



## **ICESat (GLAS) Science Computing Facility Document Series**

### **Volume 3b**

# **SCF Data Request Software Detailed Design Document Version 201205.0**

*Anita Brenner*

*Tzipi Sidel*

*Kristine Barbieri*

National Aeronautics and  
Space Administration

**Goddard Space Flight Center**  
Greenbelt, Maryland 20771

---

May 2012

## Summary of Changes since Version 200112.0

- p.14 Added the following environmental variables to readMail.ksh:
  - ISIPS\_DIST
  - SPECIAL\_REQUEST\_PATH
- p.16 Added update\_error\_table.tcl.  
The following routines call it:
  - add\_to\_log\_file.tcl
  - daily\_stats.tcl
  - isips\_request.tcl
  - parse\_mail.tcl
  - subscription.tcl
- p.17 Added to populate\_db.tcl description: for subscriptions, if no tracks are specified orbelect.f90 is called to generate list of 8 or 91 day tracks for mission
- p.17 Added convert2j2000.tcl. The following routines call it:
  - populate\_db.tcl
- p.18 Added find\_passID.f90. The following routines call it:
  - populate\_db.tcl
  - data\_select.tcl
- p.20 in data\_select.tcl: If request is quick-look, submit the request to ISIPS to get the missing files and email SCF to get approval form the CCB.
- p.21 Changed name of create\_request\_pdr.tcl to create\_por.tcl and updated arguments
- p.22 Changed name of request PDR to Product Order Request (POR) and updated example
- p.24 Added catch\_errors.tcl. The following routines call it:
  - run\_ds\_pc.tcl
- p.25 Removed read\_ref\_orbit\_mod.f90. Using one rev file now instead of one for each reference orbit, so no need for reference orbit file. Removed references to environmental variable REF\_ORBIT\_FILE or changed REV\_PATH to REV\_FILE in the following scripts and codes:
  - populate\_db.tcl
  - readMail.ksh
  - data\_select.tcl
  - run\_subscription.ksh
  - subscription.tcl
  - isips\_request.tcl
  - create\_binrev.tcl
  - create\_pass\_files.tcl
  - write\_ds\_ctrl.tcl
  - run\_ds\_pc.tcl
  - data\_select.f90
  - read\_rev\_file.f90
  - read\_ds\_ctrl\_mod.f90
  - find\_passID.f90
- p.57 Added subset.tcl
- p.57 Added sort\_unique.tcl
- p.57 Added find\_span.tcl

p.58 Added zero\_padding.tcl  
p.58 Added zero\_out.tcl  
p.64 Added "status" column to ISIPS\_SUBSCRIPTIONS table  
p.67 Added "filled" column to ISIPS\_REQUESTS table  
p.68 Removed REFERENCE\_ORBIT table from database.

## **Summary of Changes since Version 200205.0**

p.74 Added QA\_PRODUCT\_UPDATE table to database.

## Summary of Changes since Version 200208.0

- p.4 Added browser\_visualizer.pro
- p.9 Modified orbselect.f90
- p.13 Added select\_processed\_tracks.f90
- p.14 time\_track.f90
- p.16 Added qa\_mail.tcl. The following routines call it:
  - parse\_mail.tcl
- p.37 Added prepare\_browse.tcl
- p.37 Added runbrowse.ksh
- p.38 Added qabrowse.pro
- p.44 Added compare\_lists.tcl. The following routines call it:
  - data\_select.tcl
- p.45 Added lsearch\_all\_idx.tcl. The following routines call it:
  - compare\_lists.tcl
- p.46 Added the following environmental variables to run\_subscription.ksh:
  - PROC\_8\_PASS
  - PROC\_91\_PASS
- p.51 Added read\_keyword.tcl. The following routines call it:
  - read\_fn\_file.tcl
- p. 52 Added another output argument to read\_fn\_file.tcl: Quick-look indicator
- p. 56 Added modify\_processed\_tracks.tcl. The following routines call it:
  - subscription.tcl
- p. 57 Added scp\_rev\_file.ksh. The following routines call it:
  - subscription.tcl
- p. 65 Added submit\_por.tcl.
- p. 66 Added submit\_qa.tcl.
- p. 67 Added create\_qauf.tcl. The following routines call it:
  - submit\_qa.tcl
- p.72 Added columns to ISIPS\_REQUESTS database table.
- p.73 Added columns to ISIPS\_DISTRIBUTION database table.
- p.74 Added columns to QA\_PRODUCT\_UPDATE database table.
- p.74 Added ql\_flag to SPECIAL\_REQUEST\_USER database table.

## Summary of Changes since Version 200212.0

p.20 and p.50 Subscription.tcl and data\_select.tcl copies BNA01 to BNA03 and BNA04 and GRA01 to GRA03 and GRA04.

p.23 Adding release number to routine filter\_files\_by\_release\_or\_version.tcl.  
The file gives the latest release number is release=0. Otherwise it gives the files with the given release number.

p.34 Prod\_create.f90 can create GLA03 or GLA04.

p.38 Modified mkpass to retrieve from rev file the passes that associate only with the times cover by the product file.

p.41 Checking in mkunique\_index if the current unique index record is bigger than the last one. If not, exit with an error message: Index is out of order.

p.43 Mkinrev exists without creating binrev file when products are GLA03 or GLA04.

p.49 Added the following environmental variables to run\_subscription.ksh:  
- UPDATE\_REV\_FLAG  
- BROWSE\_VERSION

p.50 Subscription.tcl now calls routine filter\_files\_by\_release\_or\_version.tcl

p.58 Added sort\_pass.tcl. The following routines call it:  
- create\_binrev.tcl

p.75 Added release to SPECIAL\_REQUEST\_USER database table

p.76 Added status to SUBSCRIPTION\_USER database table

## Summary of Changes since Version 200302.0

- p.20 Added send\_mail\_user.pl. The following routines call it:
- populate\_db.tcl
- p.24 Added release as input argument to create\_por.tcl
- p.27 In data\_select.f90 , the code is looking to the keyword DATE\_STRING in the control file. If it is there, the out file name contains the date\_str in it. If not, the start date is in the out file name.
- p.48 Added send\_mail\_mscf.pl. The following routines call it:
- check\_dist.tcl
  - check\_ps.tcl
  - daily\_stats.tcl
  - data\_select.tcl
  - populate\_db.tcl
  - qa\_mail.tcl
- p.48 Added send\_mail\_ccb.pl. The following routines call it:
- data\_select.tcl
  - qa\_mail.tcl
- p.50 Added LOG\_DIRECTORY environmental variable to subscription.tcl
- p.50 Added in subscription.tcl and isips\_request.tcl a call to database to get beginningDate. The beginning Date is the date\_str in the output file name.
- p.50 Added date\_str parameter in calling to run\_ds\_pc.tcl.
- p.54 Added log\_directory global variable to create\_pdr.tcl
- p.58 In run\_ds\_pc.tcl, if the length of date\_str =8, the DATE\_STRING=date\_str is added to the data\_select.f90 control file.
- p.62 Added check\_rev\_time.tcl. The following routines call it:
- subscription.tcl
- p.63 Added check\_pid.tcl. The following routines call it:
- subscription.tcl
- p.67 Added LOG\_DIRECTORY environmental variable to isips\_request.tcl
- p.68 Added call to send\_mail\_mscf.pl to isips\_request.tcl

- p.74 Added send\_mail\_isips.pl. The following routines call it:
  - daily\_stats.tcl
- p.74 Updated synopsis for daily\_stats.tcl
- p.78 Added scripts check\_dist.tcl and check\_ingest.tcl
- p.79 Added script check\_ps.tcl
- p.79 Added send\_mail\_pager.pl. The following routines call it:
  - check\_ps.tcl
- p.84 Added beginningDate into the ISIPS\_PRODUCT\_ID table.
- p.90 Added “release” column to ISIPS\_REQUESTS table
- p.90 Add “C=couldn’t fulfill” to ISIPS\_REQUESTS “status” column
- p.90 Added “release” and “quick\_look” columns to ISIPS\_DISTRIBUTION table

## Summary of Changes since Version 200304.0

- p.30 If the requested area is the whole world, data\_select.f90 doesn't use the routines that calculate the bins in a given area .
- p.6 If the requested area is the whole world, gettracksarray.pro doesn't excute orbselect.
- p.13 In select\_processed\_tracks.f90, when the selected area is the whole world, the output is all the tracks that were processed.
- p.29 The date in the rSCF file name is the beginning time of the first file in the 14 rev product set received from the I-SIPS. This information is given from the file header. If the keyword from the header could not be found, the date is set to the requested beginning time.
- p.40 Now get the first valid value in the 40 Hz array rather than just the first value of the array.
- p.90 In ISIPS\_REQUESTS and ISIPS\_DISTRIBUTION tables requestId has been increased from 5 to 8 chars.
- p. 68 isips\_request.tcl: Files downloaded from I-SIPS are placed in tmp directory defined by environmental variable SPECIAL\_REQUEST\_TMP/requestID. An FN file listing the files and FN XFR are created in the SPECIAL\_REQUEST\_TMP directory.
- p. 50 Added environmental variable LOCK\_FILE to subscription.tcl
- p. 68 Added environmental variable LOCK\_FILE to isips\_request.tcl
- p. 80 Added environmental variables SSH\_DIR and LOCK\_FILE check\_ps\_rscf.tcl
- p. 81 Added script ssh\_rsfs.tcl. The following routines call it:  
- check\_ps\_rscf.tcl

## Summary of Changes since Version 200306.0

- p.85 Added readIStatMail.ksh and parse\_istat\_mail.tcl scripts
- p.93 Added “date” column to DISTRIBUTION\_FILES table
- p.97 Added TOO\_UPDATE table
- p.98 Added INSTRUMENT\_UPDATE table

## Summary of Changes since Version 200308.0

- p. 16 Added environmental variable DATA\_PATH to data\_select.tcl
- p. 28 Added 1 degree of data to the border of the selected area when processing to ensure that all the data within geographic range are obtained.
- p. 51 Added environmental variable PULL\_FILE to subscription.tcl
- p. 65 Added script time\_sort.tcl. The following routines call it:
  - subscription.tcl
- p. 80 Added environmental variable PULL\_FILE to check\_dist.tcl script
- p. 83 Added script daily\_cleanup.tcl.
- p. 83 Added script get\_file\_j2000.tcl. The following routines call it:
  - daily\_cleanup.tcl
  - subscription.tcl
- p. 84 Added script get\_pid\_j2000.tcl. The following routines call it:
  - data\_select.tcl
- p. 84 Added script check\_inst\_update.tcl. The following routines call it:
  - daily\_cleanup.tcl
  - subscription.tcl
  - data\_select.tcl
- p. 85 Added environmental variable MAIL\_FILE to parse\_istat\_mail.tcl script
- p. 86 Added readStatMail.ksh and parse\_stat\_mail.tcl scripts
- p. 98 Added “j2000sec” column to INSTRUMENT\_UPDATE table
- p. 99 Added RTSCM\_UPDATE table

## **Summary of Changes since Version 200311.0**

p. 51 Added environmental variable DEL\_FILE to subscription.tcl

## Summary of Changes since Version 200401.0

- p. 16 Removed environmental variable DATA\_PATH from readMail.ksh for data\_select\_req.tcl
- p. 16 Added environmental variable CREATE\_POR to readMail.ksh for data\_select\_req.tcl
- p. 21 Separated data\_select.tcl into data\_select\_req.tcl for fulfilling special requests and data\_select\_vis.tcl for running the visualization software
- p. 52 Added assumption to subscription.tcl that there is only one release per distribution ID
- p.63 Added output argument \$laser to check\_rev\_time.tcl.
- p. 87 Added run\_stat\_report.ksh and stat\_report.tcl scripts
- p. 88 Added send\_mail\_report.pl script
- p. 96 Added data\_start\_j2000 and data\_end\_j2000 to CREATION database table

## Summary of Changes since Version 200403.0

- p. 5 Updated list of IDL routines used by data request and visualization GUI's
- p. 23 Added input argument `subject_line` to `send_mail_user.pl`
- p. 51 Added input argument `subject_line` to `send_mail_mscf.pl`
- p. 51 Added input argument `subject_line` to `send_mail_ccb.pl`
- p. 51 Removed `UPDATE_REV_FLAG` environmental variable from `subscription.tcl`.
- p. 51 Added `ICESATVIS_DATA` environmental variable to `subscription.tcl`.
- p. 62 Moved 8 or 91 day processed pass file from being a global variable to an input argument in `modify_processed_tracks.tcl`. Also added `scp_flag` as an output argument. Added `mscf_path` and `anc_data_path` as global variables.
- p. 81 Added input argument `subject_line` to `send_mail_isips.pl`
- p. 85 Added environmental variables `DEL_FILE` and `QAP_PATH` to `daily_cleanup.tcl`
- p. 92 Added input argument `subject_line` to `send_mail_report.tcl`
- p. 116 Added Appendix C: File Formats and C.1 REQ File Format

## Summary of Changes since Version 200405.0

- p. 53 Added environmental variables WEB\_EA\_DIR, EA\_DIR, EA\_FLAG, and PT\_FLAG to subscription.tcl
- p. 68 Added routine write\_ea\_ctrl.tcl. The following scripts call it:
- subscription.tcl
- p. 68 Added script run\_ea.ksh. The following scripts call it:
- write\_ea\_ctrl.tcl
- p. 69 Added script ps\_to\_png.ksh. The following scripts call it:
- write\_ea\_ctrl.tcl
- p. 70 Added script scp\_file\_web.ksh. The following scripts call it:
- write\_ea\_ctrl.tcl
- p. 94 Added program prod\_trends.f90. The following scripts call it:
- subscription.tcl
- p. 96 Added script prod\_trends.tcl.
- p. 96 Added script run\_pt.ksh. The following scripts call it:
- prod\_trends.tcl

## Summary of Changes since Version 200408.0

- p. 19 Added input argument tmp\_dir to parse\_mail.tcl
- p. 21 Added input arguments tmp\_dir and call\_cnt to populate\_db.tcl
- p. 26 Added input argument error\_out to data\_select\_req.tcl
- p. 69 Write\_ea\_ctrl.tcl now outputs wf\_plots files
- p. 70 Run\_ea.ksh now calls scfplots.pro rather than EnergyAnalysis.pro
- p. 107 Added the following columns to SUBSCRIPTION\_INPUT\_FILES database table:
  - fileName
  - date
  - laser\_ref
  - laser

### **Summary of Changes since Version 200501.0**

p.114 Added a new database table: LASER\_OPERATION

### **Summary of Changes since Version 200507.0**

p. 67 Added script get\_data\_directory.tcl. The following scripts call it:

- check\_rev\_time.tcl
- stat\_report.tcl

p. 72 Added script update\_prod\_rel\_file.tcl. The following scripts call it:

- subscription.tcl

p. 112 Added the following columns to TOO\_UPDATE database table:

- start\_lat
- stop\_lat
- start\_lon
- stop\_lon
- station\_flag

p. 116 Added the following database tables:

- SUBSCRIPTION\_CYCLES
- SUBSCRIPTION\_TRACKS
- SUBSCRIPTION\_PRODUCT\_SEG

### **Summary of Changes since Version 200512.0**

p. 54 Added environmental variable CURRENT\_REFID to subscription.tcl

### **Summary of Changes since Version 200603.0**

p. 54 Added the following environmental variables to subscription.tcl:

- DUP\_FLAG
- PLOT\_FLAG
- PLOT\_FILE

## Summary of Changes since Version 200605.0

- p.9 Added create\_subset.tcl
- p.16 Orbselect.f90 now calls sort\_file.tcl script rather than spawning UNIX sort command
- p.25 Added run\_create\_maps.ksh and subsequent Perl routines to populate\_db.tcl
- p.55 Added the following environmental variable to the subscription.tcl script:
  - CURRENT\_LASER
- p.58 Added name\_flag input argument to create\_geo.tcl
- p.71 Added laser campaign as input argument to write\_ea\_ctrl.tcl
- p.91 Added the following environmental variables to the daily\_cleanup.tcl script:
  - PROC1
  - PROC2
  - PROC3
  - ACCESS\_DAYS
- p.96 Added the CURRENT\_LASER environmental variable to the stat\_report.tcl script
- p.119 Added the following columns to the LASER\_OPERATION database table:
  - end\_cycle
  - end\_track
  - laser\_end\_time

## Summary of Changes since Version 200609.0

p.74 Added scfplots.pro under subscription.tcl

p.75 Added energy\_analysis2html.pro under subscription.tcl

p.75 Added wf\_analysis2html.pro under subscription.tcl

p.76 Moved prod\_trends.f90 to under subscription.tcl

p.102 Moved prod\_trends.tcl to “SCF Web Software Detailed Design Document”

p.119 Added RTSCM\_POINTING database table

## Summary of Changes since Version 200701.0

p.57 Removed the environmental variable PULL\_FILE from the subscription.tcl script

p.57 Added the following environmental variables to the subscription.tcl script:

- PROC\_FILE1
- PROC\_FILE2
- WEB\_WF\_DIR

p.73 Added environmental variable WEB\_WF\_DIR as input to write\_ea\_ctrl.tcl.

p.94 Added the following environmental variables to the check\_dist.tcl script:

- PROC\_FILE1
- PROC\_FILE2
- PUSH\_FILE

### **Summary of Changes since Version 200707.0**

p.122 Added LASER\_GAP MySQL database table

### **Summary of Changes since Version 200708.0**

p.94 Added the following environmental variables to the check\_dist.tcl script:

- PROC\_FILEG1

p.96 Added the following environmental variables to the check\_ps\_rscf.tcl script:

- RSCF HOST NAMEs

p.96 In ssh\_rscfs.tcl replaced site\_flag input argument with host and removed site as output argument.

