n/a	4	1
L1B-INT-412		0.870 micron slope	float	n/a	4	1
L1B-INT-413		0.670 micron slope	float	n/a	4	1
L1B-INT-414		0.555 micron slope	float	n/a	4	1
L1B-INT-415		UTC at ascending node crossing, in MJD format	ul, 2* sl	MJD	12	1
L1B-INT-416		Average Monitor count	float	n/a	4	1
L1B-INT-417		Standard deviation of Monitor count	float	n/a	4	1
L1B-INT-418		Solar irradiance (1.6 micron)	float	n/a	4	1
L1B-INT-419		Solar irradiance (0.870 micron)	float	n/a	4	1
L1B-INT-420		Solar irradiance (0.670 micron)	float	n/a	4	1
L1B-INT-421		Solar irradiance (0.555 micron)	float	n/a	4	1
L1B-INT-422		Average VISCAL Pixel Counts (1.6 µm)	float	n/a	4	1
L1B-INT-423		Average VISCAL Pixel Counts (0.87 µm)	float	n/a	4	1
L1B-INT-424		Average VISCAL Pixel Counts (0.67 µm)	float	n/a	4	1
L1B-INT-425		Average VISCAL Pixel Counts (0.55 µm)	float	n/a	4	1
L1B-INT-426		VISCAL Pixel Noise (1.6 micron)	float	n/a	4	1
L1B-INT-427		VISCAL Pixel Noise (0.87 micron)	float	n/a	4	1
L1B-INT-428		VISCAL Pixel Noise (0.67 micron)	float	n/a	4	1
L1B-INT-429		VISCAL Pixel Noise (0.55 micron)	float	n/a	4	1
L1B-INT-430		Average -X BB Pixel Counts (1.6 µm)	float	n/a	4	1
L1B-INT-431		Average -X BB Pixel Counts (0.87 µm)	float	n/a	4	1
L1B-INT-432		Average -X BB Pixel Counts (0.67 µm)	float	n/a	4	1

L1B-INT-433		Average -X BB Pixel Counts (0.55 μm)	float	n/a	4	1
L1B-INT-434		-X BB Pixel Noise (1.6 micron)	float	n/a	4	1
L1B-INT-435		-X BB Pixel Noise (0.87 micron)	float	n/a	4	1
L1B-INT-436		-X BB Pixel Noise (0.67 micron)	float	n/a	4	1
L1B-INT-437		-X BB Pixel Noise (0.55 micron)	float	n/a	4	1
L1B-INT-438		(Reserved for parity indicator)	ss	n/a	2	1
(local)		bb_pixel_count_cal_period_sum[chan][bb][parity]	lo	N/A	8	28
(local)		bb_pixel_count_cal_period_tally[chan][bb][parity]	ss	N/A	2	28
(local)		mean_bb_pixel_count_cal_period[chan][bb][parity]	fl	N/A	4	28
L1B-INT-009	scp_gain(ch, s)	channel scp gain (only required for v16 at present)	float	n/a	4	
L1B-INT-010	ch_gain[i,j]	gain[parity][channel], channel = 4, .. 7	array of array of fl	TBD	4	7 * 2
L1B-INT-011	ch_offset[i,j]	offset[parity][channel], channel = 4, .. 7	array of array of fl	TBD	4	7 * 2
local	calibration_time	UT of nominal calibration time	double	days	8	1
local	t_start	UT of start of time window	double	days	8	1
local	t_end	UT of end of time window	double	days	8	1
L1B-INT-401	mid_UT_time	UT of centre of monitor period	double	days	8	1
local	ut1	UT of start of monitor period	double	days	8	1
local	ut2	UT of start of monitor period	double	days	8	1
local	s	scan index	sl		4	1
local	s1	scan number of start of monitor period	sl		4	1
local	s2	scan number of end of monitor period	sl		4	1
local	first_s	index of first scan in time window	sl		4	1
local	last_s	index of last scan in time window	sl		4	1
local	n1	index of start of calibration period	sl		4	1
local	n2	index of end of calibration period	sl		4	1
local	sum0	monitor count sum	double	n/a	8	1
local	sum1	cumulative sum	double	n/a	8	1
local	mean_s	centroid (first moment) of monitor distribution	sl	n/a	4	1
local	mon1	smoothed monitor value	float	n/a	4	1
local	found_start	local flag	flag		2	1
local	found_end	local flag	flag		2	1
local	count	local counter	ss		2	1

Table 5-8-2: Internal Data Table - Visible channel calibration

5.8.3 Algorithm Description

Module 8 can be divided into two essentially independent components.

Module 8A uses the data from the illuminated VISCAL unit to determine calibration coefficients for the visible channels, normalised to unit channel gain setting. The coefficients are not used for calibrating the current orbit, but are written to the GBTR product, and may be used for visible channel calibration in later runs of the processor. This component can only be executed if the time of maximum illumination falls within the product limits.

Module 8B uses black body data and input VISCAL coefficients to derive the calibration parameters for the current data and is applied to all source packets in a similar way to the infra-red channel calibration Module 7.

The selection of the reference black body is determined by the visible calibration option code, parameter [L1-AUX18-3].

If [L1-AUX18-3] = vis_bb_code_mx [L1-AUX18-15] then the MXBB is to be used as the reference BB.

If [L1-AUX18-3] = vis_bb_code_px [L1-AUX18-14] then the PXBB is to be used as the reference BB, and all references to the MXBB in the text and equations that follow are to be interpreted as if they refer to the PXBB Thus in this case the text and equations of this section are to be interpreted as if PXBB is substituted for MXBB throughout.

Note: This version of the algorithm has been significantly revised to account for differences between the AATSR and ATSR-2 instruments that were not previously appreciated. The changes relate principally to the identification of the VISCAL monitor peak in Steps 5.8.1.3 and 5.8.1.4.

The original algorithm, which appeared in earlier issues of this document, was based on that used for ATSR-2. In the ATSR-2 instrument, each instrument source packet contains an independent sample of the VISCAL monitor count. However, in AATSR the VISCAL monitor values are sub-commutated, so that a new independent sample appears only in every eighth source packet; the intervening source packets contain a copy of the last independent sample. Thus VISCAL monitor counts appear in blocks of 8 identical values. The ATSR-2 algorithm does not work in these circumstances.

The following limitation of the revised algorithm in its present form should be noted.

- The average monitor count and its standard deviation are calculated (Step 5.8.1.5, Reqs. 5.8-106, 5.8-107) in the same window as the averages of the pixel counts. Strictly this may be incorrect, since the plateau region of the monitor count differs from (and is narrower than) that of the illuminated pixels. If the window limits
  `calibration_window_diff1` [L1-AUX18-75] and
  `calibration_window_diff2` [L1-AUX18-76]
  define a window which is broader than the monitor count plateau, the mean monitor count may be biassed. Whether or not this happens depends on the exact choice of the window limits in the auxiliary file, and does not affect the validity of calibration itself, since the average monitor count is an auxiliary output that does not form part of the calibration. Note that a similar limitation affected the original form of the algorithm.

The algorithm in this revision has been modified to account for the possibility of missing or defective source packets (i.e. source packets that cannot be decompressed), or of data gaps, falling within the calibration period. See the implementation note in Step 5.8.1.2.

5.8.3.1 Module 8A; Slope Parameters from Illuminated VISCAL Data

Step 5.8.1.1 Select window around VISCAL illumination period.

```
Calculate the time when the VISCAL is under full solar illumination.
calibration_time = ascending_node_time + (orbit_period*
      (90-solar_declination_angle)/360 - time_offset)/86400.
                                                    (Req.5.8-98)
where
```

ascending_node_time is the ascending node crossing time given in processing format by mjdp2[0]/(1) [L1B-INT-30] from the initialization call to po_ppforb in Modules 10 and 11. (Note that this calculation is required even if a restituted orbit file is suppled.)

orbit_period is the ENVISAT-1 orbit period ~100 minutes

time_offset is the period between the time when the VISCAL is fully illuminated and the time that ENVISAT crosses the terminator, ~ 4.8 minutes. (The time of full illumination precedes the terminator crossing.)

solar_declination_angle is the solar declination angle which is given by

```
solar_declination_angle =
   (0.006918 - 0.399912*cos(day_angle) +
   0.070257*sin(day_angle) -
   0.006758*cos(2*day_angle) +
   0.000907*sin(2*day_angle) -
   0.002697*cos(3*day_angle) +
   0.00148*sin(3*day_angle)) * 180/π
```

and

```
day_angle = 2*π*(day-1)/365
```

is the day angle for the nth day of the year (Iqbal 1983).

To ensure that all the data for the VISCAL is captured when illuminated by the Sun, a window of ±5.0 minutes around calibration_time should be used. Thus

```
t_start = calibration_time - window_half_width_in_min/1440
t_end = calibration_time + window_half_width_in_min/1440
```

(Req.5.8-99)

where t_start and t_end are expressed in days, and where

```
window_half_width_in_min = 5.0.
```

Note that the value of window_half_width_in_min [L1-AUX18-78] may be found in the external file (ATS_PC1_AX).

It is possible that the window defined here does not fall within the product limits. This may be expected in the case of NRT data for the following reason.

The algorithm defined above was devised for consolidated data. In particular it was assumed that the orbit state vector supplied in the MPH and used to calculate the ascending node time (above) corresponds to the start of the data, and that the data corresponds to a complete orbit. In this case the window calculated above will certainly fall within the product limits.

However, in the case of NRT data, the start of data may not coincide with the ascending node, and the MPH may contain a state vector that is (for example) later than the viscal peak that falls within the product limits (if any does).

So we have the following cases.

If sensing_start < t_start and t_end < sensing_stop the viscal window falls entirely within the product limits; proceed to step 5.8.1.2.

Otherwise:

if t_start < sensing_start then we may have a partial peak at the beginning of data. (Actually this is unlikely if the state vector is later than t_start.) A peak at the end may be better. Calculate

```
t_orbit = orbit_period / 86400.
```

where t_orbit is expressed in days (the same units as t_start, calibration_time etc.). Note orbit_period [L1-AUX18-61] is in seconds.

```
k = integer part of [(sensing_stop - t_start)/t_orbit]
calibration_time = calibration_time + k * t_orbit
t_start = calibration_time - window_half_width_in_min/1440
```

```
      t_end = calibration_time + window_half_width_in_min/1440
else if t_start > sensing_stop a peak at the start of the data may be better. Calculate
      t_orbit = orbit_period / 86400.
      k = integer part of [(t_start - sensing_start)/t_orbit]
      calibration_time = calibration_time - k * t_orbit
      t_start = calibration_time - window_half_width_in_min/1440
      t_end = calibration_time + window_half_width_in_min/1440
end if
```

Step 5.8.1.2 Process the raw data for this window.

The remainder of Step 5.8.1 can only be done if the window centered on the calibration time computed in Step 5.8.1.1 falls within the product limits. Although this should be true for consolidated data, if NRT data is being processed it is possible that the start or end of the Level 0 data will fall within the VISCAL illumination period. In this case it may not be possible to derive a valid visible channel calibration. In particular,

```
if t_start is earlier than the time of the first scan it may not be possible to generate a valid VISCAL GADS, and Module 8A VISCAL processing should be abandoned.
Otherwise, and provided there are at least 17 scans present in the period between the start of the VISCAL window and the end of the Level 0 data
      process the raw data to extract the pixel counts and instrument telemetry as follows.
```

Implementation note:

The algorithm as defined here does not cater for the possibility of a null or invalid scan (that is to say, a scan that is not decompressed owing to the failure of one of the format tests in Module 1) falling within the time window. In the present prototype implementation, such scans are omitted entirely from the VISCAL processing. The valid scans that fall within the time window determined in Step 5.8.1.1 are copied to a scratch file, which is then re-read and processed as in the following steps.

Experience suggests that such invalid scans are rare, and the omission of one or two sporadic scans (say for a CRC failure) will not significantly bias the algorithm, but the presence of a data gap might lead to an erroneous calibration that would not be recognized as such. On the other hand, invalid scans are sufficiently frequent that they should be accounted for in the VISCAL algorithm; if the presence of a single missing or invalid source packet leads to the visible calibration being abandoned, missing VISCAL GADS may occur too frequently.

For these reasons it is proposed that the determination of the visible calibration slopes follow the prototype algorithm, but with a threshold, such that if the number of invalid or missing source packets exceed the threshold, processing is abandoned and no VISCAL GADS is generated.

Steps 5.8.1.3 to 5.8.1.5 are therefore to be interpreted as follows. Let S be the set of all valid scans falling within the time window identified at Step 5.8.1.1 above, having

$$first\_s \le s' \le last\_s,$$

where *s*' represents the instrument scan number, and let *Ns* be the number of invalid scans within the same interval. (Invalid scans are null or have not been decompressed). If

Ns > *Nvis* [L1-AUX18-77]

then a VISCAL GADS should not be generated, and Module 8A processing should be abandoned. Otherwise Steps 5.8.1.3 to 5.8.1.5 should be applied to the ordered sequence of scans *S*, which should be chronologically renumbered with a continuous index *s*, so that the index *s* represents the continuous order of the valid scans in *S*, with no gaps. Clearly if there are no invalid scans within the window, so that *Ns* = 0, then *s* = *s*', otherwise *s* ≤ *s*'.

Step 5.8.1.3. Find period within the time window when the smoothed VISCAL monitor is above the specified threshold

In the following, `first_s` and `last_s` denote the indices of the first and last scans respectively in the time window identified in Step 5.8.1.1 above. If `last_s` is the index of the last scan in the Level 0 product being processed, so that end of data falls within the time window, it may not be possible to generate a valid VISCAL GADS. The tests below involving `last_s` allow for this case.

```
      found_start = FALSE
      found_end = FALSE
If there are fewer than 17 valid scans present in the period between
the start of the VISCAL window and the end of the Level 0 data (this
situation might arise for NRT data) an initial value of mon1 cannot
be derived. In this case a valid VISCAL product cannot be generated,
and Module 8A processing should be abandoned. Otherwise:
Starting with scan s = (first_s + 8) and using all the scans in the
period identified proceed as follows:
Calculate a smoothed monitor count for the first sample
      mon1 = (monitor_count[s-8] + 2 * monitor_count[s] +
              monitor_count[s+8])/4.0
Here monitor_count[s] represents the VISCAL monitor count (A4541) at
scan number s.
If mon1 > monitor_threshold, it may not be possible to derive a valid
centroid, and processing should be abandoned. Otherwise:
while mon1 < monitor_threshold and s < (last_s - 8)
      s = s + 1
      mon1 = (monitor_count[s-8] + 2 * monitor_count[s] +
              monitor_count[s+8])/4.0
end while
if s ≥ (last_s - 8) at this point a valid VISCAL product cannot be
generated, and Module 8A processing should be abandoned.
Otherwise the end of the while loop defines the value s of the first
source packet for which the smoothed monitor count exceeds the
threshold.
      found_start = TRUE
      s1 = s                          (start of monitor window)
      ut1 = source_packet_ut_time(s)  (monitor window start time)
                                                      (Req.5.8-100)
Note that s will correspond to the index of the first sample of a
block of 8. The next loop can skip in increments of 8.
while mon1 ≥ monitor_threshold and s ≤ (last_s - 16)
```

```
        s = s + 8
        mon1 = (monitor_count[s-8] + 2 * monitor_count[s] +
                monitor_count[s+8])/4.0
end while
```

if mon1 ≥ monitor_threshold, the while loop has terminated because of the condition on last_s, and it may not be possible to define a valid centroid, because there is no more data in the monitor time window. This case may arise if the end of the Level 0 product falls within the time window. In this case a valid VISCAL GADS cannot be generated, and Module 8A processing should be abandoned. Otherwise:

```
        found_end = TRUE
        s2 = (s - 8)
        ut2 = source_packet_ut_time(s) (monitor window end time)
```

(Req.5.8-101)

Note. This is to be sure that the data used corresponds to the time when the VISCAL is being illuminated by the Sun and not in darkness.

Step 5.8.1.4. Find period of full solar illumination

Now calculate the centroid of the monitor count values between s1 and s2.

```
sum0 = 0.0 d0
sum1 = 0.0 d0
for s = s1, s2, 8 do
        sum0 = sum0 + monitor_count[s]
        sum1 = sum1 + s * monitor_count[s]
end for
mean_s = integer part of [sum1/sum0]
```

Define the start of the calibration period for the signal channels as being *calibration_window_diff1* scans before the first moment of the of the monitor signal.

n1 = mean_s - calibration_window_diff1

(Req.5.8-102)

Define the end of the calibration period for the signal channels as being *calibration_window_diff2* scans after the first moment of the monitor signal.

n2 = mean_s + calibration_window_diff2

(Req.5.8-103)

The values n1 and n2 determined above define the limits of the calibration period for the following steps. Thus the length of the calibration window is

calibration_window_diff1 + calibration_window_diff2 + 1

samples.

Calculate an estimate of the mid-point of the monitor window for inclusion in BP16);

mid_UT_time = (ut1 + ut2)/2.

Step 5.8.1.5 Average Pixel Counts over Calibration Period.

The next stage is to calculate the average and standard deviations of the VISCAL and -XBB counts over the calibration period defined in the previous step.

Note that at this point we expect that n1 ≥ first_s and n2 ≤ last_s otherwise this step cannot be carried out. In this situation (which can only arise in the event of a significant gap in the data) a valid VISCAL GADS cannot be generated, and Steps 5.8.1.5 to 5.8.1.7 should be abandoned. Otherwise:

Derive the average pixel count for all channels

$$\mathrm{average\_viscal\_pixel\_counts} = \frac{1}{M}\sum_{i=m1}^{m2}\frac{1}{N}\sum_{j=n1}^{n2}\mathrm{unpacked[j].pixels.viscal[i]}$$

$$\mathrm{average\_mxbb\_pixel\_counts} = \frac{1}{M}\sum_{i=m1}^{m2}\frac{1}{N}\sum_{j=n1}^{n2}\mathrm{unpacked[j].pixels.minus\_bb[i]}$$

(Req.5.8-104)

where m_1 and m_2 are the start and stop pixels for the view;

m1 = 0;
m2 = [A2210 - A2200] (IDF21 - IDF20) for the VISCAL equation,
m2 = [A2290 - A2280] (IDF29 - IDF28) for the mxbb or
m2 = [A2250 - A2240] (IDF25 - IDF24) for the pxbb.

M = m2 - m1 + 1 is total the number of pixels used in the view

n1 and n2 are the start and stop scans of the calibration window,

and

N is the number of valid scans in the calibration window between n1 and n2 inclusive, that contribute to the calibration.

The sums over j include only valid scans from S.

Derive the standard deviation for all the signal channels =

$$\mathrm{viscal\_pixel\_noise}^2 = \left(\frac{1}{M}\sum_{i=m1}^{m2}\frac{1}{N}\sum_{j=n1}^{n2}\mathrm{unpacked[j].pixels.viscal[i]}^2 - \left(\frac{1}{M}\sum_{i=m1}^{m2}\frac{1}{N}\sum_{j=n1}^{n2}\mathrm{unpacked[j].pixels.viscal[i]} \right)^2 \right)$$

$$\mathrm{mxbb\_pixel\_noise}^2 = \left(\frac{1}{M}\sum_{i=m1}^{m2}\frac{1}{N}\sum_{j=n1}^{n2}\mathrm{unpacked[j].pixels.minus\_bb[i]}^2 - \left(\frac{1}{M}\sum_{i=m1}^{m2}\frac{1}{N}\sum_{j=n1}^{n2}\mathrm{unpacked[j].pixels.minus\_bb[i]} \right)^2 \right)$$

(Req.5.8-105)

For all scans which see full illumination

```
    Derive the average monitor count                    (Req.5.8-106)
    Derive the standard deviation                       (Req.5.8-107)
end for
```

Step 5.8.1.6. Calculate Calibration Parameters

From the rewritten reflectance calculation

```
for each channel derive the calibration slope and solar irradiance
at the calibration time.  In the case of the visible channels (chan =
v870, v670m v555):
Calibration_Slope[chan] = Reflectance_Factor[chan]*Gain[chan] /
                          (Average_VISCAL_Pixel_Counts[chan]-
                           Average_MXBB_Pixel_Counts[chan])
where Gain[chan] is given by
        Gain[v870] = [A0391](n1)
        Gain[v670] = [A0381](n1)
        Gain[v555] = [A0371](n1)
In the case of the 1.6 micron channel (chan = v16)
Calibration_Slope[chan] = Reflectance_Factor[chan]/C′
where C′ is derived from
        (Average_VISCAL_Pixel_Counts[v16]-
         Average_MXBB_Pixel_Counts[v16]) / Gain[v16]
via the look-up table exactly as described in Section 5.17, and where
Gain[v16] is given by
        Gain[v16] = [A0041](n1)
                                                        (Req.5.8-108)
For all channels channels (chan = v16, v870, v670m v555):
solar_irradiance[chan] =
mean_solar_irradiance[chan]*bandwidth[chan]*
        (1.000110+0.034221*cos(day_angle)+0.001280*sin(day_angle)
        + 0.000719*cos(2*day_angle)+0.000077*sin(2*day_angle))
(from Iqbal 1983)                                       (Req.5.8-109)
end for
```

where the mean solar irradiance for each channel and the corresponding bandwidth are taken from L1-AUX18-67 to 70 and from L1-AUX18-71 to 74 respectively.

Step 5.8.1.7. Assemble Data for Output to File

```
Assemble the following data for o/p by Module 24:
        UTC at calibration time:
        [L1B-INT-410] = calibration_time,
        converted using pl_pmjd as in Section 5.24 (Step 5.24.3-9)
                                                       (Req. 5.8-110)
        ascending node crossing time:
        [L1B-INT-415] = ascending_node_time,
        converted using pl_pmjd as in Section 5.24 (Step 5.24.3-9)
                                                       (Req. 5.8-111)
        average monitor count:
        [L1B-INT-416] = average_monitor_count          (Req. 5.8-112)
        Standard Deviation in monitor count:
        [L1B-INT-417] = standard_deviation             (Req. 5.8-113)
```

```
    Calibration_slope:
[L1B-INT-411] = calibration_slope[4]
[L1B-INT-412] = calibration_slope[5]
[L1B-INT-413] = calibration_slope[6]
[L1B-INT-414] = calibration_slope[7]          (Req. 5.8-114)
    Solar_irradiance:
[L1B-INT-418] = solar_irradiance[4]
[L1B-INT-419] = solar_irradiance[5]
[L1B-INT-420] = solar_irradiance[6]
[L1B-INT-421] = solar_irradiance[7]           (Req. 5.8-115)
    Average_VISCAL_Pixel_Counts:
[L1B-INT-422] = average_viscal_pixel_counts[4]
[L1B-INT-423] = average_viscal_pixel_counts[5]
[L1B-INT-424] = average_viscal_pixel_counts[6]
[L1B-INT-425] = average_viscal_pixel_counts[7]
                                              (Req. 5.8-116)
    Pixel noise for VISCAL (c.f. 15% albedo):
[L1B-INT-426] = viscal_pixel_noise[4]
[L1B-INT-427] = viscal_pixel_noise[5]
[L1B-INT-428] = viscal_pixel_noise[6]
[L1B-INT-429] = viscal_pixel_noise[7]         (Req. 5.8-117)
    Average -X black-body pixel counts:
[L1B-INT-430] = average_mxbb_pixel_counts[4]
[L1B-INT-431] = average_mxbb_pixel_counts[4]
[L1B-INT-432] = average_mxbb_pixel_counts[4]
[L1B-INT-433] = average_mxbb_pixel_counts[4]  (Req. 5.8-118)
    Pixel Noise for dark -XBB (cf. dark noise):
[L1B-INT-434] = mxbb_pixel_noise[4]
[L1B-INT-435] = mxbb_pixel_noise[5]
[L1B-INT-436] = mxbb_pixel_noise[6]
[L1B-INT-437] = mxbb_pixel_noise[7]           (Req. 5.8-119)
```

Note. If no data is available for a channel (e.g. for outgassing periods) then the data should be replaced with zero.

5.8.3.2 Module 8B: Visible Calibration Parameters for the Current Product.

Step 5.8.2 Calculate the mean of valid BB pixel counts in this calibration period.

For each calibration period as for the infra-red calibration:

Calculate the mean of the black body pixel counts as in steps 5.7.3-7 and 5.7.3-8-6, but for the four visible and near-visible channels (ch = v16, v870, v670, v555). If a valid mean count is obtained, then

```
for each channel ch = v16, v870,v670, v555 set
ch_offset[ch][parity] =
      -mean_bb_pixel_count_cal_period[ch][bb][parity])
                                              (Req.5.8-120)
where bb indicates the selected (px or mx) black body.
```

Do Req. 5.8-120 for each parity, and for all source packets in the calibration period.

Step 5.8.3 Derive slope for each channel.

Extract the channel gains from the auxiliary data arrays via the MUDT as follows

```
Gain[v16][s] = [A0041](scan)
Gain[v870][s] = [A0391](scan)
Gain[v670][s] = [A0381](scan)
Gain[v555][s] = [A0371](scan)
```

For each source packet s in the calibration period, save the scp gain for the v16 channel:

scp_gain{v16, s}= Gain[v16][s]

For each parity, and for each source packet s set:

```
ch_gain[4][s][parity] = [L1-AUX17-2]
ch_gain[5][s][parity] = [L1-AUX17-3]/Gain[5][s]
ch_gain[6][s][parity] = [L1-AUX17-4]/Gain[6][s]
ch_gain[7][s][parity] = [L1-AUX17-5]/Gain[7][s]
                                        (Req.5.8-121)
```

5.9 Module Definition: Satellite Time Calibration

5.9.1 Functional Description

For each scan, the Satellite Binary Time (taken from the source packet) is converted to UTC. The conversion uses a linear relationship between SBT and UTC, defined by parameters taken from the SBT to UTC Conversion Information section of the Main Product Header (MPH) of the product model. This contains parameters UTC0, SBT0 and PER, and the conversion from SBT to UTC is done using the equation

UTC = UTC0 + (SBT - SBT0)*PER.

The arithmetic represented by the above Equation must take account of wrap-around of the satellite clock, and of the possible inclusion of a leap second in UTC between the reference time UTC0 and the converted time UTC (GMSVA 2070/96, CFI Time spec). In practice the conversion is accomplished using the time conversion subroutines documented in (CFI Time spec), which automatically includes these corrections.

5.9.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX19-3	UTCE0	Reference UTC Time (from MPH)	char	n/a	27	1
L1-AUX19-5	SBT0	Reference SBT (from MPH)	ul	counts	4	1
L1-AUX19-6	PER0	SBT clock step (from MPH)	ul	ps	4	1
L0-MDS1-8	SBT	Source Packet SBT	ul	counts	4	1

Table 5-9-1: Input Data Table - Satellite Time calibration

The SBT to UTC conversion parameters shall be taken from the MPH of the product model. The parameters identified by [L1-AUX19-3], [L1-AUX19-5], [L1-AUX19-6] correspond to the parameters in the MPH identified by the keywords "UTC_SBT_TIME=", SAT_BINARY_TIME=" and "CLOCK_STEP=" respectively. [AD11]

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-12	MJDT0[4]	Reference UTC in MJD Format:				1
L1B-INT-13	MJDT0[0]/(1)	Reference UTC days	sl	days	4	1
L1B-INT-14	MJDT0[1]/(2)	Reference UTC seconds	sl	s	4	1
L1B-INT-15	MJDT0[2]/(3)	Reference UTC micros.	sl	micros	4	1
L1B-INT-16	MJDT0[3]/(4)	Reference deltaUT1	sl	micros	4	1
(local)	MJDP0[0]/(1)	Reference UTC in processing format	double	days	8	1
(local)	MJDP0[1]/(2)	Reference deltaUT1	double	s.	8	1
L1B-INT-17	DUT1E0	delta UT1	char	n/a	8	1
L1B-INT-18	MJDT[4]	Scan UTC time in MJD format as follows:				1
L1B-INT-19	MJDT[0]/(1)	Scan UTC days	sl	days	4	1
L1B-INT-20	MJDT[1]/(2)	Scan UTC seconds	sl	s	4	1
L1B-INT-21	MJDT[2]/(3)	Scan UTC micros.	sl	micros	4	1
L1B-INT-22	MJDT[3]/(4)	Scan deltaUT1	sl	micros	4	1
L1B-INT-23	MJDP[0]/(1)	Scan UTC in processing format	double	days	8	1
L1B-INT-24	MJDP[1]/(2)	Scan deltaUT1	double	s	8	1
(local)	UTCE	Scan UTC Time	char	n/a	27	1
(local)	DUT1E	delta UT1 for scan	char	n/a	8	1
(local)	OBTM	On-board time parameter: most significant word	ul	n/a	4	1
(local)	OBTL	On-board time parameter: most significant word	ul	n/a	4	1
L1B-INT-25	ST_status	status flag	sl	n/a	4	1
	s	Index to instrument scans	sl	none	4	1
L1B-INT-400		source_packet_ut_time(s)	double	days	8	

Table 5-9-2: Internal Data Table - Satellite Time calibration

5.9.3 Algorithm Definition

Step 5.9.3-1 Convert Reference Time to Transport Format

The subroutine `emjd` from the CFI library is used to convert the reference time from external (character string) format to transport format. The time in processing format is produced as a byproduct. For this step parameter DUT1E0 is to be initialised as follows:

```
DUT1E0 = '+.000000' (ASCII string)
```

Other parameters are as defined in Tables 5-9-1 and 5-9-2.

```
status = pl_emjd(MJDT0, MJDP0, UTCE0, DUT1E0)    (Req.5.9-1)
```

Check that the value of `status` is zero; if it is not, an input error has occurred.

Exectue the following steps for each source packet s:

Step 5.9.3-2 Convert Satellite Binary Time to UTC

The subroutine `sbtutc` from the CFI library is used to convert the satellite binary time of the scan to UTC. The resultant UTC is in transport format.

```
    OBTM = SBT
    OBTL = 0
status = pl_sbtutc(MJDT0, &SBT0, &PER0, &OBTM, &OBTL, MJDT)
                                                   (Req.5.9-2)
```

Check that the value of `status` is zero; if it is not, either an input or an output error has occurred.

Step 5.9.3-3 Convert Scan UT from Transport to Processing Format

The subroutine `tmjd` from the CFI library is used to convert the scan time from transport format to processing format. The time in external (character string) format is produced as an (unwanted) byproduct.

```
status = pl_tmjd(MJDT, MJDP, UTCE, DUT1E)                  (Req.5.9-3)
```

Check that the value of `status` is zero; if it is not, an input error has occurred.

```
Source_packet_ut_time(s) = MJDP[0]/(1)                     (Req.5.9-4)
```

5.10 Module Definition: Generate Geolocation Grid

5.10.1 Functional Description

This module generates tables of the latitudes and longitudes of 'tie' points for use in the regridding. Firstly a table is derived giving the latitudes and longitudes of a series of points on the ground track, equally spaced in time at an interval corresponding to 1 granule. This table defines the ground track, for use by the geolocation algorithm, and starts sufficiently far south of the ascending node to permit the regridding of scans which are north of the equator in the forward view although the subsatellite point is south of the equator.

Further tables are then computed to give the co-ordinates of image points on a uniform grid displaced from the ground track in x in 25 km steps. The grid begins at the start of the first granule of the product (y coordinate 0) and ends sufficiently far beyond the end of data to ensure that the last instrument scan can be fully geolocated. There are 23 across-track points for each y value. The latitudes and longitudes of the image pixels will subsequently be determined by interpolation in this table.

5.10.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX20-4	UTCE1	Orbit State UTC (External Format) (from MPH)	27*uc	n/a	27	1
L1-AUX20-5	DUTC1	UT1 - UTC (from MPH)	8*uc	n/a	8	1
L1-AUX20-6	X	X coordinate (from MPH)	12*uc	m	12	1
L1-AUX20-7	Y	Y coordinate (from MPH)	12*uc	m	12	1
L1-AUX20-8	Z	Z coordinate (from MPH)	12*uc	m	12	1
L1-AUX20-9	Xdot	X velocity (from MPH)	12*uc	m/s	12	1
L1-AUX20-10	Ydot	Y velocity (from MPH)	12*uc	m/s	12	1
L1-AUX20-11	Zdot	Z velocity (from MPH)	12*uc	m/s	12	1
L1-AUX16-12	a_e	EARTH_MAJOR_AXIS	double	km	8	1
L1-AUX16-13	ΔY	Along-track sampling interval	double	km	8	1
L1-AUX16-14	ΔT	Uniform time step	double	days	8	1
L1-AUX16-15	NGRANULE	Number of image rows per granule	sl	none	4	1
L1-AUX16-16	NGRID	Number of grid rows	sl	none	4	1
L1-AUX16-17	K	Displacement of table start before ascending node	sl	none	4	1
L1-AUX16-18	e	eccentricity of ellipsoid	double	none	8	1
L1-AUX16-19	ε	Geodetic parameter (square of 'second eccentricity')	double	none	8	1
See Note	psv_flag	PREDICTED/RESTITUTED flag	10*uc	n/a	10	1
L1-AUX20-3	psv_source	Source of Orbit Vectors	2*uc	n/a	2	1
See Note	psv_fpath	psv file path	uc (note 2)	n/a		1

Table 5-10-1: Input Data Table - Generate Geolocation Grid

Note 1: The PF-HS will supply the operational processor with the name of an orbit state vector file in the "work order". This may be the name of either a predicted orbit state vector file or a restituted orbit state vector file. If the filename is that of a predicted orbit state vector, the orbit state vector provided in the MPH of the Product Model will be used, and the corresponding file will not be read. In this case the state vector values will be those values from the MPH identified by the keywords defined in Table 5-10-3.

Predicted Orbit State Vector Files may be either of:

FOS Predicted Orbit File (AUX_FPO_AX);
Doris Navigator Level 0 File.

If the operational processor is supplied with the name of a restituted Orbit state vector file, the supplied file will be used for orbit determination. In this case the file will be one of the following:

FOS Restituted Orbit State Vectors File (AUX_FRO_AX);
Doris Preliminary Orbit State Vectors Product (DOR_POR_2P);
Doris Precise Orbit State Vectors Product (DOR_VOR_2P);

The values of psv_flag and psv_source depend upon the type of the supplied orbit file as specified in Table 5-10-4.

Note 2: the size of the psv_fpath parameter must be sufficient to accommodate a file pathname on the operational processor system.

Parameter ID	Variable	Name	Type	Units	Field size	Fields
	mjdp1[2]	time of state vector (processing format)	double	days	8	2
	pos[3]	position vector	double	m	8	3
	vel[3]	velocity vector	double	m/s	8	3
L1B-INT-30/31	mjdp2[2]	ascending node time (processing format)	double	days	8	2
L1B-INT-32/37	xm[6]	mean kepler state	double	mixed	8	6
L1B-INT-38	x[0]/(1)	osc. semi-major axis	double	m	8	1
L1B-INT-39	x[1]/(2)	osc. eccentricity	double	none	8	1
L1B-INT-40	x[2]/(3)	osc. inclination	double	deg	8	1
L1B-INT-41	x[3]/(4)	osc. r.a. of node	double	deg	8	1
L1B-INT-42	x[4]/(5)	osc. arg. of perigee	double	deg	8	1
L1B-INT-43	x[5]/(6)	mean anomaly	double	deg	8	1
L1B-INT-44	acc[0]/(1)	X component of acceleration	double	m/s/s	8	1
L1B-INT-45	acc[1]/(2)	Y component of acceleration	double	m/s/s	8	1
L1B-INT-46	acc[2]/(3)	Z component of acceleration	double	m/s/s	8	1
L1B-INT-47	res[54]	results array	db array	misc.	8	54
L1B-INT-48	ierr[4]	Error Flag array	sl array	n/a	4	4
L1B-INT-56	T0	Time of first product granule	double	days	8	1
	i	index to along-track grid points (i = 0, 1, 2, ..., NGRID-1)	sl	none	4	1
	j	index to across-track grid points (j = 0, 1, ..., 22)	sl	none	4	1
	k	index to equal time samples (k = 0, 1, 2, ..., NGRID+K-1)	sl	none	4	1
L1B-INT-51	track_lat(k)	track latitude	double	deg.	8	NGRID+K
L1B-INT-52	track_long(k)	track longitude	double	deg.	8	NGRID+K
L1B-INT-53	t(k)	fixed time step array	double	days	8	NGRID+K
L1B-INT-54	s(k)	along track distance	double	km.	8	NGRID+K
L1B-INT-55	track_y(k)	along track distance relative to asc. node	double	km.	8	NGRID+K
local	ds(k)	distance increment	float	m	4	NGRID+K
local	vephi(k)	Northerly component of ground trace velocity	double	m/s	8	

local	velam(k)	Easterly component of ground trace velocity	double	m/s	8	
	i	along-track index to geolocation grid	sl	none	4	1
	j	across-track index to geolocation grid	sl	none	4	1
internal	φ_1	Latitude at first point	double	radians	8	1
internal	φ_2	Latitude at second point	double	radians	8	1
internal	ν_1	Radius of curvature (in prime vertical) at point 1	double	km	8	1
internal	ν_2	Radius of curvature (in prime vertical) at point 2	double	km	8	1
internal	S_ψ	Auxiliary trigonometric factor	double	none	8	1
internal	C_ψ	Auxiliary trigonometric factor	double	none	8	1
internal	ψ	Auxiliary angle (modified latitude)	double	radian	8	1
internal	g	coefficient of geodetic series expansion	double	none	8	1
internal	h	coefficient of geodetic series expansion	double	none	8	1
internal	r_2	coefficient of geodetic series expansion	double	none	8	1
internal	r_3	coefficient of geodetic series expansion	double	none	8	1
internal	L	across-track distance to tie point	double	km	8	1
internal	θ	Angular arc corresponding to L	double	radians	8	1
internal	ε	geodetic correction coefficient	double	none	8	1
internal	Sj	Auxiliary trigonometric factor	double	none	8	1
internal	Cj	Auxiliary trigonometric factor	double	none	8	1
internal	A	Angle at N pole	float	radian	4	1
internal	B	Auxiliary azimuth angle	float	radian	4	1
internal	C	Auxiliary azimuth angle	float	radian	4	1
internal	a	angular distance increment	float	radian	4	1
internal	b	co-latitude at first point	float	radian	4	1
internal	c	co-latitude at second point	float	radian	4	1
internal	β	Intermediate azimuth	float	radians	4	1
internal	γ	Intermediate azimuth	float	radians	4	1
internal	α'	Longitude interval to mid-point	float	radians	4	1
internal	β'	Ground track azimuth	float	radians	4	1
internal	ψ_j	co-longitude of grid point (j= 0, ... 22)	float	radians	4	1
internal	δ_j	azimuth of line segment at point j (j= 0, ... 22)	float	radians	4	1
internal	ε_j	reverse azimuth at point j (j= 0, ... 22)	float	radians	4	1
internal	Δ	angular half-interval of grid in y	float	radians	4	1
internal	D	angular interval of grid in x	float	radians	4	1
constant	π	the mathematical constant (= 3.14159265359)	float	none	4	1
L1B-INT-26	grid_lat(i, j)	grid latitude	float	deg.	4	j = 0, 22
L1B-INT-27	grid_long(i, j)	grid longitude	float	deg.	4	j = 0, 22
local	mode	mode indicator for po_ppforb or po_interpol	sl	none	4	1
L1B-INT-385	choice	Orbit file selection switch	sl	none	4	1
L1B-INT-386	ndc	Number of DORIS precise orbit files	sl	none	4	1
L1B-INT-387		doris_precise_file	char	n/a		1
L1B-INT-388	ndp	Number of DORIS preliminary files	sl	none	4	1
L1B-INT-389		doris_prelim_file	char	n/a		1
L1B-INT-390	ner	Number of FOS restiruted orbit files	sl	none	4	1
L1B-INT-391		esoc_rest	char	n/a		1
L1B-INT-392	mjdr0	Start or Requested UTC	double	days	8	1
L1B-INT-393	mjdr1	End UTC	double	days	8	1
L1B-INT-394	mjdp_int[2]	UTC of state vector (po_interpol)	double		2	1
L1B-INT-395	selected	(for use by po_interpol)	sl	none	4	1
L1B-INT-396	ierr_interpol[10]	Error Flag array for po_interpol	sl array	n/a	4	10
local	status	local status word for CFI subroutine calls	sl	n/a	4	1

Table 5-10-2: Internal Data Table - Generate Geolocation Grid

Parameter ID	Keyword
L1-AUX20-3	"VECTOR_SOURCE="
L1-AUX20-4	"STATE_VECTOR_TIME="
L1-AUX20-5	"DELTA_UT1="
L1-AUX20-6	"X_POSITION="
L1-AUX20-7	"Y_POSITION="

L1-AUX20-8	"Z_POSITION="
L1-AUX20-9	"X_VELOCITY="
L1-AUX20-10	"Y_VELOCITY="
L1-AUX20-11	"Z_VELOCITY="
L1-AUX20-12	"ABS_ORBIT="

Table 5-10-3: Correspondence between Parameter ID and SPH Keyword.

Supplied orbit file	psv_source	psv_flag
FOS Predicted Orbit File (AUX_FPO_AX)	FP	PREDICTED
Doris Navigator Level 0 File	DN	PREDICTED
FOS Restituted Orbit State Vectors File (AUX_FRO_AX)	FR	RESTITUTED
Doris Preliminary Orbit State Vectors Product (DOR_POR_2P)	DI	RESTITUTED
Doris Precise Orbit State Vectors Product (DOR_VOR_2P)	DP	RESTITUTED

Table 5-10-4. Values of psv_source and psv_flag

5.10.3 Algorithm Definition

Step 5.10.1 Initialise Orbit Propagator

Execute an initialisation call to the appropriate orbit propagator subroutine (po_ppforb or po_interpol) identical to that described in Section 5.11.3 (Step 5.11.2).

`(Req 5.10-1)`

For k = 0, NGRID + K - 1 perform steps 5.10.2 and 5.10.3 (including sub-steps 5.10.3.1 to 5.10.3.3).

Step 5.10.2 Generate along-track time-distance LUT

The look-up table defining points on the ground track at time interval ΔT is generated using the orbit propagator. The time step ΔT is chosen to correspond to one granule. The sequence of times t(k) is defined by

$$t(k) = T0 + (k - K)\Delta T, \quad k = 0, 1, 2, 3, ..., NGRID + K - 1$$

`(Req 5.10-2)`

Depending upon the source of orbit information, either po_ppforb or po_interpol is used to find the latitude and longitude of the sub-satellite point at time t(k) for each k. Thus for each t(k), the relevant subroutine is called as follows:

if psv_flag = "PREDICTED" **then**

&mode: pointer to mode = 2 (propagation call, absolute time value).

mjdr = mjdp2 (time of ascending node crossing from initialisation call).

xm = Kepler state at ascending node crossing from initialisation call.

```
mjdp[0]/(1)  = t(k)  (UTC time of required results).
mjdp[1]/(2)  = 0.0.
status = po_ppforb(&mode, mjdr, xm,
                          mjdp, x, pos, vel, acc, res, ierr);
```

Check that the value of status is zero; if it is not, check errors and warnings.

else if psv_flag = "RESTITUTED" **then**

```
mode = PO_INTERPOLATE;  /* 2 */
mjdr0 = granule_time[k];
```

```
status = po_interpol(&mode, &choice,
                     &ndc, doris_precise_file,
                     &ndp, doris_prelim_file,
                     &ner, esoc_rest,
                     &mjdr0, &mjdr1,
                     mjdp_int, x, pos, vel, acc,
                     &selected, res, ierr_interpol);
```

Check that the value of status is zero; if it is not, check errors and warnings.

end if

The required outputs are

```
track_long[k] = res[7]/(8),
track_lat[k] = res[8]/(9);
```

and the velocity components

```
velam[k] = res[37]/(38)
vephi[k] = res[36]/(37)
```

(Req 5.10-3)

The length of the line segment ds[k] between points k-1 and k is derived as follows.

Convert the latitudes to radians:

$\varphi_1 = \pi(\text{track\_lat[k-1]})/180.0$

$\varphi_2 = \pi(\text{track\_lat[k]})/180.0$

(Req 5.10-4)

Calculate the radii of curvature in prime vertical at the points k-1, k as follows:

$$\nu_1 = EARTH\_MAJOR\_AXIS/\left(1 - e^2 \sin^2(\varphi_1)\right)^{1/2}$$

$$\nu_2 = EARTH\_MAJOR\_AXIS/\left(1 - e^2 \sin^2(\varphi_2)\right)^{1/2}$$

$$S_\psi = (1 - e^2)\sin\varphi_2 + (\nu_1/\nu_2)e^2 \sin\varphi_1$$

$$C_\psi = \cos\varphi_2$$

$$\psi = \text{atan2}\left(S_\psi, C_\psi\right)$$

(Req 5.10-5)

Define the angles (expressed in radians)

$b = \pi/2 - \varphi_1$

$c = \pi/2 - \psi$

$A = \pi(\text{track\_long[k-1] - track\_long[k]})/180.0$

and use the technique described under Step 5.10.5 to solve the spherical triangle ABC (Figure 5.10-1) for side a and angles B and C. (B is a byproduct not needed at this stage.)

Solve spherical triangle {A, b, c; B, C, a}

(Req 5.10-6)

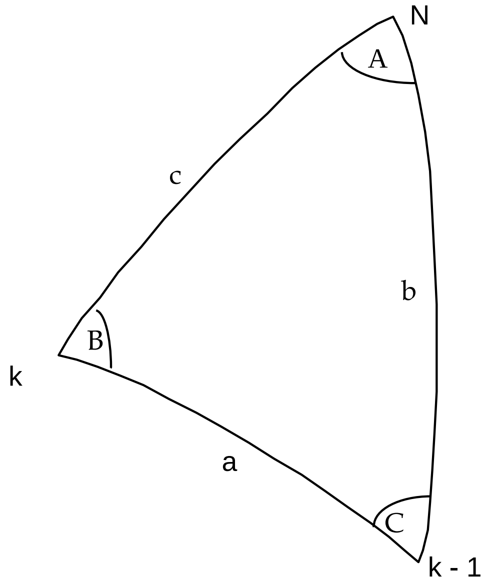


Figure 5.10-1

Calculate the geodetic correction coefficients

$$g = \varepsilon \sin\varphi_1 \cos\varphi_1 \cos C$$

$$h = \varepsilon \cos^2 \varphi_1 \cos^2 C$$

(Req 5.10-7)

where ε is parameter L1-AUX16-19, the value of which will actually be

$$\varepsilon = e^2 \big/ (1 - e^2) .$$

Calculate the line segment between points k-1 and k from the angle a:

$$\mathrm{ds(k)} = \nu_1 a \left[1 - \frac{a^2}{6} h(1-h) + \frac{a^3}{8} g(1-2h) \right]$$

(Req 5.10-8)

Calculate the along-track distance by

s(0) = 0.0

s(k) = s(k-1) + ds(k), k = 1, 2, ... NGRID + K - 1

(Req 5.10-9)

If $k \geq K$ do Step 5.10.3 (sub-steps 5.10.3.1 to 5.10.3.3).

(Req 5.10-10)

Step 5.10.3 Generate Geolocation Grids

Starting with the tabular along-track latitudes and longitudes, the lat and long of every set of 23 across-track points corresponding to the along track grid points are calculated, using spherical trigonometry. The following calculations are repeated for each value of *i (i = 0, NGRID - 1)* where *i = k - K*.

(Req 5.10-11)

Step 5.10.3.1 Fill in central column of the grid

```
grid_lat(i, 11) = track_lat[i + K]
grid_long(i, 11) = track_long[i + K],
converted to lie in the range -180 to 180 by the subtraction (or addition) of 360 if required.
β′ = π + atan2{velam[k], vephi[k]}.
```

(Note that the azimuth given by the atan2 function should be a negative angle at this point).

(Req 5.10-12)

Step 5.10.3.2 Fill in grid points North of ground track

The latitudes and longitudes of 11 points to the left and right of the sub-satellite point are calculated, again using an interval of exactly 25.0 km between the points. These points are in the across-track direction, and lie on the normal section locally orthogonal to the sub-satellite track. The azimuth of this normal section is $\beta' - \frac{1}{2}\pi$:

$$\delta = \beta' - \tfrac{1}{2}\pi$$

$\varphi = \pi($`track_lat[i + K]`$)/180.0$

Calculate

$$r_2 = -\varepsilon \cos^2\varphi \cos^2\delta$$

$$r_3 = 3\varepsilon(1 - r_2)\cos\varphi \sin\varphi \cos\delta$$

(Req 5.10-13)

where ε is as defined at (Req. 5.10-7).

Calculate the radius in prime vertical at latitude φ,

$$\nu_1 = EARTH\_MAJOR\_AXIS / \left(1 - e^2 \sin^2(\varphi)\right)^{1/2}$$

For j = 12, 13, ... 22

$L = 25\ (j - 11)$

$$\vartheta = \frac{L}{\nu_1}\left[1 - \frac{r_2(1+r_2)}{6}\left(\frac{L}{\nu_1}\right)^2 - \frac{r_3(1+3r_2)}{24}\left(\frac{L}{\nu_1}\right)^3\right]$$

$$\psi = \tfrac{1}{2}\pi - \varphi$$

(Req 5.10-14)

The quantities α_{j-1} and ψ_j are given by the solution of a further spherical triangle:

Solve spherical triangle $\{\delta, \psi, \vartheta; \varepsilon_{j-1}, \alpha_{j-1}, \psi_j\}$

$$\rho = 1 - \tfrac{1}{2} r_2 \vartheta^2 - \tfrac{1}{6} r_3 \vartheta^3$$

$$S_j = \cos\psi_j - e^2 \rho \sin\varphi$$

$$C_j = (1 - e^2)\sin\psi_j$$

(Req 5.10-15)

The latitudes of the remaining tabular grid points on the ground track are given by

grid_lat(i, j) = (180/π) atan2$\left(S_j, C_j\right)$

and their longitudes by

grid_long(i, j) = grid_long(i, 11) + $(180.0 \cdot \alpha_{j-1}) / \pi$.