### **Summary of Changes since Version 200801.0**

p.112 Removed laser\_ref from SUBSCRIPTION\_INPUT\_FILES table

p.118 Added lindex and laser to INSTRUMENT\_UPDATE table

p.119 Removed LASER\_OPERATION table

p.122 Removed LASER\_GAP database table

### **Summary of Changes since Version 200803.0**

p.31 Removed create\_por.tcl

p.88 Removed daily\_stats.tcl

p.114 Removed ISIPS\_REQUESTS database table.

p.115 Removed ISIPS\_DISTRIBUTION database table.

### **Summary of Changes since Version 200804.0**

p.103 Removed segment from REQUEST\_PRODUCT\_SEG table

p.112 Removed segment from SUBSCRIPTION\_PRODUCT\_SEG table

## Summary of Changes since Version 200807.0

- p. 21 No longer using script update\_error\_table.tcl
- p. 54 Added read\_header\_val.f90
- p. 57 Removed environmental variable EA\_FLAG.
- p. 57 Replaced environmental variables PROC\_FILE1, PROC\_FILE2, PROC\_FILEG1 with PROC\_FILE.
- p. 57 Added the following environmental variables:
  - SAVED\_DIST\_DIR
  - ACCTEST\_DIR
- p. 58 Replaced subscription.tcl with process\_data.tcl, process\_subs.tcl, process\_ea.tcl.
- p. 72 Added routine sub\_error.tcl called by process\_subs.tcl.
- p. 86 Replaced environmental variables PROC\_FILE1, PROC\_FILE2, PROC\_FILEG1 with PROC\_FILE.
- p. 88 Added mv\_saved\_files.tcl called by check\_ps\_rscf.tcl.
- p. 90 Added “Gap Checks” section.
- p.105 Added the following columns to the CREATION table:
  - data\_dir
  - laser
  - subs\_run
  - ea\_run
- p.152 Removed ERROR database table.

### **Summary of Changes since Version 200808.0**

p.106 Added index to most MySQL tables

### **Summary of Changes since Version 200809.0**

p.76 Added reference orbit as input argument to write\_ea\_ctrl.tcl

p.117 Added REQUEST\_INPUT\_FILES database table

### **Summary of Changes since Version 200810.0**

p.109 Replaced SUBSCRIPTION\_INPUT\_FILES with SUBSCRIPTION\_INPUT\_FILES1 and SUBSCRIPTION\_INPUT\_FILES2 database tables.

p.121 Replaced diagram for subscription.tcl with diagram for process\_data.tcl

### **Summary of Changes since Version 200812.0**

p.125 Added diagram for process\_ea.tcl

### **Summary of Changes since Version 200901.0**

p.68 Added environmental variable CHECK\_FULL\_PID\_FLAG

p.74 Added check\_full\_pid.tcl called by process\_subs.tcl

p.89 Modified check\_pending\_subs.tcl

p.110 Added CREATION table subs\_run options W and NF

### **Summary of Changes since Version 200906.0**

p.38 Added routine find\_uix\_delta\_mod.f90 to V5.6 scf common library. Called by data\_select.f90 and mkunique\_index.f90

### **Summary of Changes since Version 200909.0**

p.33 Updated control file example and added description of input\_files.txt for data\_select.f90.

## Summary of Changes since Version 201101.0

p.128 Updated process\_subs.tcl flowchart.

# Table of Contents

Summary of Changes since Version 200112.0 .....	iii
Summary of Changes since Version 200205.0 .....	v
Summary of Changes since Version 200208.0 .....	vi
Summary of Changes since Version 200212.0 .....	vii
Summary of Changes since Version 200302.0 .....	viii
Summary of Changes since Version 200304.0 .....	x
Summary of Changes since Version 200306.0 .....	xi
Summary of Changes since Version 200308.0 .....	xii
Summary of Changes since Version 200311.0 .....	xiii
Summary of Changes since Version 200401.0 .....	xiv
Summary of Changes since Version 200403.0 .....	xv
Summary of Changes since Version 200405.0 .....	xvi
Summary of Changes since Version 200408.0 .....	xvii
Summary of Changes since Version 200501.0 .....	xviii
Summary of Changes since Version 200507.0 .....	xviii
Summary of Changes since Version 200512.0 .....	xviii
Summary of Changes since Version 200603.0 .....	xviii
Summary of Changes since Version 200605.0 .....	xix
Summary of Changes since Version 200609.0 .....	xx
Summary of Changes since Version 200701.0 .....	xxi
Summary of Changes since Version 200707.0 .....	xxii
Summary of Changes since Version 200708.0 .....	xxii
Summary of Changes since Version 200801.0 .....	xxii
Summary of Changes since Version 200803.0 .....	xxii
Summary of Changes since Version 200804.0 .....	xxii
Summary of Changes since Version 200807.0 .....	xxiii
Summary of Changes since Version 200808.0 .....	xxiv
Summary of Changes since Version 200809.0 .....	xxiv
Summary of Changes since Version 200810.0 .....	xxiv
Summary of Changes since Version 200812.0 .....	xxiv
Summary of Changes since Version 200901.0 .....	xxiv
Summary of Changes since Version 200906.0 .....	xxiv
Summary of Changes since Version 200909.0 .....	xxiv
Summary of Changes since Version 201101.0 .....	xxv
1 Introduction .....	1
2 Data Request GUI .....	3
2.1 Invocation .....	3
2.2 Environment Definitions .....	3
2.3 List and Descriptions of the IDL Routines used by the Data Request GUI .....	5
2.3.1 orbselect.f90 .....	10
2.3.1.1 orbselect_main .....	11
2.3.1.2 georef .....	12
2.3.1.3 binrev .....	13
2.3.1.4 create_bin_config .....	13

2.3.1.5	getbins.....	14
2.3.1.6	get1bin.....	14
2.3.2	select_processed_tracks.f90.....	15
2.3.3	select_time_tracks.f90.....	15
2.3.4	time_tracks.f90.....	16
2.3.5	sort_file.tcl.....	16
3	Parsing the Email and Populating the Mysql Data Request Tables.....	18
3.1	Invocation and environment.....	19
3.2	Parse Mail scripts parse_mail.tcl.....	20
3.2.1	exist_check.tcl.....	20
3.2.2	write_log.tcl.....	21
3.2.3	qa_mail.tcl.....	21
3.3	populate_db.tcl.....	22
3.3.1	subset.tcl.....	23
3.3.2	sort_unique.tcl.....	24
3.3.3	find_span.tcl.....	24
3.3.4	zero_padding.tcl.....	24
3.3.5	zero_out.tcl.....	24
3.3.6	convert2j2000.tcl.....	25
3.3.7	find_passID.f90.....	25
3.3.8	send_mail_user.pl.....	25
3.4	data_select_req.tcl.....	25
3.5	run_create_maps.ksh.....	25
3.5.1	create_maps.pl.....	26
3.5.1.1	initialize_arrays.pm.....	26
3.5.1.2	get_area_dates.pm.....	27
3.5.1.3	get_product_seg.pm.....	27
3.5.1.4	get_tracks.pm.....	27
3.5.1.5	get_cycles.pm.....	27
3.5.1.6	send_mail_local.pm.....	28
4	Create Products for Special Requests.....	29
4.1	Main script to process special requests: data_select_req.tcl.....	30
4.1.1	add_to_log_file.tcl.....	31
4.1.2	filter_files_by_release_or_version.tcl.....	31
4.1.3	lsearch_all.tcl.....	32
4.1.4	sort_unique_version.....	32
4.1.5	run_ds_pc.tcl.....	32
4.1.5.1	write_ds_ctrl.tcl.....	33
4.1.5.2	catch_errors.tcl.....	33
4.1.6	data_select.f90.....	33
4.1.6.1	const_scf_mod.f90.....	36
4.1.6.2	Anc70_scf_mod.f90.....	36
4.1.6.3	ANC70_mod.f90.....	36
4.1.6.4	filesize_mod.f90.....	36
4.1.6.5	open_bin_file_mod.f90.....	36
4.1.6.6	fbins_mod.f90.....	36

4.1.6.7	common_files_mod.f90 .....	36
4.1.6.8	read_ds_ctrl_mod.f90 .....	36
4.1.6.9	read_rev_file_mod.f90.....	37
4.1.6.10	create_geobins_mod.f90.....	37
4.1.6.11	read_filenames_mod.f90.....	37
4.1.6.12	read_geobins_mod.f90.....	37
4.1.6.13	read_binrev_mod.f90.....	37
4.1.6.14	sort_file_mod.f90.....	37
4.1.6.15	read_unique_mod.f90 .....	37
4.1.6.16	filter_req_mod.f90 .....	37
4.1.6.17	find_uix_delta_mod.f90.....	38
4.1.6.18	rewrite_req.f90.....	39
4.1.7	write_pr_ctrl.tcl.....	40
4.1.8	write_pc_ctrl.tcl .....	40
4.1.9	parse_req.f90.....	40
4.1.9.1	read_pr_ctrl_mod.f90.....	41
4.1.9.2	read_req_mod.f90 .....	41
4.1.9.3	write_req_mod.f90.....	41
4.1.10	prod_create.f90 .....	41
4.1.10.1	prod_common_mod.f90.....	42
4.1.10.2	prod_reader_mod.f90.....	43
4.1.10.3	prod_writer_mod.f90 .....	43
4.1.10.4	read_pc_ctrl_mod.f90 .....	43
4.1.10.5	read_prod_recs_mod.f90 .....	43
4.1.11	prepare_browse.tcl.....	43
4.1.11.1	qapg.f90 .....	43
4.1.11.2	Runbrowse.ksh.....	44
4.1.11.2.1	<a href="#">Qabrowse.pro .....</a>	<a href="#">44</a>
4.1.12	create_pass_files.tcl .....	44
4.1.13	mkpass.f90 .....	44
4.1.13.1	read_pass_control_mod.f90.....	47
4.1.13.2	read_glas_record_mod.f90.....	47
4.1.13.3	qsorti_mod.f90 .....	47
4.1.13.4	find_pass_mod.f90.....	47
4.1.14	create_unique_files.tcl .....	47
4.1.15	mkunique_index.f90 .....	47
4.1.15.1	read_glas_unique_control_mod.f90.....	49
4.1.16	create_binrev.tcl.....	49
4.1.17	mkbinrev.f90.....	49
4.1.17.1	read_binrev_control_mod.f90.....	51
4.1.18	create_geo.tcl .....	51
4.1.19	mkgeo.f90 .....	51
4.1.19.1	read_geo_control_mod.f90.....	53
4.1.20	parameters_for_pdr.tcl .....	53
4.1.21	invoke_perl.ksh.....	53
4.1.22	compare_lists.tcl .....	53

4.1.23	lsearch_all_idx.tcl .....	54
4.1.24	send_mail_mscf.pl .....	55
4.1.25	send_mail_ccb.pl .....	55
4.1.26	read_header_val.f90.....	55
4.1.26.1	get_header_val_mod.f90.....	56
5	Processing Data.....	57
5.1	Invocation .....	57
5.2	Environment Definitions.....	57
5.3	Main script to process data: process_data.tcl.....	57
5.3.1	create_binrev.tcl.....	58
5.3.2	create_geo.tcl .....	59
5.3.3	create_pass_files.tcl .....	60
5.3.4	create_unique_files.tcl .....	61
5.3.5	find_track.tcl .....	61
5.3.6	read_fn_file.tcl.....	62
2.3.5.17	read_keyword.tcl.....	62
5.3.7	modify_processed_tracks.tcl.....	63
5.3.8	scp_rev_file.ksh .....	64
5.3.9	check_rev_time.tcl.....	64
2.3.5.18	get_data_directory.tcl.....	65
5.3.10	check_pid.tcl .....	66
5.3.11	time_sort.tcl .....	66
5.3.12	update_prod_rel_file.tcl .....	67
6	Fulfilling Subscriptions.....	69
6.1	Invocation .....	69
6.2	Environment Definitions.....	69
6.3	Main script to process subscriptions: process_subs.tcl.....	69
6.3.1	create_pdr.tcl.....	70
6.3.2	lsearch_all.tcl .....	71
6.3.3	run_ds_pc.tcl.....	71
6.3.3.1.	write_ds_ctrl.tcl .....	72
6.3.3.2.	write_pr_ctrl.tcl.....	73
6.3.3.3.	write_ps_ctrl.tcl .....	74
6.3.4	sub_error.tcl .....	74
6.3.5	check_full_pid.tcl.....	75
7	Energy Analysis .....	76
7.1	Invocation .....	76
7.2	Environment Definitions.....	76
7.3	Main script to run energy analysis: process_ea.tcl .....	76
7.3.1	write_ea_ctrl.tcl .....	77
7.3.1.1.	run_ea.ksh .....	77
7.3.1.1.1.	scfplots.pro.....	78
7.3.1.1.2.	energy_analysis2html.pro .....	79
7.3.1.1.3.	wf_analysis2html.pro.....	79
7.3.1.2.	ps_to_png.ksh .....	79
7.3.1.3.	scp_file_web.ksh.....	80

8	Data Distribution.....	81
8.1	parameters_for_pdr.tcl.....	82
8.2	invoke_perl.ksh.....	83
8.3	create_pdr.pl.....	83
9	Submitting Product QA Updates to the I-SIPS.....	85
9.1	Invocation .....	85
9.2	Input Arguments .....	85
9.3	Environment Definitions.....	85
9.4	submit_qa.tcl.....	85
9.4.1	create_qauf.tcl.....	86
10	Monitoring .....	87
10.1	check_dist.tcl .....	87
10.2	check_ps.tcl.....	87
10.3	check_ps_rscf.tcl.....	88
10.4.1	ssh_rscfs.tcl.....	88
10.4.2	mv_saved_files.tcl .....	89
11	Gap Checks .....	90
11.1	check_pending_subs.tcl .....	90
11.2	check_pending_ea.tcl.....	90
11.3	file_gap.tcl .....	91
11.2.1	check_laser_op.tcl.....	91
11.2.2	get_file_times.tcl.....	92
11.4	check_subs_ea.tcl.....	92
12	Clean-up.....	94
12.1	daily_cleanup.tcl .....	94
12.1.2	get_file_j2000.tcl .....	94
12.1.3	get_pid_j2000.tcl .....	95
12.1.4	check_inst_update.tcl.....	96
13	Instrument Updates .....	97
13.1	Invocation and environment .....	97
13.2	Environment Definitions.....	97
13.3	parse_istat_mail.tcl .....	97
14	I-SIPS Distribution Monitoring .....	99
14.1	Invocation and environment .....	99
14.2	Environment Definitions.....	99
14.3	parse_stat_mail.tcl .....	99
15	Daily Statistics Report .....	101
15.1	Invocation and environment .....	101
15.2	Environment Definitions.....	101
15.3	Stat_report.tcl.....	101
15.3.1	send_mail_report.pl .....	101
	Appendix A - Mysql Database Tables .....	103
A.1	Table Name: USER.....	105
A.2	Table Name: SPECIAL_REQUEST_USER .....	105
A.3	Table Name: SUBSCRIPTION_USER .....	106
A.4	Table Name: ISIPS_PRODUCT_ID.....	107

A.5 Table Name: REQUEST_PRODUCT_SEG.....	108
A.6 Table Name: REQUEST_TRACKS .....	109
A.7 Table Name: REQUEST_CYCLES.....	109
A.8 Table Name: ISIPS_SUBSCRIPTIONS.....	110
A.9 Table Name: CREATION.....	110
A.10 Table Name: SUBSCRIPTION_INPUT_FILES1 .....	111
A.11 Table Name: SUBSCRIPTION_INPUT_FILES2 .....	112
A.12 Table Name: DISTRIBUTION.....	112
A.13 Table Name: DISTRIBUTION_FILES .....	113
A.14 Table Name: [rSCF]_PRODUCT_ID.....	113
A.15 Table Name: SPECIAL_REQUEST_PID .....	114
A.16 Table Name: QA_PRODUCT_UPDATE.....	114
A.17 Table Name: TOO_UPDATE.....	115
A.18 Table Name: INSTRUMENT_UPDATE .....	116
A.19 Table Name: RTSCM_UPDATE.....	116
A.20 Table Name: RTSCM_POINTING .....	117
A.21 Table Name: SUBSCRIPTION_PRODUCT_SEG .....	118
A.22 Table Name: SUBSCRIPTION_TRACKS.....	118
A.23 Table Name: SUBSCRIPTION_CYCLES .....	119
A.24 Table Name: REQUEST_INPUT_FILES .....	119
Appendix B - Flowcharts .....	121
B.1 Flowchart for data_select.f90.....	121
B.2 Flowchart for prod_create.f90.....	123
B.3 Flowchart for process_data.tcl .....	124
B.4 Flowchart for process_ea.tcl .....	127
B.5 Flowchart for process_subs.tcl.....	128
B.6 Flowchart for data_select_req.tcl .....	132
Appendix C – File formats.....	133
C.1 REQ File Format.....	133

## 1 Introduction

The only way the rSCFs receive GLAS levels 1 and 2 data is by requesting it from the mSCF. All data is distributed in GLAS standard data product format as described in the respective User's guides. Subsetting is done by time and/or geographic area but not at the parameter level.

There are three types of requests; subscriptions, special requests, and quick look requests.

**Subscriptions** are to be submitted to the mSCF for standing orders. Subscriptions will be automatically executed every time the mSCF receives a new product set from the I-SIPS. In normal mode, each time a subscription is executed one product set will be created.

A **special request** is a one-time request. Special requests will be executed on all product sets that exist at the time the request is received. One product set will be output with all the data unless any one file size is greater than 2 Gbytes, in that case multiple product sets will be created, breaking them by time.

A **quick look request** will be similar to a data request, except that it will request special processing at the ISIPS if the data has not yet been processed

Processing a data request consists of 3 steps:

### Step 1

The rSCF user submits a special request or subscription using the data request graphical user interface that sends the request parameters in an email to user scf on icesat0. – *This part is the responsibility of the scientist.*

### Step 2

A script automatically runs on each emailed request that parses the data request parameters from the email and populates the mysql data request tables with those parameters.

### Step 3

For special requests, a script is executed right after populating the data request tables to fill the request.

For subscriptions, a cron job looks for new product sets and starts the subscription script when a new product set arrives to the mSCF ingest directory. (*Steps 2 and 3 are the responsibility of the mSCF.*)

Figure 1.1 shows the overall design of this system, and location of where the different modules are run. Basically the software consists of 5 main packages:

- The Data Request GUI
- The scripts to parse the email from the GUI and populate the data request data base
- The scripts and other software to create the products requested
- Scripts to update the data request data base
- Distribution scripts to distribute the data to the rSCF

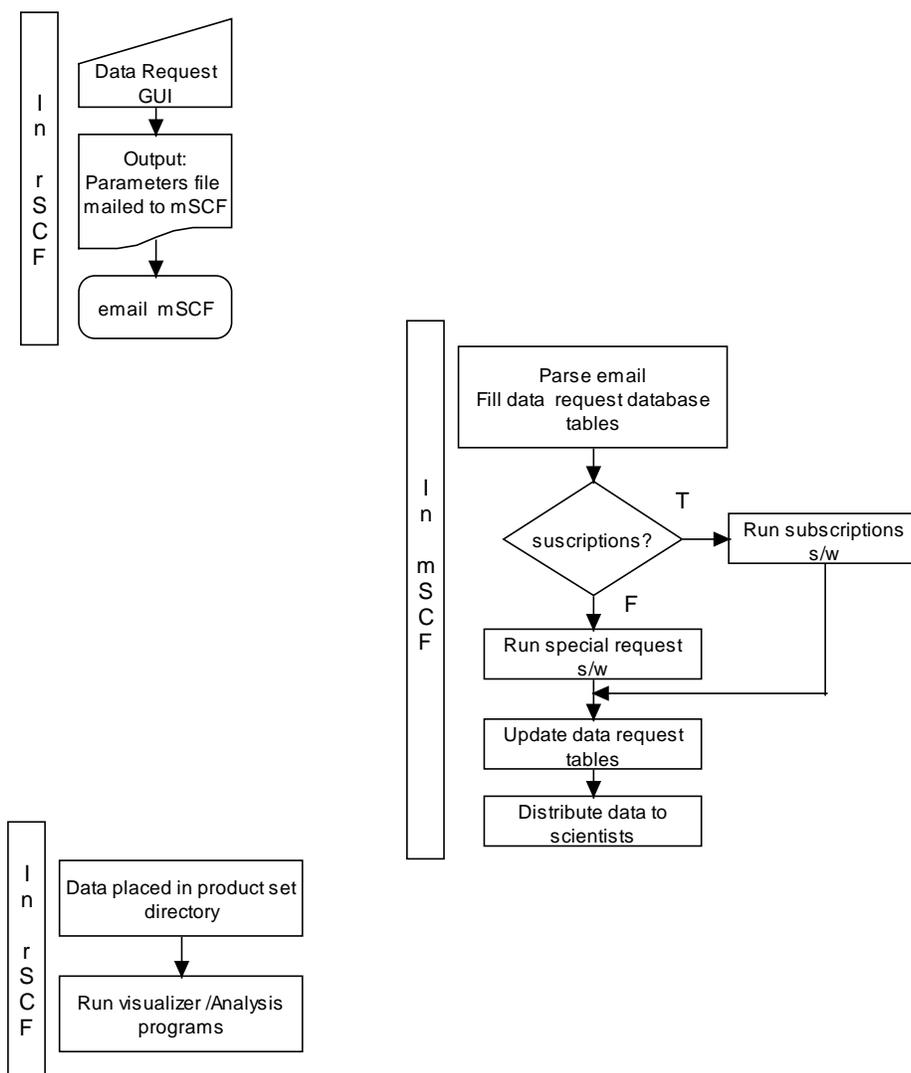


Fig 1-1 Overall Design

## 2 Data Request GUI

The data request GUI allows the user to select ICESat products by region, time, and/or set of tracks. The most restrictive information will be used - for example, if the user selects a time span and a set of tracks, he or she will receive only the subset of the tracks that are within the time span. There are two types of data requests: by subscription and by special request. The first window that pops up has two buttons for each of the data request types. After the user selects the type he wants, that window disappears and from that point the GUI is specific to the data request selected. The new user interface has two windows. The first is to define selection criteria and submit requests; the second is a help window in which the user can see a (DEM) of his/her selected region and the 8 or 91 day GLAS coverage within it. If the user submits a special request, he/she is only shown the GLAS tracks that have been processed that meet that request. If the user submits a subscription, he/she will see the GLAS coverage from the full 8-day and/or 91 day set of tracks. Note that most of this software is also used by the “front-end” of the data visualization software to select the geographical and temporal spans of data to visualize. Routines just used by the visualization software will be denoted as such.

### 2.1 Invocation

Run `/SCF/bin/ops/run_data_request.ksh`. This script defines all of the environment variables needed to run the data request GUI and creates the working directory.

### 2.2 Environment Definitions

```
export SHLIB_PATH= [location of the shared libraries]
export ICESatVIS_MAIN= [location of the data request GUI]
export ICESatVIS_TMP= [location of the temporary directory]
export ICESatVIS_HELP= [location of the help files for the data request GUI]
export ICESatVIS_PARAMETERS= [location of files in which are saved selected parameters
from the data request GUI]
export ICESatVIS_ANC= [location of the ancillary data ]
export ICESatVIS_REFORBIT= [location of the reference orbit data]
export ICESatVIS_DATA= [location of the map data]
export ICESatVIS_DEM= [location of the DEM data]
export ICESatVIS_BIN= [location of the executable and script files]
export ICESatVIS_OUT= [the directory to put the requested product set]
export TMP_DIR= [../..]dir_$$ [the working directory-has a unique name and will be deleted
when the processing is finished. $$ is the process ID]
mkdir dir_$$
cp $ICESATVIS_ANC/data/processed_8_pass.txt $TMP_DIR/. [copy the file with the list of all
the processed tracks to the temporary directory]
cp $ICESATVIS_ANC/data/processed_91_pass.txt $TMP_DIR/.
export TRACK_FILE8=t8p.reforb
export NO_TRACK8=119
```

```
export TRACK_FILE91=t91p.reforb
export NO_TRACK91=2723
# Rewrite processed_x_pass.txt to get rid of duplicated tracks
$ICESATVIS_BIN/rewrite_processed_tracks.tcl $TMP_DIR/processed_8_pass.txt
$ICESATVIS_BIN/rewrite_processed_tracks.tcl $TMP_DIR/processed_91_pass.txt
idl $ICESATVIS_BIN/run_data_request
```

## 2.3 List and Descriptions of the IDL Routines used by the Data Request GUI

### **append.pro**

Appends an element to an array

### **browser.pro**

Displays list of available browse products. Visualizer only.

### **browser\_data\_gui.pro**

Creates the main window for the Data Visualization GUI. It allows the user to select ICESat products by region, time and/or set of tracks. It consists of two main windows; a submittal window and a help window. The submittal window is used to define selection criteria and submit requests. The help window shows the user a map or DEM of his/her selected region and the 8 or 91 day GLAS coverage within it. The user is also able to load the tracks selected in the help window to the submittal window. The difference between the subscription GUI and the special request GUI is in the time span selection. In the subscription GUI, the user selects the begin and end of year, month, day and hour.

### **browser\_visualizer.pro**

Displays a window to select visualizer display, browse products display, or subsetting. Visualizer only.

### **check\_file\_exist.pro**

Checks that input file exists

### **choosemap.pro**

This program plots the map and/or DEM over the world, Antarctica, and Greenland. It allows the user to select a region by entering values in the longitude and latitude text field or by zooming the map using the mouse. The user can plot the GLAS ground tracks over the selected region. The region lat/lon values and the tracks are transferred back to the main window.

### **cmps\_form.pro**

Creates a form to configure a postscript file

### **continueevent.pro**

Summarizes and checks visualizer request once “continue” button is pressed

### **convert\_time.pro**

Converts date to J2000 secs

### **create\_save\_file.pro**

Saves all the current selected special request parameters into a file designated by the user.

### **create\_save\_file\_ds.pro**

Saves all the current selected subscription parameters into a file designated by the user.

**data\_request\_gui.pro**

Allows the user to select between special request or subscription request.

**display\_browser\_2\_mscf.pro**

Displays more browse products on the mSCF. Visualizer only.

**display\_browser\_2\_rscf.pro**

Displays more browse products on the rSCFs. Visualizer only.

**display\_browser\_mscf.pro**

Displays browse products on the mSCF. Visualizer only.

**display\_browser\_rscf.pro**

Displays browse products on the rSCFs. Visualizer only.

**display\_select.pro**

Creates a window to display selected parameters

**drawdem.pro**

Displays a Digital Elevation Model. It calculates the size of the region to be displayed and selects the proper data file with the proper resolution to use so the DEM read won't contain more values than the number of pixels that can fit in the window.

**drawmap.pro**

Displays a map within the longitude and latitude region that the user selected.

**find\_ref\_orbit.pro**

Finds the reference orbit in the start\_date-end\_date time span.

**get\_cycle.pro**

Gets a cycle number from a selected cycles string

**getcolor.pro**

Defines color names and values

**getlatlon.pro**

Gets latitude and longitude values from a text widget

**getnow.pro**

Gets the current day and time

**gettrackarray.pro**

Runs the executable orbselect to get a list of the tracks in the selected region. If the selected area is the whole world, the code skip executing orbselect.

**highres\_to\_plot.pro**

Given the size of the selected region, it selects the DEM file that has the right resolution to display the DEM without overwriting the pixels.

**icesatvis\_ds.pro**

Creates the first window for the visualizer.

**infoevent.pro**

Handles the events of the special request and subscription windows.

**loadpardsevent.pro**

Loads previously saved visualizer parameters

**loadparevent.pro**

Loads previously saved data request parameters

**maindsevent.pro**

Handles the events of the main window for the Data Visualization GUI

**mainevent.pro**

Handles the events of the main window for the Data Request GUI

**mapinfoevent.pro**

Handles the events of the map window

**mapsetdefaults.pro**

Sets map defaults

**mkunique\_index.pro**

Creates the unique index file for the GLA product

**plotcolortrack.pro**

Highlights a given track.

**plotevents.pro**

Handles the events that relate to plotting the tracks

**plotlatlon.pro**

Plots the map

**plottrack.pro**

Reads the track list and indices of tracks that cover a selected region from the output of orbselect. Reads the track files: t8p.reforb, t91p.reforb and displays the tracks in the list.

**processed\_products.pro**

Determines which GLA products are available in the data directory and creates a processed\_products.txt file listing them. Used by the visualizer only.

**product\_select\_dr.pro**

Displays the products window and handles the window events for the Data Request GUI

**product\_select\_vis.pro**

Displays the products window and handles the window events for the Data Visualization GUI

**putlatlon.pro**

Puts the lat/lon values in their text field

**puttracks.pro**

Loads the selected tracks from the map window to the main window

**region\_mask.pro**

Displays region masks

**request\_gui.pro**

Creates the main window for the Data Request GUI. It allows the user to select ICESat products by region, time and/or set of tracks. It consists of two main windows; a submittal window and a help window. The submittal window is used to define selection criteria and submit requests. The help window shows the user a map or DEM of his/her selected region and the 8 or 91 day GLAS coverage within it. The user is also able to load the tracks selected in the help window to the submittal window.

**resetcolor.pro**

Resets the color of the selected track

**savepardsevent.pro**

Saves visualizer parameters

**saveparevent.pro**

Saves data request parameters

**selectalltracks.pro**

Selects all tracks for that reference ID on the map

**set\_env\_pc.pro**

Sets environmental variables for IDL Project for visualizer use on PC.

**setdefaults.pro**

Puts default values in the GUI

**sort\_tracks.pro**

Sorts a list of tracks

**submit\_select.pro**

Writes a list of the submitted parameters to a file. Each parameter is written in a new line with a keyword prime to the value. For example: PRODUCT=GLA05. When all the parameters are listed, it calls a mail\_request script that mails the file to [scf@icesat0.gsfc.nasa.gov](mailto:scf@icesat0.gsfc.nasa.gov).

#### **submitevent.pro**

Submits the data request. If there is not enough information to submit, an error message will be displayed that prompts the user for the missing information.

Another error message will be displayed if the selected cycle is not in the time span.

Only when all required information is selected will the request be submitted.

#### **summarizeevent.pro**

Summarizes the user's selection input.

#### **swdelete.pro**

Deletes a scroll window

#### **swindow.pro**

Creates a scroll window

#### **tvimage.pro**

Displays an image

#### **Create\_subset.tcl**

Creates subsetted products.. Runs as part of the visualizer.

#### **Arguments:**

Input argument 1 temporary directory name.

Input argument 2 output directory name.

Input argument 3 is file name specified by user.

#### **Error Handling:**

- None

#### **Description of Algorithm:**

- Runs data\_select and prod\_create to create subsetted products
- Creates BN, GR, UR and PS files for subsetted products
- Moves subsetted products from temporary directory to output directory and renames files to user-specified file names

#### **Calling Routines:**

- browser\_visualizer.pro
- browser.pro

## Fortran Routines used by the Data Request GUI

### 2.3.1 orbselect.f90

The software determines the tracks and portions thereof of a reference orbit that traverse a specific geographic region. The data request interface uses orbselect to show projected GLAS coverage on a map in the help window. Given a geographic region defined by a rectangle in latitude and longitude, orbselect outputs a list of GLAS tracks that go through that rectangle and the indices within those tracks. A separate list is obtained for each reference orbit (i.e. 8-day and 91-day).

where

- Reference orbit – an orbit which is maintained to within 1 km cross-track on the ground
- Track – one orbit revolution within a reference orbit, starting and ending at the ascending node
- Track Index – index within the track latitude and longitude arrays that define a unique ground location – currently these are defined for each second along the track

The Reference orbit ground track file is defined from the reference orbit ephemerides distributed by UTCSR. It is interpolated to 1 sec intervals along the track and stored as a direct access binary file where each record has arrays giving the latitude and longitude coordinates for one track in the format defined in the *SCF Architectural Design Document*.

- 2 Bin and georeference tables as described in the *SCF Architectural Design Document* are created for the two reference orbit ground track files similar to the bin and georeference tables that are created for each data product set of data. Rather than listing the beginning and ending unique record numbers that traverse each bin, the track number and beginning and ending 1 sec indices within the track are listed in the bin table.

Given a geographic region defined as rectangle in latitude and longitude, first the geographic bins contained in that region are calculated analytically using the georeference bin configuration presented in the *SCF Architectural Design Document*. The georeference and bin tables are accessed to determine what tracks (and portions thereof) traverse the bins selected.

Now we know what tracks go through the enlarged geographic region and which portions of each of them. This only gives this information to the precision of the geographic bin configuration. To get rid of the false positives, the latitude and longitudes are read from the reference orbit ground track files for the tracks and indices selected and checked against the selected region to output a refined list of tracks and indices. The output is then sorted primarily by track number and secondarily by index within the track.