Each longitude grid_long(i, j) should be converted to lie in the range -180 to 180 by the subtraction or addition of 360 if required.

(Req 5.10-16)

Step 5.10.3.3 Fill in grid points South of ground track

$$\tilde{\varepsilon} = \beta' + \tfrac{1}{2}\pi$$

φ = π(`track_lat[i + K]`)/180.0

Calculate

$$r_2 = -\varepsilon \cos^2\varphi \cos^2\tilde{\varepsilon}$$

$$r_3 = 3\varepsilon(1 - r_2)\cos\varphi \sin\varphi \cos\tilde{\varepsilon}$$

(Req 5.10-17)

Calculate the radius in prime vertical at latitude φ,

ν_1 = *EARTH_MAJOR_AXIS*/$\left(1 - e^2 \sin^2(\varphi)\right)^{1/2}$

For j = 0, 1, ...10

L = 25 (11 - j)

$$\vartheta = \frac{L}{\nu_1}\left[1 - \frac{r_2(1 + r_2)}{6}\left(\frac{L}{\nu_1}\right)^2 - \frac{r_3(1 + 3r_2)}{24}\left(\frac{L}{\nu_1}\right)^3\right]$$

$$\psi = \tfrac{1}{2}\pi - \varphi$$

(Req 5.10-18)

The quantities α_j and ψ_j are given by the solution of a further spherical triangle:

Solve spherical triangle $\left\{\tilde{\varepsilon}, \vartheta, \psi; \alpha_j, \delta_j, \psi_j\right\}$

$$\rho = 1 - \tfrac{1}{2} r_2 \vartheta^2 - \tfrac{1}{6} r_3 \vartheta^3$$

$$S_j = \cos\psi_j - e^2 \rho \sin\varphi$$

$$C_j = (1 - e^2) \sin\psi_j$$

(Req 5.10-19)

The latitudes of the remaining tabular grid points are given by

grid_lat(i, j) = (180/π) atan2$\left(S_j, C_j\right)$

and their longitudes by

grid_long(i, j) = grid_long(i, 11) + $(180.0 \cdot \alpha_j) / \pi$.

Each longitude grid_long(i, j) should be converted to lie in the range -180 to 180 by the subtraction or addition of 360 if required.

Note that a consistent sign convention is being adopted for the azimuths, so that the sign of alpha[j] will be positive for eastward displacements, negative for westward.

(Req 5.10-20)

```
End for
```

Step 5.10.4 Redefine origin of y.

```
For all k = 0, NGRID + K -1
```

track_y(k) = s(k) - s(K)

(Req 5.10-21)

```
End for
```

Step 5.10.5 Solution of the spherical triangles

Each of the angles above is derived from the solution of a spherical triangle in which two sides and the included angle are given. All quantities are derived using a single subroutine devised to treat this case. This approach permits a simple and consistent formulation of the algorithm.

Suppose ABC is a spherical triangle, and suppose that its sides, in angular measure, are designated by a, b, c according to the usual convention. Thus a represents the length of the side BC, opposite the angle A, and so on. Given the sides b and c, together with the included angle A, the remaining quantities a, B and C are to be determined. Proceed as follows:

(i) Firstly, an application of the cosine rule gives

$$\cos a = \cos b \cos c + \sin b \sin c \cos A$$

(Req 5.10-22)

(ii) The sine rule can be written

$$\sin A / \sin a = \sin B / \sin b = \sin C / \sin c$$

from which we can deduce, multiplying by $\sin a \sin b \sin c$ throughout

$$\sin b \sin c \sin A = \sin a \sin c \sin B = \sin a \sin b \sin C$$

The LHS is known, so we can write

$$\sin a \sin c \sin B = \sin b \sin c \sin A$$

$$\sin a \sin b \sin C = \sin b \sin c \sin A$$

(Req 5.10-23)

(iii) Further applications of the cosine formula to the sides b an c respectively give after rearrangement

$$\sin a \sin c \cos B = \cos b - \cos a \cos c$$

$$\sin a \sin b \cos C = \cos c - \cos a \cos b$$

All the quantities on the RHS of (iii) are given apart from cos b, which is derived from (i).

Thus relations (ii) and (iii) give expressions for the tangent or cotangent of the angles B and C, from which the angles themselves can be derived directly by the use of the atan2 function.

The side a might be deduced directly from (i); but if the magnitude of a is small, there might be a loss of precision in this case. In order to allow for this possibility, once B has been calculated it is possible to calculate sin a by a further application of the sine rule (ii);

$$\sin a = \sin b \sin A / \sin B$$

and to use the atan2 function again.

We denote this transformation by

Solve spherical triangle{A, b, c; B, C, a}

(Req 5.10-24)

5.11 Module Definition: Predict ENVISAT Orbit

5.11.1 Functional Description

For each scan, the orbital position of ENVISAT is calculated using one of the ESA-supplied CFI orbit propagator subroutines po_ppforb or po_interpol. The subroutine used will depend on the type of the supplied orbit state vector file.

The operational processor will be supplied with the name of an orbit state vector file. This may be the name of either a predicted orbit state vector file or a restituted orbit state vector file. If the filename is that of a predicted orbit state vector, the orbit state vector provided in the MPH of the Product Model will be used with the subroutine po_ppforb, and the corresponding file will not be read. Predicted Orbit State Vector Files may be either of:

FOS Predicted Orbit File (AUX_FPO_AX);
Doris Navigator Level 0 File.

If the operational processor is supplied with the name of a restituted orbit state vector file, the supplied file will be used for orbit determination in conjunction with subroutine po_interpol. In this case the file will be one of the following:

FOS Restituted Orbit State Vectors File (AUX_FRO_AX);
Doris Preliminary Orbit State Vectors Product (DOR_POR_2P);
Doris Precise Orbit State Vectors Product (DOR_VOR_2P);

For each scan, the orbital position of ENVISAT is calculated using the appropriate orbit propagator subroutine. An initialisation call is first made to the orbit propagator subroutine. Thereafter the parameters determined during initialisation are used in further calls to the subroutine, with time argument defined by the time tag of the source packet.

5.11.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX20-4	UTCE1	Orbit State UTC (External Format) (from orbit file)	27*uc	n/a	27	1
L1-AUX20-5	DUTC1	UT1 - UTC (from orbit file)	8*uc	n/a	8	1
L1-AUX20-6	X	X coordinate (from orbit file)	12*uc	m	12	1
L1-AUX20-7	Y	Y coordinate (from orbit file)	12*uc	m	12	1
L1-AUX20-8	Z	Z coordinate (from orbit file)	12*uc	m	12	1
L1-AUX20-9	Xdot	X velocity (from orbit file)	12*uc	m/s	12	1
L1-AUX20-10	Ydot	Y velocity (from orbit file)	12*uc	m/s	12	1
L1-AUX20-11	Zdot	Z velocity (from orbit file)	12*uc	m/s	12	1
See Note 1	psv_flag	PREDICTED/RESTITUTED flag	10*uc	n/a	10	1
L1-AUX20-4	psv_source	Source of Orbit Vectors	2*uc	n/a	2	1
See Note 1	psv_fpath	psv file path	uc (note 2)	n/a		1

Table 5-11-1: Input Data Table - Geolocate Pixels

Note 1: the values of psv_flag and depend on the type of the external orbit file specified by psv_fpath, as defined it Table 5-11-4.

Note 2: the size of the psv_fpath parameter must be sufficient to accommodate a file pathname on the operational processor system.

Parameter ID	Variable	Name	Type	Units	Field size	Fields
	mode	mode indicator	sl	n/a	4	
	mjdp1[0]	UT (proc. format)	double	days	8	
	mjdp1[1]	delta UT1	double	s	8	
	MJDT1[0]	State UTC days	sl	days	4	
	MJDT1[1]	State UTC seconds	sl	s	4	
	MJDT1[2]	State UTC micros.	sl	micros	4	
	MJDT1[3]	State deltaUT1	sl	micros	4	
	pos0[0]	State X co-ordinate	double	m	8	
	pos0[1]	State Y co-ordinate	double	m	8	
	pos0[2]	State Z co-ordinate	double	m	8	
	vel0[0]	State X velocity	double	m/s	8	
	vel0[1]	State Y velocity	double	m/s	8	
	vel0[2]	State Z velocity	double	m/s	8	
L1B-INT-23	MJDP[0]/(1)	Scan UTC in processing format	double	days	8	1
L1B-INT-24	MJDP[1]/(2)	Scan deltaUT1	double	s	8	1
L1B-INT-29	geoloc_status	status flag	sl	n/a	4	1
L1B-INT-30	mjdp2[0]	ascending node time	double	days	8	1
L1B-INT-31	mjdp2[1]	delta UT1 at mjdr[0]	double	s	8	1
L1B-INT-32	xm[0]	mean semi-major axis	double	m	8	1
L1B-INT-33	xm[1]	mean eccentricity	double	none	8	1
L1B-INT-34	xm[2]	mean inclination	double	deg	8	1
L1B-INT-35	xm[3]	mean r.a. of node	double	deg	8	1
L1B-INT-36	xm[4]	mean arg. of perigee	double	deg	8	1

L1B-INT-37	xm[5]	mean anomaly	double	deg	8	1
L1B-INT-38	x[0]	osc. semi-major axis	double	m	8	1
L1B-INT-39	x[1]	osc. eccentricity	double	none	8	1
L1B-INT-40	x[2]	osc. inclination	double	deg	8	1
L1B-INT-41	x[3]	osc. r.a. of node	double	deg	8	1
L1B-INT-42	x[4]	osc. arg. of perigee	double	deg	8	1
L1B-INT-43	x[5]	mean anomaly	double	deg	8	1
L1B-INT-44	acc[0]	X component of acceleration	double	m/s/s	8	1
L1B-INT-45	acc[1]	Y component of acceleration	double	m/s/s	8	1
L1B-INT-46	acc[2]	Z component of acceleration	double	m/s/s	8	1
L1B-INT-47	res[54]	results array	double array	misc.	8	54
L1B-INT-48	ierr[4]	Error Flag array	sl array	n/a	4	4
L1B-INT-50	ve	ssp ground speed	float	km/s	4	
local	mode	mode indicator for po_ppforb or po_interpol	sl	n/a	4	
L1B-INT-385	choice	Orbit file selection switch	sl	none	4	1
L1B-INT-386	ndc	Number of DORIS precise orbit files	sl	none	4	1
L1B-INT-387		doris_precise_file	char	n/a		1
L1B-INT-388	ndp	Number of DORIS preliminary files	sl	none	4	1
L1B-INT-389		doris_prelim_file	char	n/a		1
L1B-INT-390	ner	Number of FOS restiruted orbit files	sl	none	4	1
L1B-INT-391		esoc_rest	char	n/a		1
L1B-INT-392	mjdr0	Start or Requested UTC	double	days	8	1
L1B-INT-393	mjdr1	End UTC	double	days	8	1
L1B-INT-395	selected	(for use by po_interpol)	sl	none	4	1
L1B-INT-394	mjdp_int[2]	UTC of state vector (po_interpol)	double		8	2
L1B-INT-396	ierr_interpol[10]	error flag array for po-interpol	sl	n/a	4	10
local	status	local status word for CFI subroutine calls	sl	n/a	4	1

Table 5-11-2: Internal Data Table - Geolocate Pixels

Parameter ID	Keyword
L1-AUX20-3	"VECTOR_SOURCE="
L1-AUX20-4	"STATE_VECTOR_TIME="
L1-AUX20-5	"DELTA_UT1="
L1-AUX20-6	"X_POSITION="
L1-AUX20-7	"Y_POSITION="
L1-AUX20-8	"Z_POSITION="
L1-AUX20-9	"X_VELOCITY="
L1-AUX20-10	"Y_VELOCITY="
L1-AUX20-11	"Z_VELOCITY="
L1-AUX20-12	"ABS_ORBIT="

Table 5-11-3: Correspondence between Parameter ID and SPH Keyword.

Supplied orbit file	psv_source	psv_flag
FOS Predicted Orbit File (AUX_FPO_AX)	FP	PREDICTED
Doris Navigator Level 0 File	DN	PREDICTED
FOS Restituted Orbit State Vectors File (AUX_FRO_AX)	FR	RESTITUTED
Doris Preliminary Orbit State Vectors Product (DOR_POR_2P)	DI	RESTITUTED
Doris Precise Orbit State Vectors Product (DOR_VOR_2P)	DP	RESTITUTED

Table 5-11-4. Values of psv_source and psv_flag

5.11.3 Algorithm Definition

Step 5.11.1. Convert Time of orbit state vector to Processing Format.

if psv_flag = "PREDICTED" then

The subroutine emjd from the time conversion library is used to convert the time associated with the orbit state vector from external (character string) format to processing format. The time in transport format is produced as a byproduct. Use as input parameters of the call UTCE1 (L1-AUX20-4) and DUTC1 (L1-AUX20-5).

```
status = pl_emjd(MJDT1, MJDP1, UTCE1, DUTC1)
```

Check that the value of status is zero; if it is not, an input error has occurred.

(Req 5.11-1)

end if. Perform this step only in this case.

Step 5.11.2. Initialise Orbit Propagator.

Depending upon the source of the orbit state vector information, an initialisation call is made to either po_ppforb or po_interpol.

if psv_flag = "PREDICTED" **then**

An initialisation call is made to the orbit propagator subroutine po_ppforb to generate the initial orbit state vector required by subsequent calls. The input parameters required by the subroutine call are initialised appropriately as follows:

&mode: pointer to mode indicator (mode= 0).

mjdp = MJDP1

pos = vector {X, Y, Z} = vector {L1-AUX20-6, L1-AUX20-7, L1-AUX20-8} where the external variables, in ASCII format, are converted to internal type double and correct units as defined the Product Specification (AD11) Chapter 16.

vel = vector{Xdot, Ydot, Zdot}
= vector {L1-AUX20-9, L1-AUX20-10, L1-AUX20-11}
where the external variables, in ASCII format, are converted to internal type double and correct units as defined the Product Specification (AD11) Chapter 16.

mjdr = mjdp2

The call is then

```
status =
      po_ppforb(&mode, mjdr, xm, mjdp, x, pos, vel, acc, res, ierr).
```

Check that the value of status is zero; if it is not, an input error has occurred.

(Req 5.11-2)

```
ascending_node_time[0] = mjdr[0];
ascending_node_time[1] = mjdr[1];
```

else if psv_flag = "RESTITUTED" **then**

```
mode = 0;  choice = 0;
ndc = 0;   ndp = 0;  ner = 0;
if (psv.source = "DP") then
  ndc = 1;  doris_precise_file = &psv_filepath;
else if (psv.source = "DI")
  ndp = 1;  doris_prelim_file = &psv_filepath;
else if (psv.source = "FR")
  ner = 1;  esoc_rest = &psv_filepath;
end if
```

```
mjdr0 = T0 + (-cte->kk)*cte->delta_t;
                  /* earliest grid time for interpolation */
mjdr1 = T0 + (cte->ngrid-1)*cte->delta_t;
                  /* latest   grid time for interpolation */
status = po_interpol(&mode, &choice,
                     &ndc, doris_precise_file,
                     &ndp, doris_prelim_file,
                     &ner, esoc_rest,
                     &mjdr0, &mjdr1,
                     mjdp_int, x, pos, vel, acc,
                     &selected, res, ierr_interpol);
```

Check that the value of status is zero; if it is not, an input error has occurred.

end if

Step 5.11.3. Call Orbit Propagator at time of scan.

if psv_flag = "PREDICTED" **then**

For each scan, the orbit propagator subroutine po_ppforb is entered with the scan time as parameter, the other input parameters being derived from the initialisation call.

```
&Mode: pointer to mode indicator (mode = 2)
mjdr = mjdp2
xm
mjdp = MJDP
status =
      po_ppforb(&mode, mjdr, xm, mjdp, x, pos, vel, acc, res, ierr).
```

Check that the value of status is zero; if it is not, an input error has occurred.

(Req 5.11-3)

else if psv_flag = "RESTITUTED" **then**

```
mode = PO_INTERPOLATE;  /* 2 */
mjdr0 = granule_time[k];
status = po_interpol(&mode, &choice,
                     &ndc, doris_precise_file,
                     &ndp, doris_prelim_file,
                     &ner, esoc_rest,
                     &mjdr0, &mjdr1,
                     mjdp_int, x, pos, vel, acc,
                     &selected, res, ierr_interpol);
```

Check that the value of status is zero; if it is not, an input error has occurred.

end if

Step 5.11.4. Derive auxiliary quantities from Results Vector

Calculate the magnitude of ground trace velocity, in km/s.

```
ve = {res[38]/(39)}/1000.0.
```

(Req 5.11-4)

5.12 Module Definition: Geolocate Pixels

5.12.1 Functional Description

For each tie point pixel *p* in each scan *s* the direction of the line of sight is determined in the scan reference frame. The corresponding direction cosines are determined, and transformed to the satellite reference frame, and converted back to define an azimuth and elevation. Finally, the CFI TARGET subroutine is used to derive the pixel co-ordinates on the ellipsoid.

5.12.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Size	Fields
L1-AUX16-6	aocs[0]	AOCS parameter Cx	double	deg.	8	1
L1-AUX16-7	aocs[1]	AOCS parameter Cy	double	deg.	8	1
L1-AUX16-8	aocs[2]	AOCS parameter Cz	double	deg.	8	1
L1-AUX16-9	att[0]	AOCS pitch mispointing	double	deg.	8	1
L1-AUX16-10	att[1]	AOCS roll mispointing	double	deg.	8	1
L1-AUX16-11	att[2]	AOCS yaw mispointing	double	deg.	8	1
L1-AUX16-1	κ	cone angle	float	radians	4	1
L1-AUX16-2	ϕ_0	mirror offset angle	float	radians	4	1
L1-AUX16-3	ξ	x misalignment correction	float	radians	4	1
L1-AUX16-4	η	y misalignment correction	float	radians	4	1
L1-AUX16-5	ζ	z misalignment correction	float	radians	4	1
L1-AUX18-5		MAX_NADIR_PIXELS	ss	none	2	1
L1-AUX18-7		MAX_FRWRD_PIXELS	ss	none	2	1
L1-AUX16-22		FIRST_NADIR_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-23		FIRST_FORWARD_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-26	INT_S	Along-track interpolation interval	ss	none	2	1
L1-AUX16-27	INT_P	Across-track interpolation interval	ss	none	2	1

Table 5-12-1: Input Data Table - Geolocate Pixels

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-23	mjdp[0]	Scan time, processing format	double	days	8	
L1B-INT-24	mjdp[1]	delta UT1	double	micros.	8	
	pos[0]	x coordinate	double	m	8	
	pos[1]	y coordinate	double	m	8	
	pos[2]	z coordinate	double	m	8	
	vel[0]	x velocity	double	m/s	8	
	vel[1]	y velocity	double	m/s	8	
	vel[2]	z velocity	double	m/s	8	
L1B-INT-44	acc[0]	x acceleration	double	m/s^2	8	
L1B-INT-45	acc[1]	y acceleration	double	m/s^2	8	
L1B-INT-46	acc[2]	z acceleration	double	m/s^2	8	
	datt[0]	AOCS pitch mispointing rate	double	deg./s	8	
	datt[1]	AOCS roll mispointing rate	double	deg./s	8	
	datt[2]	AOCS yaw mispointing rate	double	deg./s	8	
	idir	direction mode switch	sl	none	4	
	dir[0]	dir[0]/(1): azimuth of line of sight	double		8	
	dir[1]	dir[1]/(2): elevation of line of sight	double		8	
	dir[2]	dir[2]/(3): altitude over earth	double		8	
	dir[3]	dir[3]/(4): dummy input (Initialise to 0.0)	double		8	

	dir[4]	dir[4]/(5): azimuth-rate parameter (Initialise to 0.0)	double		8	
	dir[5]	dir[5]/(6): elevation-rate parameter (Initialise to 0.0)	double		8	
	dir[6]	dir[6]/(7): dummy input (Initialise to 0.0)	double		8	
	dir[7]	dir[7]/(8): dummy input (Initialise to 0.0)	double		8	
	iray	ray tracing model switch	sl	none	4	
	freq	signal frequency	double	Hz	8	
	ieres	extended results vector switch	sl	none	4	
	tstatus	target status flag	double	none	8	
	res[74]	target results vector	double		8	74
	status	status vector	sl		4	15
local	ier	local status word for target subroutine call	sl	n/a	4	1
		direction cosine of line of sight, scan frame		none		
		direction cosine of line of sight, scan frame		none		
		direction cosine of line of sight, scan frame		none		
		direction cosine of line of sight, satellite frame		none		
		direction cosine of line of sight, satellite frame		none		
		direction cosine of line of sight, satellite frame		none		
		transformation matrix from scan frame	fl array	none	4	9
		transformation matrix, z misalignment	fl array	none	4	9
		transformation matrix, y misalignment	fl array	none	4	9
		transformation matrix, x misalignment	fl array	none	4	9
		transformation matrix, platform to s frame	fl array	none	4	9
	azimuth	azimuth of line of sight		degrees		
	elevation	elevation of line of sight		degrees		
	s	scan index	sl	none	4	1
	p	pixel index	sl	none	4	1
	s	index to nadir tie point pixel	sl	none	4	1
	p	index to forward tie point pixel	sl	none	4	1
L1B-INT-60	nadir_lat(s, p)	nadir scan pixel latitude	float	degrees	4	575
L1B-INT-61	nadir_long(s, p)	nadir scan pixel longitude	float	degrees	4	575
L1B-INT-62	frwrd_lat(s, p)	forward scan pixel latitude	float	degrees	4	391
L1B-INT-63	frwrd_long(s, p)	forward scan pixel longitude	float	degrees	4	391

Table 5-12-2: Internal Data Table - Geolocate Pixels

5.12.3 Algorithm Definition

The following steps are carried out for each tie point pixel on each scan s_t = 0, *INT_S*, 2**INT_S*, etc. in the general case these points are

$$p_t^n = 0, INT\_P, 2*INT\_P, ..., MAX\_NADIR\_PIXELS - 1$$

on the nadir scan, and

$$p_t^f = 0, INT\_P, 2*INT\_P, ..., MAX\_FRWRD\_PIXELS - 1$$

on the forward scan. The I/O DD is defined on the basis that the expected value of INT_P is 10. In this case tie points are

$$p_t^n = 0, 10, 20, 30, ..., MAX\_NADIR\_PIXELS - 1$$

on the nadir scan, and

$$p_t^f = 0, 10, 20, 30, ..., MAX\_FRWRD\_PIXELS - 1$$

on the forward scan.

(Req 5.12-1)

For each scan s_t and for each tie point pixel p_t from the above set, the following steps are executed:

Step 5.12.1 Determine line of sight and its direction cosines in the scan reference frame.

$$\phi_p = 2\pi \cdot p / 2000 + \varphi_0$$

where p is the pixel number. The pixel number is calculated from

$$p = p_t^n + \text{FIRST\_NADIR\_PIXEL\_NUMBER}$$

or

$$p = p_t^f + \text{FIRST\_FORWARD\_PIXEL\_NUMBER}$$

as appropriate.

[Implementation note; this is a function of pixel number p only, and need be calculated only once. The direction cosines for pixels p other than the tie point pixels defined above are required for the solar angle calculations in Module 15 (see Req 5.15-3, Req 5.15-5) so it may also be convenient to perform the calculation of Steps 5.12.1 to 5.12.3 inclusive here for all pixels in the forward and nadir views and retain the results.]

$$\lambda = \sin\kappa \cos\phi_p$$

$$\mu = \sin\kappa \sin\phi_p$$

$$\nu = \cos\kappa$$

(Req 5.12-2)

Step 5.12.2 Determine the transformation matrices.

$$M_{ab}(-\kappa) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\kappa & -\sin\kappa \\ 0 & \sin\kappa & \cos\kappa \end{pmatrix}$$ (Req 5.12-3)

$$M_z(\zeta) = \begin{pmatrix} \cos\zeta & \sin\zeta & 0 \\ -\sin\zeta & \cos\zeta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ (Req 5.12-4)

$$M_y(\eta) = \begin{pmatrix} \cos\eta & 0 & -\sin\eta \\ 0 & 1 & 0 \\ \sin\eta & 0 & \cos\eta \end{pmatrix}$$ (Req 5.12-5)

$$M_x(\xi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\xi & \sin\xi \\ 0 & -\sin\xi & \cos\xi \end{pmatrix}$$ (Req 5.12-6)

$$M_{ps} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$ (Req 5.12-7)

(Note that the definition of the reference frame subscripted by *s* now differs from that used in earlier issues of this document.)

Step 5.12.3 Calculate the direction cosines of the line of sight

Calculate the direction cosines of the line of sight relative to the satellite reference frame:

$$\begin{pmatrix} \lambda_s \\ \mu_s \\ \nu_s \end{pmatrix} = M_{ps} M_x(\xi) M_y(\eta) M_z(\zeta) M_{ab}(-\kappa) \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix}$$ (Req 5.12-8)

Convert to azimuth and elevation by inverting the equations

λ_S = cos (elevation) cos (azimuth)

μ_S = cos (elevation) sin (azimuth)

ν_S = sin (elevation)

or therefore

azimuth = atan2 (μ_S, λ_S)

elevation = atan2 (ν_S, sqrt($\lambda_S^2 + \mu_S^2$))

If azimuth < 0.0 then azimuth = azimuth + 360.0.

(Req 5.12-9)

In the above atan2 represents the arc tangent function of two arguments defined in the conventional way, atan2(y, x) being the angle whose tangent is (y/x) and whose quadrant is defined by the signs of x and y.

Step 5.12.4 Use the CFI subroutine target to determine the geolocation parameters:

ier = target (mjdp, pos, vel, acc, aocs, att, datt, &idir, dir, &iray, &freq, &ieres, res, status)

where the required parameters are from the table, with the following initializations:

mjdp = [mjdp](s)

pos, vel, acc from Module 11;

aocs[0:2]/(1:3) = {L1-AUX16-6, L1-AUX16-7, L1-AUX16-8}

att[0:2]/(1:3) = {L1-AUX16-9, L1-AUX16-10, L1-AUX16-11}

datt[0:2]/(1:3) = {0.0, 0.0, 0.0}

&idir: pointer to direction mode switch: dir = 1

dir: initialised as follows:

dir[0]/(1) = azimuth

dir[1]/(2) = elevation

dir[2]/(3) = 0.0 (altitude over earth)

dir[3]/(4) = 0.0 (dummy input)

dir[4]/(5) = 0.0 (azimuth-rate parameter)

dir[5]/(6) = 0.0 (elevation-rate parameter)

dir[6]/(7) = 0.0 (dummy input)

dir[7]/(8) = 0.0 (dummy input)

&iray = pointer to ray tracing model switch iray = 0

&freq = (dummy input)

&ieres = pointer to extended results vector switch ieres = 0

Check that the value of ier is zero; if it is not, an error has occurred.

The required outputs are the geodetic pixel co-ordinates taken from the results vector.

<view>_long(s_t, p_t) = res[3]/(4),

Convert <view>_long(s_t, p_t) to lie in the range -180 to 180 by the subtraction or addition of 360 if necessary.

<view>_lat(s_t, p_t) = res[5]/(6)

(Req 5.12-10)

5.13 Module Definition: Calculate Pixel x and y coordinates

5.13.1 Functional Description

The x and y co-ordinates may in principle be calculated for each scan pixel, but in practice they are only directly calculated for a series of tie points, comprising every INT_P (= 10th) pixel along every INT_S'th scan. The co-ordinates of the remaining points are then determined by linear interpolation between these tie opoints in a later module (Interpolate Pixel Positions).

The calculation makes use of a pre-calculated table giving the latitudes and longitudes of a series of points along the ground track, equally spaced in time at an interval corresponding to 1 granule. This table is derived with the aid of the orbit propagator in an earlier module.

The first step is to identify the value of i_g corresponding to the interval within which the normal section from the pixel P intersects the ground track at right angles. The angle between the ground track and the section PX varies continuously as the point X moves along the ground track, and therefore the correct interval is identified by iteration using the criterion that the angle of intersection passes through 90° within the interval.

Thus the intersection angle is calculated at the end-points of the interval. If the magnitude of angle PQ[i]Q_[i+1] is less than 90°, and that of PQ[i+1]Q[i+2] is greater than 90°, then index i defines the correct interval. (Figure 5.13-1.)

The iteration will work regardless of the initial value of i_g, but a well-chosen initial value will minimise the number of trials. For a given scan, an initial value for the first tie point pixel can be estimated from the known time of the sub-satellite point. When each scan is processed the initial value of i_g is set to int(scan time + t_est_nadir/ΔT) for the first scan on the forward view, and int(scan time + t_est_frwrd/ΔT) for the forward view. Thereafter, as the tie point pixels are calculated regularly along the scan, the value of i found for one pixel can be used as the starting point for the next.

To calculate the intersection angle at tabular point i, the azimuth of the ground track segment Q[i]Q[i+1] is first determined, then the azimuth of the normal section from Q[i] to point P. The intersection angle is given by the difference between the azimuths (Figure 5.13-2).

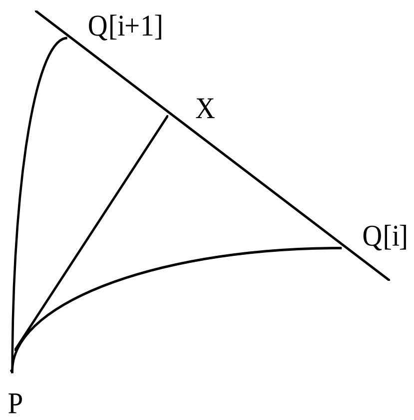


Figure 5.13-1

To determine the azimuth of the arc Q_1P, the triangle NPQ_1 is solved, where N is the north pole of the Earth. The general 'solve spherical triangle' routine (Section 5.10.3) is used, with parameters modified to account for the ellipsoidal geometry; the length of the arc PQ_1 and the other azimuth of the arc at P are also determined as a byproduct. The azimuth of the normal section is similarly determined by solution of the triangle NQ[i]Q[i+1].

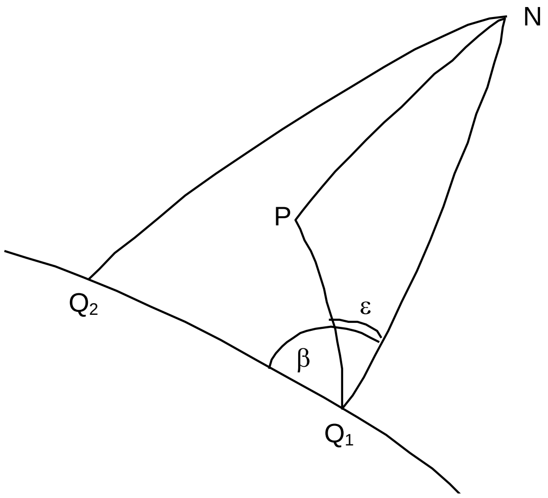


Figure 5.13-2

Thus starting from a specific tabular point i, the intersection angle at point Q[i] is calculated. If its magnitude is less than 90°, the intersection angle at point Q[i+1] is then calculated. If this is greater than 90° the interval between i and i+1 is the required interval; otherwise the process steps in the direction of increasing i until the correct interval is found. If the magnitude of the intersection angle at the initial point i is greater than 90°, the intersection angle at point Q[i-1] is calculated. If this is less than 90° the interval between i-1 and i is the required interval; otherwise the process steps in the direction of decreasing i, still until the correct interval is found.

5.13.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Size	Fields
L1-AUX18-5		MAX_NADIR_PIXELS	ss	none	4	1
L1-AUX18-7		MAX_FRWRD_PIXELS	ss	none	4	1
L1-AUX16-22		FIRST_NADIR_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-23		FIRST_FORWARD_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-24	t_est_nadir	Estimate of relative scan position, nadir scan	double	days	8	1
L1-AUX16-25	t_est_frwrd	Estimate of relative scan position, forward scan	double	days	8	1
L1-AUX16-26	INT_S	Along-track interpolation interval	ss	none	2	1
L1-AUX16-27	INT_P	Across-track interpolation interval	ss	none	2	1

Table 5-13-1: Input Data Table

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-60	nadir_lat(s, p)	nadir scan pixel latitude	float	deg.	4	575
L1B-INT-61	nadir_long(s, p)	nadir scan pixel longitude	float	deg.	4	575
L1B-INT-62	frwrd_lat(s, p)	forward scan pixel latitude	float	deg.	4	391
L1B-INT-63	frwrd_long(s, p)	forward scan pixel longitude	float	deg.	4	391
	asc_long	ascending node longitude	double	degrees	8	1
	prev_long	previous ascending node longitude	double	degrees	8	1

local	node_long	longitude current of ascending node	double	deg	8	
L1B-INT-26		grid_latitude	fl array	deg	4	
L1B-INT-27		grid_longittude	fl array	deg	4	
L1B-INT-50	ve	ssp ground speed	float	Km/s	4	
L1B-INT-55	track_y(k)	along track distance relative to asc. node	double	km.	8	NGRID+K
	s	scan index	sl	none	4	1
	p	pixel index	sl	none	4	1
	p_n	index to nadir tie point pixel	sl	none	4	1
	p_f	index to forward tie point pixel	sl	none	4	1
	i_g	index to geolocation table (i_g=0, ...,NGRID+K-1)	sl	none	4	1
L1B-INT-51	track_lat(i_g)	latitude	double	degrees	8	NGRID+K
L1B-INT-52	track_long(i_g)	longitude of ground track point	double	degrees	8	NGRID+K
	psi_1	co-latitude at point 1	double	radians	8	1
	psi_2	co-latitude at point 2	double	radians	8	1
	alpha_12	longitude differencce	double	radians	8	1
	psi_p	co_latitude of pixel	double	radians	8	1
	alpha_p	longitude difference pixel to point 1	double	radians	8	1
	beta	track azimuth	double	radians	8	1
	gamma	auxiliary azimuth (byproduct not used)	double	radians	8	1
	delta	auxiliary azimuth (byproduct not used)	double	radians	8	1
	epsilon	azimuth of pixel from point 1	double	radians	8	1
	a	arc length	double	radians	8	1
	a'	arc length	double	radians	8	1
	sigma_12	side Q1Q2	double	radians	8	1
	sigma2p	side q2p	double	radians	8	1
	sigmap1	side pQ1	double	radians	8	1
	px	angular distance, pixel to ground track	double	radians	8	1
	q2x	along-track distence	double	radians	8	1
	xi	across-track distance	double	km	8	1
	eta	along-track distance	double	km	8	1
L1B-INT-64	nadir_x(s, p)	nadir scan x coordinate	float	km	4	575
L1B-INT-65	nadir_y(s, p)	nadir scan y coordinate	float	km	4	575
L1B-INT-66	frwrd_x(s, p)	forward scan x coordinate	float	km	4	391
L1B-INT-67	frwrd_y(s, p)	forward scan y coordinate	float	km	4	391
L1B-INT-56	T0	Time of first product granule	double	days	8	1
L1B-INT-400		source_packet_ut_time(s)	double	days	8	

Table 5-13-2: Internal Data Table - Calculate Pixel x and y co-ordinates.

5.13.3 Algorithm Definition

The following steps are carried out for the same set of tie points as defined in Req. 5.12-1 on every *INT_S*'th scan s_t. The pixel latitudes and longitudes have been calculated for specific tie points on the scan: on the nadir scan these points are

$$p_t^n = 0, 10, 20, 30, ..., MAX\_NADIR\_PIXELS - 1$$

and for the forward scan

$$p_t^f = 0, 10, 20, 30, ..., MAX\_FRWRD\_PIXELS - 1$$

in the anticipated case that $INT\_P = 10$.

(Req 5.13-1)

Step 5.13.1 Co-ordinates of pixel

The geodetic latitude and longitude of the pixel are

$$\varphi_p = (\pi/180.0) \cdot nadir\_latitude(s_t, p_t^n)$$

$$\lambda_p = (\pi/180.0) \cdot nadir\_longitude(s_t, p_t^n)$$

for the nadir pixels, and

$$\varphi_p = (\pi/180.0) \cdot frwrd\_latitude(s_t, p_t^f)$$

$$\lambda_p = (\pi/180.0) \cdot frwrd\_longitude(s_t, p_t^f)$$

for the forward pixels. The longitudes should be reduced to the range (-180.0, 180.0) if necessary.

(Req 5.13-2)

Step 5.13.2 Identification of the nearest tabular points

Set interval_not_found = TRUE

Set iteration_number = 1

If first entry of scan initialise ig:

ig = K + (source_packet_ut_time(s_t) + t_est_<view> - T0)/ΔT

While (interval not found)

(Req 5.13-3)

Step 5.13.3 Calculate azimuth of Q1Q2 (beta)

Calculate

$$\varphi_1 = (\pi / 180.0) \cdot track\_lat(i_g)$$

$$\varphi_2 = (\pi / 180.0) \cdot track\_lat(i_g + 1)$$

(Req 5.13-4)

Calculate the radii of curvature in prime vertical at the points as follows:

$$\nu_1 = \mathrm{EQUATORIAL\_RADIUS} / \left(1 - e^2 \sin^2(\varphi_1)\right)^{1/2}$$

$$\nu_2 = \mathrm{EQUATORIAL\_RADIUS} / \left(1 - e^2 \sin^2(\varphi_2)\right)^{1/2}$$

$$S_\psi = (1 - e^2) \sin\varphi_2 + (\nu_1/\nu_2) e^2 \sin\varphi_1$$

$$C_\psi = \cos\varphi_2$$

$$\psi_2 = \mathrm{atan2}\left(C_\psi, S_\psi\right)$$

$$\psi_1 = \tfrac{1}{2}\pi - \varphi_1$$

$$\alpha_{12} = \pi \cdot (track\_long(i_g) - track\_long(i_g + 1)) / 180.0.$$

(Because the orbit is retrograde, the point [i+1] is always to the west of point [i], and the angle α_{12} computed as above is therefore in general positive.)

(Req 5.13-5)

Solve the following spherical triangle by the method of Step 5.10.5.

Solve spherical triangle $\{\alpha_{12}, \psi_1, \psi_2; \gamma, \beta, a\}$

The angle β will be in the range $0 \le \beta < 180°$, and will represent the azimuth of the track at Q_1, measured anticlockwise from north.

(Req 5.13-6)

Step 5.13.4 Calculate azimuth of Q1P (epsilon)

Calculate the radius of curvature in prime vertical at the point P as follows:

$$\nu_p = \text{EQUATORIAL\_RADIUS}/\left(1 - e^2 \sin^2(\varphi_p)\right)^{1/2}$$

$$S_p = (1 - e^2)\sin\varphi_p + (\nu_1/\nu_p)e^2 \sin\varphi_1$$

$$C_p = \cos\varphi_p$$

$$\psi_p = \text{atan2}\left(C_p, S_p\right)$$

(Req 5.13-7)

In the triangle NQ_1P the following additional quantity is known:

$$\alpha_p = (\pi/180.0)\cdot track\_long(i_g) - \lambda_p$$

It may be convenient to modify alpha_p, by adding or subtracting 2π, to bring it into the range -180° ≤ \alpha_p < 180°. This case arises if the 180° meridian intersects the arc Q_1P.

Again the triangle may be solved for the unknown quantities, in particular for the angle ε at q1, by the method of Section 5.10.3 (Step 5.10.5):

Solve spherical triangle $\{\alpha_p, \psi_p, \psi_1; \varepsilon, \delta, a'\}$

(Req 5.13-8)

Step 5.13.5 Iterate

```
If abs (β- ε) < π/2 then
        direction = +1
else
        direction = -1
endif

If iteration_number = 1 then prev_dir = direction

If prev_dir = direction then
        ig = ig + direction
else
        interval_not_found = FALSE
```

iteration_number = iteration_number + 1
prev_dir = direction
end while

If direction = -1 then ig = ig + direction

The difference

$$\beta - \varepsilon$$

(reduced into the range -π to π if necessary) represents the required intersection angle PQ_1Q_2.

In order that the tabular point specified be the earlier of the two, if in the last step of the iteration the value of i_g was increased by 1, it is now decremented by one.

Once the correct interval has been found, the sign of difference β - ε, reduced into the range -π to π if necessary, at either end-point determines on which side of the ground track the pixel P lies, and therefore the sign of the x co-ordinate of P. If β - ε is positive, then the point P is to the right (north) of the ground-track and its x co-ordinate will be positive. If β - ε is negative, then the point P is to the left (south) of the ground-track and its x co-ordinate will be negative.

(Req 5.13-9)

Step 5.13.6 Calculation of the sides of the triangle

Once the correct interval has been identified, the co-ordinates are calculated by spherical trigonometry. Suppose that the point on the sphere whose co-ordinates are indexed by the final value of i_g from the iteration is Q_1 and that corresponding to i_g +1 is Q_2. The latitude and longitude of each point P, Q_1 and Q_2 are known, and therefore the length of each side of the triangle PQ_1Q_2 can be determined.

Update the co-ordinates of the ground track points:

$$\varphi_1 = (\pi / 180.0) \cdot track\_lat(i_g)$$

$$\lambda_1 = (\pi / 180.0) \cdot track\_long(i_g).$$

$$\varphi_2 = (\pi / 180.0) \cdot track\_lat(i_g + 1)$$

$$\lambda_{12} = (\pi / 180.0) \cdot track\_long(i_g + 1)$$

(Req 5.13-10)

Calculate the radii of curvature in prime vertical at the two points as follows:

$$\nu_1 = \text{EQUATORIAL\_RADIUS}/\left(1 - e^2 \sin^2(\varphi_1)\right)^{1/2}$$

$$\nu_2 = \text{EQUATORIAL\_RADIUS}/\left(1 - e^2 \sin^2(\varphi_2)\right)^{1/2}$$

(Req 5.13-11)

Calculate the length of the arc between points Q_1 and Q_2 as follows:

$$S_2 = (1 - e^2)\sin\varphi_2 + (\nu_1/\nu_2)e^2 \sin\varphi_1$$

$$C_2 = \cos\varphi_2$$

$$\psi_2 = \operatorname{atan2}(C_2, S_2)$$

$$\psi_1 = \tfrac{1}{2}\pi - \varphi_1$$

$$\alpha_{12} = \lambda_1 - \lambda_2.$$

(Req 5.13-12)

Solve the following spherical triangle by the method of section 4.8.3.6.

Solve spherical triangle $\{\alpha_{12}, \psi_1, \psi_2; \gamma, \beta, \sigma_{12}\}$

Alternatively: In general the angular length of the great circle between two points A and B can be found by application of the cosine rule to the triangle ANB, where N is the North pole of the Earth: If

$$\alpha = \lambda_a - \lambda_b$$

is the longitude difference between the points then if s_{ab} is the arc AB

$$\cos\sigma_{ab} = \cos\psi_a \cos\psi_b + \sin\psi_a \sin\psi_b \cos\alpha$$

where ψ_a and ψ_b are the co-latitudes of the points. The arc length can thus be determined.]

(Req 5.13-13)

Correct the arc length to spheroidal length. Calculate

$$g = \varepsilon \sin\varphi_1 \cos\varphi_1 \cos\beta$$

$$h = \varepsilon \cos^2\varphi_1 \cos^2\beta$$

(Req 5.13-14)

Calculate the line segment between the points:

$$s_{12} = \nu_1 \sigma_{12}\left[1 - \frac{\sigma_{12}^2}{6} h(1-h) + \frac{\sigma_{12}^3}{8} g(1-2h)\right]$$

Each of the remaining sides of the triangle PQ_1Q_2 are calculated in turn using this formula, with subscripts replaced appropriately from the 3 indices {1, 2, p}. The triangle is then fully determined and any of its angles can be found.

Calculate the side Q_2P, s_{2p}, using the above equations with subscripts (1, 2) replaced by (2, p).

Calculate the side PQ_1, s_{p1}, using the above equations with subscripts (1, 2) replaced by (p, 1).

(Req 5.13-15)

Step 5.13.6-1 The mean radius of curvature.

The remaining calculations will make use of Legendre's theorem. Compute a suitable mean radius of curvature.

$$\varphi_m = (\varphi_1 + \varphi_2 + \varphi_p)/3$$

$$R = EARTH\_MAJOR\_AXIS \cdot (1-e^2)^{1/2} / (1-e^2 \sin^2 \varphi_m)$$

(Req 5.13-16)

Calculate the area of the triangle, and the spherical excess *E*:

$$s = (s_{12} + s_{2p} + s_{p1})/2$$

$$A = \sqrt{s(s-s_{12})(s-s_{2p})(s-s_{p1})}$$

$$E = A/R^2$$

radians.

(Req 5.13-17)

Step 5.13.7 The angle at Q

The next step is to compute the angle at Q_2. (The angle at Q_1 could be used instead; the choice is arbitrary.) The angle of the plane triangle having sides of the same length is computed, and then corrected by the spherical excess.

If (s_{2p} = 0) then

(trap the special case of a triangle of zero area, in which the angle is indeterminate)

$Q_2 = 0.0$

else

$$s = (s_{12} + s_{2p} + s_{p1})/2$$

$$Q_2 = 2.0 \cdot \arctan\left\{\sqrt{\frac{(s-s_{12})(s-s_{2p})}{s(s-s_{p1})}}\right\} + E/3$$

(end if)

(Req 5.13-18)

Step 5.13.8 The x and y co-ordinates

Given the side Q_2P and the angle Q_2, the right-angled triangle PQ_2X is fully determined, and the arcs PX and Q_2X can be calculated. The arc PX is determined by an application of the sine rule.

sin PX = sin (s_{2p}/R) sin Q_2.

The side Q_2X is determined from the tangent formula

tan Q_2X = tan (s_{2p}/R) cos Q_2.

The arc lengths ξ, ζ are determined by multiplying the angular lengths PX and Q2X respectively by the mean radius of curvature determined above.

ξ = R (PX)

$\zeta = R(Q_2X)$

The sign of the x co-ordinate of p is determined by inspecting the sign of the angle β - ε derived at the last value of i_g above, reduced if necessary into the range

$$-\pi < (\beta - \varepsilon) < \pi.$$

Then the x and y co-ordinates of the pixel, expressed in km, are

$$x = \mathrm{sgn}(\beta - \varepsilon) \cdot \xi$$

$$y = track\_y(i_g + 1) - \zeta + 0.15(p/2000) \cdot v_e$$

(The final term is the correction for satellite motion within the scan.)

(Req 5.13-19)

Assign the x and y co-ordinates to the relevant arrays:

$$<view>\_x(s_t, p_t^v) = x$$

$$<view>\_y(s_t, p_t^v) = y$$

(Req 5.13-20)

where <*view*> = *nadir* | *frwrd*, and where the superscript *v* represents *n* or *f*, as appropriate.

5.14 Module Definition: Interpolate Pixel Positions

5.14.1 Functional Description

Geolocation has found the pixel co-ordinates of the tie point pixels. Now linear interpolation with respect to pixel number is used to define the co-ordinates of the intermediate pixels on the scan. The process is repeated for both forward and nadir view scans.

5.14.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Size	Fields
L1-AUX18-5		MAX_NADIR_PIXELS	ss	none	4	1
L1-AUX18-7		MAX_FRWRD_PIXELS	ss	none	4	1
L1-AUX16-22		FIRST_NADIR_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-23		FIRST_FORWARD_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-26	INT_S	Along-track interpolation interval	ss	none	2	1
L1-AUX16-27	INT_P	Across-track interpolation interval	ss	none	2	1

Table 5-14-1: Input Data Table - Interpolate Pixel Positions

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-60	nadir_lat(s, p)	nadir scan pixel latitude	float	deg	4	575
L1B-INT-61	nadir_long(s, p)	nadir scan pixel longitude	float	deg	4	575
L1B-INT-62	frwrd_lat(s, p)	forward scan pixel latitude	float	deg	4	391
L1B-INT-63	frwrd_long(s, p)	forward scan pixel longitude	float	deg	4	391
L1B-INT-64	nadir_x(s, p)	nadir scan x coordinate	float	km	4	575
L1B-INT-65	nadir_y(s, p)	nadir scan y coordinate	float	km	4	575

L1B-INT-66	frwrd_x(s, p)	forward scan x coordinate	float	km	4	391
L1B-INT-67	frwrd_y(s, p)	forward scan y coordinate	float	km	4	391
	back	backward propagation flag	sl	n/a	4	1
	s	scan index	sl	none	4	1
		index to nadir scan pixels	sl	none	4	1
		index to forward scan pixels	sl	none	4	1
		index to nadir scan tie point pixels	sl	none	4	1
		index to forward scan tie point pixels	sl	none	4	1
local	w	interpolation weight	float	n/a	4	1
	(s, p)	modified pixel longitude, nadir scan	float	deg	4	
	(s, p)	modified pixel longitude, forward scan	float	deg	4	

Table 5-14-2: Internal Data Table - Interpolate Pixel Positions

5.14.3 Algorithm Definition

Step 5.14.0

If the value of *INT_S* is greater than 1, interpolate the nadir view pixel co-ordinates at each tie point defined below to derive the latitude, longitude, pixel x co-ordinate and pixel y co-ordinate for the tie point on each intermediate scan s. Interpolation is linear with respect to scan number, and in the case of longitude must allow for the possibility that the 180 degree meridian may intersect the interpolation interval, as below. (Note: It may be found more convenient for implementation to interpolate with respect to pixel number on the tie scans before interpolation with respect to scan number *s*. In this case the process would be, first, to interpolate with respect to pixel number for each 'tie' scan *s* to find the values for every pixel *p*; then to perform linear interpolation with respect ot the scan number *s* for each pixel *p*. Since the interpolations are linear, the two processes are mathematically equivalent.)