#### **Input parameters:**

Begin latitude, begin longitude, end latitude, end longitude

#### **Input files:**

orbselect.inp

**contains:** reference orbit file name, bin directory name, georeference directory name.

**Output files:**

orbselect.out

**contains:** track #, begin index, ending index (all integers)

**Error Handling:**

Calls GLAS\_error for the following conditions:

- error in listing the arguments
- error opening file
- error reading file

Returns an exit(3) to the main script.

**Description of Algorithm:**

- Open the input file to read the name of the georeference file and the bin file
- Call subroutine orbselect\_main to create a file that contains all the tracks and their indices that are in the selected area
- Call a tcl script to sort the above file by track (primarily) and index (secondarily) and write the sorted file into orbselect.sort
- Read orbselect.sort which has several contiguous records per track and rewrite a file that groups them into record spans
- Read the tracks file for the above records span and check if the longitude and the latitude of the track are within the selected area. Write into the output file the records that are within the selected area.
- Check if the above records span consists of one record number. If so, add the consecutive record, so later the track can be displayed.

**Subroutines called:**

/SCF/src/orbselect\_dir/V2\_prelim /orbselect\_mod.f90 consists of :

- orbselect\_main
- georef
- binrev
- sort\_file.tcl

### 2.3.1.1 orbselect\_main

Calls the subroutine to calculate the part of the tracks that cross the selected area

**Input parameters:**

i\_geo, i\_bin, slat, nlat, wlon, elon

where:

- slat, nlat, wlon, elon are the coordinates of the selected region.
- i\_srt, i\_geo, i\_bin are the unit file numbers of the output file (orbselect.sort), the georeference file, and the binrev file.

**Error Handling:**

None

**Description of Algorithm:**

- Call getbins to get all the bin numbers in the selected area
- Call georef to get the record in the binrev file that corresponds to the bins
- Call binrev to get the records in the tracks file that to through the bin

**2.3.1.2 georef**

Reads the georeference table to finds the records in binrev file for the input bin number

**Input parameters:**

i\_geo, ibin, irec1, irec2, lnobin, first

where:

- i\_geo is the georeference file unit
- ibin is the bin number
- irec1, irec2 are the beginning and ending records in the binrev file for bin ibin
- lnobin – logical indicating no bin number found in georeference file equal to ibin ??
- first indicates that it is the first time that the subroutine is called

**Error Handling:**

None

**Description of Algorithm:**

the first call to this routine:

- reads the georeference file until finds the first data record, the first two characters are “GD”.when the beginning of the record contains the string "GD", check if the bin number is equal to ibin. If it is equal then return, passing back irec1 and irec2 read from that record, if the bin number on the georeference file is greater than ibin, then return with lnobin set equal to true?

each succeeding call

- check if the bin retrieved from the georeference file is greater than or equal to ibin and if so, then return.
- if the bin retrieved from the georeference file is less than ibin, keep reading the file until it is greater than or equal to ibin If find a bin equal to ibin then return with the corresponding irec1 and irec2, if do not find a bin equal to ibin then return with lnobin set to true.

### 2.3.1.3 binrev

Find records in the track file of the tracks that cross through the input bin

#### **Input parameters:**

i\_bin,i\_srt,irec1,irec2

where:

- i\_bin is the bin number
- i\_srt is the output file unit for orbselect.sort
- irec1, irec2 are the beginning and ending records in the binrev file

#### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error reading a file

#### **Description of Algorithm:**

- Read the binrev file from irec1 to irec2, each record contains the geographic bins #, track #, beginning index of track within the bin, and ending index of the track within the bin.
- Write to output file orbselect.sort the track # and the beginning and ending indices within that track
- 

#### **Subroutines called:**

/SCF/src/common/V2\_prelim/fbins\_mod.f90 consists of:

- create\_bin\_config
- getbins
- get1bin

### 2.3.1.4 create\_bin\_config

Creates the bin configuration. It defines the bin size, the number of grids in longitude and latitude and the beginning and ending values of the latitude and longitude. Those values are hard coded in the file.

#### **Input parameters:**

lgren,selat,wnlnt,wlon,selon,londiv,latdiv, bin\_size\_lon,bin\_size\_lat

where:

- lgren is the crossing Greenwich flag. It is true when geographic bin configuration crosses the Greenwich
- selat and wnlnt are the southeastlatitude and northwest longitude boundaries of the geographic bin configuration. Bin 1 starts at selat and wnlnt.
- wnlnt and selon are northwest latitude and southeast longitude boundaries of the geographic bin configuration. The last bin ends here.
- londiv,latdiv are the number of bins in the longitude and the latitude direction respectively

- bin\_size\_lon,bin\_size\_lat are the sizes of the bin in the longitude and the latitude direction respectively

**Error Handling:**

None

**Description of Algorithm:**

- Determine the bin configuration

**2.3.1.5 getbins**

get all the bins in a rectangular area.

**Input parameters:**

gselat\_t,gnwlon\_t,gnwlat\_t,gselon\_t,

output parameters:

nbins,nobin,i\_err\_sev

where:

- nbins is the array giving the geographic bins in the given region
- nobin is the number of bins in the array nbins
- gselat\_t,gnwlon\_t,gnwlat\_t,gselon\_t are the geographic coordinates defining the region for which to find the bins
- i\_err\_sev is the error severity code

**Error Handling:**

Calls GLAS\_error for the following conditions:

- Number of georeference bins exceeds maximum.
- No bin was found.

**Description of Algorithm:**

- call create\_bin\_config to get the bin configuration
- check that region limits are within the geographic bin configuration limits
- determine bin group within which region starts
- check if Greenwich splits the region
- region split....divide into two pieces
- loop over the area(s) contained by the region
- locate the starting bin number
- locate all bins within lat group which are contained in region
- proceed to next bin group and determine if outside region

**2.3.1.6 get1bin**

get the bin number of specific lon and lat.

**Assumption:** In the bin configuration the Greenwich crossing flag is false.

**Input parameters:**

ibin,lat\_t,lon\_t,i\_err\_sev

where:

- ibin is the bin number
- lat\_t and lon\_t are the latitude and longitude of the spot to find the bin
- i\_err\_sev is the error severity code

**Error Handling:**

Calls GLAS\_error for the following conditions:

- Value of the bin greater than gi\_max\_geobins
- Greenwich crossing flag is true why is this an error?

**Description of Algorithm:**

- call create\_bin\_config to get the bin configuration
- calculate the bin number value only if the Greenwich crossing flag is false .

**2.3.2 select\_processed\_tracks.f90**

Read the tracks list from the orbselect output and rewrite the output file only with the tracks that were processed.

When the selected area is the whole world, the output is all the tracks that were processed.

**Input parameters:**

orbselect\_out\_file, processed\_tracks\_file, dir

where:

dir is the directory where the files are

**Error Handling:**

-iostat

**Description of Algorithm:**

- Open the processed\_tracks\_file to read the list of processed tracks
- Read the tracks list from orbselect\_out\_file
- Copy to the output file (with the extension .process) the orbselect\_out\_file leaving only the tracks that are in processed\_tracks\_file.

**2.3.3 select\_time\_tracks.f90**

Reads the tracks list from the orbselect output and copies the orbselect output file to the output file only with the tracks that are in the time span. (Takes into account only the tracks but not the cycles. For example:

if there is a track 30, cycle 2 in the time span but only track 30 cycle 1 was processed , it will still put track 30 on the list

**input parameters:**

orbselect\_out\_file, tracks\_list\_file, dir

where:

tracks\_list\_file is the file with a list of tracks in the time span.

dir is the directory where the files are.

**Error Handling:**

-Iostat

**Description of Algorithm:**

-Open the input file to read the list of tracks

-Read the tracks list from orbselect\_out\_file.

-Copy the orbselect\_out\_file to the output file (extension .time\_process) only with the tracks that show also in track\_list\_file.

### 2.3.4 time\_tracks.f90

Find the tracks in a time span.

**Input parameters:**

Begin\_time, end\_time, out\_dir, rev\_file, file8, file91, file3, err\_file

Where:

Begin\_time, end\_time is the time span (J2000 format)

File8, file91, file3 are text output file with a list of tracks from the 8 days , 91 days or transfer orbit.

**Error Handling:**

Calls Glas\_error for the following conditions:

- error opening file
- error reading file

**Description of Algorithm:**

- Call read\_rev\_file to reads rev file and find the passID's filtered by time span.

**Calling Subroutines:**

read\_rev\_file\_mod.f90

### 2.3.5 sort\_file.tcl

Sorts input file into sorted output file.

**Arguments:**

Input argument 1 is input file name with full path.

Input argument 2 is output file name with full path.

**Error Handling:**

Sets errNum =1 for the following conditions:

- error opening files to read/write

**Description of Algorithm:**

- Uses TCL lsort to sort input file into output file

**Calling Subroutines:**

orbselect.f90

### 3 Parsing the Email and Populating the Mysql Data Request Tables

The information from the subscription or the special request submitted using the Data Request Interface is sent by email to the scf account. Every new email that arrives will trigger the execution of the readMail.ksh script that will parse and retrieve the data request parameters and insert them to the mysql database tables.

The readMail.ksh script is triggered every time a new mail is received in the scf account because the full path name of the script is in the /etc/mail/forwards/scf.forward file.

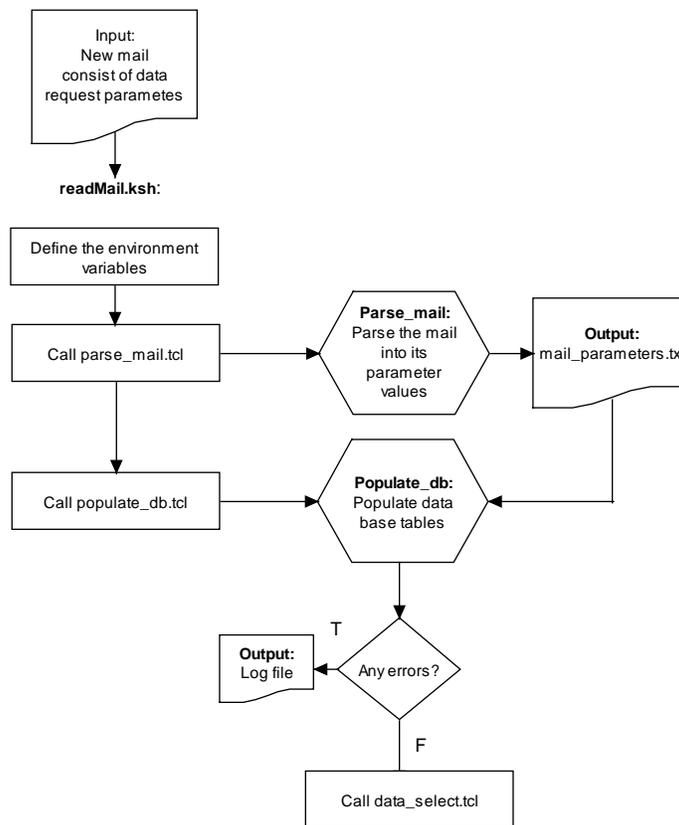


Fig 3-1

### 3.1 Invocation and environment

The readMail.ksh script automatically executes whenever a new email is received in the scf account. This script defines all the environment variables needed to process the email and executes the parse mail script and the populate db script.

#### Environment Definitions:

```
export PATH=`cat /etc/PATH`:$PATH    [Include the /etc/ directory to the path]
export ICESATVIS_TMP=                [temporary directory]
export SHLIB_PATH=                   [location of the shared libraries]
export ICESAT_PRODUCT_SET=           [location of data products – data are in L*
                                      subdirectories]

export ICESATVIS_BIN=                [location of the executable program and scripts]
export ICESATVIS Anc=                [location of the ancillary data]
export ANC07_FILE1=[location of anc07_001_01_0000.dat file for qapg]
export ANC07_FILE2=[location of anc07_001_01_0003.dat file for qapg]
export ANC07_FILE3=[location of anc07_001_01_0004.dat file for qapg]
export ANC07_FILE4=[location of anc07_001_01_0006.dat file for qapg]
export REV_FILE=                     [rev file path and name]
export SCF_ERROR_FILE=/SCF/ancillary_data/scf70_000_00_00.dat
export REQ_HEADER_SIZE=              [the size of the req file with headers and no data]
export TMP_DIR=$ICESATVIS_TMP/dir_$$ [the working directory]
export PID_NUMBER=$$                 [the process ID number]
mkdir $TMP_DIR                       [creates the working directory]
export DIST_DIR=                     [distribution directory]
export LOG_DIRECTORY=                [logfile directory]
export SRC_DIR=                       [the source directory of the perl scripts]
export LOCAL_NAME=                   [mscf or rscf]
export LOCAL_HOST=                   [host name]
export DB_NAME=                       [mysql database name]
export DB_USER=                       [mysql database user name]
export ISIPS_DIST=                   [SCF distribution cache for I-SIPS]
export SPECIAL_REQUEST_PATH=         [/SCF/product_sets special requests directory path]

#Read read-only file for database password
export DB_PASSWD=                    [mysql database password]
```

## 3.2 Parse Mail scripts parse\_mail.tcl

This script parses the mail and writes the parameters into a file.  
Calls qa\_mail.tcl if the mail came from a qa\_form .

### Input parameters:

tmp\_dir – temporary directory for mail processing  
input\_par\_file – the name of the file where the mail parameters will be written

### output files:

from\_mail\_file – text file that has all the data request parameters.

The file consist of:

request\_flag: 1 for special request, 2 for subscription

user\_name: the user name of the person that sent the data request email.

institution\_name: The name of the institution where the email was sent from.

n\_of\_products: number of products to process

product\_name

t1,t2: start and end time to process the data

lat1,lat2,lon1,lon2: the area to process

n\_of\_cycles: number of cycles to process

cycle\_number

cycle\_type: 1 for 8 days repeat, 2 for 91 days repeat

n\_of\_tracks: number of tracks to process

track\_number

track\_type: 1 for 8 days repeat, 2 for 91 days repeat

out.txt, error.txt: output and errors list text files.

**Error Handling:** The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and will be written into the log file.

- Can't open a file
- GLAS fatal error

### Description of Script:

- Define the keywords of the data request parameters
- Retrieve the values of the data request parameters

### Subroutines called:

- exist\_check.tcl
- write\_log.tcl
- update\_error\_table.tcl
- qa\_mail.tcl

### 3.2.1 exist\_check.tcl

Writes the data request parameters into the from\_mail\_file. If there is a missing parameter, sends an error and exits. The script also writes the selected track numbers into an orgTrack.txt file

**Error Handling:**

Return error number to the main script.

- Error missing parameter in a file

### 3.2.2 write\_log.tcl

Writes the data request parameters into the information log file.

**Error Handling:**

- None

### 3.2.3 qa\_mail.tcl

This script updates the QA\_PRODUCT\_UPDATE database table with product QA parameters and emails CCB and SCF personnel.

**Assumptions:**

The database is not already open.

**Input parameters:**

The following QA update parameters:

- product
- release
- start date of data
- end date of data
- user
- site
- qa\_update (passed, failed, inferred\_passed)
- description

**Global parameters:**

- \$db\_user – Database user name
- \$db\_passwd – Database password for user name
- \$db\_name – Database name
- \$bin\_path – Path to bin directory where tcl code resides
- \$tmp\_dir – Path to temporary directory where email is processed
- \$input\_qa – File containing QA update parameters
- \$errorInfo – error information string

**Error Handling:** The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and will be written into the log file.

- Can't establish database connections
- Can't write to or retrieve from database
- Can't open mail message file
- Can't send message

**Subroutines called:**

- send\_mail\_mscf.pl - Refer to section 3.1.14 for details
- send\_mail\_ccb.pl - Refer to section 3.1.15 for details

### 3.3 populate\_db.tcl

This script populates the data base tables.

A list and description of the database tables are in appendix A.

**Input parameters:**

call\_cnt – counter indicating how many times routine was called  
 tmp\_dir – temporary directory for mail processing  
 mail\_file– the name of the file where the mail parameters are

**Output files:**

requestIdFile.txt that consist of the requestId that was assign by the database.  
 output.txt, errors\_out.txt are the output and the errors list text files.

**Error Handling:**

The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and will be written into the log file.

- Can't open a file
- Sql error

**Description of Script:**

- Connect to the mysql database
- Retrieve the userID from the USER table where the user name and the institution name match.
  - If there is no match, a new userID is added to the table and contact the new user for the rest of the info required in the USER table.
- If the data request is a subscription, populate the SUBSCRIPTION\_USER table.
- If the data request is a special request, populate the SPECIAL\_REQUEST\_USER table and retrieve the requestId.
- Populate the REQUEST\_PRODUCT\_SEG table:  
 The segment is determent by the selected area.  
 If the product is no a ¼ rev the segment is zero.  
 If the product is a ¼ rev and the area include more than one segment, there will be multiple records for the same product.
- Populate the REQUEST\_TRACKS and REQUEST\_CYCLES tables:
- For subscriptions:

- If the user requested the tracks and the cycles, those values will populate the tables.
- If the user didn't select cycles then the script will populate the REQUEST\_CYCLES table with -2 to indicate use all cycles
- If the user didn't select tracks then the script will call orbselect to determine the 8-day or 91-day tracks for the mission
- Populate the REQUEST\_TRACKS and REQUEST\_CYCLES tables.
- For special requests:
  - If the user requested the tracks and the cycles, those values will populate the tables.
  - If the user didn't select cycles or tracks, than the script will evaluate all the tracks or cycles in the time span:
    - Get from the rev file all the reference orbit Ids that cover the time span.
    - For each reference orbit Id, the script will read the rev file and will get a list of all the tracks and cycles in the time span.
  - Populate the REQUEST\_TRACKS and REQUEST\_CYCLES tables.
  - Read the ISIPS\_PRODUCT\_ID table and retrieve the unique PID for each combination of reference orbit ID, reference track and cycle.  
[reference track is the first track in a 14 rev granule. For example :1,15,29....]
- Populate the SPECIAL\_REQUEST\_PID table with the PID span.

**Subroutines called:**

- add\_to\_log\_file.tcl - Refer to section 4.1.1 for details
- subset.tcl
- sort\_unique.tcl
- find\_span.tcl
- zero\_padding.tcl
- zero\_out.tcl
- orbselect.f90 – Refer to section 2.3.1 for details
- convert2j2000.tcl
- find\_passID.f90
- send\_mail\_mscf.pl - Refer to section 3.1.14 for details
- send\_mail\_user.pl
- run\_create\_maps.ksh

**3.3.1 subset.tcl**

Returns subsetted list.

**Input Arguments:**

- Input list
- List of indices to subset list by

**Output Arguments:**

- Output subsetted list

### **3.3.2sort\_unique.tcl**

Sorts a list and returns indices of elements that are different from each other.

#### **Input Arguments:**

- Input list
- Flag indicating whether first index or last index of identical elements is returned

#### **Output Arguments:**

- List of indices of unique elements

### **3.3.3find\_span.tcl**

Return minimum and maximum entries of a list.

#### **Input Arguments:**

- Input list

#### **Output Arguments:**

- Minimum value
- Maximum value

### **3.3.4zero\_padding.tcl**

Adds preceding zeros to cycle and track numbers.

#### **Input Arguments:**

- Number
- Preceding zeros (i.e. 000)

#### **Output Arguments:**

- Number with preceding zeros

### **3.3.5zero\_out.tcl**

Removes preceding zeros from cycle and track numbers.

#### **Input Arguments:**

- Number

#### **Output Arguments:**

- Number without preceding zeros

### **3.3.6 convert2j2000.tcl**

This subroutine converts date to J2000 sec

#### **Input Parameters:**

- Path where datecon resides
- Date

#### **Output Parameters:**

- Date in J2000 sec

### **3.3.7 find\_passID.f90**

This program finds the pass ID in the time span and writes them to an output file

#### **Input Parameters:**

- Rev file
- Start time
- End time
- Output directory
- Error file

#### **Output File:**

- Pass ID file

#### **Error Handling:**

- Error opening file
- Error reading file

### **3.3.8 send\_mail\_user.pl**

#### **Synopsis:**

Sends email to input email address.

#### **Invocation:**

send\_mail\_user.pl <email\_address> <file\_with\_message> <subject\_line>

### **3.4 data\_select\_req.tcl**

This is the main script to process the special request data. Detail design for the script is in Section 4.

### **3.5 run\_create\_maps.ksh**

This script runs create\_maps.pl.

**Input Parameters:**

- Subscription ID

**Environmental Variables:**

```
export INPUT_DIR=[location of html code on icesat0]
export IMAGE_DIR=[location of plot images on icesat0]
export OUTPUT_DIR=[location of html code on glas-scfweb]
export OUTPUT_JS=[main directory on glas-scfweb]
export REMOTE_HOST=[SCF website hostname]
export IDL_MAIN_DIR=[location of IDL program directory]
```

### 3.5.1 create\_maps.pl

This program creates a jpeg file of the selected tracks over the selected area for each subscription.

**Input Parameters:**

- Subscription ID

**Output File:**

- JPEG track files

**Subroutines:**

- get\_area\_dates.pm
- get\_product\_seg.pm
- get\_tracks.pm
- get\_cycles.pm
- send\_mail\_local.pm
- initialize\_arrays.pm

#### 3.5.1.1 initialize\_arrays.pm

**Description:**

Initializes arrays for database information.

**Arguments:**

None

**Subroutines Called:**

None

### 3.5.1.2 `get_area_dates.pm`

**Description:**

Retrieves from the database dates and area for each subscription.

**Arguments:**

Database connection

**Subroutines Called:**

None

### 3.5.1.3 `get_product_seg.pm`

**Description:**

Retrieves from the database products and segments for each subscription.

**Arguments:**

Database connection

**Subroutines Called:**

None

### 3.5.1.4 `get_tracks.pm`

**Description:**

Retrieves from the database a list of tracks for each subscription.

**Arguments:**

Database connection

**Subroutines Called:**

None

### 3.5.1.5 `get_cycles.pm`

**Description:**

Retrieves from the database a list of cycles for each subscription.

**Arguments:**

Database connection

**Subroutines Called:**

None

### **3.5.1.6 send\_mail\_local.pm**

**Description:**

Sends message in an email to SCF personnel.

**Arguments:**

Message

Subject line

**Subroutines Called:**

None

## 4 Create Products for Special Requests

This script is invoked by readMail.ksh

First the special request script retrieve from the database tables all the data request parameters relate to the requestId.

The special request software determines which I-SIPS product granules are required to fill the request. It then reads the parts of the product files that cover the region and the time required and writes those records to new subsetted product files. The process is finished when all the associated directories to the product files are created and transferred to the submitters.

Fig 4-2 shows the files hierarchy.

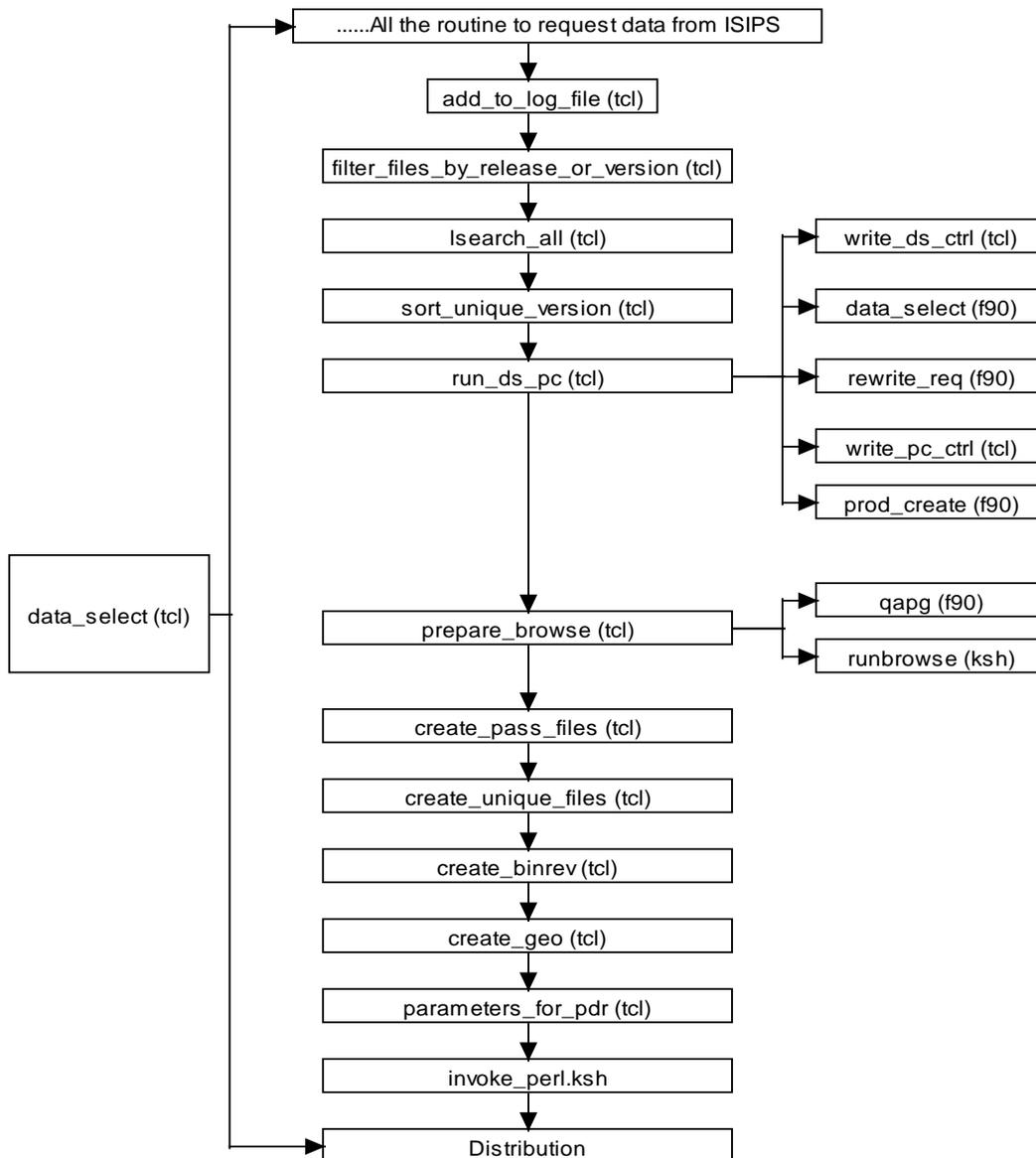


Fig 4-2

## 4.1 Main script to process special requests: data\_select\_req.tcl

### Input parameters:

requestId – the data request Id that was assign by the database.

Error file

### output files:

subsetting product files and all the associated directories to the product:

- GLAS formatted product files
- Browse product files
- Unique record index for each product file
- Pass directories for each product file
- Binrev directory
- Georeference directory
- Log file
- PDR and XFR

More details about those files are later in this section.

Also the information can be found in the *SCF Architectural Design Document*.

output\_sr.txt, errors\_sr.txt are the output and the errors list text files.

### Error Handling:

The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and is written into the log file.

- Can't open a file
- Error from an external procedure.
- No REQ files were created

### Description of Algorithm:

- Fetch from the database all the data request parameters that relate to the requestId.
- Fetch the PID span related to the requestId from the SPECIAL\_REQUEST\_PID table.
- Look at the ISIPS\_SUBSCRIPTIONS table to see if any of the selected products and segments are not stored in the mSCF.
- Make a list of all the files that are needed for the special request and are not stored in the mSCF.
- If request is quick-look, submit the request to ISIPS to get the missing files and email SCF to get approval from the CCB.
- Make a request to ISIPS to get the missing files
- Continue to complete the request with the existing files.
- For each PID:
- List all the files in mSCF data directory with the PID extension

- Create the input\_files.txt that list all the requested granules with their respected binrev files
- Create the products subsets with all their directories.
- Any time a BNA01 is created, it copies it to BNA03 and BNA04. The same with GRA01.

**Subroutines called:**

- find\_passID.f90 – Refer to section 3.3.2 for details
- get\_pid\_j2000.tcl - Refer to section 12.1.3 for details
- check\_inst\_update.tcl - Refer to section 11.1.4 for details
- add\_to\_log\_file.tcl
- filter\_files\_by\_release\_or\_version.tcl
- lsearch\_all.tcl
- sort\_unique\_version.tcl
- run\_ds\_pc.tcl
- prepare\_browse.tcl
- write\_ds\_ctrl.tcl
- data\_select.f90
- rewrite\_req.f90
- write\_pr\_ctrl.tcl
- write\_pc\_ctrl.tcl
- prod\_create.f90
- create\_pass\_files.tcl
- mkpass.f90
- create\_unique\_files.tcl
- mkunique\_index.f90
- create\_binrev.tcl
- mkbinrev.f90
- create\_geo.tcl
- mkgeo.f90
- send\_mail\_mscf.pl
- send\_mail\_ccb.pl
- read\_header\_val.f90

**4.1.1 add\_to\_log\_file.tcl**

This subroutine appends to the log file the error file that contains all the error messages

**Input Parameters:**

error\_file\_name  
tmp\_dir

**4.1.2 filter\_files\_by\_release\_or\_version.tcl**

**Input Parameters:**

files\_list: the list of files to be filtered  
release\_number:

**Output Parameters:**

filtered\_files\_list: the files list after being filtered.

**Description of Algorithm:**

- Sort the list.
- Separate the files by products.
- For each product:
- Group all the files that have the same passID.
- If release\_number=0 then:
  - Sort the group in decreasing order.
  - Take the first one and append it to the filtered files list.
- Else:
  - Take the files with the given release number

### 4.1.3 lsearch\_all.tcl

Get files that match a pattern.

Details in 5.2.7

### 4.1.4 sort\_unique\_version

**Input Parameters:**

files\_list: the list of files to be sorted

flag : flag=0 : selecting the old version

flag=1 : selecting the new version

**Output Parameters:**

filtered\_files\_list: the files list after being sorted.

**Assumption:** The version number is in the last 2 character in the file name

The subroutine sort the files list in decreasing or increasing order , depending on the flag. In each group of the same file name (accept the last 2 characters) , it selects the first one.

### 4.1.5 run\_ds\_pc.tcl

This subroutine creates the product subset.

Details in 5.2.9

**Subroutine Called:**

write\_ds\_ctrl.tcl

- catch\_errors.tcl
- data\_select.f90
- rewrite\_req.f90

- write\_pr\_ctrl.tcl
- write\_pc\_ctrl.tcl
- prod\_create.f90

#### 4.1.5.1 write\_ds\_ctrl.tcl

writes the control file to data\_select.f90  
Details in 5.2.9.1

#### 4.1.5.2 catch\_errors.tcl

Distinguishes between fatal errors and non-fatal errors.

##### **Input Arguments:**

- Message with errors written to it that is used to determine fatality of errors
- Error message upon fatal error

##### **Output Arguments:**

- None

##### **Global Variables Used:**

- errorInfo

##### **Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.

#### 4.1.6 data\_select.f90

Data\_select is the first phase of creating geographically and/or temporally sub-setted and super-setted product files. It reads a control file containing product types, latitude and longitude ranges, and start and stop times and creates REQ output files containing product file names, unique record numbers, product record numbers, and pass ID's within the geographic and temporal spans. After the REQ file is created, the program checks that there is no overlapping in time inside the file. If there is an overlapping, the program recreates the REQ file with records that not overlapping. To ensure that all the data within geographic range are obtained, a 1 degree is added to the border of the selected area.

##### **Input Parameters:**

control file name

Example of a control file:

```
=DATA_SELECT Control file SCF data selection  
DATA_PATH=/SCF/product_sets/current/L3G
```

```
REV_FILE=/SCF/ancillary_data/rev_files/total_rev_file.dat
INPUT_FILE=/SCF/tmp/dir_MSCF_12760/input_files.txt
ANC70_FILE=/SCF/ancillary_data/scf70_000_00_00.dat
OUTPUT_PATH=/SCF/tmp/dir_MSCF_12760/
OUTPUT_STRING=r9324_00_L3G.P0486_01_00
DATE_STRING=06110522
START_TIME=214574400
END_TIME=218203200
NLAT=90
SLAT=-90
ELON=180
WLON=-180
PRODUCT=01
PRODUCT=05
MSCF_FLAG=1
=END of control file
```

where:

- data\_path is the directory where the product files are.
- rev\_file is the name of the rev file.
- input\_file is the name of the file containing the list of binrev file names
- anc70\_file is the name of the file containing error numbers and constants.
- output\_path is the output directory for the REQ file.
- output\_string is the string in the output file name containing request number, laser period, and PID number.
- date\_string is the beginning time of the first file in the 14 rev product set received from the I-SIPS. This information was given from the file header. If the keyword from the header could not be found, there will be no date\_str keyword in the control file and the the date is set to the requested beginning time.
- start\_time, end\_time is the requested time span
- nlat, slat, elon, wlon is the selected region
- product is the requested product number. There may be multiple product keywords, one for each requested product.
- mscf\_flag is 1 if the data are on the main SCF meaning that all products are assumed to be present and BNA05 files are listed in the input\_file. Mscf\_flag is 0 if the data are not on the main SCF so the BN files listed in the input\_file are those of the requested products.

Example of input\_files.txt:

```
BNA05_531_2117_002_0099_1_01_0001.P1306
BNA05_531_2117_002_0099_2_01_0001.P1306
BNA05_531_2117_002_0099_3_01_0001.P1306
BNA05_531_2117_002_0099_4_01_0001.P1306
BNA05_531_2117_002_0100_1_01_0001.P1306
BNA05_531_2117_002_0100_2_01_0001.P1306
BNA05_531_2117_002_0100_3_01_0001.P1306
```

BNA05\_531\_2117\_002\_0100\_4\_01\_0001.P1306

...