(Req 5.14-0)

Step 5.14.1

The pixel co-ordinates have been calculated for at specific tie points on the scan, at intervals of *INT_P*, as defined in Req. 5.12-1: on the nadir scan these points are

$$p_t^n = 0, 10, 20, 30, ..., MAX\_NADIR\_PIXELS - 1$$

and for the forward scan

$$p_t^f = 0, 10, 20, 30, ..., MAX\_FRWRD\_PIXELS - 1$$

in the anticipated case that *INT_P* = 10. Intermediate values are now calculated by linear interpolation. The nadir and forward scans are treated separately.

(Req 5.14-1)

For the nadir scan, for each pixel p^n that is not a tie point:

$$0 \le p^n < MAX\_NADIR\_PIXELS$$

$$p^n \notin \{p_t^n\}$$

we derive

$$p_t^n = 10\,\mathrm{int}(p^n / 10)$$

$$p_{t+1}^n = \text{ the smaller of } p_t^n + INT\_P, \; MAX\_NADIR\_PIXELS \text{ - 1.}$$

and compute

$$w = (p^n - p_t^n)/(p_{t+1}^n - p_t^n)$$

(Req 5.14-2)

Then

$$x^n(s,p^n) = x^n(s,p_t^n) + w\left\{x^n(s,p_{t+1}^n) - x^n(s,p_t^n)\right\}$$

$$y^n(s,p^n) = y^n(s,p_t^n) + w\left\{y^n(s,p_{t+1}^n) - y^n(s,p_t^n)\right\}$$

$${}^n\varphi(s,p^n) = {}^n\varphi(s,p_t^n) + w\left\{{}^n\varphi(s,p_{t+1}^n) - {}^n\varphi(s,p_t^n)\right\}$$

The pixel longitude is interpolated similarly, but in this case it is possible that the 180• meridian will intersect the interval between a pair of tie points. To accommodate this case we must proceed as follows. If

$$\left|\lambda^n(s,p_{t+1}^n) - \lambda^n(s,p_t^n)\right| \le 180.0$$

then

$$\lambda^n(s,p^n) = \lambda^n(s,p_t^n) + w\left\{\lambda^n(s,p_{t+1}^n) - \lambda^n(s,p_t^n)\right\}$$

If

$$\left|\lambda^n(s,p_{t+1}^n) - \lambda^n(s,p_t^n)\right| > 180.0$$

then the meridian intersects the interpolation interval. Define modified longitudes as follows:

$$\tilde{\lambda}_t^n = \lambda^n(s,p_t^n) \text{ if } \lambda^n(s,p_t^n) \ge 0.0$$

$$\tilde{\lambda}_t^n = \lambda^n(s,p_t^n) + 360 \text{ if } \lambda^n(s,p_t^n) < 0.0$$

and similarly

$$\tilde{\lambda}_{t+1}^n = \lambda^n(s,p_{t+1}^n) \text{ if } \lambda^n(s,p_{t+1}^n) \ge 0.0$$

$$\tilde{\lambda}_{t+1}^n = \lambda^n(s,p_{t+1}^n) + 360 \text{ if } \lambda^n(s,p_{t+1}^n) < 0.0$$

Iinterpolate this modified longitude:

$$\tilde{\lambda}^n(s,p^n) = \tilde{\lambda}_t^n + w\left\{\tilde{\lambda}_{t+1}^n - \tilde{\lambda}_t^n\right\}$$

Finally if $\tilde{\lambda}^n > 180.0$ then

$$\lambda^n(s,p^n) = \tilde{\lambda}^n(s,p^n) - 360.0$$

otherwise

$$\lambda^n(s, p^n) = \tilde{\lambda}^n(s, p^n)$$

(Req 5.14-3)

Similarly for the forward view derive

$$p_t^f = 10\,\mathrm{int}(p^f / 10)$$

$$p_{t+1}^f = \text{ the smaller of } p_t^f + INT\_P\text{, } MAX\_FRWRD\_PIXELS - 1.$$

for each pixel p^f that is not a tie point, i.e:

$$0 \le p^f < MAX\_NADIR\_PIXELS$$

$$p^f \notin \{p_t^f\}$$

(Req 5.14-4)

Then repeat the process using the same equations but with the superscript *f*, to signify the forward view, replacing the superscript *n*.

(Req 5.14-5)

5.15 Module Definition: Calculate Solar Angles

5.15.1 Functional Description

A final component of the geolocation algorithm calculates various solar angles; these quantities are required for use by the 1.6 micron cloud clearing algorithms, to identify situations in which sun-glint might be present, and by cloud clearing and level 2 processing to distinguish day and night pixels. The following angles are calculated:

- The azimuth of the sun at the pixel;
- The elevation of the sun as seen from the pixel;
- The azimuth of the sub-satellite point measured at the pixel.
- The elevation of the satellite as seen from the pixel.

The derivation of the angles uses further calls to the CFI TARGET subroutine, with the extended results vector switch set to 1 (since the solar azimuth and elevation are not part of the basic set of outputs produced by the subroutine).

It is necessary to define, for the products, the tie or sample points at which the solar angles are to be recorded.

5.15.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-64	nadir_x(s, p)	nadir scan x coordinate	float	km	4	575
L1B-INT-65	nadir_y(s, p)	nadir scan y coordinate	float	km	4	575
L1B-INT-66	frwrd_x(s, p)	forward scan x coordinate	float	km	4	391
L1B-INT-67	frwrd_y(s, p)	forward scan y coordinate	float	km	4	391

L1B-INT-23	mjdp[0]	Scan time, processing format	double	days	8	
L1B-INT-24	mjdp[1]	delta UT1	double	micros.	8	
	pos[0]	x coordinate	double	m	8	
	pos[1]	y coordinate	double	m	8	
	pos[2]	z coordinate	double	m	8	
	vel[0]	x velocity	double	m/s	8	
	vel[1]	y velocity	double	m/s	8	
	vel[2]	z velocity	double	m/s	8	
	acc[0]	x acceleration	double	m/s^2	8	
	acc[1]	y acceleration	double	m/s^2	8	
	acc[2]	z acceleration	double	m/s^2	8	
	aocs[0]	AOCS parameter Cx	double	deg.	8	
	aocs[1]	AOCS parameter Cy	double	deg.	8	
	aocs[2]	AOCS parameter Cz	double	deg.	8	
	att[0]	AOCS pitch mispointing	double	deg.	8	
	att[1]	AOCS roll mispointing	double	deg.	8	
	att[2]	AOCS yaw mispointing	double	deg.	8	
	datt[0]	AOCS pitch mispointing rate	double	deg./s	8	
	datt[1]	AOCS roll mispointing rate	double	deg./s	8	
	datt[2]	AOCS yaw mispointing rate	double	deg./s	8	
	idir	direction mode switch	sl	none	4	
	dir[0]		double		8	
	dir[1]		double		8	
	dir[2]		double		8	
	dir[3]		double		8	
	dir[4]		double		8	
	dir[5]		double		8	
	dir[6]		double		8	
	dir[7]		double		8	
	iray	ray tracing model sw.	sl	none	4	
	freq	signal frequency	double	Hz	8	
	ieres	extended results vector switch	sl	none	4	
	tstatus	target status flag	double	none	8	
	res[74]	results vector	double		8	74
	status	status vector	sl		4	15
local	ier	local status word for target subroutine calls	sl	n/a	4	1
L1B-INT-120		nadir_band_edge_solar_elevation[11]	float	degrees	4	
L1B-INT-121		nadir_band_edge_satellite_elevation[11]	float	degrees	4	
L1B-INT-122		nadir_band_edge_solar_azimuth[11]	float	degrees	4	
L1B-INT-123		nadir_band_edge_satellite_azimuth[11]	float	degrees	4	
L1B-INT-140		frwrd_band_edge_solar_elevation[11]	float	degrees	4	
L1B-INT-141		frwrd_band_edge_satellite_elevation[11]	float	degrees	4	
L1B-INT-142		frwrd_band_edge_solar_azimuth[11]	float	degrees	4	
L1B-INT-143		frwrd_band_edge_satellite_azimuth[11]	float	degrees	4	
L1B-INT-124		nadir_band_centre_solar_elevation[10]	float	degrees	4	
L1B-INT-125		nadir_band_centre_satellite_elevation[10]	float	degrees	4	
L1B-INT-126		nadir_band_centre_solar_azimuth[10]	float	degrees	4	
L1B-INT-127		nadir_band_centre_satellite_azimuth[10]	float	degrees	4	
L1B-INT-144		frwrd_band_centre_solar_elevation[10]	float	degrees	4	
L1B-INT-145		frwrd_band_centre_satellite_elevation[10]	float	degrees	4	
L1B-INT-146		frwrd_band_centre_solar_azimuth[10]	float	degrees	4	
L1B-INT-147		frwrd_band_centre_satellite_azimuth[10]	float	degrees	4	
L1B-INT-380	Dphi_nadir	Latitude correction, nadir view	sl array	μdeg	4	23
L1B-INT-381	Dlam_nadir	Longitude correction, nadir view	sl array	μdeg	4	23
L1B-INT-382	H	Topographic height	ss array	m	2	23
L1B-INT-383	Dphi_frwrd	Latitude correction, forward view	sl array	μdeg	4	23
L1B-INT-384	Dlam_frwrd	Longitude correction, forward view	sl array	μdeg	4	23

Table 5-15-1: Internal Data Table - Calculate Solar Angles

5.15.3 Algorithm Definition

Step 5.15.1 Initialize Data Structures

For each image row i, the data arrays for regridded viewing angles must be initialized. This is necessary to ensure any missing data after regridding can be recognised:

<view>_band_edge_solar_elevation(i, k) = -999.0 (k = 0, 1, ...10)

<view>_band_edge_satellite_elevation(i, k) = -999.0 (k = 0, 1, ...10)

<view>_band_edge_solar_azimuth(i, k) = -999.0 (k = 0, 1, ...10)

<view>_band_edge_satellite_azimuth(i, k) = -999.0 (k = 0, 1, ...10)

<view>_band_centre_solar_elevation(i, k′) = -999.0 (k′ = 0, 1, ...9)

<view>_band_centre_satellite_elevation(i, k′) = -999.0 (k′ = 0, 1, ...9)

<view>_band_centre_solar_azimuth(i, k′) = -999.0 (k′ = 0, 1, ...9)

<view>_band_centre_satellite_azimuth(i, k′) = -999.0 (k′ = 0, 1, ...9)

for <view> = <nadir|frwrd>.

Also initialise the array elements [L1B-INT-380] to [L1B-INT-384] for use by module 16
Dphi_<view>(ig, jg) = -999999
Dlam_<view>(ig, jg) = -999999
and
H(ig, jg) = -29999,
for all ig and for jg = 0, 22.

(Req 5.15-1)

Step 5.15.2 Identify Band Edge Pixels

The image swath is imagined as divided into 10 bands, bounded by

{-250, -200, -150, -100, -50, 0, 50, 100, 150, 200, 250} km.

The band edge solar angles are calculated at the pixel positions nearest to these lines (but on the scan) and are placed in the relevant arrays.

For each view and instrument scan, and for each value of the across-track distance in turn, indexed by k = 0, 10, the pixel index p is found such that

x{p} < across_track_distance < x{p+1}

We then define p{band edge} as

p if abs (x{p} - across_track_distance) < abs (x{p+1} - across_track_distance)

otherwise

p{band edge} = p+1.

(Req 5.15-2)

Identify scan direction (azimuth and elevation) corresponding to pixel p. The algebra is that of Section 5.12.3, steps 5.12.1 to 5.12.3 inclusive. Note that as a function of pixel number these quantities need be calculated only once.

(Req 5.15-3)

Step 5.15.3 Derive using CFI

Use the CFI subroutine target to determine the geolocation parameters for each tie scan s_t defined as in Section 5.12.3 (at intervals of *INT_S* in scan number):

ier = target (mjdp, pos, vel, acc, aocs, att, datt, &idir, dir, &iray, &freq, &ieres, res, status)

where the required parameters are from the table, with the following initializations appropriate to the time of the scan s:

mjdp = [mjdp](s)

pos

vel

acc

aocs

att

datt

&idir: pointer to direction mode switch: dir = 1

dir: initialised as follows:

dir[0]/(1) = azimuth

dir[1]/(2) = elevation

dir[2]/(3) = 0.0 (altitude over earth)

dir[3]/(4) = 0.0 (dummy input)

dir[5]/(5) = 0.0 (azimuth-rate parameter)

dir[5]/(6) = 0.0 (elevation-rate parameter)

dir[6]/(7) = 0.0 (dummy input)

dir[7]/(8) = 0.0 (dummy input)

&iray = pointer to ray tracing model switch iray = 0

&freq = (dummy input)

&ieres = pointer to extended results vector switch ieres = 1

Check that the value of ier is zero; if it is not, an error has occurred.

The required outputs for the tie scan are given by the geocentric pixel co-ordinates taken from the results vector res.

```
      <view>_band_edge_satellite_azimuth(i, k) = res[8]/(9),
converted to lie in the range -180 to 180 by the subtraction (or
addition) of 360 if required.
      <view>_band_edge_satellite_elevation(i, k) = res[9](10)
      <view>_band_edge_solar_azimuth(i, k) = res[66]/(67)
converted to lie in the range -180 to 180 by the subtraction (or
addition) of 360 if required.
      <view>_band_edge_solar_elevation(i, k) = res[67]/(68)
                                                        (Req 5.15-4)
```

where *i* is the index to the regridded image row derived from the pixel y co-ordinate *<view>_y(s, p)* by the algebra of Req 5.18-5 from Step 5.18.2.

Corresponding outputs for scans intermediate between s_t and $s_t + INT\_S$ are calculated by linear interpolation with respect to scan number *s* for

$s = s_t + 1, ..., s_t + INT\_S - 1.$

In the case of the solar and satellite azimuths, care must be taken to check whether or not the value passes through -180 / +180 degrees in the interpolation interval. If it does, 360 degrees must be added to the negative value before interpolation to convert the two azimuths to the same basis, and the result corrected to lie in the range -180 to +180 degrees after interpolation, as is done in the case of longitude at Req 5.14-3.

Step 5.15.4 Identify Band Centre Pixels

The image swath is imagined as divided into 10 bands, as above. The centres of the bands are therefore at the positions, to be regarded as indexed by k′ = 0, 9,

{-225, -175, -125, -75, -25, 25, 75, 125, 175, 225} km.

The band centre solar angles are calculated at the pixel positions nearest to these lines (but on the scan) and are placed in the band centre arrays for the y index corresponging to the nearest y coordinate. Thus the angle is regridded on a nearest pixel basis. For each k′ = 0, 9 identify the pixel index p corresponding to next x co-ordinate in turn from the above set and compute the azimuth and elevation as in step 5.15.2.

```
                                                        (Req 5.15-5)
```

Step 5.15.5 Derive using CFI

Use the CFI subroutine target to determine the geolocation parameters:

ier = target (mjdp, pos, vel, acc, aocs, att, datt, &idir, dir, &iray, &freq, &ieres, res, status)

where the required parameters are from the table, with the following initializations:

mjdp

pos

vel

acc

aocs

att

datt

&idir: pointer to direction mode switch: dir = 1

dir: initialised as follows:

dir[0]/(1) = azimuth

dir[1]/(2) = elevation

dir[2]/(3) = 0.0 (altitude over earth)

dir[3]/(4) = 0.0 (dummy input)

dir[5]/(5) = 0.0 (azimuth-rate parameter)

dir[5]/(6) = 0.0 (elevation-rate parameter)

dir[6]/(7) = 0.0 (dummy input)

dir[7]/(8) = 0.0 (dummy input)

&iray = pointer to ray tracing model switch iray = 0

&freq = (dummy input)

&ieres = pointer to extended results vector switch ieres = 1

Check that the value of ier is zero; if it is not, an error has occurred.

The required outputs are the geocentric pixel co-ordinates taken from the results vector res.

```
      <view>_band_centre_satellite_azimuth(i, k′) = res[8]/(9)
converted to lie in the range -180 to 180 by the subtraction (or
addition) of 360 if required.
      <view>_band_centre_satellite_elevation(i, k′) = res[9](10)
      <view>_band_centre_solar_azimuth(i, k′) = res[66]/(67)
converted to lie in the range -180 to 180 by the subtraction (or
addition) of 360 if required.
      <view>_band_centre_solar_elevation(i, k′) = res[67]/(68)
                                                        (Req 5.15-6)
```

as in Step 5.15.3.

5.16 Module Definition: Topographic Correction

5.16.1 Functional Description

This module computes the topographic corrections to the pixel latitude and longitude at a series of tie points on each image scan, based on a terrain model.

The derivation of the corrections uses the satellite azimuth and satellite elevation angles derived in Module 15 (Section 5.15), in a geometrical model. The method avoids additional calls to 'target' in Module 16, and ensures that if the topographic height H of a pixel is zero, the corresponding corrections are also zero.

5.16.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX16-12	a_e	EARTH_MAJOR_AXIS	double	km	8	1
L1-AUX16-18	e	eccentricity of ellipsoid	double	none	8	1
L1-AUX16-19	ε	Geodetic parameter (square of 'second eccentricity')	double	none	8	1
L1-AUX24-1	height(i, j)	Surface height	float	mm	4	2160

Table 5-16-1: Input Data Table - Topographic Correction

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-64	nadir_x(s, p)	nadir scan x coordinate	float	km	4	575
L1B-INT-65	nadir_y(s, p)	nadir scan y coordinate	float	km	4	575
L1B-INT-66	frwrd_x(s, p)	forward scan x coordinate	float	km	4	391
L1B-INT-67	frwrd_y(s, p)	forward scan y coordinate	float	km	4	391
L1B-INT-060	nadir_lat(s, p)	nadir scan pixel latitude	float	degrees	4	
L1B-INT-061	nadir_long(s, p)	nadir scan pixel longitude	float	degrees	4	
L1B-INT-062	frwrd_lat(s, p)	forward scan pixel latitude	float	degrees	4	
L1B-INT-063	frwrd_long(s, p)	forward scan pixel longitude	float	degrees	4	
L1B-INT-121		nadir_band_edge_satellite_elevation[11]	float	degrees	4	k = 0, 10
L1B-INT-123		nadir_band_edge_satellite_azimuth[11]	float	degrees	4	k = 0, 10
L1B-INT-141		frwrd_band_edge_satellite_elevation[11]	float	degrees	4	k = 0, 10
L1B-INT-143		frwrd_band_edge_satellite_azimuth[11]	float	degrees	4	k = 0, 10
L1B-INT-125		nadir_band_centre_satellite_elevation[10]	float	degrees	4	k′ = 0, 9
L1B-INT-127		nadir_band_centre_satellite_azimuth[10]	float	degrees	4	k′ = 0, 9
L1B-INT-145		frwrd_band_centre_satellite_elevation[10]	float	degrees	4	k′ = 0, 9
L1B-INT-147		frwrd_band_centre_satellite_azimuth[10]	float	degrees	4	k′ = 0, 9
L1B-INT-380	Dphi_nadir	Latitude correction, nadir view	sl array	μdeg	4	23
L1B-INT-381	Dlam_nadir	Longitude correction, nadir view	sl array	μdeg	4	23
L1B-INT-382	H	Topographic height	ss array	m	2	23
L1B-INT-383	Dphi_frwrd	Latitude correction, forward view	sl array	μdeg	4	23
L1B-INT-384	Dlam_frwrd	Longitude correction, forward view	sl array	μdeg	4	23
constant	π	the mathematical constant (= 3.14159265359)	double	none	8	1
local	C	Geodetic scale factor (scaled semi-major axis)	double	none	8	1
local	N	Radius of curvature in prime vertical	double	none	8	1
local	R	Radius of curvature in meridian plane	double	none	8	1
local	φ	Pixel latitude	float	radians	4	1
local	ig	Along track tie point index	sl	none	4	1
local	jg	Across track tie point index, jg = 0, 22	sl	none	4	1
local	k	Index to band edge viewing angles, k = 0, 10	sl	none	4	1
local	k′	Index to band centre viewing angles, k′ = 0, 9	sl	none	4	1
local		satellite_azimuth	float	radians	4	1
local		satellite_elevation	float	radians	4	1
local	H_pixel	Local topographic height	float	m	4	1
local	dX	Linear correction in longitude direction	float	m	4	1
local	dY	Linear correction in latitude direction	float	m	4	1

Table 5-16-2: Internal Data Table - Topographic Correction

5.16.3 Algorithm Definition

Step 5.16.1 Initialize Data Structures

At initialisation, input the table of topographic heights height (i, j). The arrangement of the table is defined in AD11.

(Req 5.16-1)

Step 5.16.2 Identify Tie point Pixels

The nominal tie points are at across-track distances x = {-275, -250, -225, ... 275} km, each corresponding to an across-track index jg = (0, ... 22). However, if jg = 0 or jg = 22, no viewing angles will be available from Module 15, and so these cases can be omitted.

For each scan, proceed as follows. For each of the across-track distances jg = (1, ... 21) for which a correction is required, identify the pixel index p such that

x(p) < across_track distance < x(p + 1).

If there is no such pixel (i.e. if across-track distance < x(0) or across-track distance > x(max_pixel_index)) then proceed to the next case.

Otherwise if x(p + 1) is closer to the across-track distance that x(p) replace p by p + 1.

(Req 5.16-2)

Step 5.16.3

Identify the pixel latitude and longitude corresponding to pixel p, <view>_lat(s, p) and <view>_long(s, p).

(Req 5.16-3)

Step 5.16.4

Find the local altitude (over land) or bathymetry (over sea) from the topographic model. That is to say, find the indices i, j of the sample point nearest to the pixel. This is the one for which (lat(i, j) - <view>_lat(s, p)) and (long(i, j) - <view>_long(s, p)) are a minimum over i and j respectively, where lat(i, j) and long(i, j) are the latitude and longitude respectively of the tabular point indexed by (i, j). The implementation of this depends on the arrangement of the table.

Identify the topographic height/bathymetry at the pixel, height(i, j). The arrangement of the table is defined in AD11.

```
H_pixel = height(i, j)/1000.
```
(Req 5.16-4)

(Note the units conversion from mm to metres.)

Step 5.16.5

Requirement deleted.

(Req 5.16-5)

Step 5.16.6 Geolocate

Geolocate to the data structure row corresponding to pixel y. Calculate the indices ig and i′ corresponding to the pixel y co-ordinate <view>_y(s, p) as in Step 5.18.2. If i′ is not zero (so the pixel does not regrid to a tie row) omit step 5.16.7 and proceed to the next pixel. Otherwise if i′ = 0 set

```
H(ig, jg) =H_pixel
```
(Req 5.16-6)

If *height(i, j)* < 0 (note this includes the case that the pixel is over sea), set the latitude and longitude corrections to zero and omit step 5.16.7:

```
Dphi_<view>(ig, jg) = 0
Dlam_<view>(ig, jg) = 0                              (Req 5.16-7)
```

Step 5.16.7 Derive topographic corrections from viewing angles

This method makes use of to the satellite viewing angles for the appropriate view and tie point, which were computed in Module 15 (Section 5.15).

Calculate

$$C = 1000.0 \cdot EARTH\_MAJOR\_AXIS / \left(1 - e^2\right)^{1/2}.$$

(Req 5.16-8)

(Note that this is a geodetic constant and need be calculated only once. The factor of 1000 converts the units from km to metres.)

Extract pixel latitude and convert to radians. The pixel latitude in radians is

```
φ = <view>_lat(s, p) * (π/180.0)
                                                     (Req 5.16-9)
```

Calculate the two orthogonal radii of curvature of the Earth at the pixel latitude:

$$N = C / \left(1 + \varepsilon \cos^2 \varphi\right)^{1/2}$$

$$R = N / \left(1 + \varepsilon \cos^2 \varphi\right)$$

(Req 5.16-10)

(Note that in the above equations ε is the constant [L1-AUX16-19]. The quantities *N* and *R* are the radii of curvature in prime vertical and in the meridian respectively. These equations for the radii of curvature *N* and *R* are equivalent to those used in earlier modules (compare for example Req 5.13-11, Req 5.13-16), but have been re-formulated so that only the cosine of the latitude is required by this Module, to permit optimisation.)

Extract the satellite elevation and azimuth and convert to radians. The values used depend on whether *jg* is odd or even; if *jg* is odd the band edge values are used; and if *jg* is even the band centre values are used.

```
If jg is odd then
      k = (jg - 1)/2
      satellite_elevation =
        <view>_band_edge_satellite_elevation(ig, k) * (π/180.0)
      satellite_azimuth =
        <view>_band_edge_satellite_azimuth(ig, k) * (π/180.0)
                                                    (Req 5.16-11)
else if jg is even
      k′ = (jg - 2)/2
      satellite_elevation =
        <view>_band_centre_satellite_elevation(ig, k′) * (π/180.0)
      satellite_azimuth =
```

```
        <view>_band_centre_satellite_azimuth(ig, k′) * (π/180.0)
                                                        (Req 5.16-12)
end if
```

Calculate the linear corrections dX and dY:

```
dY = H_pixel * cot(satellite_elevation) * cos(satellite_azimuth)
                                                        (Req 5.16-13)
dX = H_pixel * cot(satellite_elevation) * sin(satellite_azimuth)
                                                        (Req 5.16-14)
```

The corrections in latitude and longitude, expressed in microdegrees, are respectively

```
        Dphi_<view>(ig, jg) = 1000000. * dY * (180/π)/R
        Dlam_<view>(ig, jg) = 1000000. * (dX / cos(φ)) * (180/π)/N
                                                        (Req 5.16-15)
```

5.17 Module Definition: Signal Calibration

5.17.1 Functional Description

For each scan in the scan group, the signal channels are calibrated by the application of the appropriate linear relationship between the unpacked signal counts and the calibrated radiance, in the case of the infra-red channels, or the reflectance in the case of the visible channels. In the case of the infra-red channels the calibrated radiance is then converted to a brightness temperature by means of the rasdiance to brightness temperature look-up table.

Each channel within each view is calibrated independently, and separate calibration parameters (slope and offset) are used for odd and even numbered pixels within the scan.

If valid calibration parameters for a given scan, parity and channel are not available, or if the derived channel values are out of range, the pixel is flagged by an exceptional value. If the input channel count is negative, this implies that the channel is invalid; it has already been flagged by an exceptional value, and is output from the module unchanged.

5.17.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX6-2	B(ir12,k)	temperature_lut[12 micron]	do	K	8	
L1-AUX6-3	B(ir11, k)	temperature_lut[11 micron]	do	K	8	
L1-AUX6-4	B(ir37, k)	temperature_lut[3.7 micron]	do	K	8	
L1-AUX27-8	N(ir12)	Number of entries in 12 micron Radiance to Brightness Temperature LUT	ss	none	2	1
L1-AUX27-9	increment(ir12)	Increment in 12 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-10	first_value(ir12)	First value in 12 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-11	last_value(ir12)	Last value in 12 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-13	N(ir11)	Number of entries in 11 micron Radiance to Brightness Temperature LUT	ss	none	2	1
L1-AUX27-14	increment(ir11)	Increment in 11 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-15	first_value(ir11)	First value in 11 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1

L1-AUX27-16	last_value(ir11)	Last value in 11 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-17	N(ir37)	Number of entries in 3.7 micron Radiance to Brightness Temperature LUT	ss	none	2	1
L1-AUX27-18	increment(ir37)	Increment in 3.7 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-19	first_value(ir37)	First value in 3.7 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-20	last_value(ir37)	Last value in 3.7 micron Radiance to Brightness Temperature LUT	do	Wcm-2 sr-1	8	1
L1-AUX27-24	N(v16)	Number of entries in 1.6 micron Channel Non-linearity Correction LUT	ss	none	2	1
L1-AUX27-25	increment(v16)	Increment in 1.6 micron Channel Non-linearity Correction LUT	do	counts	8	1
L1-AUX27-26	first_value(v16)	First value in 1.6 micron nonlinear correction LUT	do	counts	8	1
L1-AUX27-27	last_value(v16)	Last value in 1.6 micron nonlinear correction LUT	do	counts	8	1
L1-AUX28-1	C0(k)	Uncorrected count entry	do	counts	8	N(v16)
L1-AUX28-2	C1(k)	Corrected count table entry	do	counts	8	N(v16)
L1-AUX18-25	(constant)	CALIBRATION_UNAVAILABLE_FOR_PIXEL	ss	n/a	2	1
L1-AUX18-26	(constant)	PIXEL_RADIANCE_OUTSIDE_CALIBRATION	ss	n/a	2	1

Table 5-17-1: Input Data Table - Signal Calibration

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-80	C{ch, n; s, p}	channel[.] pixel counts nadir	ss	counts	2	
L1B-INT-81	C{ch, f; s, p}	channel[.] pixel counts frwrd	ss	counts	2	
L1B-INT-9	scp_gain(ch, s)	channel scp gain (only required for v16 at present)	float	n/a	4	
L1B-INT-10	slope{ch; s, pty}	channel[.] slope	float		4	
L1B-INT-11	intercept{ch; s, pty]	channel[.]intercept	float		4	
L1B-INT-6	cal_inv(i)	calibration invalid [.channel]	flag	n/a	4	
	p	pixel number	sl	none	4	
	s	scan number	sl	none	4	
	ch	channel index (ch = 1, 2, 3, 4, 5, 6, 7)	sl	none	4	
local	pty, parity	parity index	sl	none	4	
local	L	radiance in current channel	do	Wcm-2 sr-1	8	1
local	scale	radiance scaled in units of tabular increment	do	n/a	8	1
local	sample_count	loop counter / index for input of look-up table.	sl	none	4	1
local	k	index into temperature arrays	sl	n/a	4	1
L1B-INT-87	T{ir12, n; s, p}	channel[ir12] calibrated pixels nadir	ss	0.01K	2	
L1B-INT-88	T{ir12, f; s, p}	channel[ir12] calibrated pixels frwrd	ss	0.01K	2	
L1B-INT-87	T{ir11, n; s, p}	channel[ir11] calibrated pixels nadir	ss	0.01K	2	
L1B-INT-88	T{ir11, f; s, p}	channel[ir11] calibrated pixels frwrd	ss	0.01K	2	
L1B-INT-87	T{ir37, n; s, p}	channel[ir37] calibrated pixels nadir	ss	0.01K	2	
L1B-INT-88	T{ir37, f; s, p}	channel[ir37] calibrated pixels frwrd	ss	0.01K	2	
L1B-INT-87	R{v16, n; s, p}	channel[v16] calibrated pixels nadir	ss	0.01%	2	
L1B-INT-88	R{v16, f; s, p}	channel[v16] calibrated pixels frwrd	ss	0.01%	2	
L1B-INT-87	R{v870, n; s, p}	channel[v870] calibrated pixels nadir	ss	0.01%	2	
L1B-INT-88	R{v870, f; s, p}	channel[v870] calibrated pixels frwrd	ss	0.01%	2	
L1B-INT-87	R{v670, n; s, p}	channel[v670] calibrated pixels nadir	ss	0.01%	2	
L1B-INT-88	R{v670, f; s, p}	channel[v670] calibrated pixels frwrd	ss	0.01%	2	
L1B-INT-87	R{v555, n; s, p}	channel[v555] calibrated pixels nadir	ss	0.01%	2	
L1B-INT-88	R{v555, f; s, p}	channel[v555] calibrated pixels frwrd	ss	0.01%	2	

Table 5-17-2: Internal Data Table - Signal Calibration

5.17.3 Algorithm Definition

Step 5.17.1 Input Radiance to BT Look-up Table

This is done only once. First read in the limit and increment values [L1-AUX27-8] to [L1-AUX27-20]. Then for each of the three infra-red channels in parallel the conversion table values are read in from the file of LUT values.

```
for sample_count = 0, MAX(N(ir12), N(ir11), N(ir37)) - 1
  read B(ir12, sample_count), B(ir11, sample_count),
    B(ir37, sample_count)
end for
```

(Req 5.17-1)

Also read in the correction table for the 1.6 micron channel.

```
for sample_count = 0, N(v16) - 1
  read C0(sample_count), C1(sample_count)
end for
```

(Req 5.17-1.1)

Step 5.17.2 Calibrate infra red channels

The uncalibrated pixel count and calibrated pixel arrays are indexed by p_n and p_f such that

$p_n = p$ - FIRST_NADIR_PIXEL_NUMBER

$p_f = p$ - FIRST_FORWARD_PIXEL_NUMBER

where p is the absolute pixel number and

$0 \leq p_n <$ MAX_NADIR_PIXELS

$0 \leq p_f <$ MAX_FORWARD_PIXELS

In the following we will use the notation p_v, where the subscript v represents n or f, for the fnadir and forward views respectively.

For each scan of each view and for each infra-red channel ch = ir12, ir11, ir37 all the pixels of the scan are calibrated as follows.

If C{ch, view; s, p_v} > 0 and
(slope[channel; scan, parity] > 100000 or
intercept[channel; scan, parity] > 100000)
then the pixel is valid but there is no valid calibration for pixels of the specified parity on the scan. Set the calibrated pixel value to the corresponding exceptional value:

T{ch, view; s, p_v} = CALIBRATION_UNAVAILABLE_FOR_PIXEL

where

p = parity (mod 2)

and p is the absolute pixel number as above.

(Req 5.17-2)

If C{ch, view; s, p_v} ≤ 0 then the channel count value is invalid; pass unchanged to output

T{ch, view; s, p_v} = C{ch, view; s, p_v}

(Req 5.17-3)

Otherwise the pixel count is converted to a radiance

$L\{ch, view; s, p_v\}= C\{ch, view; s, p_v\}*slope[ch; s, parity] +intercept[ch; s, parity]$

where parity = p (mod 2)

(Req 5.17-4)

Finally, the radiance L is converted to BT by linear interpolation in the LUT.

If the radiance L lies within the range of the appropriate table, i.e. if

first_value(ch) ≤ L < last_value(ch),

find the index k such that radiance{k} ≤ L < radiance{k+1}, where radiance{k} symbolises the radiance to which the kth tabular brightness temperature corresponds, which is first_value(ch) + k * increment(ch):

scale = (L - first_value(ch))/interval(ch)
k = integer part of (scale)

Then

$T\{ch, view; s, p_v\}$ = nearest integer to
(100 * (B{ch, k}+ {B{ch, k+1} - B{ch, k}} * (scale - k))
(where the factor of 100 converts the units to 0.01 K).

(Req 5.17-5)

If the radiance L lies outside the range of the table, the calibrated pixel value is set to the corresponding exceptional value:

$T\{ch, view; scan, p_v\}$ = PIXEL_RADIANCE_OUTSIDE_CALIBRATION

(Req 5.17-6)

Step 5.17.3 Calibrate visible channels

For each scan and for each visible channel ch = v16, v870, v670, v555 all the pixels of the scan are calibrated as follows.

If $C\{ch, view; s, p_v\} > 0$ and
(slope[channel; scan, parity] > 100000 or
intercept[channel; scan, parity] > 100000
then the pixel is valid but there is no valid calibration for pixels of the specified parity on the scan. Set the calibrated pixel value to the corresponding exceptional value:

$R\{ch, view; s, p_v\}$ = CALIBRATION_UNAVAILABLE_FOR_PIXEL

where

p = parity (mod 2)

(Req 5.17-7)

If $C\{ch, view; s, p_v\} \leq 0$ then the channel count value is invalid; pass unchanged to output

$R\{ch, view; s, p_v\} = C\{ch, view; s, p\}$

(Req 5.17-8)

Otherwise the pixel count is converted to a reflectance. If ch = v870, v670, v555 then

R{ch, view; s, p_v} = nearest integer to (10000. * (C{ch, view; s, p_v}
+ intercept[ch, s, parity]) *slope[ch; s, parity])

where parity = p (mod 2) and where the factor of 10000. converts the units to 0.01% from fractional reflectance.

(Req 5.17-9)

In the case of the 1.6 micron channel the procedure is modified to account for a tabular non-linearity correction.

The value is in fact

10000 * slope * Fn{(C(v16, view; s, p_v) + intercept(v16, s, parity))/scp_gain(v16)}.

Therefore

R{v16, view; s, p_v} = nearest integer to (10000. * C′ *slope[ch; s, parity]),

where C′ is calculated as follows.

If the value of

C2 = (C(v16, view; s, p_v) + intercept(v16, s, parity))/scp_gain(v16)

lies within the range of the correction table, i.e. if

first_value(v16) ≤ C2 < last_value(v16),

find the index k such that C0{k} ≤ C2 < C0{k+1}, where C0{k} = [L1-AUX28-01](k) is the kth tabular value C0, which is first_value(v16) + k * increment(v16):

scale = (C2 - first_value(v16))/interval(v16)
k = integer part of (scale)

Then

C′ = (C1{k}+ {C1{k+1} - C1{k}} * (scale - k))

(Req 5.17-10)

If C2 lies outside the range of the table, the calibrated pixel value is set to the corresponding exceptional value:

R{ch, view; scan, p_v} = PIXEL_RADIANCE_OUTSIDE_CALIBRATION

(Req 5.17-11)

5.18 Module Definition: Regrid Pixels

5.18.1 Functional Description

This routine regrids the data from the source packets onto the rectangular 1 km along track (y) / across track (x) grid. It uses the x and y coordinates previously calculated in Geolocate_source_packets.

5.18.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX18-27		CRC_ERR_CODE_DETECTED_ERR	ss	N/A	2	1

L1-AUX18-28		BUFFERS_FULL_CHECK_ERR	ss	N/A	2	1
L1-AUX18-33		PIX_SCAN_JITTER_ERR	ss	N/A	2	1
L1-AUX18-21		RAW_PKT_FAILS_BASIC_VALIDATION_ERR	ss	N/A	2	1
L1-AUX16-17	K	Displacement of grid point table start.	sl	none	4	1
L1-AUX16-20		MAX_NADIR_PIXELS	sl	none	4	1
L1-AUX16-21		MAX_FRWRD_PIXELS	sl	none	4	1
L1-AUX16-22		FIRST_NADIR_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-23		FIRST_FORWARD_PIXEL_NUMBER	sl	none	4	1
L1-AUX16-15	NGRANULE	Number of image rows per granule	sl	none	4	1
L1-AUX16-28	eps_x	Regridding perturbation in x direction	ss	m	2	1
L1-AUX16-29	eps_y	Regridding perturbation in y direction	ss	m	2	1

Table 5-18-1: Input Data Table - Regrid Pixels

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-1	packet_error(s)		ss	N/A	2	1
L1B-INT-4	conv_aux_data	converted_auxiliary_data[i]	fl array	N/A	4	aux_tot
L1B-INT-5	auxdata(i)	unpacked_auxiliary_data[i]	ss array	N/A	2	aux_tot
L1B-INT-87	T(ch, n; s, p)	calibrated.pixels.nadir[MAX_NADIR_PIXELS], infra-red channels	ss	K/100	2	575
L1B-INT-88	T(ch, n; s, p)	calibrated.pixels.forward[MAX_FORWARD_PIXELS], infra-red channels	ss	K/100	2	391
L1B-INT-89	R(ch, n; s, p)	calibrated.pixels.nadir[MAX_NADIR_PIXELS], visible channels	ss	%/100	2	575
L1B-INT-90	R(ch, n; s, p)	calibrated.pixels.forward[MAX_FORWARD_PIXELS], visible channels	ss	%/100	2	575
	track_y[]		float	km	4	
L1B-INT-64	nadir_x_coord[][575]	source packet nadir pixel x coords	float	km	4	575
L1B-INT-65	nadir_y_coord[][575]	source packet nadir pixel y coords	float	km	4	575
L1B-INT-66	frwrd_x_coord[][391]	source packet forward pixel x coords	float	km	4	391
L1B-INT-67	frwrd_y_coord[][391]	source packet forward pixel y coords	float	km	4	391
L1B-INT-101	I(ir12, n; i, j)	regridded nadir ir12 Brightness Temp.	ss	K/100	2	512
L1B-INT-102	I(ir11, n; i, j)	regridded nadir ir11 Brightness Temp.	ss	K/100	2	512
L1B-INT-103	I(ir37, n; i, j)	regridded nadir ir37 Brightness Temp.	ss	K/100	2	512
L1B-INT-104	I(v16, n; i, j)	regridded nadir v16 Reflectance	ss	%/100	2	512
L1B-INT-105	I(v870, n; i, j)	regridded nadir v870 Reflectance	ss	%/100	2	512
L1B-INT-106	I(v670, n; i, j)	regridded nadir v670 Reflectance	ss	%/100	2	512
L1B-INT-107	I(v555, n; i, j)	regridded nadir v555 Reflectance	ss	%/100	2	512
L1B-INT-111	I(ir12, f; i, j)	regridded forward ir12 Brightness Temp.	ss	K/100	2	512
L1B-INT-112	I(ir11, f; i, j)	regridded forward ir11 Brightness Temp.	ss	K/100	2	512
L1B-INT-113	I(ir37, f; i, j)	regridded forward ir37 Brightness Temp.	ss	K/100	2	512
L1B-INT-114	I(v16, f; i, j)	regridded forward v16 Reflectance	ss	%/100	2	512
L1B-INT-115	I(v870, f; i, j)	regridded forward v870 Reflectance	ss	%/100	2	512
L1B-INT-116	I(v670, f; i, j)	regridded forward v670 Reflectance	ss	%/100	2	512
L1B-INT-117	I(v555, f; i, j)	regridded forward v555 Reflectance	ss	%/100	2	512
L1B-INT-108	nadir_x_offset[][512]	Offset of source pixel from corner of	uc	km/255	1	512
L1B-INT-109	nadir_y_offset[][512]	regridded pixel	uc	km/255	1	512
L1B-INT-118	frwrd_x_offset[][512]		uc	km/255	1	512
L1B-INT-119	frwrd_y_offset[][512]		uc	km/255	1	512
L1B-INT-100	nadir_fill_state[][512]	Nadir fill state flag.	uc		1	512
L1B-INT-110	frwrd_fill_state[][512]	Forward fill state flag.	uc		1	512
		Regridded nadir information:	struct			
L1B-INT-129	nadir_min_aux_temps	nadir_min_aux_temps[6]	float	K	4	6
L1B-INT-130	nadir_max_aux_temps	nadir_max_aux_temps[6]	float	K	4	6
L1B-INT-131	nadir_packet_invalid	Nadir source packet invalid flags.	ss	flags	2	
L1B-INT-133	nadir_pixel_maps	nadir_pixel_maps[2]	ss		2	2
L1B-INT-134	scn_nadir(ig, j)	nadir view instrument scan number	us	n/a	2	j = 0, 511

L1B-INT-135	pxl_nadir(ig, j)	nadir view instrument pixel number	us	n/a	2	j = 0, 511
		Regridded forward information:	struct			
L1B-INT-149	frwrd_min_aux_temps	frwrd_min_aux_temps[6]	float		4	6
L1B-INT-150	frwrd_max_aux_temps	frwrd_max_aux_temps[6]	float		4	6
L1B-INT-151	frwrd_.packet_invalid	Forward source packet invalid flags.	ss	flags	2	
L1B-INT-153	frwrd_pixel_maps[2]	frwrd_pixel_maps	ss		2	2
L1B-INT-154	scn_frwrd(ig, j)	forward view instrument scan number	us	n/a	2	j = 0, 511
L1B-INT-155	pxl_frwrd(ig, j)	forward view instrument pixel number	us	n/a	2	j = 0, 511
L1B-INT-400		source_packet_ut_time(s)	double	days	8	
local	Δx	x displacement	double	km	8	1
local	Δy	y displacement	double	km	8	1
local	eps_i	signed perturbation in y direction	float	km	4	1
local	eps_j	signed perturbation in x direction	float	km	4	1
local	delta_i	fractional part of y co-ordinate	float	km	4	1
local	delta_j	fractional part of x co-ordinate	float	km	4	1
local	imin	minimum row index for regridding	sl	none	4	1
local	imax	maximum row index for regridding	sl	none	4	1
local	iaux	index to auxiliary data arrays	sl	none	4	1
local	jaux	index to auxiliary temperatures	sl	none	4	1
local	aux_temp(jaux)	auxiliary temperatures	float	K	4	6
local	aux_valid	temporary array for data validation results	ss	n/a	2	6
L1B-INT-003	advr[s][iaux]	auxiliary data validation result	ss	n/a	2	
		nadir_view_confidence_flags:				
L1B-INT-200		nadir_blanking_pulse(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-201		nadir_cosmetic(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-202		nadir_scan_absent(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-203		nadir_pixel_absent(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-204		nadir_packet_validation_error(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-205		nadir_zero_count(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-206		nadir_saturation(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-207		nadir_calibration_out_of_range(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-208		nadir_calibration_unavailable(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-209		nadir_unfilled_pixel(i, j)	ss	n/a	2	j = 0, 511
		forward view confidence flags:				
L1B-INT-216		frwrd_blanking_pulse(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-217		frwrd_cosmetic(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-218		frwrd_scan_absent(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-219		frwrd_pixel_absent(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-220		frwrd_packet_validation_error(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-221		frwrd_zero_count(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-222		frwrd_saturation(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-223		frwrd_calibration_out_of_range(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-224		frwrd_calibration_unavailable(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-225		frwrd_unfilled_pixel(i, j)	ss	n/a	2	j = 0, 511
L1B-INT-071	aux_temp(s, jaux)	temporary auxiliary temperatures	float	K	4	jaux=0, 5
L1B-INT-072	aux_unconv(s, jaux)	unconverted auxiliary temperatures	us	n/a	2	jaux=0, 5
L1B-INT-073	pixel_map(s)	pixel map number from Module 4	ss	n/a	2	per s
(constant)	UNFILLED_PIXEL	UNFILLED_PIXEL = 2	uc	n/a	1	1

Table 5-18-2: Internal Data Table - Regrid pixels

5.18.3 Algorithm Definition

Step 5.18.1 Initialize Image Arrays

Initialize the regridded data arrays. This is necessary to ensure (a) that the requirement for cosmetic fill can be recognised; (b) that unfilled pixels can be identified at the conclusion of the regridding and cosmetic fill processes; and (c) that the regridded maximum and minimum temperature variables are initialized.

nadir_fill_state(i, j) = UNFILLED_PIXEL

frwrd_fill_state(i, j) = UNFILLED_PIXEL

nadir_ min_aux_temps(i, k) = +999.0 (k = 0, 1, ...5)

nadir_ max_aux_temps(i, k) = -999.0 (k = 0, 1, ...5)

frwrd_ min_aux_temps(i, k) = +999.0 (k = 0, 1, ...5)

frwrd_ max_aux_temps(i, k) = -999.0 (k = 0, 1, ...5)

nadir_ pixel_maps(i, 0) = -1

nadir_ pixel_maps(i, 1) = -1

frwrd_ pixel_maps(i, 0) = -1

frwrd_ pixel_maps(i, 1) = -1

nadir_ packet_invalid(i) = 0

frwrd_ packet_invalid(i) = 0

(Req 5.18-1)

Also initialise the instrument scan and pixel number to zero for all values of *ig*, *j*:

nadir_view_scan_number(ig, j) = 0

nadir_view_pixel_number(ig, j) = 0

frwrd_view_scan_number(ig, j) = 0

frwrd_view_pixel_number(ig, j) = 0

(Req 5.18-2)

Initialise the nadir view confidence flags[L1B-INT-200 to L1B-INT-209] and forward view confidence flags [L1B-INT-216 to L1B-INT-225] to FALSE:

nadir_blanking_pulse(i, j) = FALSE; frwrd_blanking_pulse(i, j) = FALSE

nadir_cosmetic(i, j) = FALSE; frwrd_cosmetic(i, j) = FALSE

nadir_scan_absent(i, j) = FALSE; frwrd_scan_absent(i, j) = FALSE

nadir_pixel_absent(i, j) = FALSE; frwrd_pixel_absent(i, j) = FALSE

nadir_packet_validation_error(i, j) = FALSE

frwrd_packet_validation_error(i, j) = FALSE

nadir_zero_count(i, j) = FALSE; frwrd_zero_count(i, j) = FALSE

nadir_saturation(i, j) = FALSE; frwrd_saturation(i, j) = FALSE

nadir_calibration_out_of_range(i, j) = FALSE

frwrd_calibration_out_of_range(i, j) = FALSE

nadir_calibration_unavailable(i, j) = FALSE

frwrd_calibration_unavailable(i, j) = FALSE nadir_unfilled_pixel(i, j) = FALSE; frwrd_unfilled_pixel(i, j) = FALSE

(Req 5.18-3)

Step 5.18.2 Regrid Image Pixels, Nadir View

The forward and nadir scan arrays in the unpacked data are indexed by p_f and p_n such that

p_n = p - FIRST_NADIR_PIXEL_NUMBER

p_f = p - FIRST_FORWARD_PIXEL_NUMBER

where p is the absolute pixel number and

$0 \le p_n <$ MAX_NADIR_PIXELS

$0 \le p_f <$ MAX_FORWARD_PIXELS

Each source packet scan s is processed as follows.

For each nadir pixel p_n = 0, MAX_NADIR_PIXELS in the nadir scan s with

nadir_x_coord(s, p_n) > -256. and

nadir_x_coord(s, p_n) < 256.

(Req 5.18-4)

the regridding indices are calculated:

Find the index ig such that

track_y(ig + K) ≤ nadir_y_coord(s, p_n) < *track_y(ig + K + 1)*

Calculate

$$\Delta y = \frac{NGRANULE \cdot (nadir\_y\_coord(s, p_n) - track\_y(ig + K))}{(track\_y(ig + K + 1) - track\_y(ig + K))}$$

$$i' = \text{int}(\Delta y)$$

i = $NGRANULE \cdot ig + i'$

j = FIX(nadir_x_coord(s, p_n) + 256.)

delta_x = fractional part of (nadir_x_coord(s, p_n)) + 256.)

or equivalently

delta_x = nadir_x_coord(s, p_n)) + 256. - j.

(Req 5.18-5)

If *i* is negative, the pixel is outside the image bounds. Proceed to next pixel, omitting steps up to and including Step 5.18.2.4.

If $i' = 0$ (that is, if the value of i corresponds to a granule row), or
this is the first scan to be regridded, or
nadir_view_scan_number(i_g, j) = 0
go to **Step 5.18.2.4**. Otherwise test for modified regridding.

If

relative_scan_number(s) = nadir_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)

and

p_n = nadir_view_pixel_number(i_g, j) - *FIRST_NADIR_PIXEL_NUMBER*

then go to **Step 5.18.2.4.**

(Req 5.18-6)

Otherwise the natural regridding is not that required for the present granule. Try the other possibilities.

Extract

delta_i = fractional part of (Δy)

delta_j = fractional part of (nadir_x_coord(s, p_n) + 256.)

Calculate

eps_i = sign(delta_i - 0.5)*eps_y*0.001

eps_j = sign(delta_j - 0.5)*eps_x*0.001

(Req 5.18-7)

Step 5.18.2.1 Case <1> - x displacement

The regridding indices are calculated (i is unchanged)

j = FIX(nadir_x_coord(s, p_n) + 256. + eps_j)

delta_x = fractional part of (nadir_x_coord(s, p_n)) + 256. + eps_j)

or equivalently

delta_x = nadir_x_coord(s, p_n)) + 256. + eps_j - j.

Test for modified regridding:

If

relative_scan_number(s) = nadir_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)

and

p_n = nadir_view_pixel_number(i_g, j) - *FIRST_NADIR_PIXEL_NUMBER*

then go to **Step 5.18.2.4**.

(Req 5.18-8)

Step 5.18.2.2 Case <2> - y displacement

The regridding indices are calculated:

Find the (possibly new) index ig such that

$$track\_y(ig + K) \le nadir\_y\_coord(s, p_n) + eps\_i < track\_y(ig + K + 1)$$

by the use of the same equation for Δy as that preceding Req. 5.18-5 but with *nadir_y_coord(s, p_n) + eps_i* in place of *nadir_y_coord(s, p_n).*

Calculate

$$i' = \mathrm{int}(\Delta y)$$

i = $NGRANULE \cdot ig + i'$

j = FIX(nadir_x_coord(s, p_n) + 256.)

delta_x = fractional part of (nadir_x_coord(s, p_n)) + 256.)

or equivalently

delta_x = nadir_x_coord(s, p_n)) + 256. - j.

If $i' = 0$ (that is, if the value of i corresponds to a granule row) go to **Step 5.18.2.4**. Otherwise test for modified regridding.

If
relative_scan_number(s) = nadir_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)
and
p_n = nadir_view_pixel_number(i_g, j) - *FIRST_NADIR_PIXEL_NUMBER*
then go to **Step 5.18.2.4**.

(Req 5.18-9)

Step 5.18.2.3 Case <3> - displacement in both co-ordinates

The regridding indices are calculated (i is unchanged).

j = FIX(nadir_x_coord(s, p_n) + 256. + eps_j)

delta_x = fractional part of (nadir_x_coord(s, p_n)) + 256. + eps_j)

or equivalently

delta_x = nadir_x_coord(s, p_n)) + 256. + eps_j - j.

Test for modified regridding.

If
relative_scan_number(s) = nadir_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)
and
p_n = nadir_view_pixel_number(i_g, j) - *FIRST_NADIR_PIXEL_NUMBER*
then go to **Step 5.18.2.4**.

If execution gets to here, this is an unused pixel. Proceed to next pixel.

(Req 5.18-10)

Step 5.18.2.4 Regrid nadir pixel

If $i' = 0$, or this is the first scan to be regridded, or nadir_view_scan_number(i_g, j) = 0 save the scan and pixel number for the regridded image point in the nadir view:

nadir_view_scan_number(ig, j) = relative_scan_number(s) - i′

nadir_view_pixel_number(ig, j) = p_n + FIRST_NADIR_PIXEL_NUMBER

(Req 5.18-11)

Finally the nadir source packet data for each channel are copied to the regridded data arrays.

For the infra-red channels ch = ir12, ir11, ir37

I(ch, n; i, j) = T(ch,n; s, p_n)

and for the visible channels ch = v16, v870, v670, v555

I(ch, n, i, j) = R(ch, n; s, p_n).

(Req 5.18-12)

Set the confidence flags for the relevant image pixels.

If [L1B-INT-091](s, p_n) is non-zero then set nadir_blanking_pulse(i, j) = TRUE

for each channel ch = 1, 7

if I(ch, n; i, j) < 0 then

if I(ch, n; i, j) = PIXEL_COUNT_FROM_NULL_PACKET

then set flag nadir_scan_absent = TRUE

else if = PIXEL_COUNT_INITIAL_VALUE

then set flag nadir_pixel_absent = TRUE

else if I(ch, n; i, j) =
PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED

then set flag nadir_packet_validation_error = TRUE

else if I(ch, n; i, j) = PIXEL_COUNT_ZERO

then set flag nadir_zero_count = TRUE

else if I(ch, n; i, j) = PIXEL_COUNT_SATURATED

then set flag nadir_saturation = TRUE

else if I(ch, n; i, j) = PIXEL_RADIANCE_OUTSIDE_CALIBRATION

then set flag nadir_cal_out_of_range = TRUE

else if I(ch, n; i, j) = CALIBRATION_UNAVAILABLE_FOR_PIXEL

then set flag nadir_calibration_unavailable = TRUE

endif

endif

(end loop over channels)

(Req 5.18-13)

Also the offset of the source pixel from the corner of the regridded pixel is calculated and scaled to unsigned byte values.

nadir_x_offset(i, j) = FIX(255 * (Δx))

nadir_y_offset(i, j) = FIX(255 * (fractional part of Δy))

nadir_fill_state(i, j) = *NATURAL_PIXEL*

Step 5.18.3 Regrid Image Pixels, Forward View

This process is repeated for the forward scan as follows. For each forward pixel p_f = 0, *MAX_FORWARD_PIXELS* in the forward scan s with frwrd_x_coord(s, p_f) > -256.

frwrd_x_coord(s, p_f) < 256.

the regridding indices are calculated as follows:

Find the index ig such that

track_y(ig + K) ≤ frwrd_y_coord(s, p_f) < *track_y(ig + K + 1)*

Calculate

$$\Delta y = \frac{NGRANULE \cdot (frwrd\_y\_coord(s, p_f) - track\_y(ig + K))}{(track\_y(ig + K + 1) - track\_y(ig + K))}$$

$$i' = \mathrm{int}(\Delta y)$$

i = $NGRANULE \cdot ig + i'$

j = FIX(frwrd_x_coord(s, p_f) + 256.)

delta_x = fractional part of (frwrd_x_coord(s, p_f)) + 256.)

or equivalently

delta_x = frwrd_x_coord(s, p_f)) + 256. - j.

(Req 5.18-14)

If *i* is negative, the pixel is outside the image bounds. Proceed to next pixel, omitting steps up to and including Step 5.18.3.4.

If $i' = 0$ (that is, if the value of i corresponds to a granule row), or
this is the first scan to be regridded, or
frwrd_view_scan_number(i_g, j) = 0
go to **Step 5.18.3.4.** Otherwise test for modified regridding.

If
relative_scan_number(s) = frwrd_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)
and
p_f = frwrd_view_pixel_number(i_g, j) - *FIRST_FORWARD_PIXEL_NUMBER*
then go to **Step 5.18.3.4.**

(Req 5.18-15)

Otherwise the natural regridding is not that required for the present granule. Try the other possibilities.

Extract

delta_i = fractional part of (Δy)

delta_j = fractional part of (frwrd_x_coord(s, p_f) + 256.)

Calculate

eps_i = sign(delta_i - 0.5)*eps_y*0.001

eps_j = sign(delta_j - 0.5)*eps_x*0.001

(Req 5.18-16)

Step 5.18.3.1 Case <1> - x displacement

The regridding indices are calculated (i is unchanged)

j = FIX(frwrd_x_coord(s, p_f) + 256. + eps_j)

delta_x = fractional part of (frwrd_x_coord(s, p_f)) + 256. + eps_j)

or equivalently

delta_x = frwrd_x_coord(s, p_f)) + 256. + eps_j - j.

Test for modified regridding:

If

relative_scan_number(s) = frwrd_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)

and

p_f = frwrd_view_pixel_number(i_g, j) - *FIRST_FORWARD_PIXEL_NUMBER*

then go to **Step 5.18.3.4.**

(Req 5.18-17)

Step 5.18.3.2 Case <2> - y displacement

The regridding indices are calculated:

Find the (possibly new) index ig such that

$track\_y(ig + K) \leq frwrd\_y\_coord(s, p_f) + eps\_i < track\_y(ig + K + 1)$

by the use of the same equation for Δy as that preceding Req. 5.18-14 but with *frwrd_y_coord(s, p_f) + eps_i* in place of *frwrd_y_coord(s, p_f).*

Calculate

$$i' = \mathrm{int}(\Delta y)$$

i = $NGRANULE \cdot ig + i'$

j = FIX(frwrd_x_coord(s, p_f) + 256.)

delta_x = fractional part of (frwrd_x_coord(s, p_f)) + 256.)

or equivalently

delta_x = frwrd_x_coord(s, p_f)) + 256. - j.

If $i' = 0$ (that is, if the value of i corresponds to a granule row) go to **Step 5.18.3.4**. Otherwise test for modified regridding.

If

relative_scan_number(s) = frwrd_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)

and

p_f = frwrd_view_pixel_number(i_g, j) - *FIRST_FORWARD_PIXEL_NUMBER*

then go to **Step 5.18.3.4.**

(Req 5.18-18)

Step 5.18.3.3 Case <3> - displacement in both co-ordinates

The regridding indices are calculated (i is unchanged).

j = FIX(frwrd_x_coord(s, p_f) + 256. + eps_j)

delta_x = fractional part of (frwrd_x_coord(s, p_f)) + 256. + eps_j)

or equivalently

delta_x = frwrd_x_coord(s, p_f)) + 256. + eps_j - j.

Test for modified regridding.

If

relative_scan_number(s) = frwrd_view_scan_number(i_g, j) +(i - *NGRANULE**i_g)

and

p_f = frwrd_view_pixel_number(i_g, j) - *FIRST_FORWARD_PIXEL_NUMBER*

then go to **Step 5.18.3.4.**

If execution gets to here, this is an unused pixel. Proceed to next pixel.

(Req 5.18-19)

Step 5.18.3.4 Regrid forward pixel

If $i' = 0$, or this is the first scan to be regridded, or nadir_view_scan_number(i_g, j) = 0 save the scan and pixel number for the regridded image point in the frwrd view:

frwrd_view_scan_number(ig, j) ≡ [L1B-INT-154](ig, j) = relative_scan_number(s) - i′

frwrd_view_pixel_number(ig, j) ≡ [L1B-INT-155](ig, j) =
p_f + *FIRST_FORWARD_PIXEL_NUMBER*

(Req 5.18-20)

Finally the forward source packet data for each channel are copied to the regridded data arrays.

For the infra-red channels ch = ir12, ir11, ir37

I(ch, f; i, j) = T(ch, f; s, p_f)

and for the visible channels ch = v16, v870, v670, v555

I(ch, f, i, j) = R(ch, f; s, p_f).

(Req 5.18-21)

Set the confidence flags for the relevant image pixels.

If [L1B-INT-092](s, p_f) is non-zero then set frwrd_blanking_pulse(i, j) = TRUE

for each channel ch = 1, 7

if I(ch, f; i, j) < 0 then

if I(ch, f; i, j) = PIXEL_COUNT_FROM_NULL_PACKET

then set flag frwrd_scan_absent = TRUE

else if = PIXEL_COUNT_INITIAL_VALUE

then set flag frwrd_pixel_absent = TRUE

else if I(ch, f; i, j) =
PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED

then set flag frwrd_packet_validation_error = TRUE

else if I(ch, f; i, j) = PIXEL_COUNT_ZERO

then set flag frwrd_zero_count = TRUE

else if I(ch, f; i, j) = PIXEL_COUNT_SATURATED

then set flag frwrd_saturation = TRUE

else if I(ch, f; i, j) = PIXEL_RADIANCE_OUTSIDE_CALIBRATION

then set flag frwrd_cal_out_of_range = TRUE

else if I(ch, f; i, j) = CALIBRATION_UNAVAILABLE_FOR_PIXEL

then set flag frwrd_calibration_unavailable = TRUE

endif

endif

(end loop over channels)

(Req 5.18-22)

Also the offset of the source pixel from the corner of the regridded pixel is calculated and scaled to unsigned byte values.

frwrd_x_offset(i, j) = FIX(255 * (Δx))

frwrd_y_offset(i, j) = FIX(255 * (fractional part of Δyfrwrd_y_coord(s, p_f) - i))

(Req 5.18-23)

Set the fill state for the pixel just regridded:

frwrd_fill_state(i, j) = *NATURAL_PIXEL*

(Req 5.18-24)

Step 5.18.4 Regrid Auxiliary Information

Other nadir view information from the source packet is now regridded.

(The former strip centre scan times are redundant, and the corresponding requirement is therefore deleted.)

(Req 5.18-25)

To regrid the remaining information find the first and last regridded image scans to which this instrument scan contributes. The extreme scan y positions are :-

nadir_y_coord(s, 0)

nadir_y_coord(s, number_of_scan_pixels - 1)

nadir_y_coord(s, number_of_scan_pixels / 2)

Find the minimum and maximum of these positions ymin and ymax. Then find the regrid indices *imin* and *imax* using the method for calculating *i* detailed in requirement 5.18-4.

(Req 5.18-26)

Note: The text describing the extraction of auxiliary temperatures for regridding, that appeared at this point in earlier revisions of this document, has been moved to Section 5.4.3 (Module 4).

For each scan i = imin, imax do the following:

Test packet_error(s) in turn against the following possible values.

RAWPKT_FAILS_BASIC_VALIDATION_ERR

CRC_ERROR_DETECTED_ERR

BUFFERS_FULL_CHECK_ERR

PIX_SCAN_JITTER_ERR

If equality is found the corresponding bit in the nadir packet invalid word [L1B-INT-131](i) is set. Thus

If packet_error(s) = RAWPKT_FAILS_BASIC_VALIDATION_ERR then

    set [L1B-INT-131](i) bit 1.

else if packet_error = CRC_ERROR_DETECTED_ERR then

    set [L1B-INT-131](i) bit 2.

else if packet_error = BUFFERS_FULL_CHECK then

    set [L1B-INT-131](i) bit 3.

else if packet_error = PIX_SCAN_JITTER_ERROR then

    set [L1B-INT-131](i) bit 4.

(Req 5.18-27)

Next the minimum and maximum values of each of the 6 auxiliary temperatures for the current scan are updated when the current values are exceeded by the instrument scan values.

ie for jaux = 0 to 5

if aux_unconv(s, jaux) > 0 then

    if aux_temp(s, jaux) < nadir_min_aux_temps(i, jaux) then
        nadir_min_aux_temp(i, jaux) = aux_temp(s, jaux)

    if aux_temp(s, jaux) > nadir_max_aux_temps(i, jaux) then
        nadir_max_aux_temp(i, jaux) = aux_temp(s, jaux)

(Req 5.18-28)

end if

Also the first two different pixel map numbers are recorded.

```
if nadir_pixel_maps(i, 0) = -1 then
        nadir_pixel_maps(i, 0) = pixel_map(s)
else if nadir_pixel_maps(i, 1) = -1 and nadir_pixel_maps( i, 0 ) NE pixel_map(s)
then    nadir_pixel_maps(i, 1) = pixel_map(s)
```

(End of loop over index i.)

This process is repeated to obtain the forward scan information.

(Req 5.18-29)

5.19 Module Definition: Cosmetic Fill

5.19.1 Functional Description

This routine follows the regrid source packets module and fills in any gaps in the regridded image.

5.19.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
	i	index to image scans	sl	none	4	1
	j	index to pixels, j=0, ..., 511	sl	none	4	1
L1B-INT-101	I(ir12, n; i, j)	regridded nadir ir12 Brightness Temp.	ss	0.01K	2	512
L1B-INT-102	I(ir11, n; i, j)	regridded nadir ir11 Brightness Temp.	ss	0.01K	2	512
L1B-INT-103	I(ir37, n; i, j)	regridded nadir ir37 Brightness Temp.	ss	0.01K	2	512
L1B-INT-104	I(v16, n; i, j)	regridded nadir v16 Reflectance	ss	0.01%	2	512
L1B-INT-105	I(v870, n; i, j)	regridded nadir v870 Reflectance	ss	0.01%	2	512
L1B-INT-106	I(v670, n; i, j)	regridded nadir v670 Reflectance	ss	0.01%	2	512
L1B-INT-107	I(v555, n; i, j)	regridded nadir v555 Reflectance	ss	0.01%	2	512
L1B-INT-111	I(ir12, f; i, j)	regridded forward ir12 Brightness Temp.	ss	0.01K	2	512
L1B-INT-112	I(ir11, f; i, j)	regridded forward ir11 Brightness Temp.	ss	0.01K	2	512
L1B-INT-113	I(ir37, f; i, j)	regridded forward ir37 Brightness Temp.	ss	0.01K	2	512
L1B-INT-114	I(v16, f; i, j)	regridded forward v16 Reflectance	ss	0.01%	2	512
L1B-INT-115	I(v870, f; i, j)	regridded forward v870 Reflectance	ss	0.01%	2	512
L1B-INT-116	I(v670, f; i, j)	regridded forward v670 Reflectance	ss	0.01%	2	512
L1B-INT-117	I(v555, f; i, j)	regridded forward v555 Reflectance	ss	0.01%	2	512
L1B-INT-108	nadir_x_offset(i, j)	Offset of source pixel from corner of	uc	km/255	1	512
L1B-INT-109	nadir_y_offset(i, j)	regridded pixel	uc	km/255	1	512
L1B-INT-118	frwrd_x_offset(i, j)		uc	km/255	1	512
L1B-INT-119	frwrd_y_offset(i, j)		uc	km/255	1	512
L1B-INT-100	nadir_fill_state(i, j)	Nadir fill state flag	uc	n/a	1	512
L1B-INT-110	frwrd_fill_state(i, j)	Forward fill state flag	uc	n/a	1	512
		nadir view confidence flags:				
L1B-INT-201		nadir cosmetic fill flag	ss	n/a	2	j = 0, 511
L1B-INT-209		nadir view unfilled pixel flag	ss	n/a	2	j = 0, 511
		forward view confidence flags:				
L1B-INT-217		frwrd cosmetic fill flag	ss	n/a	2	j = 0, 511
L1B-INT-225		frwrd view unfilled pixel flag	ss	n/a	2	j = 0, 511
		Regridded nadir information:	struct			
L1B-INT-120		nadir_band_edge_solar_elevation	float	degrees	4	11
L1B-INT-121		nadir_band_edge_satellite_elevation	float	degrees	4	11

L1B-INT-122		nadir_band_edge_solar_azimuth	float	degrees	4	11
L1B-INT-123		nadir_band_edge_satellite_azimuth	float	degrees	4	11
L1B-INT-124		nadir_band_centre_solar_elevation	float	degrees	4	10
L1B-INT-125		nadir_band_centre_satellite_elevation	float	degrees	4	10
L1B-INT-126		nadir_band_centre_solar_azimuth	float	degrees	4	10
L1B-INT-127		nadir_band_centre_satellite_azimuth	float	degrees	4	10
		Regridded forward information:	struct			
L1B-INT-140		frwrd_band_edge_solar_elevation	float	degrees	4	11
L1B-INT-141		frwrd_band_edge_satellite_elevation	float	degrees	4	11
L1B-INT-142		frwrd_band_edge_solar_azimuth	float	degrees	4	11
L1B-INT-143		frwrd_band_edge_satellite_azimuth	float	degrees	4	11
L1B-INT-144		frwrd_band_centre_solar_elevation	float	degrees	4	10
L1B-INT-145		frwrd_band_centre_satellite_elevation	float	degrees	4	10
L1B-INT-146		frwrd_band_centre_solar_azimuth	float	degrees	4	10
L1B-INT-147		frwrd_band_centre_satellite_azimuth	float	degrees	4	10
(constant)	NATURAL_PIXEL	NATURAL_PIXEL = 0	uc	n/a	1	1
(constant)	COSMETIC_PIXEL	COSMETIC_PIXEL = 1	uc	n/a	1	1

Table 5-19-1: Internal Data Table - Cosmetic Fill

5.19.3 Algorithm Definition

Step 5.19.1 Fill Image pixels

This process is carried out on the regridded nadir image (v = nadir) and then on the regridded forward image (v = forward).

For each pixel (i, j) in the regridded image check the fill state flag and if the pixel is unfilled, ie

Fill_state(v, i, j) = UNFILLED_PIXEL

(Req 5.19-1)

then an attempt is made to fill it using data from an adjacent pixel as follows.

Examine the 8 regridded pixels adjacent to the unfilled pixel.

Each candidate pixel (ic, jc) must satisfy the following tests:

1. It is not beyond the edge of the image
   0 ≤ ic < MAX_v_PIXELS and 0 ≤ jc < 512
2. It is a naturally filled pixel
   Fill_state(v, ic, jc) = NATURAL_PIXEL

If no candidate pixels are found then the pixel remains unfilled and the pixel values in every channel ch are set to the PIXEL_UNFILLED value.

I(ch, v, i, j) = PIXEL_UNFILLED [L1-AUX18-55]

Set the corresponding confidence flag [L1B-INT-209] or [L1B-INT-225] accordingly:

<view>_unfilled_pixel(i, j) = TRUE.

If candidate pixels are found then for each candidate the distance of the source pixel from the centre of the unfilled target pixel is calculated

dy = ic - i + (y_offset(v, ic, jc) / 255.) - 0.5 in the y direction
dx = jc - j + (x_offset(v, ic, jc) / 255.) - 0.5 in the x direction
d_squared = $dx^2 + dy^2$ distance squared.

and the candidate with the smallest distance is selected.

(Req 5.19-2)

The fill state flag for the target pixel is now set to indicate a cosmetically filled pixel and the pixel values from the selected candidate pixel are copied to the target pixel for all channels ch.

Fill_state(v, i, j) = COSMETIC_PIXEL.
I(ch, v, i, j) = I(ch, v, ic, jc)

Set the corresponding confidence flag [L1B-INT-201] or [L1B-INT-217] accordingly:

<view>_cosmetic(i, j) = TRUE.

(Req 5.19-3)

The y_offset and x_offset parameters of the target pixel are set to zero regardless of whether the pixel was filled or not.

y_offset(v, i, j) = 0
x_offset(v, i, j) = 0

(Req 5.19-4)

Step 5.19.2 Fill Auxiliary Information

Any gaps in the band edge solar angles and band centre solar angles must now be filled. A gap is filled using data from an adjacent valid point along track in the arrays, or if this fails, an adjacent valid point across track is used as follows.

The solar angles are valid for scan i and across track band k if (i, k) is within the image array bounds and the solar elevation angle at (i, k) is greater than -900.

If <view>_band_edge_solar_elevation(i, k) is invalid then check the <view>_band_edge_solar_elevation at (i-1 ,k), (i+1, k), (i, k-1) and (i, k+1), selecting the first valid one. If one is found, at say (i1, k1) then the band edge solar angles at (i, k) can be filled using the data at (i1, k1) ie

<view>_band_edge_solar_elevation(i, k) = <view>_band_edge_solar_elevation(i1, k1)
<view>_band_edge_satellite_elevation(i, k) =
<view>_band_edge_satellite_elevation(i1, k1)

The processing above is repeated in the same way for the band edge solar and satellite azimuths [L1B-INT-122], [L1B-INT-123] for the nadir view, [L1B-INT-142], [L1B-INT-143] for the forward view.

(Req 5.19-5)

Note that in the case of the solar angles, the implementation differs from that adopted for image pixels, in that all unfilled locations are filled regardless of whether or not they have naturally filled neighbours. To ensure this the image segment is searched in the four directions in turn, first from bottom up, then from the top down, then from left to right, and finally from right to left. The consequence of this is that if there is a large data gap not covered by instrument scans (this may happen at the beginning and end of data in the Near Real Time case, if the product limits exceed the valid data, including null packets), the gap is filled by copying values from the nearest available scan. If this were not done, the cloud clearing modules might attempt to use invalid angles. The specific implementation is as follows.

```
for k = 0, 10
for i = 1, 511
      if <view>_band_edge_solar_elevation( i, k ) < 900. then
      <view>_band_edge_solar_elevation(i, k) =
        <view>_band_edge_solar_elevation(i-1, k)
      <view>_band_edge_satellite_elevation(i, k) =
        <view>_band_edge_satellite_elevation(i-1, k)
      <view>_band_edge_solar_azimuth(i, k) =
        <view>_band_edge_solar_azimuth(i-1, k)
      <view>_band_edge_satellite_azimuth(i, k) =
        <view>_band_edge_satellite_azimuth(i-1, k)
      end if
end for
for i = 510, 0
      if <view>_band_edge_solar_elevation( i, k ) < 900. then
      <view>_band_edge_solar_elevation(i, k) =
        <view>_band_edge_solar_elevation(i+1, k)
      <view>_band_edge_satellite_elevation(i, k) =
        <view>_band_edge_satellite_elevation(i+1, k)
      <view>_band_edge_solar_azimuth(i, k) =
        <view>_band_edge_solar_azimuth(i+1, k)
      <view>_band_edge_satellite_azimuth(i, k) =
        <view>_band_edge_satellite_azimuth(i+1, k)
      end if
end for
end for
for i = 0, 511
for k = 1, 10
      if <view>_band_edge_solar_elevation( i, k ) < 900. then
      <view>_band_edge_solar_elevation(i, k) =
        <view>_band_edge_solar_elevation(i, k-1)
      <view>_band_edge_satellite_elevation(i, k) =
        <view>_band_edge_satellite_elevation(i, k-1)
      <view>_band_edge_solar_azimuth(i, k) =
        <view>_band_edge_solar_azimuth(i, k-1)
      <view>_band_edge_satellite_azimuth(i, k) =
        <view>_band_edge_satellite_azimuth(i, k-1)
      end if
end for
for k = 9, 0
      if <view>_band_edge_solar_elevation( i, k ) < 900. then
      <view>_band_edge_solar_elevation(i, k) =
        <view>_band_edge_solar_elevation(i, k+1)
```

```
        <view>_band_edge_satellite_elevation(i, k) =
          <view>_band_edge_satellite_elevation(i, k+1)
        <view>_band_edge_solar_azimuth(i, k) =
          <view>_band_edge_solar_azimuth(i, k+1)
        <view>_band_edge_satellite_azimuth(i, k) =
          <view>_band_edge_satellite_azimuth(i, k+1)
        end if
end for
end for
```

Note that this implementation (like the more complex cloud clearing algorithms in Section 5.22) is based on a division of the image into image segments of 512 rows (i = 0, 511). It is scientifically acceptable to reduce the size of the image segment so treated, provided it extends over at least 1 granule (32 rows), although this change may give rise to small differences in the results.

If <view>_band_centre_solar_elevation(i, k) is invalid then check the <view>_band_centre_solar_elevation at (i-1, k), (i+1, k), (i, k-1) and (i, k+1), selecting the first valid one. If one is found, at say (i1, k1) then the band centre solar angles at (i, k) can be filled using the data at (i1, k1) ie

<view>_band_centre_solar_elevation(i, k) =
<view>_band_centre_solar_elevation(i1, k1)
<view>_band_centre_satellite_elevation(i, k) =
<view>_band_centre_satellite_elevation(i1, k1)
etc.

The processing above is repeated in the same way for the band centre solar and satellite azimuths [L1B-INT-126], [L1B-INT-127] for the nadir view, [L1B-INT-146], [L1B-INT-147] for the forward view.

The specific implementation is as follows.

```
for k = 0, 9
for i = 1, 511
        if <view>_band_centre_solar_elevation( i, k ) < 900. then
        <view>_band_centre_solar_elevation(i, k) =
          <view>_band_centre_solar_elevation(i-1, k)
        <view>_band_centre_satellite_elevation(i, k) =
          <view>_band_centre_satellite_elevation(i-1, k)
        <view>_band_centre_solar_azimuth(i, k) =
          <view>_band_centre_solar_azimuth(i-1, k)
        <view>_band_centre_satellite_azimuth(i, k) =
          <view>_band_centre_satellite_azimuth(i-1, k)
        end if
end for
for i = 510, 0
        if <view>_band_centre_solar_elevation( i, k ) < 900. then
        <view>_band_centre_solar_elevation(i, k) =
          <view>_band_centre_solar_elevation(i+1, k)
        <view>_band_centre_satellite_elevation(i, k) =
          <view>_band_centre_satellite_elevation(i+1, k)
```

```
        <view>_band_centre_solar_azimuth(i, k) =
          <view>_band_centre_solar_azimuth(i+1, k)
        <view>_band_centre_satellite_azimuth(i, k) =
          <view>_band_centre_satellite_azimuth(i+1, k)
        end if
end for
end for
for i = 0, 511
for k = 1, 9
        if <view>_band_centre_solar_elevation( i, k ) < 900. then
        <view>_band_centre_solar_elevation(i, k) =
          <view>_band_centre_solar_elevation(i, k-1)
        <view>_band_centre_satellite_elevation(i, k) =
          <view>_band_centre_satellite_elevation(i, k-1)
        <view>_band_centre_solar_azimuth(i, k) =
          <view>_band_centre_solar_azimuth(i, k-1)
        <view>_band_centre_satellite_azimuth(i, k) =
          <view>_band_centre_satellite_azimuth(i, k-1)
        end if
end for
for k = 8, 0
        if <view>_band_centre_solar_elevation( i, k ) < 900. then
        <view>_band_centre_solar_elevation(i, k) =
          <view>_band_centre_solar_elevation(i, k+1)
        <view>_band_centre_satellite_elevation(i, k) =
          <view>_band_centre_satellite_elevation(i, k+1)
        <view>_band_centre_solar_azimuth(i, k) =
          <view>_band_centre_solar_azimuth(i, k+1)
        <view>_band_centre_satellite_azimuth(i, k) =
          <view>_band_centre_satellite_azimuth(i, k+1)
        end if
end for
end for
```

(Req 5.19-6)

(Requirement deleted.)

(Req 5.19-7)

Step 5.19.3 Fill Nadir Scan Number Array

In order to permit the array of nadir view instrument scan numbers (scn_nadir(ig, j), [L1B-INT-134]) to be used during Level 2 processing to time tag averaged product cells, it is also ‘cosmetically filled’ in a similar way.

```
for all ig
        for j = 1, 511
          if scn_nadir(ig, j) = 0 then
            scn_nadir(ig, j) = scn_nadir(ig, j - 1)
          end if
        end for
        for j = 510, 0
```

```
        if scn_nadir(ig, j) = 0 then
          scn_nadir(ig, j) = scn_nadir(ig, j + 1)
        end if
      end for
end for
```

(Req 5.19-8)

5.20 Module Definition: Image Pixel Positions

5.20.1 Functional Description

The module Generate Geolocation Grids has produced the latitudes and longitudes of a series of uniformly space tie point pixels. The present module derives the latitude and longitude of each of image pixel by linear interpolation, in two dimensions, between the tie points. In the case of longitude account must be taken of the possibility that the 180 degree meridian intersects the image scan. The input longitudes are geocentric; the module also transforms the output latitudes to geodetic.

5.20.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX16-15	NGRANULE	Number of image rows per granule	sl	none	4	1
L1-AUX16-16	NGRID	Number of grid rows	sl	none	4	1

Table 5-20-1: Input Data Table - Image Pixel Positions

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-26	$\varphi_g(i_g, j_g)$	Grid point latitude	float	deg.	4	jg = 0, 22
L1B-INT-27	$\lambda_g(i_g, j_g)$	Grid point longitude	float	deg.	4	jg = 0, 22
local	i_g	index to tie scans	sl	none	4	1
local	j_g	index to tie point pixels (jg = 0, 22)	sl	none	4	1
local	i	index to image scans	sl	none	4	1
local	j	index to image pixels (j = 0, 511)	sl	none	4	1
L1B-INT-160	$\varphi(i, j)$	image latitude	float	deg.	4	j = 0, 511
L1B-INT-161	$\lambda(i, j)$	image longitude	float	deg.	4	j = 0, 511
local	w_x	interpolation weight in x	float	none	4	1
local	w_y	interpolation weight in y	float	none	4	1
local	φ_1	Intermediate latitude	float	degrees	4	1
local	φ_2	Intermediate latitude	float	degrees	4	1
local	λ_1	Intermediate longitude	float	degrees	4	1
local	λ_2	Intermediate longitude	float	degrees	4	1

Table 5-20-2: Internal Data Table - Image Pixel Positions

5.20.3 Algorithm Definition

Given an image scan and pixel number i, j define

i_g = integer part of $i/NGRANULE$

$$w_y = (i / NGRANULE) - i_g$$

j_g = integer part of (j + 19)/25

$$w_x = (j+19)/25 - j_g$$

(Req 5.20-1)

Interpolate the geodetic latitudes as follows:

$$\varphi(i,j) = \varphi_1 + w_y\{\varphi_2 - \varphi_1\}$$

where

$$\varphi_1 = \varphi_g(i_g, j_g) + w_x\left\{\varphi_g(i_g, j_g+1) - \varphi_g(i_g, j_g)\right\}$$

and

$$\varphi_2 = \varphi_g(i_g+1, j_g) + w_x\left\{\varphi_g(i_g+1, j_g+1) - \varphi_g(i_g+1, j_g)\right\}$$

(Req 5.20-2)

Longitude is treated similarly unless the meridian is present:

$$\lambda(i,j) = \lambda_1 + w_y\{\lambda_2 - \lambda_1\}$$

where

$$\lambda_1 = \lambda_g(i_g, j_g) + w_x\left\{\lambda_g(i_g, j_g+1) - \lambda_g(i_g, j_g)\right\}$$

and

$$\lambda_2 = \lambda_g(i_g+1, j_g) + w_x\left\{\lambda_g(i_g+1, j_g+1) - \lambda_g(i_g+1, j_g)\right\}$$

A test for the presence of the meridian is that

$$(\lambda_{\max} - \lambda_{\min}) > 180.0$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are respectively the greatest and least of

$$\lambda_g(i_g, 0), \lambda_g(i_g, 22), \lambda_g(i_g+1, 0), \lambda_g(i_g+1, 22)$$

In this case 360.0 is added to each of the grid longitudes that is initially negative before it is substituted in the above equations. The resultant interpolated longitude translated into the range -180.0 to 180.0 degrees by subtracting 360.0 if its value exceeds 180.0.

(Req 5.20-3)

5.21 Module Definition: Determine Land/Sea Flag

5.21.1 Functional Description

The objective of this algorithm, the determination of the surface flag, is to determine, for each pixel, the setting of a flag to identify whether the surface at the pixel is land or sea, given the latitude and longitude of the pixel.

The surface flag algorithm is very simple in principle. It is based on the existence of a land/sea map at 0.01 degree resolution. The map divides the surface of the earth into cells, and for each cell a flag is defined that identifies the type of the surface within it as land or sea. Given the latitude and longitude of the pixel, the algorithm identifies in which cell the pixel lies, and sets the land flag according to the surface type specified for that cell.

In order to permit a reasonably efficient algorithm, and to allow efficient storage of the map, the algorithm used in SADIST-2 uses a hierarchical structure with three different resolutions for the surface map. For regions in the middle of an ocean, or a continental land mass, the cell size can be large, since the surface type will be uniform over a large area. In coastal regions, on the other hand, higher resolution is required.

Thus the surface flag algorithm makes use of three files. The first of these is a map of surface type at 1° resolution in both latitude and longitude. Thus the file contains an array of 180 x 360 entries, each of which refers to a cell extending over 1° of latitude by 1° of longitude. For each cell, the entry identifies the surface type if the surface within the cell is considered to be entirely sea, or entirely land. If, however, the cell is intersected by a coastline, so that both surface types are present within it, the entry for the cell identifies an entry in a second file, which contains surface type at 0.1° resolution for the coastal cells.

Each 1° by 1° coastal cell is considered to be divided into 100 subsidiary cells of extent 0.1° by 0.1°, and for each 1° by 1° coastal cell, the second file contains 100 entries, one for each subsidiary cell. If the surface type is uniform within the subsidiary cell, the entry identifies whether it is land or sea; otherwise the cell is a coastal cell, and the entry identifies the entry for the subsidiary cell in a third file which contains surface type at 0.01° resolution for the relevant cells. For each cell that has an entry in the final file, the entry consists of 100 bits each identifying the surface type for one of the 100 cells of extent 0.01° by 0.01° contained within the 0.1° by 0.1° cell.

Thus the algorithm first identifies the 1° by 1° cell within which the pixel lies; if the surface type within the cell is uniform, the land flag is set according to the surface type specified for that cell. Otherwise, if the 1° by 1° cell is coastal, the entry for the cell in the file at 0.1°resolution is located. The 0.1° by 0.1° cell within which the pixel lies is determined, and the surface type for the cell is determined; again if the surface type within the cell is uniform, the land flag is set according to the surface type specified. Otherwise the process is repeated using the file at 0.01° resolution.

The present algorithm permits an option whereby the search may be terminated at an earlier stage if the full resolution is not required. In this case, a cell of mixed surface type (a coastal cell) is flagged as land.

5.21.2 Interface Definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX16-30	long_flag	Longitude offset flag (0 = no offset)	sl	none	4	1
L1-AUX16-31	long_offset	Longitude offset (for use in system testing)	sl	μdeg	4	1
L1-AUX21-1		sfa_world_map[180][360]	ss	n/a	2	360
L1-AUX22-1		sfa_degree[.][.]	sl	n/a	4	100
L1-AUX23-1		sfa_tenth[.]	flags	n/a	8	14

Table 5-21-1: Input Data Table - Determine Land/Sea Flag

Parameter ID	Variable	Name	Type	Units	Field size	Fields
local	resolution	resolution indicator (= 1, 2 or 3)	sl	none	4	1
local	long_offs_deg	longitude offset in degrees	float	degrees	4	1
local	i	Index to image scans (grid rows)	sl	none	4	1
local	j	index to across-track pixels (j = 0, 511)	sl	none	41	
local	lat	latitude parameter	float	degrees	4	1
local	long	longitude parameter	float	degrees	4	1
L1B-INT-160		image_latitude(i, j)	float	degrees	4	j = 0, 511
L1B-INT-161		image_longitude(i, j)	float	degrees	4	j = 0, 511
L1B-INT-232		nadir_land_flag(i, j)	flag	n/a	2	j = 0, 511
L1B-INT-248		frwrd_ land_flag(i, j)	flag	n/a	2	j = 0, 511

Table 5-21-2: Internal Data Table - Determine Land/Sea Flag

5.21.3 Algorithm Definition

Step 5.21.1 Algorithm to flag all pixels within a scan

Set long_offs_deg = float(long_offset)/1000000.0 Then:

(1) For each pixel *j* within an image scan *i*:

(1.1) Is flag long_offset equal to zero?

(1.1a) Yes. Extract lat = image_latitude(i, j),
long = image_longitude(i, j)

(1.1a) No. Extract lat = image_latitude(i, j),
long = image_longitude(i, j) - long_offs_deg

(1.2) Set required resolution (resolution = 3).

(1.3) Enter surface flag algorithm (below) with above parameters lat, long.

(1.4) Inspect surface flag returned by algorithm;

(1.4a) Land. Set nadir land flag and forward land flag TRUE.

(1.4b) Sea. Set nadir land flag and forward land flag FALSE.

(Req 5.21-1)

Step 5.21.2 Surface Flag Algorithm for single pixel

(1) Enter with pixel latitude, pixel longitude, and resolution required.

(2) First entry to algorithm?
(2a) Yes. Open map files and read in 1° resolution map.
(2b) No. Step 3.

(3) Convert latitude to range 0° to 180° and longitude to range 0° to 360°.

(4) Extract map entry (int(lat), int(long))
(4a) Land: Exit returning TRUE.

(4b) Sea: Exit returning FALSE.
(4c) Mixed surface (coast): Step 5.

(5) Entry is a reference to the 'tenths' file. If not already in memory. read in the required array of 6 arc minute entries.

(6) Derive tenths digit of latitude and longitude.

(7) Extract required entry from 'tenths' file:
(7a) Land: Exit returning TRUE.
(7b) Sea: Exit returning FALSE.
(7c) Mixed surface (coast): Step 8.

(8) Entry is a reference to the 'hundredths' file. If not already in memory, read in the required array of 0.6 arc minute entries.

(9) Derive hundredths digit of latitude and longitude

(10) extract required entry from 'hundredths' file
(10a) Land: Exit returning TRUE.
(10b) Sea: Exit returning FALSE.

(Req 5.21-2)

5.22 Module Definition: Cloud Clearing

5.22.1 Functional Description

The process of cloud-clearing, or the identification of cloud affected pixels, is accomplished by applying in turn a series of tests to the brightness temperature data in the 12, 11 and 3.7 micron channels, and to the reflectance data in the 1.6 micron channel. The pixel is flagged as cloudy if any one of the tests indicates the presence of cloud. Table 5.22.1-1 below summarises the cloud clearing tests to be applied. All of the tests are of course conditional on the appropriate infra-red or 1.6 micron data being available.

Table 5.22.1-1. Cloud Clearing Tests

Test	Comments
1.6 micron histogram test	applied to nadir and forward views separately
11 micron spatial coherence test	applied to nadir and forward views separately
gross cloud test	applied to nadir and forward views separately
thin cirrus test	applied to nadir and forward views separately
medium/high level cloud test	applied to nadir and forward views separately
fog/low stratus test	applied to nadir and forward views separately
11/12 micron nadir/forward test	uses both views
11/3.7 micron nadir/forward test	uses both views
infra-red histogram test	applied to nadir and forward views separately

Some of the tests depend on the results from the tests performed previously and hence the order in which they are applied is important. The infrared histogram test is always applied last, and only uses those pixels that have not been flagged as cloudy by any of the preceding tests. The 1.6 micron test operates only on pixels not previously flagged as cloudy by the gross cloud test or the thin cirrus and 11 micron spatial coherence tests (the other single view tests that operate on daytime data), and must therefore follow these. The order in which we shall describe the tests will reflect order in which they are performed.

The 1.6 micron test operates on daytime data only. The tests involving the 3.7 micron channel, on the other hand, are only applied to night-time data, because reflected solar radiation may be significant in this channel during the day. Those tests that involve the 11 and 12 micron channels are applicable to both daytime and night-time data. Not all of the tests are implemented over land.

A series of cloud state flags is defined for each pixel and for the forward and nadir view separately. The flags are listed in Table 5.22.1-2.

Table 5.22.1-2: Cloud-clearing/land flagging flags (nadir or forward view).

#	Meaning if set
0	Pixel is over land
1	Pixel is cloudy (result of all cloud tests)
2	Sunglint detected in pixel
3	1.6 micron reflectance histogram test shows pixel cloudy (day-time only)
4	1.6 micron spatial coherence test shows pixel cloudy (day-time only)
5	11 micron spatial coherence test shows pixel cloudy
6	12 micron gross cloud test shows pixel cloudy
7	11/12 micron thin cirrus test shows pixel cloudy
8	3.7/12 micron medium/high level test shows pixel cloudy (night-time only)
9	11/3.7 micron fog/low stratus test shows pixel cloudy (night-time only)
10	11/12 micron view-difference test shows pixel cloudy
11	3.7/11 micron view-difference test shows pixel cloudy(night-time only)
12	11/12 micron thermal histogram test shows pixel cloudy

These flags are set according to the results of the tests. Thus if one of the flags numbered 3 to 12 is set, this means that the corresponding test has indicated the presence of cloud. If on completion of the cloud-clearing sequence any of these flags is not set, it may mean either that the test did not indicate the presence of cloud, or that the test was not applied because suitable data was lacking.

Each test makes use of a look-up table of parameters with which the brightness temperature or reflectance data is compared. Where tests are applied to forward and nadir view images separately, the parameters may be defined separately for the two cases. More generally, the comparison parameters may depend on the air mass in the line of sight, and this is implemented by allowing the tabular parameters to depend upon the across-track band. For the purposes of cloud clearing and Level 2 processing, the (512 km) ATSR swath is imagined as divided into 10 bands parallel to the satellite ground track. These bands are numbered from 0 to 9, and each is 50 km wide except that the outer two are 56 km wide. The across-track band into which a pixel falls is determined from its x co-ordinate by the following equation:

$$band\_no = abs(x + 250)/50 \qquad (5.6.1)$$

unless this equation would give a result of 10, in which case the band number of 9 is used; here x is the x co-ordinate of the pixel ($-256 < x < 256$).