**Input files:**

The control file described above.

The input\_file described above.

**Output files:**

REQ files for each GLAS product. The format for the REQ file is in Appendix C.

**To run:**

data\_select <control file>

Notes on running:

- the library path has to be set before running:

i.e. export SHLIB\_PATH=/SCF/lib/ops

- if data\_select is run more than once, the following error will occur:

ERROR : -10006, 3, read\_rev\_file, Error Opening File for Output:  
/SCF/tmp/dir\_25225/passid\_time.bin

Just remove the passid\_time.bin file in between runs.

**Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list
- error reading filenames file
- no BN files to process
- no UR files to process

Returns an exit(3) to the main script

**Description of Algorithm:**

Refer to Appendix B.1: Flowchart for data\_select.f90

**Calling Subroutines:**

- const\_scf\_mod.f90
- anc70\_scf\_mod.f90
- ANC70\_mod.f90
- filesize\_mod.f90
- open\_bin\_file\_mod.f90
- fbins\_mod.f90
- common\_files\_mod.f90
- read\_ds\_ctrl\_mod.f90
- read\_rev\_file\_mod.f90
- create\_geobins\_mod.f90

- read\_filenames\_mod.f90
- read\_geobins\_mod.f90
- read\_binrev\_mod.f90
- sort\_file\_mod.f90
- read\_unique\_mod.f90
- filter\_req\_mod.f90
- find\_uix\_delta\_mod.f90

#### **4.1.6.1 const\_scf\_mod.f90**

Defines constants in anc70 file.

#### **4.1.6.2 Anc70\_scf\_mod.f90**

Reads constants in anc70 file.

#### **4.1.6.3 ANC70\_mod.f90**

Reads anc70 file.

#### **4.1.6.4 filesize\_mod.f90**

Returns the size of a file in bytes

#### **4.1.6.5 open\_bin\_file\_mod.f90**

Contains routines to open direct access binary files with and without headers and return the number of lines in file.

#### **4.1.6.6 fbins\_mod.f90**

Determines bin configuration.

#### **4.1.6.7 common\_files\_mod.f90**

Contains names, lun's, record sizes, and number lines variables of files used in data\_select.

#### **4.1.6.8 read\_ds\_ctrl\_mod.f90**

Reads control file and anc70 constants/errors file.

#### **4.1.6.9 read\_rev\_file\_mod.f90**

Reads rev file and outputs file with pass ID's filtered by requested time span

#### **4.1.6.10 create\_geobins\_mod.f90**

Creates georeference bins from requested lat/lon range and outputs bin numbers to a file  
If the requested area is the whole world, data\_select.f90 doesn't use the routine.

#### **4.1.6.11 read\_filenames\_mod.f90**

Reads file containing input BN and UR file names

#### **4.1.6.12 read\_geobins\_mod.f90**

Reads georeference file and file with georeference bin numbers and outputs file with bin numbers

#### **4.1.6.13 read\_binrev\_mod.f90**

Reads binrev file, file with bin numbers based on requested lat/lon, file with pass ID's based on requested time span and outputs file with sorted unique record numbers and pass ID's filtered by time and location

#### **4.1.6.14 sort\_file\_mod.f90**

Sorts file with sorted unique record numbers and pass ID's filtered by time and location

#### **4.1.6.15 read\_unique\_mod.f90**

Reads the unique record file and file with sorted unique record numbers and pass ID's filtered by time and location and outputs the REQ file containing product file names, unique record numbers, product record numbers, and pass ID's.

#### **4.1.6.16 filter\_req\_mod.f90**

Filters REQ files for duplicate and overlapping unique record numbers.

#### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error opening input file
- error reading input file
- error writing input file
- no product files listed for product type

- index of array is greater than dimension allowed
- Returns an exit(3) to the main script

**Description of algorithm:**

- Open input file
  - If no product files are listed in input file, skip
  - Read header records into array
- Determine product number from REQ file name
- Determine number of data lines in file
- Allocate arrays for sorted array indices and data
- Read REQ file
  - Read the mode if the product is GLA01
  - Add beginning unique rec num to array for sorting
- Close REQ file after reading
- Sort array by beginning unique rec num (output is array of sorted indices)
- Write data into sorted array 2
- If file just has one line, just copy over
- Filter sorted array 2 into array 3
  - If index for array 2 is last line, assign last record and exit
  - If adjacent records are for the same GLA file, then start times are already sorted, so just copy info over and exit
  - Times do not overlap - just copy over
  - Overlapping records:
    - Second record is within first - use first record
      - Assign next record in array2 to value of current record
    - Second record overlaps first - resize first record
      - GLA01 has 1 sec unique records composed of 3 or 6 product records
      - GLA08-11 have 4 sec records
      - Other products have 1 sec records
      - First rec ends at start of overlapping second rec
- Open REQ file again for writing
- Write headers into new REQ
- Write filtered array 3 to REQ file
- Close REQ file after writing
- Deallocate arrays

**4.1.6.17 find\_uix\_delta\_mod.f90**

Determines span between unique index records by reading product header for VersionID value.

**Input Parameters:**

in\_file: product file name

**Output Parameters:**

i\_release: data release

i\_ur\_delta: span between unique index records based on data release

**Error Handling:**

Calls GLAS\_error for the following conditions:

- error opening input file
- error reading input file

**Description of algorithm:**

- Open input file (get number header lines and record length)
- Read input file (get header value for header name)
- If no header, assume it's a pre-release 31 file
  - Get release from file header
- Determine UR delta from product release
- Close input file

**4.1.6.18 rewrite\_req.f90**

**input parameters:**

- input\_file: REQ file to be modified. The format for the REQ file is in Appendix C.
- track\_file: a text file that contains a list of the tracks the user selected

**Format:**

n8: number of 8 days repeat tracks  
t1  
t2  
.  
.  
n91: number of 8 days repeat tracks  
t1  
t2  
.  
.

(When the user select no processing , the track will get the value -1.

When the user select processing without selecting specific tracks, the track will get the value 0)

- out\_file: the REQ file after being modified

**Description of algorithm:**

Read the track file

- If the first track for the 8 days repeat is zero, and first track for the 91 days repeat is zero, copy the input file to the out\_file and return.
- For each record in the input file:
  - Read the the passID
  - retrieve the tracks number and the reference ID.

- If the reference ID is 3 or the track equal to one of the tracks from the track file ,write the record into the out\_file.

#### **4.1.7 write\_pr\_ctrl.tcl**

Write the control file for parse\_req.f90  
Detail in 5.2.9.2

#### **4.1.8 write\_pc\_ctrl.tcl**

Write the control file for prod\_create.f90  
Detail in 5.2.9.3

#### **4.1.9 parse\_req.f90**

Reads REQ files and determines if an REQ will produce a GL file > 2 GB. If so, then the REQ file is divided into more than one file.

#### **Input Parameters:**

control file name

Example of a control file:

```
=PARSE_REQ Control file SCF data selection  
DATA_PATH=/SCF/tmp/dir_UTCSR_19557/  
INPUT_FILE=/SCF/tmp/dir_UTCSR_19557/REQ02_00010112_r0578.P0014_01_00  
ANC70_FILE=/SCF/src/common/2001_11_01/scf70_000_00_00.dat
```

where:

- data\_path is the directory where the product files are.
- input\_file is the REQ file path and name
- anc70\_file is where the errors list and constants are.

#### **input files:**

- The control file that was described above.
- REQ files. The format for the REQ file is in Appendix C.

#### **output files:**

REQ files for each GLAS product. The format for the REQ file is in Appendix C.

If the resultant product file is determined to be less than 2 GB, then the input REQ is output. If it is determined to be greater than 2 GB then the input REQ is parsed into multiple REQ files such that each one will not produce a product file greater than 2 GB.

#### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list

#### **Description of Algorithm:**

- Boot GLAS\_Error

- Get the control file name
- Read control file for input/output file names and read anc70 file
- Add ending / to data\_path if necessary
- For each input file (corresponding to product type)
- Read REQ file and write to separate files if one file will exceed 2 GB
- End with error severity

**Subroutines Called:**

- const\_scf\_mod.f90

The above subroutines have been documented previously

- read\_pr\_ctrl\_mod
- read\_req\_mod

**4.1.9.1 read\_pr\_ctrl\_mod.f90**

Reads control file.

**4.1.9.2 read\_req\_mod.f90**

Reads REQ file and determines if it needs to be parsed into separate files so that a resultant GL file does not exceed 2 GB

**Subroutines Called:**

- const\_scf\_mod.f90
- open\_bin\_file\_mod.f90
- common\_files\_mod.f90

The above subroutines have been documented previously

- prod\_common\_mod.f90 – Refer to section 4.1.5.7.1
- write\_req\_mod

**4.1.9.3 write\_req\_mod.f90**

Writes new REQ files from original REQ

**4.1.10 prod\_create.f90**

Runs code to create product subsets based upon input file containing product names, lat/lon range, and time span.

**input parameters:**

control file name.

Example of a control file:

```
=PROD_CREATE Control file SCF data selection
DATA_PATH=/SCF/tmp/dir_UTCSR_19557/
INPUT_FILE=/SCF/tmp/dir_UTCSR_19557/REQ02_00010112_r0578.P0014_01_00
ANC70_FILE=/SCF/src/common/2001_11_01/scf70_000_00_00.dat
```

where:

- data\_path is the directory where the product files are.
- input\_file is the REQ file path and name
- anc70\_file is where the errors list and constants are.

#### **input files:**

- control file.
- REQ files. The format for the REQ file is in Appendix C.

#### **output files:**

One output product is created per product type. The file has the GLAS format.

#### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list
- error reading a header
- error in reading the control file
- error reading product file

Returns an exit(3) to the main script

#### **Description of Algorithm:**

Refer to Appendix B.2: Flowchart for prod\_create.f90

#### **Subroutines Called:**

- const\_scf\_mod.f90
- anc70\_scf\_mod.f90
- ANC70\_mod.f90
- filesize\_mod.f90
- open\_bin\_file\_mod.f90

The above subroutines have been documented previously

- prod\_common\_mod.f90
- prod\_reader\_mod.f90
- prod\_writer\_mod.f90
- read\_pc\_ctrl\_mod.f90
- read\_prod\_recs\_mod.f90

#### **4.1.10.1 prod\_common\_mod.f90**

#### **Subroutines Included:**

- scale\_init: Initializes product scales

- prod\_init\_in: Initializes input file structure and opens input file
- prod\_init\_out: Initializes output file structure and opens output file.

#### **4.1.10.2 prod\_reader\_mod.f90**

Reads a record from a product file

#### **4.1.10.3 prod\_writer\_mod.f90**

Writes a record into a product file.

For all products except GLA01, GLA02, GLA03, GLA04- only writes output if record is within input time span and lat/lon range.

#### **4.1.10.4 read\_pc\_ctrl\_mod.f90**

Reads control file.

#### **4.1.10.5 read\_prod\_recs\_mod.f90**

Reads REQ files one record at a time. The format for the REQ file is in Appendix C.

### **4.1.11 prepare\_browse.tcl**

Creates the browse products.

Prepares the control files and runs the program to create the browse product.

#### **Input parameters:**

Bin\_path, tmp\_dir, gl\_list

Where:

Gl\_list is a list of all the products created by data\_select.f90

#### **Output parameters:**

qaplist: list of created QAP files

#### **Program Calls:**

- qapg.f90
- runbrowse.ksh

#### **4.1.11.1 qapg.f90**

Generate QAP file from product.

Code in /SCF/src/qapg/

**Input Parameters:**

Control\_file

**Consist of:**

=PROD\_VER Control file for product verification

INPUT\_FILE= anc07\_001\_01\_0000.dat

INPUT\_FILE= anc07\_001\_01\_0006.dat

INPUT\_FILE= [product file]

OUTPUT\_FILE=[product\_file].QAP

= End of control file

**Error Handling:**

Calls Glas\_error for the following conditions:

- error getting control file name from argument list

#### 4.1.11.2 Runbrowse.ksh

A script that opens idl and runs the routines to create the browse products.

**Program calls:**

Qabrowse.pro

#### 4.1.11.2.1 Qabrowse.pro

Generates the browse products

**Input Parameters:**

Control\_file

**Consists of:**

=QABROWSE

VERSION=[product version]

INPUT\_FILE=[qap file name]

OUTPUT\_DIRECTORY=

OUTPUT\_FORMAT=png

#### 4.1.12 create\_pass\_files.tcl

Create the pass directories.

Details in 5.2.3

**Program Calls:**

mkpass.f90

#### 4.1.13 mkpass.f90

Creates a directory correlating the passID and the GLAS product file unique index record.

One data record is written every time a passID is changed in the file.

### **Input parameters:**

control file name.

Here is an example of a control file:

```
=MKPASS Control file SCF data selection
INPUT_FILE=GLA05_002_1101_001_0028_2_01_01.P0001
INPUT_FILE=GLA05_002_1101_001_0028_3_01_01.P0001
INPUT_FILE=GLA06_002_1101_001_0028_2_01_01.P0001
TIME_PASS_FILE=/SCF/ancillary_data/rev_files/total_rev_file.dat
OUTPUT_PATH=
ERR_FILE=/SCF/src/common/V2_prelim/scf70_000_00_00.dat
=END of control file
```

where :

- input file is the GLAS product file
- time\_pass\_file is the rev file that has the time each pass starts
- output\_path is the output directory
- err\_file is the SCF Constants and Error File

### **input file:**

control file

### **output file:**

pass directory . (sometimes is called pass table).

The format and description of the pass directory is in the *SCF Architectural Design Document*. (5.3.2.1)

### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list
- error reading filenames file
- error opening output file
- error writing output file
- the time/pass table does not cover the input time

Returns an exit(3) to the main script.

### **Description of Algorithm:**

- Boot GLAS\_Error
- Get the control file name
- Read control file for input/output file names
- Open and read the anc70 (Error/Status) file for GLAS\_Error usage
- Read the time in the first and the last record of the product file
- Read rev files in the time covers by the product file.
- Set the time-passID table

- Initialize the pass:
  - Read the latitude in all the records in the file.
  - Loop over the GLAS file records
  - Read the time
  - Call the find\_pass routine to find the passID for the time
  - If the time is less than the first time in the time/pass table, read the next record in GLAS file
    - if a passID is found:
      - Check if you are in the forth segment.
        - If no, then your init passID is the one that was found
        - If yes, then your init passID is the pass before the one that was found. (if the passID that was found is the first one in the table then read the next record in GLAS file and do not exit the loop
      - exit the loop.
    - If no passID is found then read the next input file
    - if no passID is found for all the input files, exit the program
- After the initialization ends successfully:
  - Read the records in all the input files.
  - For each record:
    - Call the find\_pass routine to find the passID for the time
    - Check that if we are at the forth segment, the passID will be the one before.
    - If no pass is found:
      - write to the pass file the passID and the unique record indices that refers to the beginning and the ending of the last pass
      - Exit the program
    - If the pass is different from the last one:
      - Write to the pass file the passID and the unique record indices that refer to the beginning and the ending of the last pass
  - After the last record is read:
    - Write to the pass file the passID and the unique record indices that refer to the beginning and the ending of the last pass

**Subroutines Called:**

- const\_scf\_mod.f90
- anc70\_scf\_mod.f90
- filesize\_mod.f90\*
- ANC70\_mod.f90
- prod\_common\_mod.f90
- prod\_reader\_mod.f90
- open\_bin\_file\_mod.f90

The above subroutines have been documented previously.

- read\_pass\_control\_mod.f90
- read\_glas\_record\_mod.f90
- qsorti\_mod.f90
- find\_pass\_mod.f90

#### **4.1.13.1 read\_pass\_control\_mod.f90**

Reads the control file

#### **4.1.13.2 read\_glas\_record\_mod.f90**

Reads a record from a GLAS product file and passes to the main time, lon ,lat , unique record index and record type.

It gives the first valid value in the 40 Hz array rather than just the first value of the array.

#### **4.1.13.3 qsorti\_mod.f90**

Sorts an array.

#### **Input Parameters:**

n, a, ord

where:

- N: length of the array
- A: the array to be sorted
- Ord: a list of indices of the ordered array.

#### **4.1.13.4 find\_pass\_mod.f90**

Finds in the time/pass table a corresponding pass to an input time.

#### **4.1.14 create\_unique\_files.tcl**

Create the control file for mkunique\_index.f90

Details at 5.2.5

#### **Program Calls:**

mkunique\_index.f90

#### **4.1.15 mkunique\_index.f90**

Creates a directory that allows one to calculate the data records within a file that correspond to specific unique record index numbers.

One data record is written every time there is a break in unique record index in the file or for GLA01, a record also has to be written every time the waveform record mode changes.

#### **Input parameters:**

control file name

Here is an example of a control file:

```
=GLAS_INDEX Control file SCF data selection
INPUT_FILE=GLA08_001_1101_001_0015_0_01_01.P0003
INPUT_FILE=GLA09_001_1101_001_0029_0_01_01.P0003
INPUT_FILE=GLA09_001_1101_001_0043_0_01_01.P0003
OUTPUT_FILE_PATH=
ERR_FILE=/SCF/src/common/V2_prelim/scf70_000_00_00.dat
=END of control file
```

where:

- input file is the GLAS product file name
- output\_file\_path is the output directory
- err\_file is the SCF constants and errors file.

### **input file:**

control file

### **output file:**

unique record index file.

The format and description of the unique record index file is in the *SCF Architectural Design Document. (5.3.2.1)*

### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list
- error reading filenames file
- error opening output file
- error writing output file
- Unique record is out of order.

### **Description of Algorithm:**

- Boot GLAS\_Error
- Get the control file name
- Read control file for input/output file names
- Open and read the anc70 (Error/Status) file for GLAS\_Error usage
- Read the records in all the input files.
- For each record and each file:
- Get the unique record index and the UTC time
- if the unique record index is different from the last one, and bigger than the last one:
  - write to the output file the unique record indices that refer to the beginning and the ending of the last unique record and the file record number and the UTC time that refers to the beginning of that unique record.
- After the last record is read:
  - write to the output file the unique record indices that refer to the beginning and the ending of the last unique record and the file record number and the UTC time that refers to the beginning of that unique record.

**Subroutines Called:**

- anc70\_scf\_mod.f90
- filesize\_mod.f90
- ANC70\_mod.f90
- prod\_common\_mod.f90
- prod\_reader\_mod.f90
- uniq\_glas\_record\_mod.f90
- open\_bin\_file\_mod.f90
- qsorti\_mod.f90
- const\_scf\_mod.f90

The above subroutines have been documented previously.

- find\_uix\_delta\_mod.f90 – Refer to section 4.1.6.17
- read\_glas\_unique\_control\_mod.f90

#### **4.1.15.1 read\_glas\_unique\_control\_mod.f90**

Reads the control file.

#### **4.1.16 create\_binrev.tcl**

Create the control file to run mkbinrev.f90

Details at 5.3.1

**Program Calls:**

mkbinrev.f90

#### **4.1.17 mkbinrev.f90**

Creates a directory correlating the bin number and the passID to the unique record indices in the GLAS product files. The program reads from a file the UTC time, longitude, latitude and unique record number and calculates the bin index from the longitude and latitude, and the pass from a rev table and writes to the output file the old bin, the pass that belongs to the record where the old bin was calculated for the first time, and the beginning and ending unique record for the old bin number.

No binrev file is created for GLA03 and GLA04. These products have no longitudes and latitudes.

**Input Parameters:**

control file name.

Example of the control file:

```
=MKBINREV Control file SCF data selection
INPUT_FILE=GLA05_002_1101_001_0028_2_01_01.P0001
INPUT_FILE=GLA05_002_1101_001_0028_3_01_01.P0001
INPUT_FILE=GLA05_002_1101_001_0028_4_01_01.P0001
```

TIME\_PASS\_FILE=/SCF/ancillary\_data/rev\_files/total\_rev\_file.dat  
OUTPUT\_FILE=BNA\_1101\_001\_0015.P0001\_00  
ERR\_FILE=/SCF/src/common/V2\_prelim/scf70\_000\_00\_00.dat  
=END of control file

where:

- input file is the GLAS product file
- time\_pass\_file is the rev file that has the time each pass starts
- output\_file is the binrev file name
- err\_file is the SCF Constants and Error File

**input file:**

control file

**output files:**

binrev directory (sometimes is called bin table).

The format and description of the binrev directory is in the *SCF Architectural Design Document*.  
(5.3.2.1)

**Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list
- error reading the control file
- error reading GLAS product file
- error opening output file
- error writing output file
- the time/pass table does not cover the input time

Returns an exit(3) to the main script

**Description of Algorithm:**

- Boot GLAS\_Error
- Get the control file name
- Read control file.
- Open and read the anc70 (Error/Status) file for GLAS\_Error usage
- Read rev file
- Sort the rev records by time
- Initialize the bin and passID:
  - Loop over the GLAIn file records
    - Read the time, longitude and latitude
    - Call get1bin to calculate the bin index
    - Call the find\_pass routine to find the passID for the time
    - If the time is less than the first time in the time/pass table, read the next record in GLAIn file; exit the loop if a pass is found.
    - If no pass is found then read the next input file; if no pass is found for all the input files, exit the program.
- After the initialization ends successfully:

- Read the records in all the input files.
- For each record:
  - Call get1bin to calculate the bin index
  - Call the find\_pass routine to find the passID for the time and the end time for this passID
  - If no pass is found:
    - Write to the bin/rev file the bin, passID and the unique record indices that refers to the beginning and the end of the last bin
    - Exit the program
  - if the bin is different from the last one or time is greater than the end time of the passID:
    - Write to the bin/rev file the bin, passID, and the unique record indices that refer to the beginning and ending of the last bin
- After the last record is read (of the last input file),
  - Write to the bin/rev file the bin, passID and the unique record indices that refer to the beginning and the endong of the last bin

**Subroutines Called:**

- const\_scf\_mod.f90
- anc70\_scf\_mod.f90
- filesize\_mod.f90
- ANC70\_mod.f90
- prod\_common\_mod.f90
- prod\_reader\_mod.f90
- fbins\_mod.f90
- open\_bin\_file\_mod.f90
- read\_gls\_record\_mod.f90
- qsorti\_mod.f90
- find\_pass\_mod.f90

The above subroutines have been documented previously.

- read\_binrev\_control\_mod.f90

**4.1.17.1 read\_binrev\_control\_mod.f90**

Read the control file.

**4.1.18 create\_geo.tcl**

Create the control file to run mkgeo.f90

Details at 5.3.3

**Program Calls:**

mkgeo.f90

**4.1.19 mkgeo.f90**

The program reads the bin/rev file and sorts the file by bins. Then it rewrites the bin/rev file. The program reads the file again and if a bin has more than one row in the file than it sorts the rows of the same bin by beginning unique record, and rewrites the bin/rev file.

The new bin/rev file is now sorted by bin and by unique records in each bin.

The program reads the bin/rev again and writes to a geo file the bin and the beginning and the ending record of each bin as it appears in the bin/rev file.

### **Input Parameters:**

control file name.

Here is an example of a control file:

```
=MKGEO Control file SCF data selection
```

```
INPUT_FILE=BNA_1101_001_0029.P0001_00
```

```
OUTPUT_PATH=
```

```
ERR_FILE=/SCF/src/common/V2_prelim/scf70_000_00_00.dat
```

```
=END of control file
```

where:

- input file is the binrev file name
- output path is the output directory
- err\_file is the SCF constants and errors file.

### **input file:**

control file

### **ouput file:**

georeference directory. (sometimes is called georeference table).

The format and description of the georeference directory is in the *SCF Architectural Design Document*.(5.3.2.1).

### **Error Handling:**

Calls GLAS\_error for the following conditions:

- error getting control file name from argument list
- error opening input file
- error reading bin/rev file
- error opening output file
- error writing to the output file
- array size is bigger than the size declared

Returns an exit(3) to the main script

### **Description of Algorithm:**

- Boot GLAS\_Error
- Get the control file name
- Read control file for input/output file names

- Open and read the anc70 (Error/Status) file for GLAS\_Error usage
- Open the outfile and write the headers
- Open the bin/rev file and read it
- Sorting the records by bin index
- Rewrite the bin/rev file with the records sorted
- Reading the bin/rev file again
- When there are identical bins, sort them by unique record index
- Rewrite the bin/rev file with the unique record indices sorted
- Write into the output file the bin index and the begin and end of the bin/rev record number of that bin
- Close the file units

**Subroutines Called:**

- const\_scf\_mod.f90
- anc70\_scf\_mod.f90
- filesize\_mod.f90
- ANC70\_mod.f90
- prod\_common\_mod.f90
- prod\_reader\_mod.f90
- open\_bin\_file\_mod.f90
- qsorti\_mod.f

The above subroutines have been documented previously.

- read\_geo\_control\_mod.f90

**4.1.19.1 read\_geo\_control\_mod.f90**

Reads the control file

**4.1.20 parameters\_for\_pdr.tcl**

See 6.0

**4.1.21 invoke\_perl.ksh**

See 6.0

**4.1.22 compare\_lists.tcl**

Compares multiple lists to see if they each have the corresponding parameter for the same index (agree).

Returns 1 if they agree

Returns 0 if they do not agree or if any parameters are not found

Returns -1 upon error

**Input Arguments:**

- Any number of lists and parameters
- Argument list has to be of the form:  
list1 parameter1 list2 parameter2...
- It doesn't make sense to run this routine with less than two lists to compare

- Each list must have a parameter after it even if it is a duplicate

examples:

```
{GLA01 GLA02} GLA02 {2332 2334} 2334
```

Do we have track 2334 for GLA02? – yes

```
{GLA01 GLA02} GLA01 {2332 2334} 2334
```

Do we have track 2334 for GLA01? - no

**Output Arguments:**

- None

**Global Variables Used:**

- errorInfo
- bin\_path

**Output Files:**

- None

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error running subroutine

**Routines Called:**

- lsearch\_all\_idx.tcl

### 4.1.23 lsearch\_all\_idx.tcl

Returns list of all list indices that match lsearch pattern.

**Input Arguments:**

- Input list
- Pattern

**Output Arguments:**

- Output list

**Global Variables Used:**

- None

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

**4.1.24 send\_mail\_mscf.pl****Synopsis:**

Sends email to mSCF personnel.

**Invocation:**

send\_mail\_mscf.pl <file\_with\_message> <subject\_line>

**4.1.25 send\_mail\_ccb.pl****Synopsis:**

Sends email to CCB personnel.

**Invocation:**

send\_mail\_ccb.pl <file\_with\_message> <subject\_line>

**4.1.26 read\_header\_val.f90**

Reads input file and returns requested header value.

**Input parameters:**

- in\_file
- header

**Error Handling:**

Calls Glas\_error for the following conditions:

- error opening file
- error reading file

**Description of Algorithm:**

- Open and read the anc70 (Error/constants) file for GLAS\_Error usage
- Open input file (get number header lines and record length)
- Read input file (get header value for header name)
- Close input file

**Calling Subroutines:**

- open\_bin\_head – Refer to open\_bin\_file\_mod.f90 in section 4.1.6.5 for details

- get\_header\_val\_mod.f90

#### 4.1.26.1 get\_header\_val\_mod.f90

Returns value of requested header in input file.

##### **Input parameters:**

- file
- i\_lun
- i\_rec\_len
- i\_num\_head
- header

##### **Output parameters**

- header\_value
- i\_err\_sev

##### **Error Handling:**

Calls Glas\_error for the following conditions:

- error reading file

##### **Description of Algorithm:**

- Read up to i\_num\_head headers
- Parse the header line by the delimiter ";"
- May have up to gi\_max\_num\_seg header statements on one line
- Parse header into keyword=value
- Assign remaining header line to temp line

##### **Calling Subroutines:**

- parse\_keyval – In GSAS common libraries
- compare\_kval - In GSAS common libraries

## 5 Processing Data

### 5.1 Invocation

Processing and storing data is invoked with the script /SCF/bin/ops/run\_process\_data.ksh. This script defines environment variables needed to run the data processing software and creates an error file that is sent to SCF personnel via email upon error.

### 5.2 Environment Definitions

```
export ISIPS_PATH=[location of I-SIPS files before processing]
export ICESAT_PRODUCT_SET=[location of mSCF files after processing]
export ICESATVIS_BIN=[location of the executable and script files]
export ICESATVIS_DATA =[location of the processed_pass files ]
export DELETED_DATA_PATH=[path for deleted files]
export ACCTEST_DIR=[name of acctest directory]
export REV_FILE=[rev file]
export PROC_8_PASS=[name of process_8_pass.txt file]
export PROC_91_PASS=[name of process_91_pass.txt file]
export SCF_ERROR_FILE=[location of the error/constants file]
export TMP_DIRECTORY=[name of the temporary directory]
export SRC_DIR=[path to the Perl scripts]
export DB_NAME=[name of mysql database]
export DB_USER=[username for mysql database]
export DB_PASSWD=[password for mysql database]
export PROC_FILE=[name of processing file to check for]
export DEL_FILE=[name of file containing deleted file names]
export PROD_REL_FILE=[name of file containing products and releases]
export CURRENT_REFID=[current reference orbit to be processed first]
export PLOT_FILE=[name of file containing plot information]
export DUP_FLAG=[flag indicating if check for duplicate files should be run (0=no,1=yes)]
export PLOT_FLAG=[flag indicating if plot file should be written (0=no,1=yes)]
```

### 5.3 Main script to process data: process\_data.tcl

#### Synopsis:

As each product set is received from the I-SIPS, they are each renamed with a PID and database management files are created including binrev directories, georeference directories, pass directories, and unique index files. They are then stored at the mSCF. Entries are created in the CREATION database table with columns set to run subscriptions and energy analysis programs.

#### Assumptions:

- Files downloaded from I-SIPS are placed in tmp directory defined by environmental variable ISIPS\_PATH along with an FN file and FN XFR. For a description of the FN file, refer to the *SCF Interface Software Detailed Design Document*.

- There is only one PID per distribution ID
- There is only one release per distribution ID
- Only GLA product files are received from the I-SIPS.
- One BN, GR, PS, and UR file will be created per product.

### **Error Handling:**

The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and is written into the log file.

- Can't open or connect to the database
- Error from an external procedure.
- Error from a subroutine.

### **Description of Algorithm:**

Refer to Appendix B.3: Flowchart for subscription.tcl

### **Output Files:**

- /SCF/tmp/plot\_info.txt.

### **Subroutines called:**

- get\_file\_j2000.tcl - Refer to section 12.1.2 for details
- check\_inst\_update.tcl - Refer to section 11.1.4 for details
- read\_header\_val.f90 - Refer to section 4.1.26.1 for details
- create\_binrev.tcl
- create\_geo.tcl
- create\_pass\_files.tcl
- create\_unique\_files.tcl
- find\_track.tcl
- read\_fn\_file.tcl
- modify\_processed\_tracks.tcl
- scp\_rev\_file.ksh
- check\_rev\_time.tcl
- check\_pid.tcl
- time\_sort.tcl
- update\_prod\_rel\_file.tcl

#### **5.3.1 create\_binrev.tcl**

Creates the control file and runs mkbinrev, which creates the binrev directory for the input list of product files.

### **Input Arguments:**

- Output directory
- Control file name
- Product file list
- Flag indicating if product names are "mscf" convention or "rscf" convention
- Reference orbit number

**Output Arguments:**

- BN file name

**Global Variables Used:**

- errorInfo
- bin\_path
- scf\_error\_file
- rev\_file

**Output Files:**

BNA or BNL file

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error from an external procedure.
- Error from a subroutine.
- Error opening control file
- Error if name\_flag indicates neither mscf or rscf naming convention

**Routines Called:**

- find\_track.tcl
- sort\_pass.tcl
- mkbinrev.f90 – Refer to section 4.2.10 for more information

### 5.3.2 create\_geo.tcl

Creates the control file and runs mkgeo, which creates the georeference directory for the input binrev file.

**Input Arguments:**

- Output directory
- Control file name
- BN file name
- Flag indicating if products are “mscf” or “rscf”: if “mscf” then BNA03-4/GRA03-4 are copied from GLA02; if “rscf” then they are copied from GLA01

**Output Arguments:**

- None

**Global Variables Used:**

- errorInfo
- bin\_path

- scf\_error\_file

**Output Files:**

GRA or GRL file

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error from an external procedure.
- Error opening control file

**Routines Called:**

- mkgeo.f90 - Refer to section 4.2.11 for more information

### 5.3.3 create\_pass\_files.tcl

Creates the control file and runs mkpass, which creates the pass directories for the input product files.

**Input Arguments:**

- Output directory
- Control file name
- Product file list

**Output Arguments:**

- None

**Global Variables Used:**

- errorInfo
- bin\_path
- scf\_error\_file
- rev\_file

**Output Files:**

PS file(s)

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error from an external procedure.
- Error opening control file

**Routines Called:**

- mkpass.f90 - Refer to section 4.2.13 for more information

### **5.3.4 create\_unique\_files.tcl**

Creates the control file and runs mkunique\_index, which creates the unique index files for the input product files.

#### **Input Arguments:**

- Output directory
- Control file name
- Product file list

#### **Output Arguments:**

- None

#### **Global Variables Used:**

- errorInfo
- bin\_path
- scf\_error\_file

#### **Output Files:**

UR file(s)

#### **Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error from an external procedure.
- Error opening control file

#### **Routines Called:**

- mkunique\_index.f90 - Refer to section 4.2.12 for more information

### **5.3.5 find\_track.tcl**

Determines the integer track from a 4 character track string

#### **Input Arguments:**

- String track (4 characters containing preceding zeros)

#### **Output Arguments:**

- Integer track

#### **Global Variables Used:**

- None

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

### 5.3.6 read\_fn\_file.tcl

Reads the FN file. For a description of the FN file, refer to the *SCF Interface Software Detailed Design Document*.