5.22.2 Interface Definition

The following table details the look-up table data used by the cloud clearing algorithms:

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1-AUX7-1		v16_histogram_spread[nadir]	float	%	4	1
L1-AUX7-2		v16_histogram_spread[forward]	float	%	4	1
L1-AUX7-3		v16_histogram_peak_count[nadir]	ss	none	2	1

L1-AUX7-4		v16_histogram_peak_count[forward]	ss	none	2	1
L1-AUX7-5		min_for_v16_histogram	ss	n/a	2	1
L1-AUX7-6		tilt_threshold	float	degrees	4	1
L1-AUX7-7		threshold_3	float	none	4	1
L1-AUX7-8		near_glint_range	float	degrees	4	1
L1-AUX7-9		min_for_passed	ss	n/a	2	1
L1-AUX7-10		(deleted)				0
L1-AUX7-11		search_range_for_peak	float	%	4	1
L1-AUX7-12		tilt_weight_limit	float	degrees	4	1
L1-AUX7-13		range_weight_limit	float	degrees	4	1
L1-AUX7-14		tilt_weight_factor	float	n/a	4	1
L1-AUX7-15		min_peak_value	ss	n/a	2	1
L1-AUX7-16		min_for_detrend	ss	n/a	2	1
L1-AUX7-17		max_glint_threshold	float	%	4	1
L1-AUX8-1		sea_max_dev	ss	K/100	2	1
L1-AUX8-2		land_day_max_dev	ss	K/100	2	1
L1-AUX8-3		land_night_max_dev	ss	K/100	2	1
L1-AUX8-4		coherence_reset_thresh	ss	K/100	2	1
L1-AUX8-5		coh_area_size	ss	Km	2	1
L1-AUX8-6		coh_fraction_passed	float	n/a	4	1
L1-AUX8-7		coh_adj_thresh_land	ss	K/100	2	1
L1-AUX8-8		coh_adj_dif_land	float	n/a	4	1
L1-AUX8-9		coh_area_dif_nv	ss	K/100	2	1
L1-AUX8-10		coh_area_dif_fv	ss	K/100	2	1
L1-AUX8-11		coh_min_dif_nv	ss	K/100	2	1
L1-AUX8-12		coh_min_dif_fv	ss	K/100	2	1
L1-AUX8-13		coh_area_thresh_nv	ss	K/100	2	1
L1-AUX8-14		coh_area_thresh_fv	ss	K/100	2	1
L1-AUX8-15		cloudy_box_thresh (Dependency test flag)	ss	n/a	2	1
L1-AUX9-1		12 micron threshold nadir (For each latitude index i =0, 179 and for each month j = 0, 11)	ss	K/100	2	2160
L1-AUX9-2		12 micron threshold forward (For each latitude index i =0, 179 and for each month j = 0, 11)	ss	K/100	2	2160
L1-AUX10-1		nadir_threshold[i][j] (i = 0, 60, j = 5, 9)	ss	K/100	2	305
L1-AUX10-2		frwrd_threshold[i][j] (i = 0, 60, j = 5, 9)	ss	K/100	2	305
L1-AUX11-1		med_high_level_threshold[i][nadir] (i = 0, 120)	ss	K/100	2	121
L1-AUX11-2		med_high_level_threshold[i][forward] (i = 0, 120)	ss	K/100	2	121
L1-AUX12-1		fog_threshold[j][nadir] (j = 5,... 9)	ss	K/100	2	5
L1-AUX12-2		fog_threshold[j][forward] (j = 5,... 9)	ss	K/100	2	5
L1-AUX13-1		view_diff_slope[j] (j = 5,... 9)	float	none	4	5
L1-AUX13-2		view_diff_offset[j] (j = 5,... 9)	float	none	4	5
L1-AUX13-3		ir11_ir12_view_diff_thresh	ss	K/100	2	1
L1-AUX14-1		constant coefficient a0[j] (j = 5,...9)	real	none	4	5
L1-AUX14-2		linear coefficient a1[j] (j = 5,...9)	real	none	4	5
L1-AUX14-3		quadratic coefficient a2[j] (j = 5,...9)	real	none	4	5
L1-AUX14-4		ir37_ir11_view_diff_thresh	integer	K/100	2	1
L1-AUX15-1		min_for_ir11_ir12_histogram	ss	TBD	2	1
L1-AUX15-5		peak_frac_min	float	TBD	4	1
L1-AUX15-6		latitude_threshold	float	TBD	4	1
L1-AUX15-9		second_low_fraction	float	TBD	4	1
L1-AUX15-10		half_width_m_nv	float	TBD	4	1
L1-AUX15-11		half_width_b_nv	float	TBD	4	1
L1-AUX15-12		half_width_m_fv	float	TBD	4	1
L1-AUX15-13		half_width_b_fv	float	TBD	4	1
L1-AUX15-14		max_dif_ave_chan_1	float	TBD	4	1
L1-AUX15-15		max_dif_peak_chan_1	float	TBD	4	1
L1-AUX15-16		ratio_b	float	TBD	4	1
L1-AUX15-17		ir_spread_nv	float	TBD	4	1
L1-AUX15-18		ir_spread_fv	float	TBD	4	1
L1-AUX15-19		slope_max_allowed	float	TBD	4	1

L1-AUX15-20		ir_peak_min	float	TBD	4	1
L1-AUX29-1		12 micron land threshold nadir (For each latitude index i =0, 179 and for each month j = 0, 11)	ss	K/100	2	2160
L1-AUX29-2		12 micron land threshold forward (For each latitude index i =0, 179 and for each month j = 0, 11)	ss	K/100	2	2160
L1-AUX30-1	N_VERT	Total number of zone vertices	ss	none	2	1
L1-AUX30-2	N_ZONES	Number of defined zones	ss	none	2	1
L1-AUX31-1	X[j]	X co-ordinate of vertex *j* (*j* = 0, N_VERT-1)	float	none	4	N_VERT
L1-AUX31-1	Y[j]	Y co-ordinate of vertex *j* (*j* = 0, N_VERT-1)	float	none	4	N_VERT
L1-AUX32-2	v[k, i_zone]	Array of vertex identifiers,. (i_zone = 0, N_ZONE - 1	float	none	4	k = 0, 4
L1-AUX33-1	NDSI_THRESH	NDSI threshold	ss	0.0001	2	1
L1-AUX33-2	R87_THRESH	0.87 micron reflectance threshold	ss	0.01 %	2	1
L1-AUX33-3	T11_THRESH	11 micron BT threshold	ss	0.01 K	2	1

Table 5-22-1: Input Data Table - Cloud Clearing

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-030	mjdp2[0]	ascending node time	double	days	8	1
L1B-INT-101	I(ir12, n; i, j)	regridded nadir ir12 Brightness Temp.	ss	0.01 K	2	j = 0, 511
L1B-INT-102	I(ir11, n; i, j)	regridded nadir ir11 Brightness Temp.	ss	0.01 K	2	j = 0, 511
L1B-INT-103	I(ir37, n; i, j)	regridded nadir ir37 Brightness Temp.	ss	0.01 K	2	j = 0, 511
L1B-INT-104	I(v16, n; i, j)	regridded nadir v16 Reflectance	ss	0.01%	2	j = 0, 511
L1B-INT-105	I(v870, n; i, j)	regridded nadir v870 Reflectance	ss	%/100	2	j = 0, 511
L1B-INT-106	I(v670, n; i, j)	regridded nadir v670 Reflectance	ss	%/100	2	j = 0, 511
L1B-INT-107	I(v555, n; i, j)	regridded nadir v555 Reflectance	ss	%/100	2	j = 0, 511
L1B-INT-111	I(ir12, f; i, j)	regridded forward ir12 Brightness Temp.	ss	0.01 K	2	j = 0, 511
L1B-INT-112	I(ir11, f; i, j)	regridded forward ir11 Brightness Temp.	ss	0.01 K	2	j = 0, 511
L1B-INT-113	I(ir37, f; i, j)	regridded forward ir37 Brightness Temp.	ss	0.01 K	2	j = 0, 511
L1B-INT-114	I(v16, f; i, j)	regridded forward v16 Reflectance	ss	0.01%	2	j = 0, 511
L1B-INT-115	I(v870, f; i, j)	regridded forward v870 Reflectance	ss	%/100	2	j = 0, 511
L1B-INT-116	I(v670, f; i, j)	regridded forward v670 Reflectance	ss	%/100	2	j = 0, 511
L1B-INT-117	I(v555, f; i, j)	regridded forward v555 Reflectance	ss	%/100	2	j = 0, 511
L1B-INT-100		nadir_fill_state(i, j)	byte	none	1	j = 0, 511
L1B-INT-110		frwrd_fill_state(i, j)	byte	none	1	j = 0, 511
L1B-INT-160		image_latitude(i, j)	float	degrees	4	j = 0, 511
L1B-INT-161		image_longitude(i, j)	float	degrees	4	j = 0, 511
L1B-INT-120		nadir_band_edge_solar_elevation(i, k)	float	degrees	4	k = 0, 10
L1B-INT-121		nadir_band_edge_satellite_elevation(i, k)	float	degrees	4	k = 0, 10
L1B-INT-122		nadir_band_edge_solar_azimuth(i, k)	float	degrees	4	k = 0, 10
L1B-INT-123		nadir_band_edge_satellite_azimuth(i, k)	float	degrees	4	k = 0, 10
L1B-INT-124		nadir.band_centre_solar_elevation(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-125		nadir.band_centre_satellite_elevation(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-126		nadir.band_centre_solar_azimuth(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-127		nadir.band_centre_satellite_azimuth(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-140		frwrd_band_edge_solar_elevation(i, k)	float	degrees	4	k = 0, 10
L1B-INT-141		frwrd_band_edge_satellite_elevation(i, k)	float	degrees	4	k = 0, 10
L1B-INT-142		frwrd_band_edge_solar_azimuth(i, k)	float	degrees	4	k = 0, 10
L1B-INT-143		frwrd_band_edge_satellite_azimuth(i, k)	float	degrees	4	k = 0, 10
L1B-INT-144		frwrd.band_centre_solar_elevation(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-145		frwrd.band_centre_satellite_elevation(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-146		frwrd.band_centre_solar_azimuth(i, k′)	float	degrees	4	k′ = 0, 9
L1B-INT-147		frwrd.band_centre_satellite_azimuth(i, k′)	float	degrees	4	k′ = 0, 9
	band(j)	across track band number	integer	none	2	512
		ir11_coherence_result[171][171]	integer		2	
		image_month	integer		2	
		nadir_cloud_state[512][512]	struct			

		Cloud state flags are:				
L1B-INT-232		land	flag	n/a	2	
L1B-INT-233		cloud	flag	n/a	2	
L1B-INT-234		sunglint	flag	n/a	2	
L1B-INT-235		v16 histogram test	flag	n/a	2	
L1B-INT-236		v16 spatial coherence test	flag	n/a	2	
L1B-INT-237		ir11 spatial coherence test	flag	n/a	2	
L1B-INT-238		ir12 gross cloud test	flag	n/a	2	
L1B-INT-239		ir11 ir12 thin cirrus test	flag	n/a	2	
L1B-INT-240		ir37 ir12 med high level test	flag	n/a	2	
L1B-INT-241		ir11 ir37 fog low stratus test	flag	n/a	2	
L1B-INT-242		ir11 ir12 view diff test	flag	n/a	2	
L1B-INT-243		ir37 ir11 view diff test	flag	n/a	2	
L1B-INT-244		ir11 ir12 histogram test	flag	n/a	2	
L1B-INT-245		Visible channel cloud test	flag	n/a	2	
L1B-INT-246		Snow flag	flag	n/a	2	
		frwrd_cloud_state[512][512]	struct			
		Cloud state flags are:				
L1B-INT-248		land	flag	n/a	2	
L1B-INT-249		cloud	flag	n/a	2	
L1B-INT-250		sunglint	flag	n/a	2	
L1B-INT-251		v16 histogram test	flag	n/a	2	
L1B-INT-252		v16 spatial coherence test	flag	n/a	2	
L1B-INT-253		ir11 spatial coherence test	flag	n/a	2	
L1B-INT-254		ir12 gross cloud test	flag	n/a	2	
L1B-INT-255		ir11 ir12 thin cirrus test	flag	n/a	2	
L1B-INT-256		ir37 ir12 med high level test	flag	n/a	2	
L1B-INT-257		ir11 ir37 fog low stratus test	flag	n/a	2	
L1B-INT-258		ir11 ir12 view diff test	flag	n/a	2	
L1B-INT-259		ir37 ir11 view diff test	flag	n/a	2	
L1B-INT-260		ir11 ir12 histogram test	flag	n/a	2	
L1B-INT-261		Visible channel cloud test	flag	n/a	2	
L1B-INT-262		Snow flag	flag	n/a	2	
L1B-INT-261		x_index	int	n/a	2	
L1B-INT-262		y_index	int	n/a	2	
L1B-INT-263		solar_elevation	float	deg	4	
L1B-INT-264		ir11[3][3]	int	K/100	2	
L1B-INT-265		group_land_flag[171][171]	flag	n/a	2	
L1B-INT-266		average_11	double	K/100	8	
L1B-INT-267		sigma_11	double	K/100	8	
L1B-INT-268		j	int	n/a	2	
L1B-INT-269		n	int	n/a	2	
L1B-INT-270		average_11_array[171][171]	int	K/100	2	
L1B-INT-271		threshold_sd	int	K/100	2	
L1B-INT-272		group_cloud_flag[171][171]	flag	n/a	2	
L1B-INT-273		average_11_12_dif_cloudy	double	K/100	8	
L1B-INT-274		average_11_12_dif_clear	double	K/100	8	
L1B-INT-275		n_cloudy (number of cloudy pixels in group)	ss	none	2	
L1B-INT-276		valid_pixel_pairs	ss	none	2	
L1B-INT-277		previous_tests[171][171]	flag	n/a	2	
L1B-INT-279		extended_land_flag[171][171]	flag	n/a	2	
L1B-INT-280		sub_area_n	int	Km	2	
L1B-INT-281		sub_area_max_11[sub_area_n][sub_area_n]	int	K/a	2	
L1B-INT-282		sub_area_use_11[sub_area_n][sub_area_n]	int	K/100	2	
L1B-INT-283		sub_area_dif[sub_area_n][sub_area_n]	int	K/100	2	
L1B-INT-284		land_sub_area[sub_area_n][sub_area_n]	flag	n/a	2	
L1B-INT-287		x_index_1	int	n/a	2	
L1B-INT-288		y_index_1	int	n/a	2	
L1B-INT-289		max_bt_11	int	K/100	2	

L1B-INT-291		threshold_11	int	K/100	2	
L1B-INT-293		land_in_areas	flag	n/a	2	
L1B-INT-294		bt_dif_max	int	K/100	2	
L1B-INT-295		difference_threshold	int	K/100	2	
L1B-INT-297		lowest_max_bt	int	K/100	2	
L1B-INT-298		valid_sub_area_flag[sub_area_n][sub_area_n]	flag	n/a	2	
L1B-INT-301		ir16[32][32]	int	%/100	2	
L1B-INT-302		x[32][32]	int	n/a	2	
L1B-INT-303		index_valid[1024]	int	n/a	2	
L1B-INT-304		total_valid	int	n/a	2	
L1B-INT-305		sol_elev	float	degree	4	
L1B-INT-306		sat_elev	float	degree	4	
L1B-INT-307		azim_dif	float	degree	4	
L1B-INT-308		v_x	float	n/a	4	
L1B-INT-309		v_y	float	n/a	4	
L1B-INT-310		v_z	float	n/a	4	
L1B-INT-311		tilt	float	degree	4	
L1B-INT-312		magnitude	float	n/a	4	
L1B-INT-313		glint_present	flag	n/a	2	
L1B-INT-314		histogram_16[1000]	int	n/a	2	
L1B-INT-315		low_interval	float	%	4	
L1B-INT-316		high_interval	float	%	4	
L1B-INT-317		hist_range	float	%	4	
L1B-INT-318		peak_box_no	int	n/a	2	
L1B-INT-319		peak_value	int	n/a	2	
L1B-INT-320		average_value	float	n/a	4	
L1B-INT-321		peak_interval	float	%	4	
L1B-INT-322		D	float	n/a	4	
L1B-INT-323		delta	float	n/a	4	
L1B-INT-324		spread_adjusted	float	%	4	
L1B-INT-325		f	float	n/a	4	
L1B-INT-326		g	float	n/a	4	
L1B-INT-327		spread	float	%	4	
L1B-INT-328		peak_factor	float	%	4	
L1B-INT-329		reflectance_threshold	float	%	4	
L1B-INT-330		n	float	%	4	
L1B-INT-331		gradient_16	float	%/Km/100	4	
L1B-INT-332		a	float	n/a	4	
L1B-INT-333		b	float	n/a	4	
L1B-INT-334		i	int	n/a	2	
L1B-INT-335		detrended_16[32][32]	int	%/100	2	
L1B-INT-336		sd_threshold	float	%/100	4	
L1B-INT-337		x_y_12um_max[2]	int	n/a	2	
L1B-INT-338		reflectance_at_12um_max	float	%	4	
L1B-INT-340		scan_loop	int	2	1	
L1B-INT-341		pixel_loop	int	2	1	
L1B-INT-342		valid_pixels	int	2	1	
L1B-INT-343		histogram[2][1000]	int	2	2000	
L1B-INT-344		bin_size	int	2	1	
L1B-INT-345		MAJOR	constant	2	1	
L1B-INT-346		MINOR	constant	2	1	
L1B-INT-347		CHOSEN	int	2	1	
L1B-INT-348		ir11_ir12_diff	int	K/100	2	
L1B-INT-349		binned_diff	int	K/10	2	
L1B-INT-351		ir12boxtotal[1000]	int	K/100	4	
L1B-INT-352		hist_index	int	2	1	
L1B-INT-353		peak_value[2]	int	2	2	
L1B-INT-354		peak_interval[2]	int	K/10	2	
L1B-INT-355		exact_peak_value[2]	float	4	2	
L1B-INT-356		exact_peak_interval[2]	float	K/10	4	

L1B-INT-357		ir11_ir12_diff_at_peak[2]	float	K/100	4	
L1B-INT-358		lower_limit	int	K/10	2	
L1B-INT-359		higher_limit	int	K/10	2	
L1B-INT-360		low_limit[2]	int	K/10	2	
L1B-INT-361		high_limit[2]	int	K/10	2	
L1B-INT-362		average_bt_mode[2]	float	K/100	4	
L1B-INT-363		average_bt[1000]	float	K/100	4	
L1B-INT-364		highest_av_bt_box[2]	int	2	2	
L1B-INT-365		slope_at_peak	float	4	1	
L1B-INT-366		slope[1000]	float	4	1000	
L1B-INT-367		half_width_threshold[2]	float	K/100	4	
L1B-INT-368		half_width[2]	float	K/100	4	
L1B-INT-369		upperhalfwidth_index	float	K/10	4	
L1B-INT-370		lowerhalfwidth_index	float	K/10	4	
L1B-INT-371		peak_valid[2]	flag	2	2	
L1B-INT-372		nightime	flag	2	1	

Table 5-22-2: Internal Data Table - Cloud Clearing

5.22.3 Algorithm Definition

Each of the following tests is applied independently to the regridded image, and therefore comprises a distinct processing module.

Note that the overall implementation is based on the notion that for each pixel in each view, there is a series of flags (cloud_flag, [specific_test]_flag) that are initialised to false, and then set as required as the tests are carried out.

A valid pixel is one which is not represented by a negative (exceptional) value.

The tests should be applied in the order stated except that the 11 micron spatial coherence test on the nadir and forward views (Section 5.22.3.1) should follow the fog/low stratus test (Section 5.22.3.5). That is to say, Steps 5.22.3-1 to 5.22.3-13 inclusive (Reqs. 5.22-1 to 5.22-27) should follow Step 5.22.3-24. The 11 micron spatial coherence test includes a dependency on the previous tests, such that the tests represented by Steps 5.22.3-14 to 5.22.3-24 should have been applied to all pixels in the image segment before the 11 micron spatial coherence test is attempted.

While six of the tests operate on each pixel independently, three of the tests operate on larger image areas. The 11 micron spatial coherence test (Section 5.22.3.1), and the infrared histogram test (Section 5.22.3.9) operate on image segments of 512 by 512 pixels, and the 1.6 micron histogram test (Section 5.22.3.6) operates on areas of 32 by 32 pixels. Like the 11 micron spatial coherence test, the two histogram tests also include dependencies on the results of previous tests, and so previous tests should have been applied to all pixels in the relevant image area before these tests are attempted.

In the case of those tests that operate on the nadir and forward images separately, each test should be applied to the nadir and the forward image before proceeding to the next test.

5.22.3.1 Spatial coherence test (11 micron)

The test uses a look-up table containing the following parameters:

SEA_MAX_DEV (L1-AUX8-1)

LAND_DAY_MAX_DEV (L1-AUX8-2)

LAND_NIGHT_MAX_DEV (L1-AUX8-3)

COHERENCE_RESET_THRESH (L1-AUX8-4)

COH_AREA_SIZE (L1-AUX8-5)

COH_FRACTION_PASSED (L1-AUX8-6)

COH_ADJ_THRESH_LAND (L1-AUX8-7)

COH_ADJ_DIF_LAND (L1-AUX8-8)

(Req 5.22-1)

These are not view dependent. The view dependent parameters are:

COH_AREA_DIF (L1-AUX8-9 in the nadir view; L1-AUX8-10 in the forward view)

COH_MIN_DIF (L1-AUX8-11 in the nadir view; L1-AUX8-12 in the forward view)

COH_AREA_THRESH (L1-AUX8-13 in the nadir view; L1-AUX8-14 in the forward view)

(Req 5.22-2)

The test is performed separately for the nadir and forward views. The nadir view 12µm and 11µm data (L1B-INT-101 and L1B-INT-102) are used in the nadir view test and the forward nadir view 12µm and 11µm data (L1B-INT-111 and L1B-INT-112) are used in the forward view test. In the following, it is assumed that the along-track size of a view is 512 Km. That is, the orbit is imagined as tiled with 512 by 512 pixel images, each of which is treated separately. If an incomplete image is encountered at the end of data, the image should be filled up to 512 by 512 pixels with unfilled pixels.

(Req 5.22-3)

The small scale spatial coherence test uses a square array of 3 x 3 pixels and is described below in steps a to g. The groups are identified by their x_index and y_index (L1B-INT-261, L1B-INT-262), and the pixels used in testing a group are in pixel positions - with respect to the 256th across-track pixel on the first line in view - (x_index*3-256:x_index*3-254, y_index*3:y_index*3+2) The groups with x_index = 170 are at across-track pixel positions 253:255 and those with y_index = 170 at along-track pixel positions of 509:511, so that these also contain 9 pixels.

(Req 5.22-4)

The large scale spatial coherence test uses data from sub-areas of COH_AREA_SIZE x COH_AREA_SIZE and is described in steps h to l. (N.B. The value of COH_AREA_SIZE must be chosen so that it is one of the factors of 512. Its current value is 128 Km.) The sub areas are, similarly to the 3x3 groups above, non-overlapping and congruent.

(Req 5.22-5)

For each group do Steps 5.22.3-1 to 5.22.3-5 inclusive:

Step 5.22.3-1

a) Determine solar_elevation (L1B-INT-263) at the centre of the across-track band which contains the central pixel of the group. The band is given by equation 5.6.1 where x is the across-track distance of the centre pixel in the group.

Transfer the 11µm for the group being investigated into the 3 x 3 element array ir11 (L1B-INT-264)

(Req 5.22-6)

Step 5.22.3-2

b) Determine whether the pixels in group are over land or over sea, using the appropriate land flag (L1B-INT-232 in the nadir view and L1B-INT-248 in the forward view) for pixels over land.

If one or more pixels in a group are over land then set the group_land_flag (L1B-INT-265) for the group and store for later use.

If there are 2 or fewer valid pixels, set the flag for the group in group_cloud_flag [L1B-INT-272] and proceed to the next group. A valid pixel has the 11 micron brightness temperature ([L1B-INT-102] or [L1B-INT-112]) > 0 and <view>_fill_state = NATURAL_PIXEL as in step (c) below.

(Req 5.22-7)

Step 5.22.3-3

c) Calculate the average and the standard deviation of the 11µm brightness temperatures for the valid pixels in the group, excluding cosmetic fill pixels, as follows. Filled pixels are identified by

<view>*_fill_state(i, j) = NATURAL_PIXEL.*

average_11 = (1/n) * Σ *I(ir11, v; i, j)*

sigma_11= sqrt((1/(n-1)) * Σ(*I(ir11, v; i, j)*-average_11)^2)

where where the sums are over the indices (*i, j*) is the index of the pixels in the group that satisfy the selection criteria above, and n their total number.

Store the average_11 in average_11_array (L1B-INT-270) for later use.

(Req 5.22-8)

Step 5.22.3-4

d) Select the appropriate threshold; if the group is over sea set threshold_sd (L1B-INT-271) to SEA_MAX_DEV. If the group is over land (i.e. if group_land_flag is set), inspect the solar elevation angle determined in Step a. (Note that cells of mixed surface type are treated here as land cells.)

If
solar_elevation > 5°

then set threshold_sd to the day-time value LAND_DAY_MAX_DEV; otherwise set threshold_sd to the night-time value LAND_NIGHT_MAX_DEV.

(Req 5.22-9)

Step 5.22.3-5

e) If
sigma_11 > threshold_sd

then set flag for the group in group_cloud_flag (L1B-INT-272) and proceed to the next group.

(Req 5.22-10)

Step 5.22.3-6

f) When all the groups in a view have been processed investigate whether the spatial variation may result from temperature gradients on the surface rather than from cloud. For each group flagged as cloudy (i.e. when group_cloud_flag has been set) in step e, do the following:

f.1) If four or more of the up to 8 surrounding groups are clear (i.e. group_cloud_flag not set) and the group contains more than two valid pixels (as at step (b) above), then proceed to Step f.2; otherwise move to the next flagged group.

(Req 5.22-11)

f.2) If the central group is open sea (i.e. group_land_flag not set) and if it is not a border group (having x_index = 0 or 170, y_index = 0 or 170), calculate the average difference between the 11 and 12 micron channel brightness temperatures for the central (cloudy) group (average_11_12_dif_cloudy: L1B-INT-273) and for those of the surrounding groups that have not been flagged as cloudy (average_11_12_dif_clear: L1B-INT-274)

Specifically,

$$average\_11\_12\_dif\_cloudy = \frac{1}{N_{cloud}} \sum_{i,j} (I(ir11, v; i, j) - I(ir12, v; i, j))$$

where the pixels indexed by *i* and *j* fall within the central cloudy group, are *NATURAL* pixels, and are valid in both channels: $I(ir11, v; i, j) > 0$ and $I(ir12, v; i, j) > 0$. N_{cloud} is the total number of pixels contributing to the sum. Similarly

$$average\_11\_12\_dif\_clear = \frac{1}{N_{clear}} \sum_{i,j} (I(ir11, v; i, j) - I(ir12, v; i, j))$$

where the pixels indexed by *i* and *j* fall within the clear groups, are *NATURAL* pixels, and are valid in both channels: $I(ir11, v; i, j) > 0$ and $I(ir12, v; i, j) > 0$. N_{clear} is the total number of pixels contributing to the sum.

If

abs(average_11_12_dif_cloudy - average_11_12_dif_clear) < COHERENCE_RESET_THRESH,

then unflag the central group by setting to zero the appropriate element in group_cloud_flag.

(Req 5.22-12)

Step 5.22.3-7

g) Flag all pixels in the cloud-flagged groups (i.e. where group_cloud_flag is set) for which the land flag (L1B-INT-232 in the nadir view and L1B-INT-248 in the forward view) is not set) as cloudy by setting L1B-INT-233 and L1B-INT-237 in the nadir view and L1B-INT-249 and L1B-INT-253 in the forward view.

(Req 5.22-13)

[Note: the objective of the modification above is to effectively disable the spatial coherence test over land without introducing inadvertent side-effects. The cloud flags for each pixel are to be set to group_cloud_flag AND (NOT land). Of course for a given regridded pixel the nadir and forward view land flags L1B-INT-232 and L1B-INT-248 should be the same.]

Step 5.22.3-8

h) By using group_land_flag (L1B-INT-265), loop through all the groups in view and flag groups within approximately 6 km of land, as well as all-land groups. A group's flag extended_land_flag(x_index, y_index) in (L1B-INT-279) is set if

$$\Sigma\Sigma \text{group\_land\_flag}(i, j) > 0$$

where the summations are for indices in the ranges of

$$\text{x\_index-3} < i < \text{x\_index} + 3 \text{ and}$$

$$\text{y\_index-3} < j < \text{y\_index} + 3$$

i and j must also be in the range of 0 to 170.

(Req 5.22-14)

Step 5.22.3-9

i) Create arrays arrays used in the large-scale coherence test.

i.1) Determine the number of sub-arrays across the view, sub_area_ (L1B-INT-280), given by

$$\text{sub\_area\_n} = 512 / \text{COH\_AREA\_SIZE}$$

(Req 5.22-15)

i.2) Create arrays L1B-INT-281 to 284 of size [sub_area_n][sub_area_n], as specified in Table 5-22-2.

(Req 5.22-16)

(The image segment is divided into sub_area_n by sub_area_n sub-areas; thus if the adopted value of COH_AREA_SIZE is 128 there are 16 sub-areas.)

Step 5.22.3-10

j) For all sub-areas, find the group(s) with the highest average 11μm brightness temperature, and the average (11μm - 12μm) brightness temperature difference(s) of these:

j.1) Select sub-areas for all combinations of x_index_1, y_index_1 (L1B-INT-287 and L1B-INT-288). The ranges of x_index_1 and y_index_1 are 0 to (sub_area_n - 1).

The groups for sub-area(x_index_1, y_index_1) are the array elements average(ix1:ix2, iy1:iy2) where

$$ix1 = \text{INT}((\text{x\_index\_1}*\text{coh\_area\_size})/3)$$

$$ix2 = \text{INT}(((\text{x\_index\_1} + 1)*\text{coh\_area\_size})/3) - 1$$

$$iy1 = \text{INT}((\text{y\_index\_1}*\text{coh\_area\_size})/3)$$

$$iy2 = \text{INT}(((\text{y\_index\_1} + 1)*\text{coh\_area\_size})/3) - 1$$

Step (j.1) associates pixel groups with the sub-areas. The set of groups that corresponds to the sub-area indexed by x_index_1, y_index_1 comprises those groups whose indices x_index, y_index satisfy (x_index, y_index) $\in$ *A*(x_index_1, y_index_1), where

the set of index pairs *A*(x_index_1, y_index_1) is formally defined by

A(x_index_1, y_index_1) = {(x_index, y_index): ix1 ≤ x_index ≤ ix2;
iy1 ≤ y_index ≤ iy2}

Since the number of groups is not a multiple of the sub_area_n, the sets *A*(x_index_1, y_index_1) do not each contain the same number of groups.

For all sub-areas,

j.2) find the highest average 11μm brightness temperature (max_bt_11: L1B-INT-289) and its/their coordinates. Use valid groups only.

'valid' implies that the group satisfies the following conditions:

- its extended_land_flag is not set;

- it has passed the small-scale coherence test; that is, if group_cloud_flag is not set;

- it has not been shown to contain significant cloud by any previous test;

- it contains at least 3 natural pixels having valid 11 micron and 12 micron brightness temperatures; that is, at least 3 NATURAL pixels are available to contribute to the computation of the average (11 micron - 12 micron) brightness temperature difference at j.3) below.

Determine the flag previous_tests(x_index, y_index) as follows:

If cloudy_box_thresh [L1-AUX8-15] = -1, the value of previous_tests(x_index, y_index) is determined by inspecting only the central pixel of the group:

previous_tests(x_index, y_index) =
(<view>_cloud_state[i][j].ir12_gross_cloud_test = TRUE or
<view>_cloud_state[i][j].ir11_ir12_thin_cirrus_test = TRUE or
<view>_cloud_state[i][j].ir37_ir12_med_high_level_test = TRUE or
<view>_cloud_state[i][j].ir11_ir37_fog_low_stratus_test = TRUE)

where *i* is the lesser of (3*y_index + 1, 510) and *j* is the lesser of (3*x_index + 1, 510); *i* and *j* index the central pixel of the group.

If cloudy_box_thresh [L1-AUX8-15] > 0, the value of previous_tests(x_index, y_index) is determined by inspecting all pixels of the group:

previous_tests(x_index, y_index) = (n_cloudy ≥ cloudy_box_thresh)

where n_cloudy is the total number of pixels that fall within the group (x_index, y_index) and for which

(<view>_cloud_state[i][j].ir12_gross_cloud_test = TRUE or
<view>_cloud_state[i][j].ir11_ir12_thin_cirrus_test = TRUE or
<view>_cloud_state[i][j].ir37_ir12_med_high_level_test = TRUE or
<view>_cloud_state[i][j].ir11_ir37_fog_low_stratus_test = TRUE).

The pixels that fall within the group defined by (x_index, y_index) are those whose indices satisfy

$$i_0 \le i \le i_0 + 2,\ j_0 \le j \le j_0 + 2,$$

where i_0 is the smaller of (3*y_index + 1, 509) and j_0 is the smaller of (3*x_index + 1, 509).

Then

max_bt_11 = maximum value of average_11_array(x_index, y_index)

over the set of indices (x_index, y_index) ∈ A_{valid}(x_index_1, y_index_1)

The set A_{valid}(x_index_1, y_index_1) is the set of indices of VALID groups, and is defined by

A_{valid}(x_index_1, y_index_1) = {x_index, y_index:
(x_index, y_index) ∈ *A*(x_index_1, y_index_1)
and NOT group_cloud_flag(x_index, y_index)
and NOT extended_land(x_index, y_index)
and NOT previous_test(x_index, y_index)
and (valid_pixel_pairs ≥ 3)}

where valid_pixel_pairs is the number of pixels in the group defined by (x_index, y_index) for which *<view>_fill_state(i, j) = NATURAL_PIXEL*, and both *I(ir11, v; i, j)* > 0, *I(ir12, v; i, j)* > 0. The set of pixels in the group (x_index, y_index) is as defined above.

Then

sub_area_max_11(x_index_1, y_index_1) = max_bt_11.

It is possible that there are no valid groups; A_{valid} is an empty set. In this case sub_area_max_11(x_index_1, y_index_1) and sub_area_dif(x_index_1, y_index_1) should each be set to an exceptional value (≤ 0) to ensure that they are ignored in Step (m) below.

j.3) Calculate the average (11µm - 12µm) brightness temperature difference(s) for the groups with max_bt_11. Use valid pixels only, as in step c).

Use pixels with x coordinates in the range of

max(0, 3*x_max - 1) and min(511, 3*x_max + 1)

and y coordinates in the range of

max(0, 3*y_max - 1) and min(511, 3*y_max + 1)

If a single group had the average 11µm brightness temperature value of max_bt_11 then assign its brightness temperature difference value to sub_area_dif (L1B-INT-283)

If more than one group has the same max_bt_11 value then find the highest of differences, and assign this value to sub_area_dif.

(Req 5.22-17)

This step calculates the average difference between the 11 micron and 12 micron channels for those valid groups for which

average_11_array(x_index, y_index) = max_bt_11

Thus

sub_area_dif(x_index_1, y_index_1) = maximum value of
(11 micron - 12 micron difference) over the groups having
(x_index, y_index) $\in A_{valid}$(x_index_1, y_index_1) and
average_11_array(x_index, y_index) = max_bt_11

Step 5.22.3-11

k) Set the land_sub_area flag to 1 for all sub-areas in which one or more groups are over or near land i.e, set the flag if

Σextended_land_flag(x_index, y_index) > 0

where the sum includes all (x_index, y_index) $\in$ *A*(x_index_1, y_index_1) as defined in step j.

(Req 5.22-18)

Step 5.22.3-12

l) Set valid_sub_area_flag to 1 for all sub-areas for which the number of clear sea groups is greater than COH_FRACTION_PASSED of the total number, i.e.

Σ (1-group_cloud_flag(x_index, y_index)) * (1-extended_land_flag(x_index, y_index)) / (COH_AREA_SIZE/3)**2 > COH_FRACTION_PASSED,

where the sum includes all (x_index, y_index) $\in$ *A*(x_index_1, y_index_1),

and sub_area_dif(x_index_1, y_index_1) is greater than COH_MIN_DIF.

(Req 5.22-19)

Step 5.22.3-13

m) For invalid sub-areas, set the 11μm brightness temperature threshold threshold_11 (L1B-INT-291) to 32000 cK i.e. to an an unrealistically high value so that all the sea pixels in these sub-areas are flagged as cloudy and omit Steps (m.1) to (m.7).

For sub-areas that are valid, determine threshold_11 as follows (Steps (m.1) to (m.7)):

m.1) Select the up to 9 sub-areas (valid or not) centred on the one being investigated. These are the sub-areas whose indices fall in the set

S(p, q) = {(x_index_1, y_index_1): 0 ≤x_index_1 < sub_area_n,
0 ≤y_index_1 < sub_area_n,
p-1 ≤x_index_1 ≤p+1, q-1 ≤y_index_1 ≤q+1}

where p, q, 0 ≤p < sub_area_n, 0 ≤q < sub_area_n, are the indices of the sub-area currently under consideration.

(Req 5.22-20)

m.2) Set land_in_areas (L1B-INT-293) to 1 if one or more of the sub-areas selected has land:

land_in_areas = (Σland_sub_area(x_index_1, y_index_1)) ge 1

where the sum is over index pairs (x_index_1, y_index_1) $\in$ *S*(p, q).

(Req 5.22-21)

Step (m.2) sets the flag land_in_areas if, for any (x_index_1, y_index_1) ∈ *S*(p, q),
land_sub_area(x_index_1, y_index_1)
is set.

m.3) Find the highest sub_area_dif , bt_dif_max (L1B-INT-294), for valid selected sub-areas.

bt_dif_max = max (sub_area_dif(x_index_1, y_index_1)),

over the set of index pairs (x_index_1, y_index_1) ∈ *S*(p, q) that identify valid sub-areas.

(Req 5.22-22)

Notice that at least one sub-area (the central one) must be VALID, so this operation is always possible.

m.4) Select those sub_area_max_11 values for which the corresponding sub_area_dif is within difference_threshold (L1B-INT-295) of bt_dif_max, found in step m.3. That is,

sub_area_dif(x_index_1, y_index_1) > bt_dif_max - difference_threshold.

The value of difference_threshold is given by:-

COH_AREA_DIF*(1 + land_in_areas * COH_ADJ_DIF_LAND)

(Req 5.22-23)

m.5) Find the lowest_max_bt (L1B-INT-297) of those sub-areas selected in step m.4

lowest_max_bt = minimum of sub_area_max_11(x_index_1, y_index_1)
over the set of index pairs such that (x_index_1, y_index_1) ∈ *S*(p, q), the sub-area is valid, and sub_area_dif(x_index_1, y_index_1) satisfies the test in step m.4.

(Req 5.22-24)

m.6) Calculate threshold_11 for the sub-area investigated by decreasing lowest_max_bt by

COH_AREA_THRESH + land_in_areas*COH_ADJ_THRESH_LAND.

That is:
threshold_11 = lowest_max_bt - COH_AREA_THRESH
- land_in_areas * COH_ADJ_THRESH_LAND

(Req 5.22-25)

m.7) If only one valid sub-area was selected at step m.4, and the land_in_area flag is set, lower threshold_11 by 200 cK.

(Req 5.22-26)

m.8) Flag all sea pixels in the sub-area investigated for those groups for which average_11_array is below threshold_11 by setting L1B-INT-233 and L1B-INT-237 in the nadir view and L1B-INT-249 and L1B-INT-253 in the forward view.

(Req 5.22-27)

5.22.3.2 Gross cloud test

The test uses a look-up table with values of the threshold tabulated at intervals of 1° of latitude for each view (forward and nadir), and for each calendar month. The test uses different look-up tables for the land and sea pixels.

Step 5.22.3-14

a) Derive the month in which the data was collected from the time of the ascending node [L1B-INT-30].

`(Req 5.22-28)`

Step 5.22.3-15

b) Do the gross cloud check for each valid pixel (over both land and sea) in each view ($v = n \mid f$) as follows:

b.1) Extract the pixel latitude *image_latitude(i, j)* (in degrees), and convert to an integer *latitude_index* in the range 0 to 180 by adding 90 and taking the integer part.

`(Req 5.22-29)`

b.2) Determine threshold *gross_cloud_threshold* using the latitude index just determined and the month to enter the gross cloud check threshold array. If the pixel is a sea pixel, use the [L1-AUX9-1] threshold array in the nadir view, and the [L1-AUX9-2] threshold array in the forward view. If the pixel is a land pixel, use the [L1-AUX29-1] threshold array in the nadir view, and the [L1-AUX29-2] threshold array in the forward view.

`(Req 5.22-30)`

b.3) If the 12μm brightness temp
I(ir12, v; i, j) < gross_cloud_threshold,
flag the pixel as cloudy by setting L1B-INT-233 and L1B-INT-238 in the nadir view, and L1B-INT-249 and L1B-INT-254 in the forward view.

`(Req 5.22-31)`

5.22.3.3 Thin cirrus test

The test uses a look-up table that defines a threshold THIN_CIRRUS_THRESHOLD[temperature index][across-track band] for the nadir and forward views separately, for each across-track band = 5, 9, and for values of the brightness temperature index, defined by (T-250) K, from 0 to 60 inclusive. (Note that by symmetry the coefficients for the across-track band i = 0, 1, ...4 are the same as for band 9 - i.)

It is applied to the forward ($v = f$) and nadir ($v = n$) views separately. For each pixel:

Step 5.22.3-16

a) Determine the pixel across track band number (band_no) using the across-track co-ordinate of the pixel for x in equation 5.6.1.

`(Req 5.22-32)`

Step 5.22.3-17

b) Determine the brightness temperature index using the 11μm value I(ir11, v; i, j)

bt_index = integer part of (*I(ir11, v; i, j)* - 25000) / 100
if (bt_index > 60 then bt_index = 60
else if (bt_index < 0) then bt_index = 0
end if

(Note that brightness temperature values are stored in units of 0.01 K)

(Req 5.22-33)

Step 5.22.3-18

c) Calculate the difference

I(ir11, v; i, j) - I(ir12, v; i, j)

If this difference is greater than
THIN_CIRRUS_THRESHOLD[bt_index][band_no]
for the appropriate view, flag the pixel as cloudy by setting L1B-INT-233 and L1B-INT-239 in the nadir view and L1B-INT-249 and L1B-INT-255 in the forward view.

Use the L1-AUX10-1 threshold array in the nadir view, and the L1-AUX10-2 threshold array in the forward view.

(Req 5.22-34)

5.22.3.4 Medium/high level cloud test

The test uses a look-up table that defines a threshold MED_HIGH_LEVEL_THRESH[temperature index] for the nadir and forward views separately, for values of the brightness temperature index, defined by 2*(T-250) K, from 0 to 120 inclusive.

It is applied to the forward and nadir views separately.

Step 5.22.3-19

a) For each scan *i*, retrieve the solar elevation at each end of the scan. The test is only performed if these are less than 5°: i.e. if
<view>*_band_centre_solar_elevation*(*i*, 0) < 5.0 or
<view>*_band_centre_solar_elevation*(*i*, 9) < 5.0,
so excluding day-time data. Then for each pixel:

(Req 5.22-35)

Step 5.22.3-20

b) Determine threshold med_high_level_thresh[bt_index] for the appropriate view from the table holding the threshold values for this test. The bt_index is calculated from

bt_index = integer part of (*I(ir12, v; i, j)* - 25000) / 50.
if (bt_index > 120 then bt_index = 120
else if (bt_index < 0) then bt_index = 0
end if

(Note that brightness temperature values are stored in units of 0.01 K)

(Req 5.22-36)

Step 5.22.3-21

c) If the 3.7 and 12 micron brightness temperatures are valid, and the difference

I(ir37, v; i, j) - I(ir12, v; i, j)

is greater than MED_HIGH_LEVEL_THRESH[bt_index] for the appropriate view then flag the pixel as cloudy by setting L1B-INT-233 and L1B-INT-240 in the nadir view and L1B-INT-249 and L1B-INT-256 in the forward view.

Use the L1-AUX11-1 threshold array in the nadir view, and the L1-AUX11-2 threshold array in the forward view.

(Req 5.22-37)

5.22.3.5 Fog/low stratus test

The test uses a look-up table that defines a threshold FOG_LOW_STRATUS_THRESHOLD[across-track band] for the nadir and forward views separately, for each across-track band = 5, 9. (Note that by symmetry the coefficients for the across-track band i = 0, 1, ...4 are the same as for band 9 - i.)

Apply the fog/low stratus test as follows:

Step 5.22.3-22

a) For each image scan *i*, retrieve the solar elevation at each end of the scan. The test is only performed if these are less than 5°: i.e. if
<view>*_band_centre_solar_elevation*(*i,* 0) < 5.0 or
<view>*_band_centre_solar_elevation*(*i,* 9) < 5.0,
so excluding day-time data. Then for each pixel:

(Req 5.22-38)

Step 5.22.3-23

b) Determine the across track band number band_no from the pixel x co-ordinate.using equation 5.6.1.

(Req 5.22-39)

Step 5.22.3-24

c) For each pixel for which a valid 11 micron and 3.7 micron brightness temperature is available, calculate the difference

$$I(ir11, v; i, j) - I(ir37, v; i, j).$$

If this difference is greater than FOG_LOW_STRATUS_THRESHOLD[band_no] for the appropriate view then flag the pixel as cloudy by setting L1B-INT-233 and L1B-INT-241 in the nadir view and L1B-INT-249 and L1B-INT-257 in the forward view.

Use the L1-AUX12-1 threshold array in the nadir view, and the L1-AUX12-2 threshold array in the forward view.

(Req 5.22-40)

5.22.3.6 Histogram test (1.6 micron)

Note. In the following, this test is specified in two forms: text describing the algorithm, and pseudo-code clarifying the text description. In the event of a conflict, the numbered requirements of the pseudo-code shall take precedence. The pseudo-code starts at Step 5.22.3-28.

The 1.6μm test is applied to sub-arrays of 32 x 32 pixels. It uses the 1.6μm reflectance data (L1B-INT-104 in the nadir view and L1B-INT-114 in the forward view).

Note: The input reflectance data are in units of 0.01% whereas the threshold parameters in the LUTs, and the derived reflectance thresholds, are in %.

Step 5.22.3-25

a) on initial entry, input the relevant cloud detection parameters (L1-AUX7-1 to L1-AUX7-17) from the look-up table.

(Req 5.22-41)

Step 5.22.3-26

b) Loop over the nadir and forward view.

(Req 5.22-42)

Step 5.22.3-27

c) Loop through the view by sequentially transferring subgroups of 32 x 32 1.6μm pixel values into array ir16 (L1B-INT-301) and their across-track indeces into array x (L1B-INT-302). The subgroups are non-overlapping.

c.1) Check that the number of valid pixels in the subgroup are sufficient to perform the test. Find then index of valid pixels, index_valid (L1B-INT-303) and the number of them, total_valid (L1B-INT-304). If total_valid is less than MIN_FOR_HISTOGRAM (L1-AUX7-5) then flag all pixels as cloudy, and exit.

'valid' implies that a pixel satisfies the following conditions:

- it has a valid 1.6μm value (i.e. reflectance greater than zero) AND

- it is not over land (i.e. L1B-INT-232 (nadir view) or L1B-INT-248 (forward view) is not set) AND

- it has not been flagged by the gross test OR the spatial coherence test OR the thin cirrus test (i.e. L1B-INT-233 (nadir view) or L1B-INT-249 (forward view) is not set)

(Req 5.22-43)

c.2) Check for sun glint:

c.2.1) Interpolate with respect to distance for the mid-point of the subgroup, using the band-edge solar angles appropriate for the view (L1B-INT-120 to L1B-INT-123 in the nadir view and L1B-INT-140 to L1B-INT-143 in the forward view) to get the following:

solar elevation angle	sol_elev (L1B-INT-305)
satellite elevation angle	sat_elev (L1B-INT-306)
(satellite azimuth - solar azimuth)	
difference angle	azim_dif (L1B-INT-307)

(Req 5.22-44)

c.2.2) Compute the x, y and z components of vector V bisecting the sun - subgroup midpoint - satellite angle. (LIB-INT-308 to 308). z axis: local vertical; x axis: sun's azimuth; y axis: completes right-handed co-ordinate system.

v_x = cos(sol_elev) + cos(sat_elev)*cos(azim_dif)

v_y = cos(sat_elev)*sin(azim_dif)

v_z = sin(sol_elev) + sin(sat_elev)

(Req 5.22-45)

c.2.3) Calculate the 'tilt' angle (L1B-INT-311), the angle by which the sea surface would need to be tilted to give specular reflection). It is given by

tilt = arccos(v_z/magnitude)

where

magnitude = sqrt($v\_x^2 + v\_y^2 + v\_z^2$)

(Req 5.22-46)

c.2.4) Compare tilt with TILT_THRESHOLD (L1-AUX7-6)

If

tilt < TILT_THRESHOLD.

then sun glint is deemed to be present. If true then set glint_present flag (L1B-INT-313) to 1,

(Req 5.22-47)

c.2.5) If glint_present is not set then generate histogram_16 (L1B-INT-314) using the 1.6µm values with a bin size of 0.1% reflectance (i.e. 10 product units), and continue.

If glint_present is set then set sunglint flag (L1B-INT-124 in nadir view, L1B-INT-250) in forward view) for the pixels and go to step c.7.

(Req 5.22-48)

c.3) Calculate for histogram_16 the following:

low_interval (L1B-INT-315) the reflectance of the first box with at least one pixel contributing

high_interval (L1B-INT-316) the reflectance of the last box with at least one pixel contributing

hist_range (L1B-INT-317) (high_interval - low_interval) + 0.1

peak_box_no (L1B-INT-318) the box number at the histogram peak, within the reflectance range of low_interval and (low_interval + SEARCH_RANGE_FOR_PEAK)

(N.B. % reflectance is obtained from histogram_16 box-numbers by dividing by 10.0. SEARCH_RANGE_FOR_PEAK: L1-AUX7-11)

peak_value (L1B-INT-319) histogram_16(peak_box_no)

average_value (L1B-INT-320) the average number of pixels per box for histogram_16, between the reflectance limits of low_interval and high_interval

By using the 3 histogram values centred on peak_box_no, calculate peak_interval (L1B-INT-321), the more accurate peak position, in percent. Fit a quadratic through the 3 points, and then find the ordinate of the function maximum as follows:

Calculate denominator D (L1B-INT-322):

D = 2.*histogram_16(peak_box_no-1) - 4.*histogram_16(peak_box_no + 2.*histogram_16(peak_box_no+1)

If D is not equal to zero then calculate delta L1B-INT-323)

delta = (histogram_16(peak_box_no-1) - histogram_16(peak_box_no+1)) / D

else delta = 0.0

The precise peak_interval, in percent, is given by

peak_interval = (peak_box_no + delta)/10.0

Also modify low_interval to the minimum non-zero value of the reflectance values, in % units

(Req 5.22-49)

c.4) The histogram must satisfy the following conditions:

- (peak_interval - low_interval) < spread_adjusted

- peak_value > (peak_factor +f(hist_range)) * g(tilt) * average_value

- peak_value > MIN_PEAK_VALUE

where, if tilt ≥ TILT_WEIGHT_LIMIT, then

spread_adjusted = spread

or, if tilt < TILT_WEIGHT_LIMIT, then

spread_adjusted = spread*(1.+(TILT_WEIGHT_LIMIT-tilt)/TILT_WEIGHT_LIMIT)

If hist_range < RANGE_WEIGHT_LIMIT then

f = (1.-peak_factor) *sqrt((RANGE_WEIGHT_LIMIT-hist_range) / (RANGE_WEIGHT_LIMIT-.2)))

otherwise f = 0

If tilt < TILT_WEIGHT_LIMIT then

g = 1 - (TILT_WEIGHT_LIMIT - tilt) * TILT_WEIGHT_FACTOR

otherwise g = 1.0

The tabulated constants, used in both views in the above, are:

TILT_WEIGHT_LIMIT (L1-AUX7-12)

RANGE_WEIGHT_LIMIT (L1-AUX7-13)

TILT_WEIGHT_FACTOR (L1-AUX7-14)

MIN_PEAK_VALUE (L1-AUX7-15)

In addition, in the nadir view for spread (L1B-INT-327) use (L1-AUX7-1) and for peak_factor (L1B-INT-328) use (L1-AUX7-3). The corresponding constants in the forward view are (L1-AUX7-2) and (L1-AUX7-4).

(Req 5.22-50)

c.4.1) If all three conditions are satisfied then

- set reflectance_threshold (L1B-INT-329):

reflectance_threshold = peak_interval + spread_adjusted/2.

- if the area is in a near-glint region i.e. where

TILT_THRESHOLD < tilt < (TILT_THRESHOLD + NEAR_GLINT_RANGE)

then go to c.5. otherwise go to c.6

(N.B. NEAR_GLINT_RANGE: L1-AUX7-8)

(Req 5.22-51)

c.4.2) If any of the three conditions are NOT satisfied then

If the area is not in a near-glint region, as defined in c.4.1, then flag all as cloudy and go to the next subgroup of 32 x 32 pixels.

(Req 5.22-52)

If the area is in a near-glint region then do the following:

c.4.2.1) calculate n, such that

n = 1.5 * spread_adjusted

(Req 5.22-53)

c.4.2.2) if the total number of valid pixels with reflectance values between low_interval and low_interval+n is greater than MIN_FOR_DETREND (L1-AUX7-16) then

- reflectance_threshold = low_interval + n

- go to c.5

otherwise go to step c.7 to do a spatial coherence test on reflectances.

(Req 5.22-54)

c.5) De-trend the reflectance data before applying a threshold as follows:

c.5.1) Calculate the across-track reflectance gradient gradient_16 (L1B-INT-331) in the data by regressing the reflectance values against their across-track distances. Only pixels having reflectance values less than reflectance_threshold are used in the fit.

The gradient is obtained by computing, using valid pixels only,

$a = \text{total\_valid} * \Sigma(x(i) * ir16(i)) - \Sigma x(i) * \Sigma ir16(i)$ and

$b = \text{total\_valid} * \Sigma x(i)^2 - (\Sigma x(i))^2$

where the summations are for total_valid values for the valid pixels only, and i takes the values of index_valid determined in step c.1.

If b is not equal to zero then

gradient_16 = a / b

otherwise

gradient_16 = 0.0

(Req 5.22-55)

c.5.2) De-trend the reflectance value of all valid pixels using the gradient calculated above: -1

detrended_16(i) = ir16(i) - gradient_16 * x(i)

(Req 5.22-56)

c.5.3) Form a new histogram_16 as in c.2.5

(Req 5.22-57)

c.5.4) Determine low_interval and peak_interval as in c.3

(Req 5.22-58)

c.5.5) Check that

(peak_interval - low_interval) < spread

If it is not true then go to step c.7)

(Req 5.22-59)

c.5.6) Determine the reflectance threshold, appropriate to the de-trended data:

reflectance_threshold = peak_interval + spread/2.

(Req 5.22-60)

c.6) Flag all the pixels with (de-trended, if appropriate) reflectance values greater than reflectance_threshold - in same units -by setting L1B-INT-233 and L1B-INT-235 in the nadir view and L1B-INT-249 and L1B-INT-251 in the forward view.

Go to step c.9)

(Req 5.22-61)

c.7) Calculate sd_threshold (L1B-INT-336) for the standard deviation of the reflectance in the following way:

c.7.1) Find the index of the highest 12µm brightness temperature in the 32 x 32 array, using the 'valid' pixels only., x_y_12um_max (L1B-INT-337)

A 'valid' pixel here, and also in section c.8) below, satisfies:

- the conditions listed in c.1 AND

- it has a 1.6µm value less than MAX_GLINT_THRESHOLD (L1-AUX7-17) AND

it has not been 'cosmetically filled'. i.e. <view>*_fill_state(i, j) = NATURAL_PIXEL.*

Req 5.22-62)

c.7.2) Set reflectance_at_12um_max (L1B-INT-338) equal to the 1.6µm reflectance value of this pixel (or, if more than one pixel is found, to that with the highest 1.6µm value).

(Req 5.22-63)

c.7.3) The standard deviation threshold, in units of %/100, is given by sd_threshold =100 * reflectance_at_12um_max * THRESHOLD_3

(N.B. THRESHOLD_3: L1-AUX7-7)

(Req 5.22-64)

c.8) Loop through the subgroup by examining the standard deviations of valid pixels from grouplets of (2 x 4) pixel arrays and flag those whose standard deviation exceeds the threshold value from c.7.3.

c.8.1) Compute the reflectance standard deviation for the sixteen (2 x 4) pixel grouplets for strips of 4 pixels by 32 pixels. (N.B. The long side of the rectangle is in the along-track direction.) Do it for all 8 strips in the (32 x 32) array.

(Req 5.22-65)

Only use 'valid' pixels, as re-defined above.

c.8.1.1) Set the flag as 'cloudy' for those (2 x 4) grouplets which had less than two valid pixels.

(Req 5.22-66)

c.8.1.2) Set the flag as 'cloudy' for those (2 x 4) grouplets which had a standard deviation greater than sd_threshold

(Req 5.22-67)

c.8.1.3) If an unflagged (i.e. 'clear') grouplet is between two 'cloudy' grouplets in a strip then set the flag for the group as 'cloudy'.

(Req 5.22-68)

c.8.2) Flag all those pixels cloudy which are in a 'cloudy' grouplet, and the pixels with reflectance values higher than MAX_GLINT_THRESHOLD by setting L1B-INT-233 and L1B-INT-236 in the nadir view and L1B-INT-249 and L1B-INT-252 in the forward view.

(Req 5.22-69)

c.9) Sum the number of clear pixels in the (32 x 32) array.

If the total is less than MIN_FOR_PASSED (L1-AUX7-1) then flag these also as cloudy as in c.8.2, then exit.

(Req 5.22-70)

Pseudo code for V16 Histogram test:

Step 5.22.3-28 Initialise

Note that the following pseudo-code is formulated to operate on a sequence of images of 512 by 512 pixels for each view (forward or nadir). The orbit is then imagined as tiled with such images, to which the outermost loop below refers. This matches the inner loops over y = 0, 511. Of course since subgroups of 32 by 32 pixels are treated independently, the length of the image segment to be processed at one time may be any multiple of 32, provided the loop over y is changed accordingly. It is assumed that processing will always include an integer number of granules, so if there is an incomplete image segment at the end of processing, it will always contain a multiple of 32 rows.

```
for each 512 by 512 pixel image segment
```

Initialise variables

```
set FirstTime = TRUE
                                                    (Req 5.22-71)
set FirstCall = TRUE
                                                    (Req 5.22-72)
```

If this is the first call, read the lookup table parameters from disk.

```
if (FirstCall==TRUE) then
  load_lookup_table
                                                    (Req 5.22-73)
  set FirstCall = FALSE
                                                    (Req 5.22-74)
end if

/* [SPEC: c.4.1] */
```

Step 5.22.3-29 Check that the solar elevation (at the centre of the image) is greater than zero

```
if(regridded_info[ROW(256)].band_edge_solar_angles[5].
      solar_elevation < 5.0) then
   return to calling routine
                                                    (Req 5.22-75)
  end if

/* [SPEC: c.2.1] */
Perform sunglint test on all sea pixels (Steps 30 to 32):
```

Step 5.22.3-30 Determine the "tilt angle" at the centre of each 32km square box

```
for each y ∈ {0..511} step 32
  for each Band ∈ {0..10} step 1
    set SolarElev = regridded_info[ROW(y+16)].
       band_edge_solar_angles[Band].solar_elevation
                                                    (Req 5.22-76)
    set SatelElev = regridded_info[ROW(y+16)].
       band_edge_solar_angles[Band].satellite_elevation
                                                     Req 5.22-77)
    set SolarAzim = regridded_info[ROW(y+16)].
       band_edge_solar_angles[Band].solar_azimuth
                                                    (Req 5.22-78)
    set SatelAzim = regridded_info[ROW(y+16)].
       band_edge_solar_angles[Band].satellite_azimuth
                                                    (Req 5.22-79)

    set TO_RADIANS = PI/180.0
                                                    (Req 5.22-80)
    set AzimuthDifference = SolarAzim-SatelAzim
                                                    (Req 5.22-81)
    set theta = SolarElev*TO_RADIANS
                                                    (Req 5.22-82)
    set n_theta = SatelElev*TO_RADIANS
```

```
                                                        (Req 5.22-83)
      set n_phi = AzimuthDifference*TO_RADIANS
                                                        (Req 5.22-84)
      set i_vec =
         0.5*(cos(theta) + (cos(n_theta) * cos(n_phi)))
                                                        (Req 5.22-85)
      set j_vec = 0.5*(cos(n_theta) * sin(n_phi))
                                                        (Req 5.22-86)
      set k_vec = 0.5*(sin(theta) + sin(n_theta))
                                                        (Req 5.22-87)
      set magnitude =
         SQRT(SQUARE(i_vec) + SQUARE(j_vec) + SQUARE(k_vec))
                                                        (Req 5.22-88)
      set TiltAtBands(Band) =
         ABS((acos(k_vec/magnitude)*TO_DEGREES))
                                                        (Req 5.22-89)
    end for

/* [SPEC: c.2.4] */
```

Step 5.22.3-31 Check if sunglint is present

```
    for each x ∈ {16..511} step 32
      set Band = x/51.2
                                                        (Req 5.22-90)
      set Weight = x - Band*51.2
                                                        (Req 5.22-91)
      set TiltAngle[y/32][x/32]=((TiltAtBands[Band]*(51.2-Weight)) +
         (TiltAtBands[Band+1]*Weight)) / 51.2
                                                        (Req 5.22-92)

      if (TiltAngle[y/32][x/32]<TiltThreshold) then
        set SunGlintResult[y/32][x/32]=SUNGLINT
        for each j ∈ {0..31} step 1
          for each i ∈ {0..31} step 1
            if (pixel_cloud_state[y+j][x+(i-16)].land==FALSE) then
              set pixel_cloud_state[y+j][x+(i-16)].sunglint==TRUE
                                                        (Req 5.22-93)
            end if
          end for
        end for
      else

/* [SPEC: c.4] */
```

Step 5.22.3-32 Check if in near glint conditions

```
        if (TiltAngle[y/32][x/32]<(TiltThreshold+NearGlintRange) then
           set SunGlintResult[y/32][x/32]=NEARGLINT
                                                        (Req 5.22-94)
```

```
          else
            set SunGlintResult[y/32][x/32]=NOGLINT
                                                          (Req 5.22-95)
          end if
        end if
      end for
    end for
```

Step 5.22.3-33 Create array of "scaled" reflectances

```
  /* [SPEC: c]  */
    Loop through the image by examining SUBGROUPS of 32x32 pixels
    for each y ∈ {0..511} step 32
      for each x ∈ {0..511} step 32
        /* [SPEC: c.1] */
        /* Check the number of 'valid' pixels in subgroup are enough to
        perform test  */
        /* [SPEC: b] */
        /* Create array of "scaled" reflectances */
        set PixelCount=0
                                                          (Req 5.22-96)
        for each j ∈ {0..31} step 1
          for each i ∈ {0..31} step 1
            set Reflectance[j][i] =
              (v16[ROW(y+j)][x+i]/100.0)                  (Req 5.22-97)
            if (Reflectance[j][i]>0 AND
               pixel_cloud_state[y+j][x+i].cloud==FALSE AND
               pixel_cloud_state[y+j][x+i].land==FALSE )
               set Infrared[j][i]=ir12[ROW(y+j)][x+i]
                                                          (Req 5.22-98)
               set PixelCount=Pixelcount+1
                                                          (Req 5.22-99)
            else
              set Reflectance[j][i]=IGNORE_PIXEL
                                                         (Req 5.22-100)
            end if
          end for
        end for

  /* [SPEC: c.1] */
```

Step 5.22.3-34 If too few pixels, flag all pixels as cloudy

```
        if ( PixelCount<MinForHistogram) then
          for each j ∈ {0..31} step 1
            for each i ∈ {0..31} step 1
              if (Reflectance[j][i]!=IGNORE_PIXEL) then
                set pixel_cloud_state[y+j][x+i].cloud=TRUE
                                                         (Req 5.22-101)
```

```
                set pixel_cloud_state[y+j][x+i].
                    v16_histogram_test=TRUE
                                                    (Req 5.22-102)
              end if
            end for
          end for
        exit to loop over 32 by 32 subgroups.
        end if

  /* [SPEC: c.2.5] */
```

Step 5.22.3-35 If sunglint is present, do spatial coherence test

```
        if (SunGlintResult[y/32][x/32]==SUNGLINT) then
           goto SpatialCoherenceTest
        end if                                      (Req 5.22-103)

  /* [SPEC: c.4]  */
```

Step 5.22.3-36 Calculate the "spread_adjusted" value using tilt angle

```
        if (TiltAngle[y/32][x/32]>=TiltWeightLimit) then
          set SpreadAdjusted=HistSpread(View)
                                                    (Req 5.22-104)
        else
          set SpreadAdjusted=HistSpread(View)*(1.0+
            (TiltWeightLimit-TiltAngle[y/32][x/32])
             /TiltWeightLimit)
                                                    (Req 5.22-105)
        end if

  /* [SPEC: c.2.5,c.3] */
```

Step 5.22.3-37 Construct the histogram and obtain the statistics

```
  /* [SPEC: c.4.2.1] */
        set n=1.5*SpreadAdjusted
                                                    (Req 5.22-106)
        if (FirstTime==TRUE) then
          set FirstTime=FALSE
                                                    (Req 5.22-107)
          for each Box ∊ {0..1499} step 1
            set Histogram[Box]=0
                                                    (Req 5.22-108)
          end for
        end if

  /* [SPEC:c.2.5 & c.4.3] */
```

Step 5.22.3-38 Generate histogram using 1.6um values with a binsize of 0.1%

```
        set PixelCount = 0
                                                    (Req 5.22-109)
```

```
set HighInterval = 0.0
                                                   (Req 5.22-110)
set LowInterval = 200.0
                                                   (Req 5.22-111)
for each j ∈ {0..31} step 1
  for each i ∈ {0..31} step 1
    if (Reflectance[j][i]!=IGNORE_PIXEL  AND
        Reflectance[j][i] > 0  AND
      Reflectance[j][i]<150.0) then
      set Box = INTEGER(Reflectance[j][i]*10.0)
                                                   (Req 5.22-112)
      set Histogram[Box] = Histogram[Box] + 1
                                                   (Req 5.22-113)
      set PixelCount = PixelCount + 1
                                                   (Req 5.22-114)
      set LowInterval =
        MINIMUM_OF(LowInterval,Reflectance[j][i])
                                                   (Req 5.22-115)
      set HighInterval =
        MAXIMUM_OF(HighInterval,Reflectance[j][i])
                                                   (Req 5.22-116)
    end if
  end for
end for
```

```
/* [SPEC:c.3] */
```

Step 5.22.3-39 Calculate HIST_RANGE, AVERAGE_VALUE, and PEAK_INTERVAL

```
set HistRange = HighInterval - LowInterval+0.1
                                                   (Req 5.22-117)
set AverageValue = PixelCount/(HistRange*10)
                                                   (Req 5.22-118)
set PeakInterval = LowInterval*10
                                                   (Req 5.22-119)
for each Box ∈ {INTEGER(LowInterval*10.0)..
MINIMUM_OF(INTEGER(LowInterval*10.0 + SearchRangeForPeak*10.0),
                        INTEGER(HighInterval*10.0))} step 1
  if (Histogram[Box]>Histogram[INTEGER(PeakInterval)] then
    set PeakInterval = Box
                                                   (Req 5.22-120)
  end if
end for

set PeakValue = Histogram[INTEGER(PeakInterval)]
                                                   (Req 5.22-121)
set PeakInterval = PeakInterval/10.0
                                                   (Req 5.22-122)
```

```
/* [SPEC:c.3] */
```

Step 5.22.3-40 Calculate PEAK_INTERVAL more accurately by doing a quadratic fit

```
if (PeakInterval!=LowInterval
   AND PeakInterval!=HighInterval) then
  set Box = NEAREST_INTEGER(PeakInterval*10.0)
                                                (Req 5.22-123)
  set Denominator = 2.0*Histogram[Box-1] - 4.0*Histogram[Box] +
      2.0*Histogram[Box+1]
                                                (Req 5.22-124)
  if (Denominator!=0) then
    set PeakDelta = (Histogram[Box-1] -
       Histogram[Box+1])/Denominator
                                                (Req 5.22-125)
    set PeakInterval = PeakInterval + PeakDelta/10
                                                (Req 5.22-126)
  end if
end if
```

```
/* [SPEC:c.4] */
```

Step 5.22.3-41 Count "the number of valid pixels with reflectance values between low_interval and low_interval+n

```
set LowestPixels = 0
                                                (Req 5.22-127)
for each Box ∈
   {INTEGER(LowInterval*10.0)..INTEGER((LowInterval+n)*10.0)}
  set LowestPixels = LowestPixels + Histogram[Box]
                                                (Req 5.22-128)
end for

/* Reset histogram */
for each Box ∈
  {INTEGER(LowInterval*10.0)..INTEGER(HighInterval*10.0)}
  set Histogram[Box] = 0
                                                (Req 5.22-129)
end for

/* [SPEC: c.4.1] */
/* If all three [of the following] conditions are satisfied */
(Requirement deleted)
                                                (Req 5.22-130)
if(HistRange>RangeWeightLimit) then
  set temp_factor=PeakFactor(View)
                                                (Req 5.22-131)
else
  set temp_factor = PeakFactor(View) +
      (1.0 - PeakFactor(View))*
```

```
      SQRT((RangeWeightLimit-HistRange)/
      (RangeWeightLimit-0.2))
                                              (Req 5.22-132)
end if

if (TiltAngle[y/32][x/32]<TiltThreshold
  + NearGlintRange) then
   set temp_factor =
     temp_factor * (1.0 - (TiltWeightLimit -
     TiltAngle[y/32][x/32])*TiltWeightFactor
                                              (Req 5.22-133)
end if

if (PeakValue > MinPeakValue AND
  PeakInterval-LowInterval <SpreadAdjusted AND
  PeakValue > temp_factor * average_value) then

set ReflectanceThreshold =
   PeakInterval+SpreadAdjusted/2.0
                                              (Req 5.22-134)

/* [SPEC: c.4.1] */
```

Step 5.22.3-42 If the area is in a near-glint region" detrend data

```
if (SunGlintResult[y/32][x/32]==NEARGLINT) then
    set SX = 0.0                              (Req 5.22-135)
    set SY = 0.0                              (Req 5.22-136)
    set SYY = 0.0                             (Req 5.22-137)
    set SXY = 0.0                             (Req 5.22-138)
    set SXX = 0.0                             (Req 5.22-139)
    (Requirement deleted)                     (Req 5.22-140)

    /* [SPEC: c.4.1] */
```

Step 5.22.3-42.1. Calculate the across track reflectance gradient in the data by regressing the reflectance values against their across track distances

```
for each j ∈ {0..31} step 1
  for each i ∈ {0..31} step 1
    if(Reflectance[i][j]!=IGNORE_PIXEL  AND
       Reflectance[i][j]<ReflectanceThreshold) then
      set SX = SX + j                         (Req 5.22-141)
      set SY = SY + Reflectance[i][j]         (Req 5.22-142)
      set SYY = SYY +
         Reflectance[i][j]*Reflectance[i][j]
                                              (Req 5.22-143)
      set SXY = SXY + j*Reflectance[i][j]
                                              (Req 5.22-144)
      set SXX = SXX + j*j                     (Req 5.22-145)
      set NP = NP + 1                         (Req 5.22-146)
```

```
    end if
  end for
end for

set DELTA=(NP*SXX)-(SX*SX)                      (Req 5.22-147)
if (DELTA!=0) then
  set DELTA=(NP*SXX)-(SX*SX)                    (Req 5.22-148)
  set SLOPE=((NP*SXY)-(SX*SY))/DELTA            (Req 5.22-149)
  set OFFSET=((SY*SXX)-(SX*SXY))/DELTA          (Req 5.22-150)

  for each j ∈ {0..31} step 1
    for each i ∈ {0..31} step 1
      if (Reflectance[j][i]!=IGNORE_PIXEL) then
         Reflectance[j][i] =
               Reflectance[j][i] - i*SLOPE
                                                (Req 5.22-151)
      end if
    end for
  end for
end if

if (FirstTime==TRUE) then
  set FirstTime=FALSE                           (Req 5.22-152)
  for each Box ∈ {0..1499} step 1
    set Histogram[Box]=0                        (Req 5.22-153)
  end for
end if
```

```
/* [SPEC:c.2.5 & c.4.3] */
```

Step 5.22.3-42.2. Generate histogram using 1.6um values with a binsize of 0.1%

```
set PixelCount = 0                              (Req 5.22-154)
set HighInterval = 0.0                          (Req 5.22-155)
set LowInterval = 200.0                         (Req 5.22-156)
for each j ∈ {0..31} step 1
  for each i ∈ {0..31} step 1
    if (Reflectance[j][i]!=IGNORE_PIXEL  AND
        Reflectance[j][i] > 0  AND
      Reflectance[j][i]<150.0) then
      set Box = INTEGER(Reflectance[j][i]*10.0)
                                                (Req 5.22-157)
      set Histogram[Box] = Histogram[Box] + 1
                                                (Req 5.22-158)
      set PixelCount = PixelCount + 1           (Req 5.22-159)
      set LowInterval =
         MINIMUM_OF(LowInterval,Reflectance[j][i])
```

```
                                                   (Req 5.22-160)
          set HighInterval =
             MAXIMUM_OF(HighInterval,Reflectance[j][i])
                                                   (Req 5.22-161)
        end if
      end for
    end for
```

/* [SPEC:c.3] */

Step 5.22.3-43 Calculate HIST_RANGE, AVERAGE_VALUE, and PEAK_INTERVAL

```
    set HistRange = HighInterval - LowInterval+0.1
                                                   (Req 5.22-162)
    set AverageValue = PixelCount/(HistRange*10)
                                                   (Req 5.22-163)
    set PeakInterval = LowInterval*10              (Req 5.22-164)
    for each Box ∈ {INTEGER(LowInterval*10.0)..
        MINIMUM_OF(INTEGER(LowInterval*10.0 +
        SearchRangeForPeak*10.0),
               INTEGER(HighInterval*10.0))} step 1

      if (Histogram[Box]>Histogram[INTEGER(PeakInterval)] then
        set PeakInterval = Box                     (Req 5.22-165)
      end if
    end for
    set PeakValue = Histogram[INTEGER(PeakInterval)]
                                                   (Req 5.22-166)
    set PeakInterval = PeakInterval/10.0           (Req 5.22-167)
```

/* [SPEC:c.3] */

Step 5.22.3-44 Calculate PEAK_INTERVAL more accurately by doing a quadratic fit

```
    if (PeakInterval!=LowInterval  AND
    PeakInterval!=HighInterval) then
      set Box = NEAREST_INTEGER(PeakInterval*10.0)
                                                   (Req 5.22-168)
      set Denominator = 2.0*Histogram[Box-1] -
          4.0*Histogram[Box] + 2.0*Histogram[Box+1]
                                                   (Req 5.22-169)

      if (Denominator!=0) then
        set PeakDelta = (Histogram[Box-1] -
             Histogram[Box+1])/Denominator         (Req 5.22-170)
        set PeakInterval =
             PeakInterval + PeakDelta/10           (Req 5.22-171)
      end if
    end if
```

```
/* [SPEC:c.4] */
```

Step 5.22.3-45 Count "the number of valid pixels with reflectance values between low_interval and low_interval+n

```
set LowestPixels = 0                          (Req 5.22-172)
for each Box ∈ {INTEGER(LowInterval*10.0)..
                INTEGER((LowInterval+n)*10.0)}
  set LowestPixels = LowestPixels + Histogram[Box]
                                              (Req 5.22-173)
end for

/* Reset histogram */

for each Box ∈
{INTEGER(LowInterval*10.0)..INTEGER(HighInterval*10.0)}
  set Histogram[Box] = 0                      (Req 5.22-174)
end for

if(PeakInterval-LowInterval)>HistSpread(View)) then
  goto SpatialCoherenceTest                   (Req 5.22-175)
else
  set ReflectanceThreshold = PeakInterval +
     HistSpread(View) / 2.0                   (Req 5.22-176)
       (Requirement deleted)                  (Req 5.22-177)
end if

  for each j ∈ {0..31} step 1
    for each i ∈ {0..31} step 1
      if (Reflectance[j][i]!=IGNORE_PIXEL  AND
      Reflectance[j][i]>ReflectanceThreshold) then
        set pixel_cloud_state[y+j][x+i].cloud = TRUE
                                              (Req 5.22-178)
        set pixel_cloud_state[y+j][x+i].v16_histogram_test
      = TRUE                                  (Req 5.22-179)
      end if
    end for
  end for
end if
```

```
/* [SPEC: c.4.1] */
```

Step 5.22.3-46 Otherwise, do "reflectance_threshold" check

```
else
    for each j ∈ {0..31} step 1
      for each i ∈ {0..31} step 1
        if (Reflectance[j][i]!=IGNORE_PIXEL  AND
```

```
                 Reflectance[j][i]>ReflectanceThreshold) then
            set pixel_cloud_state[y+j][x+i].cloud=TRUE
            set pixel_cloud_state[y+j][x+i].v16_histogram_test
                    =TRUE                          (Req 5.22-180)
          end if
        end for
      end for
  end if
    set PixelCount = 0                             (Req 5.22-181)
    set xx = 0                                     (Req 5.22-182)
    set yy = 0                                     (Req 5.22-183)

/* [SPEC: c.9] */
```

Step 5.22.3-47 . Count the number of clear pixels

```
    for each yy ∊ {0..31} step 1
      if (PixelCount<INTEGER(MinForPassed)) then
        for each xx ∊ {0..31} step 1
          if (PixelCount<INTEGER(MinForPassed) AND
              Reflectance[yy][xx])!=IGNORE_PIXEL  AND
            pixel_cloud_state[y+yy][x+xx].cloud !=TRUE) then
            set xpos[PixelCount] = x + xx          (Req 5.22-184)
            set ypos[PixelCount] = y + yy          (Req 5.22-185)
            set PixelCount = PixelCount + 1        (Req 5.22-186)
          end if
        end for
      end if
    end for

/* [SPEC: c.9] */
```

Step 5.22.3-48 If less than "MinForPassed" pixels remain clear, flag these as cloudy

```
    if (PixelCount<MinForPassed  AND
        PixelCount>0) then
      for each i E {0..PixelCount-1} step 1
        set pixel_cloud_state[ypos[i]][xpos[i]].cloud
         = TRUE                                    (Req 5.22-187)
        set
  pixel_cloud_state[ypos[i]][xpos[i]].v16_histogram_test
          = TRUE                                   (Req 5.22-188)
      end for
    end if

/* [SPEC: c.4.2] */
/* If any of the three conditions are NOT satisfied */
```

```
        else

    /* [SPEC: c.4.2] */
```

Step 5.22.3-49 If in a near-glint region, do the following

```
        if(SunGlintResult[y/32][x/32]==NEARGLINT) then

    /* [SPEC: c.4.2.2]  */

          if(LowestPixels>MinForDetrend) then
            set ReflectanceThreshold=LowInterval+n    (Req 5.22-189)
              set SX = 0.0                            (Req 5.22-190)
              set SY = 0.0                            (Req 5.22-191)
              set SYY = 0.0                           (Req 5.22-192)
              set SXY = 0.0                           (Req 5.22-193)
              set SXX = 0.0                           (Req 5.22-194)
              (Requirement deleted)                   (Req 5.22-195)

    /* [SPEC: c.4.1] */
```

Step 5.22.3-50 Calculate the across track reflectance gradient in the data by regressing the reflectance values against their across track distances

```
              for each j ∈ {0..31} step 1
                for each i ∈ {0..31} step 1
                  if(Reflectance[i][j]!=IGNORE_PIXEL  AND
                     Reflectance[i][j]<ReflectanceThreshold) then
                    set SX = SX + j                   (Req 5.22-196)
                    set SY = SY + Reflectance[i][j]   (Req 5.22-197)
                    set SYY = SYY +
                        Reflectance[i][j]*Reflectance[i][j]
                                                      (Req 5.22-198)
                    set SXY = SXY + j*Reflectance[i][j]
                                                      (Req 5.22-199)
                    set SXX = SXX + j*j               (Req 5.22-200)
                    set NP = NP + 1                   (Req 5.22-201)
                  end if
                end for
              end for

              set DELTA=(NP*SXX)-(SX*SX)              (Req 5.22-202)
              if (DELTA!=0) then
                set DELTA=(NP*SXX)-(SX*SX)            (Req 5.22-203)
                set SLOPE=((NP*SXY)-(SX*SY))/DELTA    (Req 5.22-204)
                set OFFSET=((SY*SXX)-(SX*SXY))/DELTA  (Req 5.22-205)

                for each j ∈ {0..31} step 1
```

```
      for each i ∈ {0..31} step 1
        if (Reflectance[j][i]!=IGNORE_PIXEL) then
           Reflectance[j][i] =
                Reflectance[j][i] - i*SLOPE(Req 5.22-206)
        end if
      end for
    end for
  end if

  if (FirstTime==TRUE) then
    set FirstTime=FALSE                    (Req 5.22-207)
    for each Box ∈ {0..1499} step 1
      set Histogram[Box]=0                 (Req 5.22-208)
    end for
  end if
```

/* [SPEC:c.2.5 & c.4.3] */

Step 5.22.3-51 Generate histogram using 1.6um values with a binsize of 0.1%

```
set PixelCount = 0                         (Req 5.22-209)
set HighInterval = 0.0                     (Req 5.22-210)
set LowInterval = 200.0                    (Req 5.22-211)

for each j ∈ {0..31} step 1
  for each i ∈ {0..31} step 1
    if (Reflectance[j][i]!=IGNORE_PIXEL  AND
        Reflectance[j][i] > 0  AND
      Reflectance[j][i]<150.0) then
      set Box = INTEGER(Reflectance[j][i]*10.0)
                                           (Req 5.22-212)
      set Histogram[Box] = Histogram[Box] + 1
                                           (Req 5.22-213)
      set PixelCount = PixelCount + 1      (Req 5.22-214)
      set LowInterval =
      MINIMUM_OF(LowInterval,Reflectance[j][i])
      set HighInterval =
      MAXIMUM_OF(HighInterval,Reflectance[j][i])
                                           (Req 5.22-215)
    end if
  end for
end for
```

/* [SPEC:c.3] */

Step 5.22.3-52 Calculate HIST_RANGE, AVERAGE_VALUE, and PEAK_INTERVAL

```
set HistRange = HighInterval - LowInterval+0.1
```

```
                                                  (Req 5.22-216)
      set AverageValue = PixelCount/(HistRange*10)
                                                  (Req 5.22-217)
      set PeakInterval = LowInterval*10           (Req 5.22-218)

      for each Box ∈ {INTEGER(LowInterval*10.0)..

  MINIMUM_OF(INTEGER(LowInterval*10.0 +

  SearchRangeForPeak*10.0),

  INTEGER(HighInterval*10.0))} step 1
        if (Histogram[Box]>Histogram[INTEGER(PeakInterval)]
                                                          then
          set PeakInterval = Box                  (Req 5.22-219)
        end if
      end for

      set PeakValue = Histogram[INTEGER(PeakInterval)]
                                                  (Req 5.22-220)
      set PeakInterval = PeakInterval/10.0        (Req 5.22-221)

/* [SPEC:c.3] */
```

Step 5.22.3-53 Calculate PEAK_INTERVAL more accurately by doing a quadratic fit

```
      if (PeakInterval!=LowInterval  AND
  PeakInterval!=HighInterval) then
        set Box = NEAREST_INTEGER(PeakInterval*10.0)
                                                  (Req 5.22-222)
        set Denominator = 2.0*Histogram[Box-1] -
            4.0*Histogram[Box] + 2.0*Histogram[Box+1]
                                                  (Req 5.22-223)

        if (Denominator!=0) then
          set PeakDelta = (Histogram[Box-1] -
                     Histogram[Box+1])/Denominator
                                                  (Req 5.22-224)
          set PeakInterval = PeakInterval +
                                   PeakDelta/10 (Req 5.22-225)
        end if
      end if

/* [SPEC:c.4] */
```

Step 5.22.3-54 Count "the number of valid pixels with reflectance values between low_interval and low_interval+n

```
      set LowestPixels = 0
```

```
      for each Box ∈ {INTEGER(LowInterval*10.0)..
                      INTEGER((LowInterval+n)*10.0)}
        set LowestPixels = LowestPixels +
                                    Histogram[Box](Req 5.22-226)
      end for

      /* Reset histogram */
      for each Box ∈
  {INTEGER(LowInterval*10.0)..INTEGER(HighInterval*10.0)}
        set Histogram[Box] = 0                  (Req 5.22-227)
      end for

    if(PeakInterval-LowInterval)>HistSpread(View)) then
      goto SpatialCoherenceTest                 (Req 5.22-228)
    else
      set ReflectanceThreshold =
           PeakInterval+HistSpread(View)/2.0  (Req 5.22-229)

        for each j ∈ {0..31} step 1
          for each i ∈ {0..31} step 1
            if (Reflectance[j][i]!=IGNORE_PIXEL  AND
               Reflectance[j][i]>ReflectanceThreshold) then
               set pixel_cloud_state[y+j][x+i].cloud
                                    = TRUE
                                                (Req 5.22-230)
               set pixel_cloud_state[y+j][x+i].
                    v16_histogram_test = TRUE
                                                (Req 5.22-231)
            end if
          end for
        end for

        set PixelCount = 0                      (Req 5.22-232)
        set xx = 0                              (Req 5.22-233)
        set yy = 0                              (Req 5.22-234)

/* [SPEC: c.9] */
```

Step 5.22.3-55 Count the number of clear pixels

```
        for each yy ∈ {0..31} step 1
          if (PixelCount<INTEGER(MinForPassed)) then
            for each xx ∈ {0..31} step 1
              if (PixelCount<INTEGER(MinForPassed) AND
                  Reflectance[yy][xx])!=IGNORE_PIXEL  AND
                  pixel_cloud_state[y+yy][x+xx].cloud !=TRUE)
                  then
```

```
                  set xpos[PixelCount] = x + xx
                                                 (Req 5.22-235)
                  set ypos[PixelCount] = y + yy
                                                 (Req 5.22-236)
                  set PixelCount = PixelCount + 1
                                                 (Req 5.22-237)
                end if
              end for
            end if
          end for

/* [SPEC: c.9] */
```

Step 5.22.3-56 If less than "MinForPassed" pixels remain clear, flag these as cloudy

```
          if (PixelCount<MinForPassed  AND
              PixelCount>0) then
            for each i ∈ {0..PixelCount-1} step 1
              set pixel_cloud_state[ypos[i]][xpos[i]].
               cloud = TRUE                      (Req 5.22-238)

              set pixel_cloud_state[ypos[i]][xpos[i]].
               v16_histogram_test = TRUE         (Req 5.22-239)
            end for
          end if
        end if
      else
        goto SpatialCoherenceTest                (Req 5.22-240)
      end if
    else

/* [SPEC: c.4.2] */
```

Step 5.22.3-57 If the area is NOT near-glint, flag all pixels as cloudy

```
      for each j ∈ {0..31} step 1
        for each i ∈ {0..31} step 1
          if (Reflectance[j][i]!=IGNORE_PIXEL) then
             set pixel_cloud_state[y+j][x+i].cloud=TRUE
                                                 (Req 5.22-241)
             set pixel_cloud_state[y+j][x+i].v16_histogram_test
                          = TRUE                 (Req 5.22-242)
          end if
        end for
      end for
    end if
    end if  (Matches if following Req 5.22-133, Step 5.22.3-41)
```

```
Go to OmitSpatialCoherenceTest (label following Req. 5.22-271)
```

Label: SpatialCoherenceTest

```
/* [SPEC: C.8] */
```

Step 5.22.3-58 Ignore "filled" pixels, and pixels values that exceed "MAX_GLINT_THRESHOLD"

```
for each j ∈ {0..31} step 1
  for each i ∈ {0..31} step 1
                                                   (Req 5.22-243)
```

```
/* [SPEC: c.8.2] */
```

Step 5.22.3-59 Flag...pixels with reflectance greater than MAX_GLINT_THRESHOLD

```
    if (Reflectance[j][i]!=IGNORE_PIXEL  AND
      Reflectance[j][i]>=MaxGlintThreshold) then
      set pixel_cloud_state[y+j][x+i].cloud=TRUE
                                                   (Req 5.22-244)
      set pixel_cloud_state[y+j][x+i].
         v16_spatial_coherence_test=TRUE           (Req 5.22-245)
      set Reflectance[j][i]=IGNORE_PIXEL           (Req 5.22-246)
    else if (<view>_fill_state[y+j][x+i] != NATURAL_PIXEL) then
      set Reflectance[j][i]=IGNORE_PIXEL
    end if
  end for
end for
```

```
/* [SPEC: c.7] */
```

Step 5.22.3-60 Calculate SD Threshold

Derive the sd_threshold value described in [SPEC: c.7], using the array of 32 x 32 1.6 micron percentage reflectance values (Reflectance), and the corresponding array of 32 x 32 12 micron pixel values (Infrared).

```
Set Result = 0.0
for each j ∈ {0..31} step 1
  for each i ∈ {0..31} step 1
    if (Reflectance[j][i]!=IGNORE_PIXEL) then
      if (Infrared[j][i]>Highest12umValue) then
         Highest12umValue =Infrared[j][i];
         Result=Reflectance[j][i];
      else if ((Infrared[j][i]==Highest12umValue) &&
               (Reflectance[j][i]>Result))
            Result=Reflectance[j][i];
      end if
    end if
  end for
end for
```

```
        set SDThreshold = Result * (Threshold3)
                                                      (Req 5.22-247)

    /* [SPEC: c.8] */
```

Step 5.22.3-61 Do spatial coherence test

```
    /* For every "grouplet" do the following */
        for each yy ∊ {0..31} step 4
          for each xx ∊ {0..31} step 2
            set GroupletMean = 0.0                    (Req 5.22-248)
            set GroupletSD = 0.0                      (Req 5.22-249)
            set GroupletPixelCount = 0                (Req 5.22-250)
            set GroupletCategory[yy/4][xx/2]=CLEAR    (Req 5.22-251)

    /* For every pixel in grouplet do the following */
            for each j ∊ {yy..(yy+3)} step 1
              for each i ∊ {xx..(xx+1)} step 1
              /* If pixel is "valid", use value to calculate mean */
                if (Reflectance[j][i]!=IGNORE_PIXEL) then
                  set GroupletMean =
                      GroupletMean + Reflectance[j][i] (Req 5.22-252)
                  set GroupletPixelCount =
                      GroupletPixelCount + 1          (Req 5.22-253)
                end if
              end for
            end for

    /* [SPEC: c.8.1.1] */
    /* If there are less than 2 clear pixels flag grouplet as cloudy */
            if (GroupletPixelCount<2) then
              set GroupletCategory[yy/4][xx/2]=CLOUD  (Req 5.22-254)
            else

    /* [SPEC: c.8.1] */
    /* Otherwise, calculate the standard deviation for the grouplet */
              set GroupletMean = GroupletMean/GroupletPixelCount
                                                      (Req 5.22-255)
              for each j ∊ {yy..yy+3} step 1
                for each i ∊ {xx..xx+1} step 1
                  if (Reflectance[j][i]!=IGNORE_PIXEL) then
                    set GroupletSD = GroupletSD +
                       SQUARE(Reflectance[j][i] -
                       GroupletMean)                  (Req 5.22-256)
                  end if
                end for
              end for
              set GroupletSD =
```

```
                    SQRT(GroupletSD/(GroupletPixelCount-1))
                                                  (Req 5.22-257)
```

Step 5.22.3-62 Flag pixels with SD greater than SD_THRESHOLD

```
[SPEC: c.8.1.2]
            if (GroupletSD>SDThreshold) then
              set GroupletCategory[yy/4][xx/2]=CLOUD (Req 5.22-258)
            end if
          end if
       end for
     end for

/* [SPEC: c.8.1.3] */
```

Step 5.22.3-63 If an unflagged grouplet is between two cloudy grouplets do the following

```
for each yy ∈ {0..7} step 1
  for each xx ∈ {1..14} step 1
    if (GroupletCategory[yy][xx]!=LAND  AND
      GroupletCategory[yy][xx-1]==CLOUD  AND
      GroupletCategory[yy][xx+1]==CLOUD) then
      set TempCategory[yy][xx]=CLOUD          (Req 5.22-259)
    else
      TempCategory[yy][xx]=CLEAR              (Req 5.22-260)
    end if
  end for
end for
```

Step 5.22.3-64 Now merge the temporary results with the grouplet results.

```
for each yy ∈ {0..7} step 1
  for each xx ∈ {1..14} step 1
    if (TempCategory[yy][xx]==CLOUD) then
      set GroupletCategory[yy][xx]=CLOUD      (Req 5.22-261)
    end if
  end for
end for

/* [SPEC: 8.2] */
```

Step 5.22.3-65 Flag all those pixels which are in a cloudy grouplet

```
for each yy ∈ {0..31} step 4
  for each xx ∈ {0..31} step 2
    if (GroupletCategory[yy/4][xx/2]==CLOUD) then
      for each j ∈ {yy..yy+3} step 1
```

```
                for each i ∈ {xx..xx+1} step 1
                  if (Reflectance[j][i]!=IGNORE_PIXEL) then
                    set pixel_cloud_state[y+j][x+i].cloud = TRUE
                                                             (Req 5.22-262)
                    set pixel_cloud_state[y+j][x+i].
                             v16_spatial_coherence_test = TRUE
                                                             (Req 5.22-263)
                  end if
                end for
              end for
            end if
          end for
        end for

  /* [SPEC: c.9] */
```

Step 5.22.3-66 Do cloud quantity test

```
        set PixelCount = 0                                   (Req 5.22-264)
        set xx = 0                                           (Req 5.22-265)
        set yy = 0                                           (Req 5.22-266)

  /* [SPEC: c.9] */
        /* Count the number of clear pixels */
        for each yy ∈ {0..31} step 1
          if (PixelCount<INTEGER(MinForPassed)) then
            for each xx ∈ {0..31} step 1
              if (PixelCount<INTEGER(MinForPassed) AND
                  Reflectance[yy][xx])!=IGNORE_PIXEL  AND
                pixel_cloud_state[y+yy][x+xx].cloud !=TRUE) then
                set xpos[PixelCount] = x + xx        (Req 5.22-267)
                set ypos[PixelCount] = y + yy        (Req 5.22-268)
                set PixelCount = PixelCount + 1      (Req 5.22-269)
              end if
            end for
          end if
        end for

  /* [SPEC: c.9] */

  /* If less than "MinForPassed" pixels remain clear, flag these as
  cloudy */

        if (PixelCount<MinForPassed  AND
            PixelCount>0) then
          for each i ∈ {0..PixelCount-1} step 1
            set pixel_cloud_state[ypos[i]][xpos[i]].cloud = TRUE
```

```
                                                  (Req 5.22-270)
          set pixel_cloud_state[ypos[i]][xpos[i]].
              v16_histogram_test = TRUE          (Req 5.22-271)
        end for
      end if
Label: OmitSpatialCoherenceTest
    end for
  end for
end for
```

5.22.3.7 11/12 micron nadir/forward test

This test is only applied to sea pixels. Loop through the pixels in the 10 across-bands of an image and check the relationship between the measured brightness temperature differences for each pixel, using the appropriate parameters and threshold. Use the nadir view 12 micron and 11 micron brightness temperatures (L1B-INT-101 and L1B-INT-102), and the forward view 11µ brightness temperatures (L1B-INT-112). For each pixel i, j perform steps (a) to (d) inclusive if

the nadir land flag [L1B-INT-232](*i, j*) is FALSE and if
the brightness temperatures used in Steps (a) and (b) are valid:

a) Derive the nadir view (11µm - 12µm) difference

$$dif\_11\_12 = I(ir11, n; i, j) - I(ir12, n; i, j)$$

b) Derive the 11µm (nadir-view - forward-view) differences

$$dif\_nv\_fv = I(ir11, n; i, j) - I(ir11, f; i, j)$$

c) By using the appropriate coefficients for the band of every pixel, given by equation 5.6.1, compute the expected 11µm (nadir-view - forward-view) differences, given by

$$exp\_dif\_nv\_fv = a0 + a1 * dif\_11\_12$$

where

a0 = [L1-AUX13-2](*band_no*);
a1 = [L1-AUX13-1](*band_no*)

d) Flag those pixels as cloudy for which the difference

abs(*exp_dif_nv_fv* - *dif_nv_fv*) > [L1-AUX13-3]

by setting L1B-INT-233 and L1B-INT-242 in the nadir view and L1B-INT-249 and L1B-INT-258 in the forward view

5.22.3.8 11/3.7 micron nadir/forward test

If night-time and the 3.7µm channel is available, repeat the preceding test, now using the brightness temperatures measured in the 3.7 µm and 11 µm channels with the appropriate parameters and threshold. Use the nadir view 11 µm and 3.7 µm brightness temperatures (L1B-INT-102 and L1B-INT-103), and the forward view 3.7 µm brightness temperatures (L1B-INT-113). The test is only applied to sea pixels.

The test is only applied to pixels on scans for which
nadir_band_centre_solar_elevation(*i*, 0) < 5.0 and
nadir_band_centre_solar_elevation(*i*, 9) < 5.0 and
frwrd_band_centre_solar_elevation(*i*, 0) < 5.0 and
frwrd_band_centre_solar_elevation(*i*, 9) < 5.0.

Step 5.22.3-67

For each scan i for which the above condition is true, and for each pixel j, perform steps (a) to (d) inclusive if
the nadir land flag [L1B-INT-232]*(i, j*) is FALSE and if
the brightness temperatures used in Steps (a) and (b) are valid:

a) Derive the nadir view (3.7µm - 11µm) difference
dif_37_11 = I(ir37, n; i, j) - I(ir11, n; i, j)

(Req 5.22-272)

b) Derive the 3.7µm (nadir-view - forward-view) difference
dif_nv_fv = I(ir37, n; i, j) - I(ir37, f; i, j)

(Req 5.22-273)

c) By using the appropriate coefficients for the band of every pixel, given by **equation 5.6.1,** compute the expected 3.7µm (nadir-view - forward-view) differences, given by

*exp_dif_nv_fv = b0 + b1 * dif_37_11 + b2 * (dif_37_11)**2*

where
b0 = [L1-AUX14-1](*band_no*);
b1 = [L1-AUX14-2](*band_no*);
b2 = [L1-AUX14-3](*band_no*).

(Req 5.22-274)

d) Flag those pixels as cloudy for which the difference

abs(*exp_dif_nv_fv - dif_nv_fv*) > [L1-AUX14-4]

by setting L1B-INT-233 and L1B-INT-243 in the nadir view and L1B-INT-249 and L1B-INT-259 in the forward view.

(Req 5.22-275)

5.22.3.9 Infrared histogram test

Step 5.22.3-68

The histogram test operates on a 512 by 512 pixel image segment, so the orbit should be divided into image segments of this size; it should be imagined as 'tiled' with 512 row image segments. Note that if (at the end of an orbit or at the end of a segment in stripline processing) the data is insufficient to form a complete image segment of 512 rows the histogram can be based on an incomplete image segment. (Because histograms are made up of those pixels which are valid sea pixels, if an image is largely over land, one has an incomplete image anyway.) Provided the number of pixels exceeds MIN_FOR_11_12_HISTOGRAM the test can be applied. It is expected that processing will always include an integer number of granules, so if there is an incomplete image segment at end of data, it will always have at least 32 rows.

For each image segment, loop over the entire segment, using control variables scan_loop (L1B-INT-340) and pixel_loop (L1B-INT-341):

```
for (scan_loop = beginning to end)
   {
    for (pixel_loop = beginning to end)
       {
                                                   (Req 5.22-276)
```

Step 5.22.3-69

Make sure each pixel has not been flagged by any previous cloud test and is not over land

```
        if (pixel_cloud_state[scan_loop][pixel_loop].cloud is FALSE
       (L1B-INT-233 or 249) and
            pixel_cloud_state[scan_loop][pixel_loop].land  is FALSE)
           (L1B-INT-232 or 248)
           {
```

Calculate the difference (L1B-INT-348) between the 11um and 12um brightness temperatures for each pixel from the input arrays ir11 (L1B-INT-112) and ir12 (L1B-INT-111).

```
                                                   (Req 5.22-277)
              ir11_ir12_diff = ir11[scan_loop][pixel_loop] -
                                      ir12[scan_loop][pixel_loop]
```

Form a histogram, histogram[MAJOR][ ] (L1B-INT-343 and L1B-INT-345) of these differences with a bin_size (L1B-INT-344) of 10 cK:

```
                                                   (Req 5.22-278)
             binned_diff = ir11_ir12_diff / bin_size   (L1B-INT-349)
(Note the above is an integer division)
             Increment histogram[MAJOR][binned_diff + 200]
```

Accumulate the 12um brightness temperatures in each histogram bin (L1B-INT-351) Requirement 2.3

```
             ir12boxtotal[binned_diff + 200] =
                                  ir12boxtotal[binned_diff + 200]
                                    + ir12[scan_loop][pixel_loop]
```

Increment the number of valid pixels

```
                                                   (Req 5.22-279)
             Increment valid_pixels  (L1B-INT-342)
           }
        else
             do nothing - it's either over land or already cloudy
       }
```

```
        end of pixel loop
    }
    end of scan loop
```

Step 5.22.3-70

If the number of pixels in the array is less than MIN_FOR_11_12_HISTOGRAM (L1-AUX15-1) then flag all so far clear pixels as cloudy, and exit.

(Req 5.22-280)

```
if (valid_pixels is less than MIN_FOR_11_12_HISTOGRAM)
   {
    for (scan_loop = beginning to end)
       {
        for (pixel_loop = beginning to end)
           {
            if (pixel_cloud_state[scan_loop][pixel_loop].cloud is
                                                       FALSE and
      pixel_cloud_state[scan_loop][pixel_loop].land is FALSE)
               {
                set pixel_cloud_state[scan_loop][pixel_loop].cloud to
                  TRUE
                set pixel_cloud_state[scan_loop][pixel_loop].
                ir11_ir12_histogram_test to TRUE
               }
           }
       }
     return
    }
```

Step 5.22.3-71

Find the position (abscissa) and value (ordinate) of the histogram mode, looping through the histogram array using control variable hist_index (L1B-INT-352):

(Req 5.22-281)

```
for (hist_index = beginning to end)
    {
     find the largest value of histogram[MAJOR][hist_index]
        {
         set peak_interval[MAJOR] to hist_index
              (L1B-INT-354) (abscissa)
         set peak_value[MAJOR] to histogram[MAJOR][hist_index]
              (L1B-INT-353) (ordinate)
        }
    }
```

Step 5.22.3-72

Find the position of a local minimum (below PEAK_FRAC_MIN (L1-AUX15-5) of the ordinate of the histogram mode) or of an empty histogram box, on the left (lower) side of the peak,using control variable lower_limit (L1B-INT-358). Set lower_limit to the value of the histogram mode and decrement.

(Req 5.22-282)

```
for (lower_limit = peak_interval[MAJOR] to beginning)
    {
     if ((histogram[MAJOR][lower_limit] is not greater than
      (PEAK_FRAC_MIN * peak_value[MAJOR]) and
         ((histogram[MAJOR][lower_limit] is 0) or
           (histogram[MAJOR][lower_limit] is less than
           histogram[MAJOR][lower_limit-1])))
        {
```

Step 5.22.3-73

Check - if the lower_limit is below 10% of the histogram mode - that the *first* trough only is genuine and is not caused by a single spike.

(Req 5.22-283)

```
         if ((histogram[MAJOR][lower_limit] is not greater than
              (0.1 * peak_value[MAJOR])) or
             (histogram[MAJOR][lower_limit] is less than
              histogram[MAJOR][lower_limit-2]) and
             (this is the first trough))
            {
              set lower_limit
            }
         }
     }
```

Step 5.22.3-74

Find the position of a local minimum on the right (higher) side of the peak in a similar way,using control variable higher_limit (L1B-INT-359). Set higher_limit to the value of the histogram mode and increment... Check the first trough for a single spike as in Step 5.22.3-73)

(Req 5.22-284)

```
for (higher_limit = peak_interval[MAJOR] to end)
    {
     if ((histogram[MAJOR][higher_limit] is as above Step 5.22.3-72
        {
         if ((histogram[MAJOR][higher_limit] is as above Step 5.22.3-
73
            {
             set higher_limit
            }
        }
     }
Save lower_limit as low_limit[MAJOR] (L1B-INT-360)
Save higher_limit as high_limit[MAJOR] (L1B-INT-361)
```

(Step 5.22.3-75 does not exist.)

(Step 5.22.3-76 does not exist.)

(Step 5.22.3-77 does not exist.)

Step 5.22.3-78

Test the validity of the MAJOR peak:

```
                                                    (Req 5.22-285)
if ((peak_interval[MAJOR] is not less than 0) or
      (image_latitude[256][256] is not less than LATITUDE_THRESHOLD)
      (L1B-INT-160 and L1-AUX15-6) or (peak_value[MAJOR] is not less
      than IR_PEAK_MIN)) (L1-AUX15-20)

    set peak_valid[MAJOR] as valid (L1B-INT-371)
else
    set peak_valid[MAJOR] as not valid
```

Step 5.22.3-79

Compute the average 12um brightness temperature, average_bt_mode[MAJOR] (L1B-INT-362), for all the pixels falling into the histogram mode box:

```
                                                    (Req 5.22-286)
    (ir12boxtotal[peak_interval[MAJOR]] / peak_value[MAJOR])
```

Step 5.22.3-80

In the night time check that the histogram is not the sum of two histograms, one having mostly clear pixels and one having mostly cloudy pixels. Check that the histogram is not excessively broad:

```
if peak_valid[MAJOR] is valid
   {
    test for night time by checking elevation angle of the sun in the
    centre of the image                             (Req 5.22-287)

    if (regridded_info[256].band_centre_solar_angles[5].
      solar_elevation(L1B-INT-124 and L1B-INT-144)
      is less than +5 degrees) then
       {
        nightime is TRUE (L1B-INT-372)
        if peak_value[MAJOR] is less than IR_PEAK_MIN then
           {
```

Step 5.22.3-81

Determine the histogram peak position to sub-bin accuracy by fitting a quadratic to the histogram values at the histogram mode and the two adjacent boxes, and computing the abscissa of the maximum of the quadratic.

```
                                                    (Req 5.22-288)
```

The general quadratic is

```
            y = Ax**2 + Bx + C
```

Solve the three equations - centred with the origin at the mode, such that x will equal -1, 0 and 1 respectively - giving:

```
y1   =  A  - B  + C
y2   =  C
y3   =  A  + B  + C,
```

then

```
set exact_peak_interval[MAJOR] (L1B-INT-356) to the
x-value (abscissa) of the quadratic
```

where dy/dx = 0

(Req 5.22-289)

```
Ensure that
          ir11_ir12_diff_at_peak[MAJOR] =
     exact_peak_interval[MAJOR] * bin_size(L1B-INT-357)
```

to keep the units as centiKelvins.

(Req 5.22-290)

Also, from the three equations above,

```
set exact_peak_value[MAJOR] (L1B-INT-355) to the
y-value (ordinate) of the quadratic
```

where dy/dx = 0

(Req 5.22-291)

Note. If any denominator goes to zero then leave ir11_ir12_diff_at_peak[MAJOR] set to peak_interval[MAJOR] * bin_size and exact_peak_value[MAJOR] to peak_value[MAJOR] (ie leave them unchanged).

Step 5.22.3-82

Compute the half-width of the histogram (ie the width of the histogram at half its height, in term of centiKelvins) to sub-box accuracy by interpolation (using the two bins either side of 0.5 * exact_peak_value[MAJOR] on both the left (low) and right (high) sides of the histogram:

(Req 5.22-292)

If (x1,y1) and (x2,y2) are two bracketing values, the interpolated values are given by:

$$x = x2 + \frac{y - y2}{y1 - y2} (x1 - x2)$$

where y will be 0.5 * exact_peak_value[MAJOR]

Set lowerhalfwidth_index (L1B-INT-370) to the left (low) side abscissa value and upperhalfwidth_index (L1B-INT-369) to the right (high) side abscissa value. Then

```
half_width[MAJOR] = (L1B-INT-368) (upperhalfwidth_index
    - lowerhalfwidth_index) * bin_size
```

Step 5.22.3-83

Calculate the half width threshold:

```
                                                 (Req 5.22-293)
```

Note. The values of HALF_WIDTH_M and HALF_WIDTH_B are View-dependent (L1-AUX15-10 or L1-AUX15-12 and L1-AUX15-11 or L1-AUX15-13.

```
            half_width_threshold[MAJOR] = (L1B-INT-367)
                HALF_WIDTH_M * ir11_ir12_diff_at_peak[MAJOR] +
                HALF_WIDTH_B
```

Step 5.22.3-84

Test the half width of the histogram peak is not too broad:

```
                                                 (Req 5.22-294)
            if half_width[MAJOR] is greater than
               half_width_threshold[MAJOR]
               set peak_valid[MAJOR] as not valid
            else
               set peak_valid[MAJOR] as valid
           }
       }
   }
```

Step 5.22.3-85

```
Generate a new histogram, histogram[MINOR][ ], (L1B-INT-346) by
setting to zero the histogram values of histogram[MAJOR][ ] between,
but not including, the two limits determined in Steps 5.22.3-72 to
5.22.3-74 inclusive (low_limit[MAJOR] and high_limit[MAJOR])
                                                 (Req 5.22-295)
```

Set the number of valid_pixels in the new histogram. If valid_pixels is less than MIN_FOR_11_12_HISTOGRAM then set peak_valid[MINOR] as not valid, otherwise investigate the new histogram.

```
                                                 (Req 5.22-296)
If peak_valid[MINOR] is valid
   {
    repeat Steps 5.22.3-71 to 5.22.3-84 inclusive, to determine the
      position of the minor peak and its limits, setting as
      appropriate:
                                                 (Req 5.22-297)
        peak_value[MINOR],
        peak_interval[MINOR],
        low_limit[MINOR],
        high_limit[MINOR],
        average_bt_mode[MINOR],
        ir11_ir12_diff_at_peak[MINOR],
        exact_peak_value[MINOR],
        exact_peak_interval[MINOR],
        half_width[MINOR],
        half_width_threshold[MINOR], and
        peak_valid[MINOR]
   }
```

Step 5.22.3-86

```
If peak_valid[MINOR] is valid
   {                                                          (Req 5.22-298)
```

Step 5.22.3-87

```
if high_limit[MINOR] - low_limit[MINOR] is not greater than 2
histogram bins
   {
    set peak_valid[MINOR] to not valid
   }                                                           Req 5.22-299)
```

Step 5.22.3-88

Test if the major peak is highly likely to be cloudy, and the minor peak is not sufficiently distinct from the major peak:

```
                                                              (Req 5.22-300)
if peak_interval[MINOR] is greater than peak_interval[MAJOR] and
 (average_bt_mode[MINOR] - average_bt_mode[MAJOR]) is greater
 than MAX_DIF_PEAK_CHAN_1 and (L1-AUX15-15)
   low_limit[MINOR] is less than high_limit[MAJOR] and
   histogram[MAJOR][high_limit[MAJOR]] is greater than
   (SECOND_LOW_FRACTION * peak_value[MINOR]) (L1-AUX15-9)
   {
    set peak_valid[MINOR] to not valid
   }
```

Step 5.22.3-89

Test if the minor peak is highly likely to be cloudy:

```
                                                              (Req 5.22-301)
if peak_interval[MINOR] is less than peak_interval[MAJOR] and
  (average_bt_mode[MAJOR] - average_bt_mode[MINOR]) is greater
  than MAX_DIF_PEAK_CHAN_1
    {
     set peak_valid[MINOR] to not valid
    }
```

Step 5.22.3-90

It is possible that there are no empty histogram boxes between the minor and major histogram peaks. If the minor peak is valid then check that the histogram at the lower brightness temperature differences does not contaminate the other one beyond the latter's lower limit

```
                                                              (Req 5.22-302)
     if peak_valid[MINOR] is valid and peak_interval[MINOR] is
greater than peak_interval[MAJOR]
        {
         if low_limit[MINOR] is less than high_limit[MAJOR]
            {
             set low_limit[MINOR] to high_limit[MAJOR]
            }
```

Step 5.22.3-91

```
if high_limit[MAJOR] - peak_interval[MAJOR] is greater than
3 histogram bins
```

```
                {
                  Extrapolate high_limit[MAJOR] to the x-axis. - using
                  peak_value[MAJOR] and the ordinate of the bin
                  immediately to the left (low) side of
                  high_limit[MAJOR]                      (Req 5.22-303)
```

If (x1,y1) are the histogram mode abscissa and ordinate, while (x2,y2) are the co-ordinates of the adjacent bin (referred to above), the linearly extrapolated value of x when y = 0 is:

$$x = x1 + \frac{x2 - x1}{y1 - y2} \, y1$$

```
                }
            else
                {
```

Step 5.22.3-92

```
         Extrapolate high_limit[MAJOR] to the x-axis (as
         above) - using peak_value[MAJOR] and the ordinate
         of high_limit[MAJOR] itself Requirement 9.3.1.3.1
                }


            Set high_limit[MAJOR] to its new extrapolated value.
                                                         (Req 5.22-304)
```

Step 5.22.3-93

```
            if low_limit[MINOR] is less than high_limit[MAJOR]
               {
                set low_limit[MINOR] to high_limit[MAJOR] or to
                 (peak_interval[MINOR] - 1 histogram bin),
                 whichever is the lower                  (Req 5.22-305)
               }
           }
           (End of if clause that began following Req. 5.22-302)
}
   (End of if clause that began at Req 5.22-298)
```

Step 5.22.3-94

```
If peak_valid[MINOR] is not valid and peak_interval[MINOR] is less
than peak_interval[MAJOR]
   {
```

Step 5.22.3-95

```
    if (high_limit[MINOR] - peak_interval[MINOR] is greater than 3
      histogram bins
       {
        Extrapolate high_limit[MINOR] to the x-axis - using
        peak_value[MINOR] and the ordinate of the bin second to the
```

```
          left (low) side of high_limit[MINOR].  Use the formula given
          above accordingly.                                (Req 5.22-306)
        }
      else
        {
```

Step 5.22.3-96

```
    Extrapolate high_limit[MINOR] to the x-axis (as above) -
          using peak_value[MINOR] and the ordinate of the bin
          immediately to the left (low) side of high_limit[MINOR]
                                                 Requirement 9.4.2.1
        }
      Set high_limit[MINOR] to its new extrapolated value.
                                                            (Req 5.22-307)
```

Step 5.22.3-97

```
      if low_limit[MAJOR] is less than high_limit[MINOR]
        {
          set low_limit[MAJOR]  to  high_limit[MINOR]   or to
          (peak_interval[MAJOR] + peak_interval[MINOR]) / 2),
          whichever is the lower                            (Req 5.22-308)
        }
      }
      (End of if clause that began at Step 5.22.3-94)
```

Step 5.22.3-98

```
    if peak_interval[MINOR] is greater than peak_interval[MAJOR] and
        (average_bt_mode[MINOR] - average_bt_mode[MAJOR]) is greater
        than MAX_DIF_PEAK_CHAN_1
            {
             set peak_valid[MAJOR] to not valid
            }
```

Step 5.22.3-99

Choose between the MAJOR and MINOR histograms:

```
      If both peaks are (still) 'valid' then
         choose the one with the lower value.

      If there is only one 'valid' peak then
         choose it.

      If there is no 'valid' peak then
         flag all so-far clear pixels as cloudy and exit

      Replace the [MAJOR],[MINOR] nomenclature with whichever is
         [CHOSEN] (L1B-INT-347)                             (Req 5.22-309)
```

Step 5.22.3-100

Check that, for the peak chosen, the difference between the corresponding histogram mode and lower limit is not greater than IR_SPREAD.

(Req 5.22-310)

Note. The value of IR_SPREAD is View-dependent (L1-AUX15-17 or L1_AUX15-18))

```
if ((peak_interval[CHOSEN] - low_limit[CHOSEN]) * bin_size) is
      greater than IR_SPREAD
   {
    set low_limit[CHOSEN] to peak_interval[CHOSEN] -
      (IR_SPREAD / bin_size)
   }
```

Step 5.22.3-101

Check the way the 12um brightness temperature varies with the (11um minus 12um) difference. Under normal conditions, it decreases as the difference increases.

Average the 12um brightness temperature values for the pixels in each histogram bin:

(Req 5.22-311)

```
for (hist_index = beginning to end)
   {
    average_bt[hist_index] = (float) (ir12boxtotal[hist_index] /
      histogram[CHOSEN][hist_index] (L1B-INT-363))
   }
```

Step 5.22.3-102

Find the histogram bin which has the highest average 12um brightness temperature and set it to highest_av_bt_box[CHOSEN] (L1B-INT-364)

(Req 5.22-312)

Step 5.22.3-103

Compute the slope of the average 12um brightness temperatures with respect to the histogram at the histogram peak: (The slope is defined as +ve if uphill from left to right. The slope (L1B-INT-365) is measured using the difference between the *next* point and the current point.)

(Req 5.22-313)

```
    slope_at_peak =
     (average_bt[peak_interval[CHOSEN] + 1] -
      average_bt[peak_interval[CHOSEN]]) / bin_size
```

Step 5.22.3-104

If slope_at_peak exceeds threshold SLOPE_MAX_ALLOWED (L1-AUX15-19) then flag all so-far clear pixels as cloudy, and exit.

(Req 5.22-314)

Step 5.22.3-105

At night, use the 12um average brightness temperatures to tighten the lower limit, if necessary:

(Req 5.22-315)

```
if (nightime)
   {
    if peak_interval[CHOSEN] is greater than
      highest_av_bt_box[CHOSEN]
       {
        for hist_index = low_limit[CHOSEN] to the end
           {
            if histogram[CHOSEN][hist_index] is not less than
      (1 /RATIO_B)*histogram[CHOSEN][highest_av_bt_box[CHOSEN]]
                                                       (L1-AUX15-16)
                {
                 re-set low_limit[CHOSEN] to this value
                 and exit to Step 3.108
                }
           }
       }
    else
       {
```

Step 5.22.3-106

Compute slopes of the 12um brightness temperatures with respect to the chosen histogram for each non-empty histogram box and store in array slope[ ] (L1B-INT-366) (The slope is defined as +ve if uphill from left to right. The slope is measured using the difference between the *next* point and the current point.)

(Req 5.22-316)

Step 5.22.3-107

```
        set hist_index to the current value of low_limit[CHOSEN]


        while ((((average_bt[hist_index] + MAX_DIF_AVE_CHAN_1)
            (L1-AUX15-14)is less than average_bt_mode[CHOSEN]) or
            (slope[hist_index] is greater than SLOPE_MAX_ALLOWED))
              and (hist_index is less than peak_interval[CHOSEN]))
                {
                 Increment hist_index
                }
        re-set low_limit[CHOSEN] to this value     (Req 5.22-317)
       }
   }
```

Step 5.22.3-108

Flag as cloudy all, so far clear pixels which have a (11um minus 12um) brightness temperature difference value less than the value of low_limit[CHOSEN], determined above.

(Req 5.22-318)

Step 5.22.3-109

```
If the number of valid_pixels remaining is less than threshold value
      MIN_FOR_11_12_HISTOGRAM then
   {
      flag all remaining clear pixels as cloudy.
                                                        (Req 5.22-319)
   }
```

5.22.3.10 Visible channel cloud test

This test operates on a pixel-by-pixel basis, and applies to daytime pixels only. It is applied to each view separately.

The following steps are applied for each view ($v = n \mid f$).

Step 5.22.3-110. Extract solar elevations

For each scan *i*, retrieve the solar elevation at each end of the scan. The test is only performed if these are not less than 5°: i.e. if
<view>*_band_centre_solar_elevation*(*i*, 0) ≥ 5.0 or
<view>*_band_centre_solar_elevation*(*i*, 9) ≥ 5.0,
so excluding night-time data.

(Req 5.22-320)

Step 5.22.3-111. Calculate normalized difference indices

For each day-time pixel *j* identified in step (a) above for which the visible channel reflectance values are valid, i.e. for which $I(ch, v; i, j) > 0$ for $ch = v870, v670, v555$ calculate the normalised difference indices as follows:

$$NDVI = (I(v870, v; i, j) - I(v670, v; i, j))/(I(v870, v; i, j) + I(v670, v; i, j))$$

(Req 5.22-321)

$$NDI2 = (I(v670, v; i, j) - I(v555, v; i, j))/(I(v670, v; i, j) + I(v555, v; i, j))$$

(Req 5.22-322)

Step 5.22.3-112. Test to see if pixel falls within clode zone

Perform the following procedure to identify within which cluster defined by *NDVI* and *NDV2* the pixel falls.

For each zone $iz = 0, N\_ZONE - 1$

Extract array of vertices:

$vertex[k] = v[k, i\_zone], k = 0, 4.$

$N\_SIDES = 5$

If $v[5] < 0$ then $N\_SIDES = N\_SIDES - 1$; (*N_SIDES* is the number of sides, equal to the number of vertices, of the polygon defining the zone. It will not be less than 4.)

(Req 5.22-323)

Define arrays of dimension N_SIDES + 1 and extract the vertex co-ordinates into them:

For $k = 0, N\_SIDES - 1$

$X[k] = [\text{L1-AUX31-1}](k)$

$Y[k]$ = [L1-AUX31-2](k)

end for

(Req 5.22-324)

$X[N\_SIDES] = X[0]$

$Y[N\_SIDES] = Y[0]$

(Req 5.22-325)

Identify whether the point PX, PY lies within this zone.

$FLAG = TRUE$

$PX = NDI2$

$PY = NDVI$

(Req 5.22-326)

For $k = 0$, $N\_SIDES - 1$

$SX = X[k + 1] - X[k]$

$SY = Y[k + 1] - Y[k]$

(Req 5.22-327)

$QX = PX - X[k]$

$QY = PY - Y[k]$

(Req 5.22-328)

Calculate the vector cross product Q × S:

$ZZ = (QX \times SY - QY \times SX)$

(Req 5.22-329)

$FLAG = FLAG$ AND $(ZZ \geq 0)$

(Req 5.22-330)

If $FLAG = FALSE$ then exit loop; the point is not in this zone

End for

If $FLAG = TRUE$, exit loop ; we have found the zone

(Req 5.22-331)

End for

Step 5.22.3-113. Flag pixel

The table of zones will only contain those regions that are associated with cloud. Thus the value of *FLAG* at this point tells whether or not the pixel is cloudy.

If *FLAG = TRUE* and the pixel is a land pixel for which the land flag (L1B-INT-232 in the nadir view and L1B-INT-248 in the forward view) is set then flag the pixel as cloudy by setting L1B-INT-233 and L1B-INT-245 in the nadir view, or L1B-INT-249 and L1B-INT-261 in the forward view.

(Implementation note: although this algorithm in theory works over sea as well as land, use over sea is disabled for the time being. This is done in this step rather than by applying the test as a whole to land pixels only so that it can be easily enabled for sea pixels if desired at some time in the future.)

(Req 5.22-332)

5.22.3.11 Snow-covered surface test

This test operates on a pixel-by-pixel basis, and applies to daytime pixels only. It is applied to each view separately.

The following steps are applied for each view ($v = n \mid f$).

Step 5.22.3-114. Extract solar elevation

For each scan *i*, retrieve the solar elevation at each end of the scan. The test is only performed if these are not less than 5°: i.e. if
<view>*_band_centre_solar_elevation*(*i*, 0) ≥ 5.0 or
<view>*_band_centre_solar_elevation*(*i*, 9) ≥ 5.0,
so excluding night-time data.

(Req 5.22-333)

Step 5.22.3-115. Calculate NDSI

For each day-time pixel *j* identified in step (5.22.3-114) above for which the 1.6, 0.87 and 0.55 micron channel reflectance values are valid, i.e. for which $I(ch, v; i, j) > 0$ for $ch = v16, v870, v555$ calculate the normalised difference snow index *NDSI* as follows:

$$NDSI = (I(v555, v; i, j) - I(v16, v; i, j))/(I(v555, v; i, j) + I(v16, v; i, j))$$

(Req 5.22-334)

Step 5.22.3-116. Correct 0.86 micron reflectance for solar incidence angle

The across-track band number for each pixel in the regridded image is given by

```
band(j) = 0 IF j < 6
band(j) = integer part of (j - 6 ) / 50  IF  6 ≤ j < 506
band(j) = 9 IF j ≥ 506
```

where j is the pixel across track index.

(Req 5.22-335)

A linear interpolation may be used to determine the satellite elevation.

$$w = float(j - 6)/50.0 - band(j)$$

$$sol\_elev = (1.0 - w) \times nadir\_band\_edge\_solar\_elevation(i, band(j)) + w \times nadir\_band\_edge\_solar\_elevation(i, band(j) + 1)$$

(Req 5.22-336)

The solar zenith angle in radians is $z = \pi(90 - sol\_elev)/180.0$. The calibrated reflectance at 0.87 microns, corrected to normal solar incidence, is then

$$R87 = (I(v870, v; i, j) \times \sec(z)$$

(Req 5.22-337)

Extract the 11 micron brightness temperature:

T11 = *I(ir11, v; i, j)*

(Req 5.22-338)

Step 5.22.3-117. Snow flag

Convert the units of the NDSI threshold: NDSI_THRESHOLD = NDSI_THRESH / 10000.

If *T11* < *T11_THRESH and* (*R*87 > R87*_THRESHOLD and NDSI* > *NDSI_THRESHOLD),* the pixel is considered snow-covered. Flag the pixel accordingly by setting L1B-INT-246 in the nadir view, and L1B L1B-INT-262 in the forward view.

(Req 5.22-339)

5.23 Module Definition: Generate Browse Product

5.23.1 Functional Description

The AATSR BROWSE product is to comprise a subsampled image at 4 km resolution based on the nadir view brightness temperature or reflectance images, together with associated subsampled cloud flags. A coding scheme is to be used to derive a 3-colour composite image from the channel values and associated cloud flags. The cloud flags themselves will not be retained in the product; cloud will be identified visually.

The browse product is false-colour image generated from the AATSR data set and is intended to provide the user with a quick-look illustrating the image contents (ie. the land, sea, cloud etc.). The product will be partitioned into a day form that exploits the visible channel data available on the illuminated part of the orbit, and a night form derived using the 11 µm channel brightness temperatures in conjunction with the image segmentation information available with the GBT products.

Daytime browse product

For the daytime browse product there are several different channel combinations that could be used to generate this product, but, for the sake of compatibility with existing systems run at ESRIN, the scheme that will be used in the IONIA AVHRR browser described by Melinotte and Arino (Melinotte, J.-M and O. Arino, The IONIA 1-km Net-Browser Experience - Quicklook Processing and Access Statistics, Earth Observation Quarterly, 50, pp 6-10, 1995). This scheme generates a quick-look which is a three-colour composite of two AATSR visible channels and a third channel which is a brightness temperature; the red channel is given by the 0.67 µm channel, the green channel is given by the 0.87 µm, and the blue band is the the inverted 11µm channel (ie. light is cold and dark is hot). These data are then translated via a look-up table to corresponding values of red, green and blue (RGB). In the IONIA scheme these LUTS have been chosen to enhance certain features of the data. They are basically a series of linear ramps between specified knots. The colour table is interpolated between the knots, in a way that doesn't maintain the gradient (ie. there may be discontinuous changes across the knots).

Nighttime browse product

Nighttime browse product will be based on the AATSR 11µm channel. This will give rise to a monochromatic image. Transition processing is applied to the day-time scans to reduce the visual discontinuity at the day-night transition.

5.23.2 Interface definition

Parameter ID	Variable	Name	Type	Units	Field size	Fields
BR-AUX1-1	N_red	Number of red coefficients in LUT	ss	n/a	2	1
BR-AUX1-2		Red channel reflectance	ss	0.01 %	2	10
BR-AUX1-3		Red channel colour coefficients	uc	n/a	1	10
BR-AUX1-4	N_green	Number of green coefficients in LUT	ss	n/a	2	1
BR-AUX1-5		Green channel reflectance	ss	0.01 %	2	10
BR-AUX1-6		Green channel colour coefficients	uc	n/a	1	10
BR-AUX1-7	N_blue	Number of blue coefficients in LUT	ss	n/a	2	1
BR-AUX1-8		Blue channel brightness temperature	ss	0.01 K	2	10
BR-AUX1-9		Blue channel colour coefficients	uc	n/a	1	10
L1-AUX16-15	NGRANULE	Number of image rows per granule	sl	none	4	1

Table 5-23-1: Input Data Table - Generate Browse Product

Parameter ID	Variable	Name	Type	Units	Field size	Fields
L1B-INT-102	I(ir11, n; i, j)	regridded nadir ir11 Brightness Temp.	ss	0.01K	2	j = 0, 511
L1B-INT-105	I(v870, n; i, j)	regridded nadir v870 Reflectance	ss	0.01%	2	j = 0, 511
L1B-INT-106	I(v670, n; i, j)	regridded nadir v670 Reflectance	ss	0.01%	2	j = 0, 511
local	IB[ir11; ib, jb]	Sub-sampled ir11 Brightness Temperature	ss	0.01K	2	jb = 0, 127
local	IB[v870; ib, jb]	Sub-sampled v870 reflectance	ss	0.01%	2	jb = 0, 127
local	IB[v670; ib, jb]	Sub-sampled v670 reflectance	ss	0.01%	2	jb = 0, 127
L1B-INT-120		nadir_band_edge_solar_elevation(i, k)	float	degrees	4	k = 0, 10
local	Red[ib, jb]	Red channel intensity	uc	intensity	1	jb = 0, 127
local	Blue[ib, jb]	Blue channel intensity	uc	itnensity	1	jb = 0, 127
local	Green[ib, jb]	Green channel intensity	uc	intensity	1	jb = 0, 127
local	C[ib, jb]	Notation for current channel intensity	uc	intensity	1	jb = 0, 127
	j	nadir pixel index (j = 0, 1, ...511)	sl	none	4	
local	jb	browse product pixel index (jb = 0, 1, 127)	sl	none	4	
	i	image scan index	sl	none	4	
local	i′	Index of image row within granule				
local	i_0	Index of first row of granule				
local	ib	browse product image scan index	sl	none	4	
local	V	Current brightness temperature/reflectance	ss	0.01K / 0.01%	2	
local	V_ref(index)	Reference brightness temperature/reflectance	ss	0.01K / 0.01%	2	
local	coeff(index)	corresponding tabular colour coefficient	uc	n/a	1	
local	index	index to colour conversion table	sl	none	4	1
local	k	index into histogram arrays	sl	none	4	1
local	F(k)	Histogram array	sl	none	4	256
local	h(k)	Cumulative histogram array	sl	none	4	256
local	T_MIN_LAND	minimum temperature over clear land	ss	0.01K	2	1
local	T_MAX_LAND	maximum temperature over clear land	ss	0.01K	2	1
local	T_MIN_SEA	minimum temperature over clear land	ss	0.01K	2	1
local	T_MAX_SEA	maximum temperature over clear land	ss	0.01K	2	1
local	T_MIN_CLOUD	minimum temperature over cloud	ss	0.01K	2	1
local	T_MAX_CLOUD	maximum temperature over cloud	ss	0.01K	2	1
local	LAND_GRAD	Scale factor for pixels over clear land	float	$(0.01K)^{-1}$	2	1
local	SEA_GRAD	Scale factor for pixels over clear sea	float	$(0.01K)^{-1}$	2	1

local	CLOUD_GRAD	Scale factor for cloudy pixels	float	$(0.01K)^{-1}$	2	1

Table 5-23-2: Internal Data Table - Generate Browse Product

Parameter ID	Field Number	Name	Type	Units	Field Size	Fields
BRWS-MDS1-1	1	Nadir UTC time in MJD format	sl, 2*ul	MJD	12	1
BRWS-MDS1-2	2	Record Quality Indicator	sc	n/a	1	1
BRWS-MDS1-3	3	Spare	uc	n/a	3	1
BRWS-MDS1-4	4	Image scan y coordinate	sl	km	4	1
BRWS-MDS1-5	5 - 132	Red, Green, Blue channel intensities, interleaved	3*uc	intensity	3	128

Table 5-23-3: Output Data Table - Generate Browse Product

5.23.3 Algorithm Definition

The Browse Processing steps are as follows.

The nadir view data is extracted from each granule in turn until the complete orbit has been processed. The following steps are applied to each granule:

Step 5.23.1 Select sub-sampled image

Pixels are selected from every 4th image record in granule by copying each 4th pixel along the nadir record into the browse arrays thus:

```
FOR i' = 0, NGRANULE - 1 in steps of 4
  ib = i' / 4
  FOR j = 0, 511 in steps of 4
    jb = j / 4
    i = i0 + i'
    where i0 is the index of the first row of the granule.
    IB[ir11; ib, jb] = I(ir11, n; i, j)
    IB[v670; ib, jb] = I(v670, n; i, j)
    IB[v870; ib, jb] = I(v870, n; i, j)
  END FOR (over pixels)
END FOR (i loop over all the scans)
                                                  (Req 5.23-1)
```

Step 5.23.2 Main loop

Define the following constants:

```
SOLAR_ANGLE_NIGHT = 5.0 degrees
SOLAR_ANGLE_DAY   = 6.0 degrees                   (Req 5.23-2)


FOR ib = 0, 7
FOR jb = 0, 127
current_solar_angle =
  nadir_band_edge_solar_elevation(i, ACROSS_TRACK_BANDS_COUNT/2)
                                                  (Req 5.23-3)
```

```
if (IB[ir11;ib,jb] > 0) OR (IB[ir11;ib,jb] == PIXEL_COUNT_SATURATED)
{
  if ( current_solar_angle >= SOLAR_ANGLE_NIGHT )     // daytime
  {
    if ( IB[v870;ib,jb] > 0 ) AND ( IB[v670;ib,jb] > 0 )
    {
      Proceed with Day time Processing as at Step 5.23.2.1 below;
    }                                                 (Req 5.23-4)
  }
  else // nighttime
  {
    Proceed with Night time Processing as at Step 5.23.2.2 below;
  }                                                   (Req 5.23-5)
  if (IB[ir11;ib,jb] == PIXEL_COUNT_SATURATED )
  {
    Blue[line][pixel] = 0;
  }                                                   (Req 5.23-6)
  if ( ( SOLAR_ANGLE_NIGHT <= current_solar_angle ) &&
       ( current_solar_angle <= SOLAR_ANGLE_DAY ) )
  { // Smoothing Day-Night transition as at Step 5.23.3 below

    float hue, saturation, value;
    RGB_to_HSV( Red[ib,jb],
                Green[ib,jb],
                Blue[ib,jb],
                hue,
                saturation,
                value );
                                                      (Req 5.23-7)
    float adjusted_saturation;
    adjust_saturation( current_solar_angle,
                       saturation,
                       adjusted_saturation );
                                                      (Req 5.23-8)
    HSV_to_RGB( hue,
                adjusted_saturation,
                value,
                Red[ib,jb],
                Green[ib,jb],
                Blue[ib,jb] );
  }                                                   (Req 5.23-9)
}

Where the functions RGB_to_HSV(), adjust_saturation and HSV_to_RGB()
are defined below.
```

Step 5.23.2.1 Day time Processing

Day time processing generates a false colour composite from 2 visible channels plus one infrared. It is applied to each scan opf the sub-sampled image for which the solar elevation at the centre of the scan,

nadir_band_edge_solar_elevation(i, 5) ≥ 5.0

(Req 5.23-10)

Step 5.23.2.1 Day time Processing

Calculate the pixel intensities for each color (red, green, blue), in function of the calibrated reflectances or brightness temperatures, using a pre-defined reflectance/temperature lookup table. The auxiliary file ATS_BRW_AX defines a look-up table for each basic color. The 3 possible conversion functions are called LUT_RED(), LUT_GREEN(), LUT_BLUE().

Interpolate and calculate the pixel intensities:

```
Red[ib, jb] = LUT_RED( IB[v670; ib, jb] )
Green[ib, jb] = LUT_GREEN( IB[v870; ib, jb] )
Blue[ib, jb] = LUT_BLUE( IB[ir11; ib, jb] )
```

The transformation in each case is as follows.

(Req 5.23-11)

Step 5.23.2.1.1 Generate RGB Values

The channel conversion is carried out as follows. For each colour (e.g. red) the look-up table provides a set of N_<colour> coefficient pairs (knots) as follows:

V_ref(index), reference brightness temperature/reflectance, as appropriate;
coeff(index), corresponding colour value;

for i = 0, N_<colour> - 1. Thus for the red channel,

V_ref(index) = [BR-AUX1-2](index);
coeff(index) = [BR-AUX1-3](index).

Then given the channel reflectance (or brightness temperature, as appropriate) *V*, we interpolate as follows.

```
if V ≤ V_ref(0) then
      C[ib, jb] = coeff(0)
else if V > V_ref(N_<colour>-1)
      C[ib, jb] = coeff(N_<colour>-1)
else
      for index = 0, N_<colour>-1
      if V ≤ V_ref(index) then exit
      end for
C[ib, jb] = nearest integer to
           {[coeff(index) *(V - V_ref(index-1)) +
             coeff(index - 1) * (V_ref(index) - V)] /
            (V_ref(index) - V_ref(index - 1))}
endif
```

rounded to nearest integer.

In the above T is T(ib, jb) and colour is R(ib, jb). The Green and blue channels are converted similarly.

(Req 5.23-12)

Step 5.23.2.2 Night-Time Processing

Night time processing is based on use of the 11 µm channel. It is performed when

nadir_band_edge_solar_elevation(i, 5) < 5.0

(Req 5.23-13)

Calculate the pixel intensities for each color (red, green, blue), in function of the calibrated reflectances or brightness temperatures, using a pre-defined reflectance/temperature lookup table. The auxiliary file ATS_BRW_AX defines a look-up table for each basic color. The 3 possible conversion functions are called LUT_RED(), LUT_GREEN(), LUT_BLUE().

Interpolate and calculate the pixel intensities:

```
Red[ib, jb] = LUT_BLUE( IB[ir11; ib, jb] )
Green[ib, jb] = LUT_BLUE( IB[ir11; ib, jb] )
Blue[ib, jb] = LUT_BLUE( IB[ir11; ib, jb] )
```

Step 5.23.3 Function definitions: RGB_to_HSV(), HSV_to_RGB() and adjust_saturation()

Definition of RGB_to_HSV():

```
r, g and b in [0,1], h in [0,360], s and v in [0,1]

RGB_to_HSV (r,g,b: float; h,s,v: float)
{
  max = Maximum(r,g,b);
  min = Minimum(r,g,b);
  v = max;
  if (max != 0)
  {
    s = (max - min)/max;
  }
  else
  {
    s = 0;
  }
  if (s == 0)
  {
    h = UNDEFINED;
  }
  else
  {
    delta = max - min;
    if (r == max)
    {
      h = (g - b) / delta;
    }
    else
    {
      if (g == max)
```

```
        {
          h = 2 + (b - r) / delta;
        }
        else
        {
          if (b == max)
          {
            h = 4 + (r - g) / delta;
          }
        }
      }
      h = h * 60;
      if (h < 0)
      {
        h = h + 360;
      }
    }
}
```

Definition of HSV_to_RGB():

r, g and b in [0,1], h in [0,360], s and v in [0,1]

```
HSV_to_RGB (r,g,b: float; h,s,v: float)
{
  if (s == 0)
  {
    r = v;
    g = v;
    b = v;
  }
  else
  {
    if (h == 360)
    {
      h = 0;
    }
    h = h / 60;
    i = Floor (h); /* Largest integer <= h */
    f = h - I;
    p = v * (1 - s);
    q = v * (1 - (s * f));
    t = v * (1 -(s * (1 -f)));
    switch (i)
    {
      case 0: (r,g,b) = (v,t,p);
```

```
        case 1: (r,g,b) = (q,v,p);
        case 2: (r,g,b) = (p,v,t);
        case 3: (r,g,b) = (p,q,v);
        case 4: (r,g,b) = (t,p,v);
        case 5: (r,g,b) = (v,p,q);
      }
    }
  }
```

Definition of adjust_saturation():

```
adjust_saturation( current_solar_angle,
                   s_in,
                   s_out)
{
  s_out = s_in * ( current_solar_angle - SOLAR_ANGLE_NIGHT )
          / ( SOLAR_ANGLE_DAY - SOLAR_ANGLE_NIGHT )
}
```

Step 5.23.4 Write Browse Product Records to Output

Step 5.23.4.1 Product Headers.

The MPH is to be prepared as in document PO-RS-MDA-GS-2009.

The SPH is identical to that of the GBTR product (Section 24) as far as the data fields are concerned. DSDs shall be supplied as specified in the IODD.

Step 5.23.4.2. Summary Quality ADS.

The Summary Quality ADS is to be identical to that prepared for the GBTR (Section 24).

Step 5.23.4.3. ADS #1.

This ADS is to be identical to ADS #3 prepared for the GBTR (Section 24), except for the omission of the topographic correction fields [GBTR-ADS3-7 to 12].

Step 5.23.4.4. MDS #1

The single measurement data set for this product is prepared as follows.

[BRWS-MDS1-1] = (Identical to GBTR)

[BRWS-MDS1-2] = (Identical to GBTR)

[BRWS-MDS1-3] = (Identical to GBTR)

[BRWS-MDS1-4] = (Identical to GBTR)

[BRWS-MDS1-5](ib, jb) = {R[ib, jb], G[ib, jb], B[ib, jb]}, (jb = 0, 127)

5.24 Module Definition: Output GBTR Records

5.24.1 Functional Description

The GBTR product is written to the output medium. First the SPH and ADS records are output, then for each image line, an MDS record is assembled and written.

5.24.2 Interface Definition

See IODD Tables and internal parameter list.

5.24.3 Algorithm Definition

Step 5.24.3-1 MPH record.

A standard Main Product Header (MPH) shall be prepared and written to the output medium.

Step 5.24.3-2 SPH record.

Suitable DSD records for the ADS and MDS defined below should be prepared, as per PO-RS-MDA-GS-2009, and appended to the SPH of the level 1b product. Formatting considerations as defined in the reference should be followed; in particular output parameters must be converted to ASCII format as specified therein.

In the following, first_i and last_i are the indices of the first and last image rows in the product.

DSDs as per as per PO-RS-MDA-GS-2009.

Note that the Visible Channel Calibration Data auxiliary file [L1-AUX17] may contain in its SPH a second DSD following the DSD that points to its Data Set. This second DSD is a reference DSD that refers to the Visible Calibration Drift Table file used in its generation. If present, this reference DSD should be copied to the final DSD field [GBTR-SPH-86] of the product SPH, formerly a spare DSD.

```
[GBTR-SPH-5] = MIN(min_aux_temp(i, 0))                (Req 5.24-14)
[GBTR-SPH-8] = MIN(min_aux_temp(i, 1))                (Req 5.24-15)
[GBTR-SPH-11] = MIN(min_aux_temp(i, 2))               (Req 5.24-16)
[GBTR-SPH-14] = MIN(min_aux_temp(i, 3))               (Req 5.24-17)
[GBTR-SPH-17] = MIN(min_aux_temp(i, 4))               (Req 5.24-18)
[GBTR-SPH-20] = MIN(min_aux_temp(i, 5))               (Req 5.24-19)
[GBTR-SPH-23] = MAX(max_aux_temp(i, 0))               (Req 5.24-20)
[GBTR-SPH-26] = MAX(max_aux_temp(i, 1))               (Req 5.24-21)
[GBTR-SPH-29] = MAX(max_aux_temp(i, 2))               (Req 5.24-22)
[GBTR-SPH-32] = MAX(max_aux_temp(i, 3))               (Req 5.24-23)
[GBTR-SPH-35] = MAX(max_aux_temp(i, 4))               (Req 5.24-24)
[GBTR-SPH-38] = MAX(max_aux_temp(i, 5))               (Req 5.24-25)
```

In the above, the notation MIN(<view>_min_aux_temps(i, jaux)) refers to the minimum value of the set of values

```
{nadir_min_aux_temps(i, jaux), frwrd_min_aux_temps(i, jaux), all i}
```

Similarly the notation MAX(<view>_max_aux_temps(i, jaux)) refers to the minimum value of the set of values

```
{nadir_max_aux_temps(i, jaux), frwrd_max_aux_temps(i, jaux), all i}
```

The logical generation of these quantities is as follows:

```
for jaux = 0, 5
Initialise:
      min_aux_temp[jaux] = 999.0
      max_aux_temp[jaux] = -999.0
end for (jaux)
for jaux = 0, 5
for <view> = nadir, frwrd
for i = first_i, last_i
Update min and max temps as follows:
min_aux_temp(jaux) = smaller of (<view>_min_aux_temps(i, jaux),
min_aux_temp(jaux))
max_aux_temp(jaux) = greater of (<view>_max_aux_temps(i, jaux),
max_aux_temp(jaux))
end for (i)
end for (<view>)
end for (jaux)
```

Then

```
MIN(<view>_min_aux_temps(i, jaux)) = min_aux_temp(jaux)
MAX(<view>_max_aux_temps(i, jaux)) = max_aux_temp(jaux)


[GBTR-SPH-41] = cst(x coord. of lat/long tie points):
LAT_LONG_TIE_POINTS={-275, -250, -225, -200, -175, -150,
      -125, -100, -75, -50, -25, 0, 25, 50, 75, 100, 125, 150,
      175, 200, 225, 250, 275}                      (Req 5.24-26)
[GBTR-SPH-44] = cst(x coord.of solar angle tie-points):
VIEW_ANGLE_TIE_POINTS={-250, -200, -150, -100, -50, 0,
      50, 100, 150, 200, 250}                       (Req 5.24-27)
[GBTR-SPH-47] = character string(pixel numbers of x-y tie points):
      {FIRST_NADIR_PIXEL_NUMBER + (0, (INT_P), MAX_NADIR_PIXELS-1);
      FIRST_FORWARD_PIXEL_NUMBER + (0, (INT_P), MAX_FRWRD_PIXELS-1)}
                                                    (Req 5.24-28)
SPH_DESCRIPTOR=??
STRIPLINE_CONTINUITY_INDICATOR=??
SLICE_POSITION=??
NUM_SLICES=??
FIRST_LINE_TIME= nadir_time(first_i) converted to external string
format.
LAST_LINE_TIME= nadir_time(last_i) converted to external string
format.
(Note that nadir_time is calcuated as at Step 5.24-69, and that the
timer in external string format is available from the CFI subroutine
call.)
FIRST_FIRST_LAT= image_latitude(first_i, 0) <10-6degN>
FIRST_FIRST_LONG= image_longitude(first_i, 0) <10-6degN>
FIRST_MID_LAT= image_latitude(first_i, 256) <10-6degN>
FIRST_MID_LONG=image_longitude(first_i, 256) <10-6degN>
FIRST_LAST_LAT= image_latitude(first_i, 511) <10-6degN>
FIRST_LAST_LONG= image_longitude(first_i, 511) <10-6degE>
LAST_FIRST_LAT= image_latitude(last_i, 0) <10-6degN>
LAST_FIRST_LONG= image_latitude(last_i, 0)<10-6degE>
```

```
LAST_MID_LAT= image_latitude(first_i, 256) <10-6degN>
LAST_MID_LONG= image_latitude(last_i, 256) <10-6degE>
LAST_LAST_LAT= image_latitude(first_i, 511) <10-6degN>
LAST_LAST_LONG= image_latitude(last_i, 511) <10-6degE>
```

Step 5.24.3-2.1 Summary Quality ADS (ADS #0).

Note that all ADS records correspond to a product granule except for ADS#0. In the case of ADS #0 each record corresponds to an image segment of 512 rows (512/*NGRANULE* granules). Thus in this step the phrase ‘number of scans’ refers to the number of scans from the set of 512 scans defined by

$$NGRANULE * ig \le i < NGRANULE * ig + 512,$$

where

$$ig = 0,\ 512/NGRANULE,\ 1024/NGRANULE, \text{ etc.}$$

In the following the attachment flag is shown as zero. In the case of ADS #0 and ADS #3 only this flag may, in accordance with ESA guidelines, be set to 1 if all of the MDS records in the granule have the record quality indicator set set to -1. In this case the corresponding MDS records should be omitted (Reference [AD11], Chapter 5). This mechanism may be used used to identify gaps in the sequence of MDS records. In the case of ADS #0 the attachment flag should only be set if all 512 MDS records are omitted under this provision.

The time tag for all ADS records other than ADS #4 (which is scan-based) is derived by converting the corresponding time to transport format.

Let ig (above) be the index corresponding to the current granule row and let kg = ig + K. The corresponding time tag is t(kg) = [L1B-INT-53](kg). It is converted to transport format using the ESA CFI library subroutine `pl_pmjd`.

```
      mjdpg[0]/(1) = t(kg)
      mjdpg[1]/(2) = 0.0 (dummy value since output not required)
      status = pl_pmjd(mjdtg, mjdpg, utceg, dut1eg)
[GBTR-ADS0-1](ig) = [mjdtg[0:2]/(1:3)](kg)
[GBTR-ADS0-2](ig) = 0 (but see note on Attachment Flag above.)
[GBTR-ADS0-3](ig) = 3 zero bytes
[GBTR-ADS0-4](ig) = ig
[GBTR-ADS0-5](ig) = number of scans for which bit 0 of
nadir_packet_invalid(i) is set
[GBTR-ADS0-6](ig) = number of scans for which bit 1 of
nadir_packet_invalid(i) is set
[GBTR-ADS0-7](ig) = number of scans for which bit 2 of
nadir_packet_invalid(i) is set
[GBTR-ADS0-8](ig) = number of scans for which bit 3 of
nadir_packet_invalid(i) is set
[GBTR-ADS0-9](ig) = number of scans for which bit 4 of
nadir_packet_invalid(i) is set
[GBTR-ADS0-10](ig) = 0
[GBTR-ADS0-11](ig) = 0
[GBTR-ADS0-12](ig) = 0
[GBTR-ADS0-13](ig) = 0
[GBTR-ADS0-14](ig) = number of scans for which bit 9 of
nadir_packet_invalid(i) is set
```

```
[GBTR-ADS0-15](ig) = number of scans for which bit 0 of
frwrd_packet_invalid(i) is set
[GBTR-ADS0-16](ig) = number of scans for which bit 1 of
frwrd_packet_invalid(i) is set
[GBTR-ADS0-17](ig) = number of scans for which bit 2 of
frwrd_packet_invalid(i) is set
[GBTR-ADS0-18](ig) = number of scans for which bit 3 of
frwrd_packet_invalid(i) is set
[GBTR-ADS0-19](ig) = number of scans for which bit 4 of
frwrd_packet_invalid(i) is set
[GBTR-ADS0-20](ig) = 0
[GBTR-ADS0-21](ig) = 0
[GBTR-ADS0-22](ig) = 0
[GBTR-ADS0-23](ig) = 0
[GBTR-ADS0-24](ig) = number of scans for which bit 9 of
frwrd_packet_invalid(i) is set
[GBTR-ADS0-35](ig) = 28 null bytes
```

Step 5.24.3-3 Ancillary Data Set 1.

ADS #1 (scan and pixel number). This is a new 'image array' generated in regrid source packets. For each *ig* = 0, 1, ...

```
[GBTR-ADS1-1](ig) = [mjdtg[0:2]/(1:3)](kg)           (Req 5.24-29)
[GBTR-ADS1-2](ig) = 0                                (Req 5.24-30)
[GBTR-ADS1-3](ig) = 3 zero bytes                     (Req 5.24-31)
[GBTR-ADS1-4](ig) = fix(1000 * track_y(kg) + 0.5)    (Req 5.24-32)
[GBTR-ADS1-5](ig, j) = [L1B-INT-134](ig, j), j= 0, 511
                                                     (Req 5.24-33)
[GBTR-ADS1-6](ig, j) = [L1B-INT-135](ig, j), j= 0, 511
                                                     (Req 5.24-34)
```

Step 5.24.3-4 Ancillary Data Set 2.

ADS #2 (scan and pixel number). This is a new 'image array' generated in regrid source packets. For each *ig* = 0, 1, ...

```
[GBTR-ADS2-1](ig) = [mjdtg[0:2]/(1:3)](kg)           (Req 5.24-35)
[GBTR-ADS2-2](ig) = 0                                (Req 5.24-36)
[GBTR-ADS2-3](ig) = 3 zero bytes                     (Req 5.24-37)
[GBTR-ADS2-4](ig) = fix(1000 * track_y(kg) + 0.5)    (Req 5.24-38)
[GBTR-ADS2-5](ig, j) = [L1B-INT-154](ig, j), j= 0, 511
                                                     (Req 5.24-39)
[GBTR-ADS2-6](ig, j) = [L1B-INT-155](ig, j), j= 0, 511
                                                     (Req 5.24-40)
```

Step 5.24.3-5 Ancillary Data Set 3.

ADS #3. The numbers here (grid pixel lat and long) are in fact the geolocation grids,. and so are generated in 'Generate Geolocation Grid'. For each *ig* = 0, 1, ...

```
[GBTR-ADS3-1](ig) = [mjdtg[0:2]/(1:3)](kg)           (Req 5.24-41)
[GBTR-ADS3-2](ig) = 0                                (Req 5.24-42)
[GBTR-ADS3-3](ig) = 3 zero bytes                     (Req 5.24-43)
[GBTR-ADS3-4](ig) = fix(1000 * track_y(kg) + 0.5)    (Req 5.24-44)
[GBTR-ADS3-5](ig, j) =
      fix(1000000 * [L1B-INT-26](ig, j) + 0.5), j = 0, 22
                                                     (Req 5.24-45)
[GBTR-ADS3-6](ig, j) =
      fix(1000000 * [L1B-INT-27](ig, j) + 0.5), j = 0, 22
                                                     (Req 5.24-46)
[GBTR-ADS3-7](ig, j) = [L1B-INT-380](ig, j), j = 0, 22
                                                     (Req 5.24-47)
[GBTR-ADS3-8](ig, j) = [L1B-INT-381](ig, j), j = 0, 22
                                                     (Req 5.24-48)
[GBTR-ADS3-9](ig, j) = [L1B-INT-383](ig, j), j = 0, 22
                                                     (Req 5.24-49)
[GBTR-ADS3-10](ig, j) = [L1B-INT-384](ig, j), j = 0, 22
                                                      (Req 5.24-1)
[GBTR-ADS3-11](ig, j) = [L1B-INT-382](ig, j), j = 0, 22
                                                      (Req 5.24-2)
```

Step 5.24.3-6 Ancillary Data Set 4.

ADS #4 (scan pixel x and y) are generated in the geolocation modules. They and the scan times must be saved separately in a new structure indexed by instrument scan number. Only save every NGRANULE.

For ADS #4 (which is scan-based) the time tag is derived by converting the scan UTC to transport format. Let sg be the index corresponding to the current record of the ADS, and let s be the index to the instrument scans. Then s increments from 0 to end of data in steps of *NGRANULE*. Thus s = sg * *NGRANULE*, except that when (as will usually be the case) the total number of instrument scans is not a multiple of 32, the final record of ADS #4 shall be derived from the final instrument scan. The time tag corresponding to scan s is [L1B-INT-400](s). It is converted to transport format using the ESA CFI library subroutine pl_pmjd as below.

For s = 0, NGRANULE, ..., (last scan); sg = 0, 1, ...

```
      mjdp[0]/(1) = source_packet_ut_time(s)
      mjdp[1]/(2) = 0.0 (dummy value since output not required)
      status = pl_pmjd(mjdt, mjdp, utce, dut1e)
[GBTR-ADS4-1](sg) = [mjdt[0:2]/(1:3)](s)             (Req 5.24-50)
[GBTR-ADS4-2](sg) = 0 (but see note on Attachment Flag
      in Step 5.24.3-2.1)                            (Req 5.24-51)
[GBTR-ADS4-3](sg) = 3 zero bytes                     (Req 5.24-52)
[GBTR-ADS4-4](sg) = relative_scan_number(s)          (Req 5.24-53)
[GBTR-ADS4-5](sg) = {[L1B-INT-64](0), [L1B-INT-64](10),
      [L1B-INT-64](20), ... [L1B-INT-64](570), [L1B-INT-64](574),
      [L1B-INT-66](0), [L1B-INT-66](10), [L1B-INT-66](20), ...
      [L1B-INT-66](390)} * 1000                      (Req 5.24-54)
[GBTR-ADS4-6](sg) = {[L1B-INT-65](0), [L1B-INT-65](10),
      [L1B-INT-65](20), ... [L1B-INT-65](570), [L1B-INT-65](574),
      [L1B-INT-67](0), [L1B-INT-67](10), [L1B-INT-67](20), ...
      [L1B-INT-67](390)} * 1000                      (Req 5.24-55)
```

Each of [GBTR-ADS4-5], [GBTR-ADS4-6] comprises the specified list of values, in order. The indices correspond to the tie points in [GBTR-SPH-47] in Req. 5.24-25 above; these are, given the adopted value of INT_P = 10,

nadir view: 0 to 574 in steps of 10 (59 values)
forward view: 0 to 390 in steps of 10 (40 values).

Note that each value in these lists is to be multiplied by 1000 to convert its units to metres.

Step 5.24.3-7 Ancillary Data Set 5.

ADS #5. (Solar angles). These are taken from every *NGRANULE* row of the regridded parameters structure. For each *ig* = 0, 1, ...

```
[GBTR-ADS5-1](ig) = [mjdtg[0:2]/(1:3)](kg)          (Req 5.24-56)
[GBTR-ADS5-2](ig) = 0                               (Req 5.24-57)
[GBTR-ADS5-3](ig) = 3 zero bytes                    (Req 5.24-58)
[GBTR-ADS5-4](ig) = fix(1000 * track_y(kg) + 0.5)   (Req 5.24-59)
[GBTR-ADS5-5](ig) = 1000 * [L1B-INT-120](i, k), k = 0, 10
      where i = ig * NGRANULE                       (Req 5.24-60)
[GBTR-ADS5-6](ig) = 1000 * [L1B-INT-121](i, k), k = 0, 10
                                                    (Req 5.24-61)
[GBTR-ADS5-7](ig) = 1000 * [L1B-INT-122](i, k), k = 0, 10
                                                    (Req 5.24-62)
[GBTR-ADS5-8](ig) = 1000 * [L1B-INT-123](i, k), k = 0, 10
                                                    (Req 5.24-63)
```

Step 5.24.3-8 Ancillary Data Set 6.

ADS #6. (Solar angles). These are taken from every *NGRANULE* row of the regridded parameters structure. For each *ig* = 0, 1, ...

```
[GBTR-ADS6-1](ig) = [mjdtg[0:2]/(1:3)](kg)          (Req 5.24-64)
[GBTR-ADS6-2](ig) = 0                               (Req 5.24-65)
[GBTR-ADS6-3](ig) = 3 zero bytes                    (Req 5.24-66)
[GBTR-ADS6-4](ig) = fix(1000 * track_y(kg) + 0.5)   (Req 5.24-67)
[GBTR-ADS6-5](ig) = 1000 * [L1B-INT-140](i, k), k = 0, 10
      where i = ig * NGRANULE                       (Req 5.24-68)
[GBTR-ADS6-6](ig) = 1000 * [L1B-INT-141](i, k), k = 0, 10
                                                    (Req 5.24-69)
[GBTR-ADS6-7](ig) = 1000 * [L1B-INT-142](i, k), k = 0, 10
                                                    (Req 5.24-70)
[GBTR-ADS6-8](ig) = 1000 * [L1B-INT-143](i, k), k = 0, 10
                                                    (Req 5.24-71)
```

Step 5.24.3-8.1 Ancillary Data Set 7

Output the VISCAL data from Module 8 (Table 5-8-2) to the Visible Calibration ADS (ADS #7)

```
[GBTR-ADS7-1] = [L1B-INT-410]
[GBTR-ADS7-31] = 0 (Attachment flag)
[GBTR-ADS7-1] = (3 zero bytes)
[GBTR-ADS7-<n>] = [L1B-INT-<409 + n>] for n = 2, ... 29.
[GBTR-ADS7-30] = 0
```

Step 5.24.3-9 MDSR Header words

(a) UTC Nadir Time in MDS records.

```
Suppose we are dealing with image scan i.  Let
      i = ig*NGRANULE + i',
where
      ig = integer part of (i/NGRANULE)
and
      i' = i - i_g*NGRANULE.
```

We can also define

```
      k_g = i_g + K.
```

The time argument of record i is obtained by linear interpolation

```
      nadir_time(i) = t[k_g] + (i') (t[k_g + 1] - t[k_g])/(NGRANULE).
```

We can then use the subroutine pmjd from the CFI time conversion library to convert as above the nadir time (t3) from processing to transport and external formats (the latter is an unwanted byproduct as above.

```
      mjdp3[0]/(1) = nadir_time(i)
      mjdp3[1]/(2) = 0.0 (dummy value since output not required)
      status = pl_pmjd(mjdt3, mjdp3, utce, dut1e)
```

The latter are local, as usual. mjdt3 then provides the output.

(Req 5.24-72)

(b) The corresponding image scan y co-ordinate is

```
      nadir_y(i) = track_y(kg) +
      (i') (track_y(kg + 1) - track_y(kg))/(NGRANULE).
```

in km. (On output this is to be converted to metres and fixed.)

(Req 5.24-73)

(c) The Record Quality Indicator. In the following this is shown as zero. If required it may be assembled either at output or at regridding time. A flag for each row of the buffer, initialized to false, would be set to true if a valid datum is geolocated to that row.

(Req 5.24-74)

Step 5.24.3-10 Measurement Data Sets #1 - 7.

These measurement data sets represent the nadir view channel brightness/reflectance values. They are to be assembled as follows. For each channel ch = 1, 7 and for each image scan i:

```
[GBTR-MDS<ch>-1](i) = [mjdt3[0:2]/(1:3)](i)          (Req 5.24-75)
[GBTR-MDS<ch>-2](i) = 0                              (Req 5.24-76)
[GBTR-MDS<ch>-3](i) = 3 zero bytes                   (Req 5.24-77)
[GBTR-MDS<ch>-4](i) = fix(1000 * nadir_y(i)+0.5)     (Req 5.24-78)
```

For each pixel j = 0, 511

```
      [GBTR-MDS<ch>-5](i, j) = I(ch, n; i, j)        (Req 5.24-79)
```

No type conversion is required here.

Step 5.24.3-11 Measurement Data Sets #8 - 14.

These measurement data sets represent the forward view channel brightness/reflectance values. They are to be assembled as follows. For each channel ch = 1, 7 and for each image scan i:

```
[GBTR-MDS<7 + ch>-1](i) = [mjdt3[0:2]/(1:3)](i)      (Req 5.24-80)
[GBTR-MDS<7 + ch>-2](i) = 0                           (Req 5.24-81)
[GBTR-MDS<7 + ch>-3](i) = 3 zero bytes                (Req 5.24-82)
[GBTR-MDS<7 + ch>-4](i) = fix(1000 * nadir_y(i)+0.5) (Req 5.24-83)
```

For each pixel j = 0, 511

```
      [GBTR-MDS<7 + ch>-5](i, j) = I(ch, f; i, j)    (Req 5.24-84)
```

No type conversion is required here.

Step 5.24-3-12 Measurement Data Sets #15, 16.

These measurement data sets represent the nadir and forward view confidence words respectively. They are to be assembled as follows. For n = 15, 16 and for each image scan i:

```
[GBTR-MDS<n>-1](i) = [mjdt3[0:2]/(1:3)](i)           (Req 5.24-85)
[GBTR-MDS<n>-2](i) = 0                                (Req 5.24-86)
[GBTR-MDS<n>-3](i) = 3 zero bytes                     (Req 5.24-87)
[GBTR-MDS<n>-4](i) = fix(1000 * nadir_y(i)+0.5)      (Req 5.24-88)

      (Requirement deleted)                           (Req 5.24-89)
```

Assemble confidence words for each pixel j = 0, 511:

For each gbtr confidence flag, the corresponding bit of the gbtr confidence word is to be set according to the truth value (1 = TRUE; 0 = FALSE) of the flag as follows:

[gbtr_confidence_<view>(i, j)](bit 0) = <view>_blanking_pulse(i, j)

(Req 5.24-90)

[gbtr_confidence_<view>(i, j)](bit 1) = <view>_cosmetic(i, j)

(Req 5.24-91)

[gbtr_confidence_<view>(i, j)](bit 2) = <view>_scan_absent(i, j)

(Req 5.24-92)

[gbtr_confidence_<view>(i, j)](bit 3) = <view>_pixel_absent(i, j)

(Req 5.24-93)

[gbtr_confidence_<view>(i, j)](bit 4) = <view>_packet_validation_error(i, j)

(Req 5.24-94)

[gbtr_confidence_<view>(i, j)](bit 5) = <view>_zero_count(i, j)

(Req 5.24-95)

[gbtr_confidence_<view>(i, j)](bit 6) = <view>_saturation(i, j)

(Req 5.24-96)

[gbtr_confidence_<view>(i, j)](bit 7) = <view>_cal_out_of_range(i, j)

(Req 5.24-97)

[gbtr_confidence_<view>(i, j)](bit 8) = <view>_calibration_unavailable(i, j)

(Req 5.24-98)

[gbtr_confidence_<view>(i, j)](bit 9) = <view>_unfilled_pixel(i, j)

(Req 5.24-99)

[gbtr_confidence_<view>(i, j)](bit 10) = 0

(Req 5.24-100)

[gbtr_confidence_<view>(i, j)](bit 11) = 0

(Req 5.24-101)

[gbtr_confidence_<view>(i, j)](bit 12) = 0

(Req 5.24-102)

[gbtr_confidence_<view>(i, j)](bit 13) = 0

(Req 5.24-103)

[gbtr_confidence_<view>(i, j)](bit 14) = 0

(Req 5.24-104)

[gbtr_confidence_<view>(i, j)](bit 15) = 0

(Req 5.24-105)

where in each case <view> = <nadir | frwrd>

For each pixel j = 0, 511 [GBTR-MDS15-5](i, j) = gbtr_confidence_nadir(i, j)

(Req 5.24-106)

For each pixel j = 0, 511 [GBTR-MDS16-5](i, j) = gbtr_confidence_frwrd(i, j)

(Req 5.24-107)

Step 5.24.3-13 Measurement Data Sets #17, 18.

These measurement data sets represent the nadir and forward view cloud/land flag words respectively. They are to be assembled as follows. For n = 17, 18 and for each image scan i:

```
[GBTR-MDS<n>-1](i) = [mjdt3[0:2]/(1:3)](i)          (Req 5.24-108)
[GBTR-MDS<n>-2](i) = 0                              (Req 5.24-109)
[GBTR-MDS<n>-3](i) = 3 zero bytes                   (Req 5.24-110)
[GBTR-MDS<n>-4](i) = fix(1000 * nadir_y(i))         (Req 5.24-111)

      (Requirement deleted)                         (Req 5.24-112)
```

Assemble cloud state words for each pixel j = 0, 511:

For each gbtr cloud/land state flag, the corresponding bit of the gbtr cloud state word is to be set according to the truth value (1 = TRUE; 0 = FALSE) of the flag as follows:

[gbtr_cloud_state_<view>(i, j)](bit 0) = <view>_land(i, j)

(Req 5.24-113)

[gbtr_cloud_state_<view>(i, j)](bit 1) = <view>_cloud(i, j)

(Req 5.24-114)

[gbtr_cloud_state_<view>(i, j)](bit 2) = <view>_sunglint(i, j)

(Req 5.24-115)

[gbtr_cloud_state_<view>(i, j)](bit 3) = <view>_v16_histogram_test(i, j)

(Req 5.24-116)

[gbtr_cloud_state_<view>(i, j)](bit 4) = <view>_v16_spatial_coherence_test(i, j)

(Req 5.24-117)

[gbtr_cloud_state_<view>(i, j)](bit 5) = <view>_ir11_spatial_coherence_test(i, j)

(Req 5.24-118)

[gbtr_cloud_state_<view>(i, j)](bit 6) = <view>_ir12_gross_cloud_test(i, j)

(Req 5.24-119)

[gbtr_cloud_state_<view>(i, j)](bit 7) = <view>_ir11_ir12_thin_cirrus_test(i, j)

(Req 5.24-120)

[gbtr_cloud_state_<view>(i, j)](bit 8) = <view>_ir37_ir12med_high_level_test(i, j)

(Req 5.24-121)

[gbtr_cloud_state_<view>(i, j)](bit 9) = <view>_ir11_ir37_fog_low_stratus_test(i, j)

(Req 5.24-122)

[gbtr_cloud_state_<view>(i, j)](bit 10) = <view>_ir11_ir12_view_diff_test(i, j)

(Req 5.24-123)

[gbtr_cloud_state_<view>(i, j)](bit 11) = <view>_ir37_ir11_view_diff_test(i, j)

(Req 5.24-124)

[gbtr_cloud_state_<view>(i, j)](bit 12) = <view>_ir11_ir12_histogram_test(i, j)

(Req 5.24-125)

[gbtr_cloud_state_<view>(i, j)](bit 13) = <view>_visible_channel_cloud_test(i, j)

(Req 5.24-126)

[gbtr_cloud_state_<view>(i, j)](bit 14) = <view>_snow_flag(i, j)

(Req 5.24-127)

[gbtr_cloud_state_<view>(i, j)](bit 15) = 0

(Req 5.24-128)

where in each case <view> = <nadir | frwrd>

For each pixel j = 0, 511 [GBTR-MDS17-5](i, j) = gbtr_cloud_state_nadir(i, j)

(Req 5.24-129)

For each pixel j = 0, 511 [GBTR-MDS18-5](i, j) = gbtr_cloud_state_frwrd(i, j)

(Req 5.24-130)

6. INTERNAL PARAMETER LIST

Parameter ID	Variable	Name	Type	Units	Field Size	Fields	Comments
		Global Constants:					
L1-AUX18-13	constant	INIT_CAL_PARAM	fl	N/A	4	1	
L1-AUX18-21	constant	RAW_PKT_FAILS_BASIC_VALIDATION_ERR	ss	N/A	2	1	
L1-AUX18-22	constant	PIXEL_COUNT_SCIENCE_DATA_NOT_DECOMPRESSED	ss	N/A	2	1	
L1-AUX18-23	constant	PIXEL_COUNT_ZERO	ss	N/A	2	1	
L1-AUX18-24	constant	PIXEL_COUNT_SATURATED	ss	N/A	2	1	
L1-AUX18-27	constant	CRC_ERR_CODE_DETECTED_ERR	ss	N/A	2	1	
L1-AUX18-28	constant	BUFFERS_FULL_CHECK_ERR	ss	N/A	2	1	
L1-AUX18-30	constant	TEMP_OUT_OF_RANGE_FOR_LUT_ERR	ss	N/A	2	1	
L1-AUX18-33	constant	PIX_SCAN_JITTER_ERR	ss	N/A	2	1	
L1-AUX18-34	constant	TMZ_AT_LIMIT_ERR	ss	N/A	N/A	1	
L1-AUX18-35	constant	TMZ_ROGUE_PRT_ERR	ss	N/A	N/A	1	
L1-AUX18-36	constant	TMZ_CALIBRATION_ERR	ss	N/A	N/A	1	
L1-AUX18-37	constant	TMZ_BB_OVERRANGE_ERR	ss	N/A	N/A	1	
L1-AUX18-38	constant	TMZ_SURVEILLANCE_ERR	ss	N/A	N/A	1	
L1-AUX18-39	constant	TMZ_PRT8_ERR	ss	N/A	N/A	1	
L1-AUX18-40	constant	TMZ_ROGUE_SCP_ERR	ss	N/A	N/A	1	
L1-AUX18-41	constant	TMZ_BB_OUT_OF_LIMIT_ERR	ss	N/A	N/A	1	
L1-AUX18-42	constant	PIXEL_COUNT_INITIAL_VALUE	ss	N/A	2	1	
L1-AUX18-43	constant	BB_PIX_COUNT_OUT_OF_RANGE_ERR	ss	N/A	2	1	
L1-AUX18-44	constant	BB_PIX_COUNT_OUT_OF_RANGE_ALL_CHANS	ss	N/A	2	1	
L1-AUX18-45	constant	TMZ_ROGUE_BB_ERR	ss	N/A	N/A	1	
L1-AUX18-46	constant	MAX_HBB_PRT_MEAN_DIFF	do	deg K	8	1	
L1-AUX18-47	constant	MAX_CBB_PRT_MEAN_DIFF	do	deg K	8	1	
L1B-INT-001	packet_error	packet_error	ss	N/A	2	1	
L1B-INT-002		Deleted	n/a	n/a	n/a	n/a	
L1B-INT-003	advr(i)	auxilary_data_validation_result[i]	array of us	N/A	2	aux_tot	
L1B-INT-004	conv_aux_data	converted_auxiliary_data[i]	fl array	N/A	4	aux_tot	
L1B-INT-005	auxdata(i)	unpacked_auxiliary_data[i]	us array	N/A	2	aux_tot	
L1B-INT-006	cal_inv(i)	calibration_invalid[channel]	ss	N/A	2	7	Used if scan jitter detected
L1B-INT-006		.calibration_invalid	sl				
L1B-INT-007	pixmap_id	pixel_map_id	ss	N/A	2		
L1B-INT-009	scp_gain(ch, s)	channel scp gain (only required for v16 at present)	float	n/a	4		
L1B-INT-010		.slope[PARITY_LEVELS]	float				
L1B-INT-010	ch_gain[i,j] slope{ch; s, pty}	gain[parity][channel] channel[.] slope	array of array of fl	TBD	4	7 * 2	From section 4.5
L1B-INT-011	ch_offset[i,j] intercept{ch; s, pty}	offset[parity][channel] channel[.]intercept	array of array of fl	TBD	4	7 * 2	From section 4.5
L1B-INT-011		.intercept[PARITY_LEVELS]	float				
L1B-INT-012	MJDT0[4]	Reference UTC in MJD Format:				1	
L1B-INT-013	MJDT0[0]/(1)	Reference UTC days	sl	days	4	1	
L1B-INT-014	MJDT0[1]/(2)	Reference UTC seconds	sl	s	4	1	
L1B-INT-015	MJDT0[2]/(3)	Reference UTC micros.	sl	micros	4	1	
L1B-INT-016	MJDT0[3]/(4)	Reference deltaUT1	sl	micros	4	1	
L1B-INT-017	DUT1E0	delta UT1	char	n/a	8	1	
L1B-INT-018	MJDT[4]	Scan UTC Time in MJD Format:				1	
L1B-INT-019	MJDT[0]/(1)	Scan UTC days	sl	days	4	1	
L1B-INT-020	MJDT[1]/(2)	Scan UTC seconds	sl	s	4	1	
L1B-INT-021	MJDT[2]/(3)	Scan UTC micros.	sl	micros	4	1	
L1B-INT-022	MJDT[3]/(4)	Scan deltaUT1	sl	micros	4	1	

L1B-INT-023	MJDP[0]/(1)	Scan UTC in processing format	double	days	8	1	
L1B-INT-024	MJDP[1]/(2)	Scan deltaUT1	double	s	8	1	
L1B-INT-025	ST_status	status flag	sl	n/a	4	1	
L1B-INT-026		grid latitude[1680][23]	float	deg.	4		
L1B-INT-027		grid longitude[1680][23]	float	deg.	4		
L1B-INT-028		y_to_around orbit angle [1680]	float	km	4		
L1B-INT-029	geoloc_status	status flag	sl	n/a	4	1	
L1B-INT-030	mjdp2[0]	ascending node time	double	days	8	1	
L1B-INT-031	mjdp2[1]	delata UT1 at mjdr[0]	double	s	8	1	
L1B-INT-032	xm[0]	mean semi-major axis	double	m	8	1	
L1B-INT-033	xm[1]	mean eccentricity	double	none	8	1	
L1B-INT-034	xm[2]	mean inclination	double	deg	8	1	
L1B-INT-035	xm[3]	mean r.a. of node	double	deg	8	1	
L1B-INT-036	xm[4]	mean arg. of perigee	double	deg	8	1	
L1B-INT-037	xm[5]	mean anomaly	double	deg	8	1	
L1B-INT-038	x[0]	osc. semi-major axis	double	m	8	1	
L1B-INT-039	x[1]	osc. eccentricity	double	none	8	1	
L1B-INT-040	x[2]	osc. inclination	double	deg	8	1	
L1B-INT-041	x[3]	osc. r.a. of node	double	deg	8	1	
L1B-INT-042	x[4]	osc. arg. of perigee	double	deg	8	1	
L1B-INT-043	x[5]	mean anomaly	double	deg	8	1	
L1B-INT-044	acc[0]	x component of acceleration	double	m/s/s	8	1	
L1B-INT-045	acc[1]	Y component of acceleration	double	m/s/s	8	1	
L1B-INT-046	acc[2]	Z component of acceleration	double	m/s/s	8	1	
L1B-INT-047	res[54]	results array	db array	misc.	8	54	
L1B-INT-048	ierr[4]	Error Flag array	sl array	n/a	4	4	
L1B-INT-050	ve	ssp ground speed	float	Km/s	4	1	
L1B-INT-51	track_lat(k)	track latitude	double	deg.	8	NGRID+K	
L1B-INT-52	track_long(k)	track longitude	double	deg.	8	NGRID+K	
L1B-INT-53	t(k)	fixed time step array	double	days	8	NGRID+K	
L1B-INT-54	s(k)	along track distance	double	km.	8	NGRID+K	
L1B-INT-55	track_y(k)	along track distance relative to asc. node	double	km.	8	NGRID+K	
L1B-INT-56	T0	Time of first product granule	double	days	8	1	
		Geolocation Structures:					
L1B-INT-060	nadir_lat(s, p)	nadir scan pixel latitude	float	deg	4		
L1B-INT-061	nadir_long(s, p)	nadir scan pixel longitude	float	deg	4		
L1B-INT-062	frwrd_lat(s, p)	forward scan pixel latitude	float	deg	4		
L1B-INT-063	frwrd_long(s, p)	forward scan pixel longitude	float	deg	4		
L1B-INT-064	nadir_x(s, p) nadir_x_coord[][575]	nadir scan x coordinate source packet nadir pixel x coords	float	km	4	575	
L1B-INT-065	nadir_y(s, p) nadir_y_coord[][575]	nadir scan y coordinate source packet nadir pixel y coords	float	km	4	575	
L1B-INT-066	frwrd_x(s, p) frwrd_x_coord[][391]	forward scan x coordinate source packet forward pixel x coords	float	km	4	391	
L1B-INT-067	frwrd_y(s, p) frwrd_y_coord[][391]	forward scan y coordinate source packet forward pixel y coords	float	km	4	391	
L1B-INT-070		relative_scan_numnber(s)	sl	none	4		
L1B-INT-071	aux_temp(s, jaux)	temporary auxiliary temperatures for Module 18	float	K	4	jaux = 0, 5	
L1B-INT-072	aux_unconv(s, jaux)	unconverted auxiliary temperatures for Module 18	us	n/a	2	jaux = 0, 5	
L1B-INT-073	pixel_map(s)	pixel map number for use by Module 18	ss	n/a	2	per s	
L1B-INT-080	C(ch, n; s, p)	unpacked.pixels.nadir[MAX_NADIR_PIXELS]	ss				
L1B-INT-081	C(ch, f; s, p)	unpacked.pixels.forward[MAX_FORWARD_PIXELS]	ss				
L1B-INT-082		unpacked.pixels.plus_bb[MAX_PXBB_PIXELS]	ss				
L1B-INT-083		unpacked.pixels.minus_bb[MAX_MXBB_PIXELS]	ss				

L1B-INT-084		unpacked.pixels.viscal[MAX_VISCAL_PIXELS]	ss				
L1B-INT-087	T(ch, n; s, p)	calibrated.pixels.nadir[MAX_NADIR_PIXELS], infra-red channels	ss				
L1B-INT-088	T(ch, f; s, p)	calibrated.pixels.forward[MAX_FORWARD_PIXELS], infra-red channels	ss				
L1B-INT-089	R(ch, n; s, p)	calibrated.pixels.nadir[MAX_NADIR_PIXELS], visible channels	ss				
L1B-INT-090	R(ch, f; s, p)	calibrated.pixels.forward[MAX_FORWARD_PIXELS], visible channels	ss				
L1B-INT-091		unpacked_blanking_nadir[MAX_NADIR_PIXELS]	ss				
L1B-INT-092		unpacked_blanking_forward[MAX_FORWARD_PIXELS]	ss				
L1B-INT-093		unpacked_blanking_plus_bb[MAX_PXBB_PIXELS]	ss				
L1B-INT-094		unpacke_blanking.minus_bb[MAX_MXBB_PIXELS]	ss				
L1B-INT-095		unpacked_blanking._viscal[MAX_VISCAL_PIXELS]	ss				
		Regridded image data structures:					
L1B-INT-100		nadir_fill_state(i, j)	byte	n/a	1	j = 0, 511	
L1B-INT-101	I(ir12, n; i, j)	regridded nadir ir12 Brightness Temp.	ss	K/100	2	j = 0, 511	MDS1-5
L1B-INT-102	I(ir11, n; i, j)	regridded nadir ir11 Brightness Temp.	ss	K/100	2	j = 0, 511	MDS2-5
L1B-INT-103	I(ir37, n; i, j)	regridded nadir ir37 Brightness Temp.	ss	K/100	2	j = 0, 511	MDS3-5
L1B-INT-104	I(v16, n; i, j)	regridded nadir v16 Reflectance	ss	%/100	2	j = 0, 511	MDS4-5
L1B-INT-105	I(v870, n; i, j)	regridded nadir v870 Reflectance	ss	%/100	2	j = 0, 511	MDS5-5
L1B-INT-106	I(v670, n; i, j)	regridded nadir v670 Reflectance	ss	%/100	2	j = 0, 511	MDS6-5
L1B-INT-107	I(v555, n; i, j)	regridded nadir v555 Reflectance	ss	%/100	2	j = 0, 511	MDS7-5
L1B-INT-108		nadir_x_offset(i, j)	byte	n/a	1	j = 0, 511	
L1B-INT-109		nadir_y_offset(i, j)	byte	n/a	1	j = 0, 511	
L1B-INT-110		frwrd_fill_state(i, j)	byte	n/a	1	j = 0, 511	
L1B-INT-111	I(ir12, f; i, j)	regridded forward ir12 Brightness Temp.	ss	K/100	2	j = 0, 511	MDS8-5
L1B-INT-112	I(ir11, f; i, j)	regridded forward ir11 Brightness Temp.	ss	K/100	2	j = 0, 511	MDS9-5
L1B-INT-113	I(ir37, f; i, j)	regridded forward ir37 Brightness Temp.	ss	K/100	2	j = 0, 511	MDS10-5
L1B-INT-114	I(v16, f; i, j)	regridded forward v16 Reflectance	ss	%/100	2	j = 0, 511	MDS11-5
L1B-INT-115	I(v870, f; i, j)	regridded forward v870 Reflectance	ss	%/100	2	j = 0, 511	MDS12-5
L1B-INT-116	I(v670, f; i, j)	regridded forward v670 Reflectance	ss	%/100	2	j = 0, 511	MDS13-5
L1B-INT-117	I(v555, f; i, j)	regridded forward v555 Reflectance	ss	%/100	2	j = 0, 511	MDS14-5
L1B-INT-118		frwrd_x_offset(i, j)	byte	n/a	1	j = 0, 511	
L1B-INT-119		frwrd_y_offset(i, j)	byte	n/a	1	j = 0, 511	
		regridded nadir information:					
L1B-INT-120		nadir_band_edge_solar_elevation(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-121		nadir_band_edge_satellite_elevation(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-122		nadir_band_edge_solar_azimuth(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-123		nadir_band_edge_satellite_azimuth(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-124		nadir_band_centre_solar_elevation(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-125		nadir_band_centre_satellite_elevation(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-126		nadir_band_centre_solar_azimuth(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-127		nadir_band_centre_satellite_azimuth(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-128		deleted: no entry					
L1B-INT-129		nadir_min_aux_temps[6]	float	K	4	6	
L1B-INT-130		nadir_max_aux_temps[6]	float	K	4	6	
L1B-INT-131	nadir_packet_invalid	Nadir source packet invalid flags.	ss	flags	2		
L1B-INT-133		nadir_pixel_maps[2]	short int				
L1B-INT-134	scn_nadir(ig, j)	nadir view instrument scan number	us	n/a	2	j = 0, 511	
L1B-INT-135	pxl_nadir(ig, j)	nadir view instrument pixel number	us	n/a	2	j = 0, 511	
		regridded forwrad information:					
L1B-INT-140		frwrd_band_edge_solar_elevation(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-141		frwrd_band_edge_satellite_elevation(i, k)	float	degrees	4	k = 0, 10	

L1B-INT-142		frwrd_band_edge_solar_azimuth(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-143		frwrd_band_edge_satellite_azimuth(i, k)	float	degrees	4	k = 0, 10	
L1B-INT-144		frwrd_band_centre_solar_elevation(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-145		frwrd_band_centre_satellite_elevation(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-146		frwrd_band_centre_solar_azimuth(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-147		frwrd_band_centre_satellite_azimuth(i, k′)	float	degrees	4	k′ = 0, 9	
L1B-INT-148		deleted: no entry					
L1B-INT-149		frwrd_min_aux_temps[6]	float				
L1B-INT-150		frwrd_max_aux_temps[6]	float				
L1B-INT-151	frwrd_packet_in valid	Forward source packet invalid flags.	ss	flags	2		
L1B-INT-153		frwrd_pixel_maps[2]	short int				
L1B-INT-154	scn_frwrd(ig, j)	forward view instrument scan number	us	n/a	2	j = 0, 511	
L1B-INT-155	pxl_frwrd(ig, j)	forward view instrument pixel number	us	n/a	2	j = 0, 511	GBTR-ADS-6
L1B-INT-160		image_latitude(i, j)	float	degrees	4	j = 0, 511	
L1B-INT-161		image_longitude(i, j)	float	degrees	4	j = 0, 511	
L1B-INT-170	IDF20	pixel_map_viscal_start_pixel	ss	N/A	2	1	
L1B-INT-171	IDF21	pixel_map_viscal_end_pixel	ss	N/A	2	1	
L1B-INT-172	IDF22	pixel_map_nadir_start_pixel	ss	N/A	2	1	
L1B-INT-173	IDF23	pixel_map_nadirl_end_pixel	ss	N/A	2	1	
L1B-INT-174	IDF24	pixel_map_pxbb_start_pixel	ss	N/A	2	1	
L1B-INT-175	IDF25	pixel_map_pxbb_end_pixel	ss	N/A	2	1	
L1B-INT-176	IDF26	pixel_map_along_track_start_pixel	ss	N/A	2	1	
L1B-INT-177	IDF27	pixel_map_along_track_end_pixel	ss	N/A	2	1	
L1B-INT-178	IDF28	pixel_map_mxbb_start_pixel	ss	N/A	2	1	
L1B-INT-179	IDF29	pixel_map_mxbb_end_pixel	ss	N/A	2	1	
		nadir view confidence flags:					
L1B-INT-200		nadir blanking pulse flag	ss	n/a	2	j = 0, 511	
L1B-INT-201		nadir cosmetic fill flag	ss	n/a	2	j = 0, 511	
L1B-INT-202		nadir view scan absent flag	ss	n/a	2	j = 0, 511	
L1B-INT-203		nadir view pixel absent flag	ss	n/a	2	j = 0, 511	
L1B-INT-204		nadir view packet validation error flag	ss	n/a	2	j = 0, 511	
L1B-INT-205		nadir view zero count flag	ss	n/a	2	j = 0, 511	
L1B-INT-206		nadir view saturation flag	ss	n/a	2	j = 0, 511	
L1B-INT-207		nadir view calibration out of range flag	ss	n/a	2	j = 0, 511	
L1B-INT-208		nadir view calibration unavailable flag	ss	n/a	2	j = 0, 511	
L1B-INT-209		nadir view unfilled pixel flag	ss	n/a	2	j = 0, 511	
		forward view confidence flags:					
L1B-INT-216		frwrd blanking pulse flag	ss	n/a	2	j = 0, 511	
L1B-INT-217		frwrd cosmetic fill flag	ss	n/a	2	j = 0, 511	
L1B-INT-218		frwrd view scan absent flag	ss	n/a	2	j = 0, 511	
L1B-INT-219		frwrd view pixel absent flag	ss	n/a	2	j = 0, 511	
L1B-INT-220		frwrd view packet validation error flag	ss	n/a	2	j = 0, 511	
L1B-INT-221		frwrd view zero count flag	ss	n/a	2	j = 0, 511	
L1B-INT-222		frwrd view saturation flag	ss	n/a	2	j = 0, 511	
L1B-INT-223		frwrd view calibration out of range flag	ss	n/a	2	j = 0, 511	
L1B-INT-224		frwrd view calibration unavailable flag	ss	n/a	2	j = 0, 511	
L1B-INT-225		frwrd view unfilled pixel flag	ss	n/a	2	j = 0, 511	
		nadir land and cloud flags:					
L1B-INT-232		nadir land flag	ss	n/a	2	j = 0, 511	
L1B-INT-233		nadir cloud flag	ss	n/a	2	j = 0, 511	
L1B-INT-234		nadir sunglint flag	ss	n/a	2	j = 0, 511	
L1B-INT-235		nadir v16 histogram test flag	ss	n/a	2	j = 0, 511	
L1B-INT-236		nadir v16 spatial coherence test flag	ss	n/a	2	j = 0, 511	
L1B-INT-237		nadir ir11 spatial coherence test flag	ss	n/a	2	j = 0, 511	
L1B-INT-238		nadir ir12 gross cloud test flag	ss	n/a	2	j = 0, 511	
L1B-INT-239		nadir ir11 ir12 thin cirrus test flag	ss	n/a	2	j = 0, 511	
L1B-INT-240		nadir ir37 ir12 med high level test flag	ss	n/a	2	j = 0, 511	
L1B-INT-241		nadir ir11 ir37 fog low stratus test flag	ss	n/a	2	j = 0, 511	

L1B-INT-242		nadir ir11 ir12 view difference test flag	ss	n/a	2	j = 0, 511	
L1B-INT-243		nadir ir37 ir11 view difference test flag	ss	n/a	2	j = 0, 511	
L1B-INT-244		nadir ir11 ir12 histogram test flag	ss	n/a	2	j = 0, 511	
L1B-INT-245		Visible channel cloud test	ss	n/a	2	j = 0, 511	
L1B-INT-246		Snow flag	ss	n/a	2	j = 0, 511	
		forward land and cloud flags:					
L1B-INT-248		frwrd land flag	ss	n/a	2	j = 0, 511	
L1B-INT-249		frwrd cloud flag	ss	n/a	2	j = 0, 511	
L1B-INT-250		frwrd sunglint flag	ss	n/a	2	j = 0, 511	
L1B-INT-251		frwrd v16 histogram test flag	ss	n/a	2	j = 0, 511	
L1B-INT-252		frwrd v16 spatial coherence test flag	ss	n/a	2	j = 0, 511	
L1B-INT-253		frwrd ir11 spatial coherence test flag	ss	n/a	2	j = 0, 511	
L1B-INT-254		frwrd ir12 gross cloud test flag	ss	n/a	2	j = 0, 511	
L1B-INT-255		frwrd ir11 ir12 thin cirrus test flag	ss	n/a	2	j = 0, 511	
L1B-INT-256		frwrd ir37 ir12 med high level test flag	ss	n/a	2	j = 0, 511	
L1B-INT-257		frwrd ir11 ir37 fog low stratus test flag	ss	n/a	2	j = 0, 511	
L1B-INT-258		frwrd ir11 ir12 view difference test flag	ss	n/a	2	j = 0, 511	
L1B-INT-259		frwrd ir37 ir11 view difference test flag	ss	n/a	2	j = 0, 511	
L1B-INT-260		frwrd ir11 ir12 histogram test flag	ss	n/a	2	j = 0, 511	
L1B-INT-261		Visible channel cloud test	ss	n/a	2	j = 0, 511	
L1B-INT-262		Snow flag	ss	n/a	2	j = 0, 511	
		Parameters for cloud clearing algorithms:					
L1B-INT-261		x_index	int	n/a	2		
L1B-INT-262		y_index	int	n/a	2		
L1B-INT-263		solar_elevation	float	deg	4		
L1B-INT-264		ir11[3][3]	int	K/100	2		
L1B-INT-265		group_land_flag[171][171]	flag	n/a	2		
L1B-INT-266		average_11	int	K/100	2		
L1B-INT-267		sigma_11	int	K/100	2		
L1B-INT-268		j	int	n/a	2		
L1B-INT-269		n	int	n/a	2		
L1B-INT-270		average_11_array[171][171]	int	K/100	2		
L1B-INT-271		threshold_sd	int	K/100	2		
L1B-INT-272		group_cloud_flag[171][171]	flag	n/a	2		
L1B-INT-273		average_11_12_dif_cloudy	double	K/100	8		
L1B-INT-274		average_11_12_dif_clear	double	K/100	8		
L1B-INT-275		n_cloudy (number of cloudy pixels in group)	ss	none	2		
L1B-INT-276		valid_pixel_pairs	ss	none	2		
L1B-INT-277		previous_tests[171][171]	flag	n/a	2		
L1B-INT-279		extended_land_flag[171][171]	flag	n/a	2		
L1B-INT-280		sub_area_n	int	Km	2		
L1B-INT-281		sub_area_max_11[sub_area_n][sub_area_n]	int	K/a	2		
L1B-INT-282		sub_area_use_11[sub_area_n][sub_area_n]	int	K/100	2		
L1B-INT-283		sub_area_dif{sub_area_n][sub_ara_n]	int	K/00	2		
L1B-INT-284		land_sub_area[sub_area_n][sub_area_n]	flag	n/a	2		
L1B-INT-285		(deleted)					
L1B-INT-286		(deleted)					
L1B-INT-287		x_index_1	int	n/a	2		
L1B-INT-288		y_index_1	int	n/a	2		
L1B-INT-289		max_bt_11	int	K/100	2		
L1B-INT-290		(deleted)					
L1B-INT-291		threshold_11	int	K/100	2		
L1B-INT-292		(deleted)					
L1B-INT-293		land_in_areas	flag	n/a	2		
L1B-INT-294		bt_dif_max	int	K/100	2		
L1B-INT-295		difference_threshold	int	K/100	2		
L1B-INT-296		(deleted)					
L1B-INT-297		lowest_max_bt	int	K/100	2		
L1B-INT-298		valid_sub_area_flag[sub_area_n][sub_area_n]	flag	n/a	2		
L1B-INT-301		ir16[32][32]	int	%/100	2		

L1B-INT-302		x[32][32]	int	n/a	2		
L1B-INT-303		index_valid[1024]	int	n/a	2		
L1B-INT-304		total_valid	int	n/a	2		
L1B-INT-305		sol_elev	real	degree	4		
L1B-INT-306		sat_elev	float	degree	4		
L1B-INT-307		azim_dif	float	degree	4		
L1B-INT-308		v_x	float	n/a	4		
L1B-INT-309		v_y	float	n/a	4		
L1B-INT-310		v_z	float	n/a	4		
L1B-INT-311		tilt	float	degree	4		
L1B-INT-312		magnitude	float	n/a	4		
L1B-INT-313		glint_present	flag	n/a	2		
L1B-INT-314		histogram_16[1000]	int	n/a	2		
L1B-INT-315		low_interval	float	%	4		
L1B-INT-316		high_interval	float	%	4		
L1B-INT-317		hist_range	float	%	4		
L1B-INT-318		peak_box_no	int	n/a	2		
L1B-INT-319		peak_value	int	n/a	2		
L1B-INT-320		average_value	float	n/a	4		
L1B-INT-321		peak_interval	float	%	4		
L1B-INT-322		D	float	n/a	4		
L1B-INT-323		delta	float	n/a	4		
L1B-INT-324		spread_adjusted	float	%	4		
L1B-INT-325		f	float	n/a	4		
L1B-INT-326		g	float	n/a	4		
L1B-INT-327		spread	float	%	4		
L1B-INT-328		peak_factor	float	%	4		
L1B-INT-329		reflectance_threshold	float	%	4		
L1B-INT-330		n	float	%	4		
L1B-INT-331		gradient_16	float	%/Km/100	4		
L1B-INT-332		a	float	n/a	4		
L1B-INT-333		b	float	n/a	4		
L1B-INT-334		i	int	n/a	2		
L1B-INT-335		detrended_16[32][32]	int	%/100	2		
L1B-INT-336		sd_threshold	float	%/100	4		
L1B-INT-337		x_y_12um_max[2]	int	n/a	2		
L1B-INT-338		reflectance_at_12um_max	float	%	4		
L1B-INT-340		scan_loop	int	2	1		
L1B-INT-341		pixel_loop	int	2	1		
L1B-INT-342		valid_pixels	int	2	1		
L1B-INT-343		histogram[2][1000]	int	2	2000		
L1B-INT-344		bin_size	int	2	1		
L1B-INT-345		MAJOR	constant	2	1		
L1B-INT-346		MINOR	constant	2	1		
L1B-INT-347		CHOSEN	int	2	1		
L1B-INT-348		ir11_ir12_diff	int	K/100	2		
L1B-INT-349		binned_diff	int	K/10	2		
L1B-INT-351		ir12boxtotal[1000]	int	K/100	4		
L1B-INT-352		hist_index	int	2	1		
L1B-INT-353		peak_value[2]	int	2	2		
L1B-INT-354		peak_interval[2]	int	K/10	2		
L1B-INT-355		exact_peak_value[2]	float	4	2		
L1B-INT-356		exact_peak_interval[2]	float	K/10	4		
L1B-INT-357		ir11_ir12_diff_at_peak[2]	float	K/100	4		
L1B-INT-358		lower_limit	int	K/10	2		
L1B-INT-359		higher_limit	int	K/10	2		
L1B-INT-360		low_limit[2]	int	K/10	2		
L1B-INT-361		high_limit[2]	int	K/10	2		
L1B-INT-362		average_bt_mode[2]	float	K/100	4		
L1B-INT-363		average_bt[1000]	float	K/100	4		

L1B-INT-364		highest_av_bt_box[2]	int	2	2		
L1B-INT-365		slope_at_peak	float	4	1		
L1B-INT-366		slope[1000]	float	4	1000		
L1B-INT-367		half_width_threshold[2]	float	K/100	4		
L1B-INT-368		half_width[2]	float	K/100	4		
L1B-INT-369		upperhalfwidth_index	float	K/10	4		
L1B-INT-370		lowerhalfwidth_index	float	K/10	4		
L1B-INT-371		peak_valid[2]	flag	2	2		
L1B-INT-372		nightime	flag	2	1		
L1B-INT-380	Dphi_nadir	Latitude correction, nadir view	sl	μdeg	2	23	
L1B-INT-381	Dlam_nadir	Longitude correction, nadir view	sl	μdeg	2	23	
L1B-INT-382	H	Topographic height	ss	m	2	23	
L1B-INT-383	Dphi_frwrd	Latitude correction, forward view	sl	μdeg	2	23	
L1B-INT-384	Dlam_frwrd	Longitude correction, forward view	sl	μdeg	2	23	
L1B-INT-385	choice	Orbit file selection switch	sl	none	4	1	
L1B-INT-386	ndc	Number of DORIS precise orbit files	sl	none	4	1	
L1B-INT-387		doris_precise_file	char	n/a		1	
L1B-INT-388	ndp	Number of DORIS preliminary files	sl	none	4	1	
L1B-INT-389		doris_prelim_file	char	n/a		1	
L1B-INT-390	ner	Number of FOS restiruted orbit files	sl	none	4	1	
L1B-INT-391		esoc_rest	char	n/a		1	
L1B-INT-392	mjdr0	Start or Requested UTC	double	days	8	1	
L1B-INT-393	mjdr1	End UTC	double	days	8	1	
L1B-INT-394	mjdp_int[2]	UTC of state vector (po_interpol)	double		2	1	
L1B-INT-395	selected	(for use by po_interpol)	sl	none	4	1	
L1B-INT-396	ierr_interpol[10]	Error Flag array for po_interpol	sl array	n/a	4	10	
L1B-INT-400		source_packet_ut_time(s)	double	days	8		
L1B-INT-401	mid_UT_time	UT of centre of monitor period	double	days	8	1	
L1B-INT-410		Time of cal in MJD format	ul, 2* sl	MJD	12	1	
L1B-INT-411		1.6 micron slope	float	n/a	4	1	
L1B-INT-412		0.870 micron slope	float	n/a	4	1	
L1B-INT-413		0.670 micron slope	float	n/a	4	1	
L1B-INT-414		0.555 micron slope	float	n/a	4	1	
L1B-INT-415		UTC at ascending node crossing, in MJD format	ul, 2* sl	MJD	12	1	
L1B-INT-416		Average Monitor count	float	n/a	4	1	
L1B-INT-417		Standard deviation of Monitor count	float	n/a	4	1	
L1B-INT-418		Solar irradiance (1.6 micron)	float	n/a	4	1	
L1B-INT-419		Solar irradiance (0.870 micron)	float	n/a	4	1	
L1B-INT-420		Solar irradiance (0.670 micron)	float	n/a	4	1	
L1B-INT-421		Solar irradiance (0.555 micron)	float	n/a	4	1	
L1B-INT-422		Average VISCAL Pixel Counts (1.6 μm)	float	n/a	4	1	
L1B-INT-423		Average VISCAL Pixel Counts (0.87 μm)	float	n/a	4	1	
L1B-INT-424		Average VISCAL Pixel Counts (0.67 μm)	float	n/a	4	1	
L1B-INT-425		Average VISCAL Pixel Counts (0.55 μm)	float	n/a	4	1	
L1B-INT-426		VISCAL Pixel Noise (1.6 micron)	float	n/a	4	1	
L1B-INT-427		VISCAL Pixel Noise (0.87 micron)	float	n/a	4	1	
L1B-INT-428		VISCAL Pixel Noise (0.67 micron)	float	n/a	4	1	
L1B-INT-429		VISCAL Pixel Noise (0.55 micron)	float	n/a	4	1	
L1B-INT-430		Average -X BB Pixel Counts (1.6 μm)	float	n/a	4	1	
L1B-INT-431		Average -X BB Pixel Counts (0.87 μm)	float	n/a	4	1	
L1B-INT-432		Average -X BB Pixel Counts (0.67 μm)	float	n/a	4	1	
L1B-INT-433		Average -X BB Pixel Counts (0.55 μm)	float	n/a	4	1	
L1B-INT-434		-X BB Pixel Noise (1.6 micron)	float	n/a	4	1	
L1B-INT-435		-X BB Pixel Noise (0.87 micron)	float	n/a	4	1	
L1B-INT-436		-X BB Pixel Noise (0.67 micron)	float	n/a	4	1	
L1B-INT-437		-X BB Pixel Noise (0.55 micron)	float	n/a	4	1	

L1B-INT-438		(Reserved for parity indicator)	ss	n/a	2	1	

Table 6-1: Summary List of Internal Parameters and Global Constants