**Input Arguments:**

- FN file name

**Output Arguments:**

- List of file names from FN file
- Distribution ID
- Quick-look indicator (0/1)

**Global Variables Used:**

- errorInfo

**Output Files:**

None

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error opening the FN file

**Routines Called:**

- Read\_keyword.tcl

### 2.3.5.17 read\_keyword.tcl

This routine reads a line and separates the value from the keyword. If the line follows the standard GLAS header format it returns 0. If the line does not follow this format it returns -1.

**Assumptions:**

Assumes that the line follows standard GLAS header format: keyword = value;  
Whitespace doesn't matter.

**Input Arguments:**

- line (string)

**Output Arguments:**

- keyword
- keyvalue

**Global Variables Used:**

- none

**Output Files:**

- none

**Error Handling:**

- none

**Routines Called:**

- none

### 5.3.7 modify\_processed\_tracks.tcl

Update the processed\_pass file if a new track has been processed.

**Input Arguments:**

- List of GLA01 files
- 8 or 91 day processed pass file

**Output Arguments:**

- scp flag (0/1=no/yes)

**Global Variables Used:**

- bin\_path
- mscf\_path
- anc\_data\_path

**Output Files:**

- Creates or appends to 8 or 91 day processed pass file in mscf\_path
- Appends to 8 or 91 day processed pass file in anc\_data\_path

**Error Handling:**

- Returns the error number to the calling script

**Routines Called:**

- None

### 5.3.8 scp\_rev\_file.ksh

Scp a file to all rSCF's.

**Assumptions:**

The file is copied to same directory on the remote site that it is taken from on the mSCF

**Input Arguments:**

- Directory to copy from and to
- File name

**Output Arguments:**

- None

**Global Variables Used:**

- None

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

### 5.3.9 check\_rev\_time.tcl

This routine checks time of file against last rev file time. It also returns the laser time period and laser refID.

If file time  $\leq$  rev time return 0.

If file time  $>$  rev time return 1.

Returns 2 upon error.

**Assumptions:**

Assumes that rev file is a binary file of 28 bytes per record and the first real\*8 is the time in J2000sec.

**Input Arguments:**

- Product file name
- Rev file name

**Output Arguments:**

- Laser time period (from get\_data\_directory.tcl)
- Laser refID (from get\_data\_directory.tcl)

**Global Variables Used:**

- errorInfo
- bin\_path

**Output Files:**

- None

**Error Handling:**

- Returns the error number to the calling script
- Error from an external procedure.
- Error from a subroutine.

**Routines Called:**

- get\_data\_directory.tcl
- read\_keyword.tcl: Refer to section 2.3.5.17 for details
- convert2j2000.tcl: Refer to section 2.6.1 for details

### 2.3.5.18 get\_data\_directory.tcl

This routine Finds the data directory based on time from the database.

**Assumptions:**

Assumes that the MySQL database is already open.

**Input Arguments:**

- Time in J2000 secs

**Output Arguments:**

- Laser time period
- Laser refID

**Global Variables Used:**

- errorInfo
- bin\_path
- conn

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

### 5.3.10 check\_pid.tcl

This routine checks if files are in the same 14 rev PID as first file.

If yes return 0.

If no return 1.

**Assumptions:**

Assumes mSCF naming convention

**Input Arguments:**

- Product file list

**Output Arguments:**

- None

**Global Variables Used:**

- errorInfo
- bin\_path

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- find\_track.tcl: Refer to section 5.3.5 for details

### 5.3.11 time\_sort.tcl

This routine sorts input file list by ascending time and returns sorted file list.

**Assumptions:**

- File names in list must include full path

**Input Arguments:**

- List to be sorted

**Output Arguments:**

- Sorted list

**Global Variables Used:**

- None

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

### 5.3.12 update\_prod\_rel\_file.tcl

This routine updates the product\_release file if the input release is greater than the release in the file for that product.

**Assumptions:**

- None

**Input Arguments:**

- Product\_release file name
- Release
- List of unique reference products (i.e. GLA01, GLA02)

**Output Arguments:**

- None

**Global Variables Used:**

- errorInfo
- bin\_path

**Output Files:**

- Product\_release file is overwritten

**Error Handling:**

- Returns the error number to the calling script
- Error opening product\_release file
- Error if new release < file release

**Routines Called:**

- None

## 6 Fulfilling Subscriptions

### 6.1 Invocation

Fulfilling subscriptions is invoked with the script `/SCF/bin/ops/run_process_subs.ksh`. This script defines environment variables needed to run the subscription software and creates an error file that is sent to SCF personnel via email upon error.

### 6.2 Environment Definitions

```
export SHLIB_PATH= [location of the shared libraries]
export ICESATVIS_BIN=[location of the executable and script files]
export ANC07_FILE1=[location of anc07_001_01_0000.dat file for qapg]
export ANC07_FILE2=[location of anc07_001_01_0003.dat file for qapg]
export ANC07_FILE3=[location of anc07_001_01_0004.dat file for qapg]
export ANC07_FILE4=[location of anc07_001_01_0006.dat file for qapg]
export REV_FILE=[rev file]
export SCF_ERROR_FILE=[location of the error/constants file]
export TMP_DIRECTORY=[name of the temporary directory]
export SRC_DIR=[path to the Perl scripts]
export DB_NAME=[name of mysql database]
export DB_USER=[username for mysql database]
export DB_PASSWD=[password for mysql database]
export REQ_HEAD_BYTES=[number of bytes in the REQ header]
export LOCK_FILE=[name of lock file to check for]
export CURRENT_LASER=[name of active laser campaign]
export SAVED_DIST_DIR=[name of directory containing saved distribution data]
export CHECK_FULL_PID_FLAG =[flag to indicate if check_full_pid.tcl is called (0/1)]
```

### 6.3 Main script to process subscriptions: `process_subs.tcl`

#### **Synopsis:**

The CREATION database table is accessed to see if subscriptions need be fulfilled on a batch of files. If yes, then the subscriptions software is invoked.

#### **Assumptions:**

- Files are processed in batches per PID per product.
- Two file batches can be processed at the same time.

#### **Error Handling:**

The script terminates when an error occurs. The error information from all the calling routines is saved in `errorInfo` and is written into the log file.

- Can't open or connect to the database
- Error from an external procedure.
- Error from a subroutine.

**Description of Algorithm:**

Refer to Appendix B.3: Flowchart for subscription.tcl

**Output Files:**

- Subsetted datasets including GLA, BN, GR, PS, UR, and possible browse files along with a PDR file for data transfer.

**Subroutines called:**

- create\_binrev.tcl – Refer to section 5.3.1 for more information
- create\_geo.tcl – Refer to section 5.3.2 for more information
- create\_pass\_files.tcl – Refer to section 5.3.3 for more information
- create\_unique\_files.tcl – Refer to section 5.3.4 for more information
- find\_track.tcl – Refer to section 5.3.5 for more information
- create\_pdr.tcl
- lsearch\_all.tcl
- run\_ds\_pc.tcl
- sub\_error.tcl
- check\_full\_pid.tcl

**6.3.1 create\_pdr.tcl**

Creates the PDR input file and runs create\_pdr.pl, which creates the PDR for the input list of files. Refer to section 6.3 for information on create\_pdr.pl.

**Input Arguments:**

- Output directory
- File list
- Distribution ID
- Request ID

**Output Arguments:**

- None

**Global Variables Used:**

- errorInfo
- perl\_path
- log\_directory

**Output Files:**

PDR and XFR

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.

- Error from an external procedure.
- Error opening input file

**Routines Called:**

- create\_pdr.pl - Refer to section 6.3 for more information

### **6.3.2 lsearch\_all.tcl**

Returns list of all list entries that match lsearch pattern.

**Input Arguments:**

- Input list
- Pattern

**Output Arguments:**

- Output list

**Global Variables Used:**

- None

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

### **6.3.3 run\_ds\_pc.tcl**

Creates control files and runs data\_select, parse\_req, and prod\_create.

**Input Arguments:**

- Tmp directory
- Input file for data\_select containing list of BN and UR files (File must already exist)
- Control file name for data\_select (just the name, the file will be created)
- Output string for data\_select output files containing request ID, product set ID and version
- Track file for rewrite\_req containing requested tracks (File must already exist)
- Control file name for prod\_create (just the name, the file will be created)
- Control file name for parse\_req (just the name, the file will be created)
- Data path for data\_select
- Data path for prod\_create
- Flag indicating if prod\_create should be run (0=no, 1=yes)

**Output Arguments:**

- List of subsetted product files from prod\_create. If prod\_create is not run, "" is returned.

**Global Variables Used:**

- errorInfo
- bin\_path
- rev\_file
- anc70\_file
- startJ2000
- endJ2000
- maxLat
- minLat
- maxLon
- minLon
- ds\_prod\_list

**Output Files:**

- REQ files in tmp directory from data\_select. The format for the REQ file is in Appendix C.
- Subsetted product files from prod\_create

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error from an external procedure.
- Error from a subroutine.

**Routines Called:**

- write\_ds\_ctrl.tcl
- write\_pr\_ctrl.tcl
- write\_pc\_ctrl.tcl
- data\_select.f90 - Refer to section 4.1.6 for more information
- rewrite\_req.f90 - Refer to section 4.1.6.18 for more information
- prod\_create.f90 - Refer to section 4.1.10 for more information

**6.3.3.1.write\_ds\_ctrl.tcl**

Creates the control file for the data\_select software.

**Input Arguments:**

- Control file name
- Input file name containing list of BN and UR files
- Output string for data\_select output files containing request ID, product set ID and version
- Date\_str for data select output file. If it is null, the start date will replace it.
- Tmp directory

- Data path

**Output Arguments:**

- None

**Global Variables Used:**

- bin\_path
- rev\_file
- anc70\_file
- startJ2000
- endJ2000
- maxLat
- minLat
- maxLon
- minLon
- ds\_prod\_list

**Output Files:**

- Control file for data\_select

**Error Handling:**

- Returns the error number to the calling script
- Error from an external procedure.
- Error opening the control file

**Routines Called:**

- None

### 6.3.3.2.write\_pr\_ctrl.tcl

Creates the control file for the parse\_req software.

**Input Arguments:**

- Control file name
- Tmp directory
- List of created REQ files
- Data path

**Output Arguments:**

- None

**Global Variables Used:**

- anc70\_file

**Output Files:**

- Control file for prod\_create

**Error Handling:**

- Returns the error number to the calling script
- Error from an external procedure.
- Error opening the control file

**Routines Called:**

- None

**6.3.3.3.write\_ps\_ctrl.tcl**

Creates the control file for the prod\_create software.

**Input Arguments:**

- Control file name
- Tmp directory
- List of created REQ files
- Data path

**Output Arguments:**

- None

**Global Variables Used:**

- anc70\_file

**Output Files:**

- Control file for prod\_create

**Error Handling:**

- Returns the error number to the calling script
- Error from an external procedure.
- Error opening the control file

**Routines Called:**

- None

**6.3.4sub\_error.tcl**

Changes pending subscriptions to "B" in CREATION table.

**Input Arguments:**

- Process number (P1 or P2)

**Output Arguments:**

- None.

**Global Variables Used:**

- errorInfo
- bin\_path
- conn

**Output Files:**

- None

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.
- Error from failed database action.

**Routines Called:**

- None

### 6.3.5 check\_full\_pid.tcl

Checks that input file list has complete number of files for PID.

**Input Arguments:**

- file list

**Output Arguments:**

- PID full flag (0/1)

**Global Variables Used:**

- errorInfo
- bin\_path

**Output Files:**

- None

**Error Handling:**

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into a log file and will terminate.

**Routines Called:**

- None

## 7 Energy Analysis

### 7.1 Invocation

Energy analysis is invoked with the script /SCF/bin/ops/run\_process\_ea.ksh This script defines environment variables needed to run the energy analysis software and creates an error file that is sent to SCF personnel via email upon error.

### 7.2 Environment Definitions

```
export ISIPS_PATH=[location of I-SIPS files before processing]
export ICESATVIS_TMP=[location for temporary directories]
export DB_NAME=[name of mysql database]
export DB_USER=[username for mysql database]
export DB_PASSWD=[password for mysql database]
export WEB_EA_DIR=[name of directory on glas-scfweb for energy analysis files]
export WEB_WF_DIR=[name of directory on glas-scfweb for waveform analysis files]
export EA_DIR=[name of directory containing energy analysis code]
```

### 7.3 Main script to run energy analysis: process\_ea.tcl

#### Synopsis:

The CREATION database table is accessed to see if energy analysis needs be performed on a batch of GLA01 files. If yes, then the energy analysis software is invoked.

#### Assumptions:

- Only GLA01 files are processed.
- Files are processed in batches per PID.
- File batches are processed one at a time.

#### Error Handling:

The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and is written into the log file.

- Can't open or connect to the database
- Error from an external procedure.
- Error from a subroutine.

#### Description of Algorithm:

Refer to Appendix B.3: Flowchart for subscription.tcl

#### Output Files:

- Png files for display on the SCF website.

#### Subroutines called:

- write\_ea\_ctrl.tcl

### 7.3.1 write\_ea\_ctrl.tcl

This routine writes the control file for the energy and waveform analysis software and runs it. Copies resultant png and html files to glas-scfweb.

#### **Assumptions:**

- File names in GLA01 list must include full path

#### **Input Arguments:**

- Control file
- Output directory
- Directory on web host for energy analysis web output
- Directory on web host for waveform analysis web output
- List of GLA01 files
- Laser campaign
- Reference orbit

#### **Output Arguments:**

- Control file for scfplot.pro

#### **Global Variables Used:**

- errorInfo
- bin\_path
- ea\_dir

#### **Output Files:**

- None

#### **Error Handling:**

- Returns the error number to the calling script
- Error opening control file
- Error from an external procedure

#### **Routines Called:**

- run\_ea.ksh
- ps\_to\_png.ksh
- scp\_file\_web.ksh

#### 7.3.1.1.run\_ea.ksh

Script to run scfplots.pro. Converts resultant data files to html files.

#### **Assumptions:**

- None

**Input Arguments:**

- Control file name
- Energy analysis plot flag
- Waveform analysis plot flag
- Energy analysis text file name
- Energy analysis HTML file name
- Waveform analysis text file name
- Waveform analysis HTML file prefix

**Output Arguments:**

- None

**Global Variables Used:**

- Uses scf\_envIRON.ksh

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- /SCF/IDL/EnergyAnalysis/ops/scfplots.pro
- /SCF/IDL/EnergyAnalysis/ops/energy\_analysis2html.pro
- /SCF/IDL/EnergyAnalysis/ops/wf\_analysis2html.pro

### 7.3.1.1.1. scfplots.pro

Creates energy and waveform analysis plots based on GLA01. Outputs data files then creates plots output to png files.

**Input Arguments:**

- Control file name

**Output Files:**

- energy\_analysis.txt
- energy\_analysis.ps
- wf\_log.txt
- wfdailystats.txt
- wf\_plots.dat
- wf\_plots\_1.ps to wf\_plots\_5.ps

**Routines Called:**

- Several in /SCF/IDL/EnergyAnalysis/ops

#### **7.3.1.1.2. energy\_analysis2html.pro**

Converts energy analysis data file to html file.

##### **Input Arguments:**

- Input text file name
- Output HTML file name

##### **Output Files:**

- energy\_analysis\_table.html

##### **Routines Called:**

- None

#### **7.3.1.1.3. wf\_analysis2html.pro**

Converts waveform analysis data file to html files.

##### **Input Arguments:**

- Input text file name
- Output HTML file name

##### **Output Files:**

- wf\_analysis\_COMBINED.html
- wf\_analysis\_ICESHEET.html
- wf\_analysis\_LAND.html
- wf\_analysis\_OCEAN.html
- wf\_analysis\_SEAICE.html

##### **Routines Called:**

- None

#### **7.3.1.2.ps\_to\_png.ksh**

Converts a postscript file to PNG file using image\_convert. This file is in /SCF/IDL/EnergyAnalysis/ops.

##### **Assumptions:**

- Rotates the image 270 degrees

##### **Input Arguments:**

- PS file name
- PNG file name

**Output Arguments:**

- None

**Global Variables Used:**

- None

**Output Files:**

- PNG file

**Error Handling:**

- None

**Routines Called:**

- None

### **7.3.1.3.scp\_file\_web.ksh**

Scp a file to glas-scfweb web site.

**Assumptions:**

- None

**Input Arguments:**

- Input directory on icesat0
- Output directory on glas-scfweb
- File name to transfer

**Output Arguments:**

- None

**Global Variables Used:**

- None

**Output Files:**

- None

**Error Handling:**

- None

**Routines Called:**

- None

## 8 Data Distribution

The data distribution procedure is invoked by the special request script.

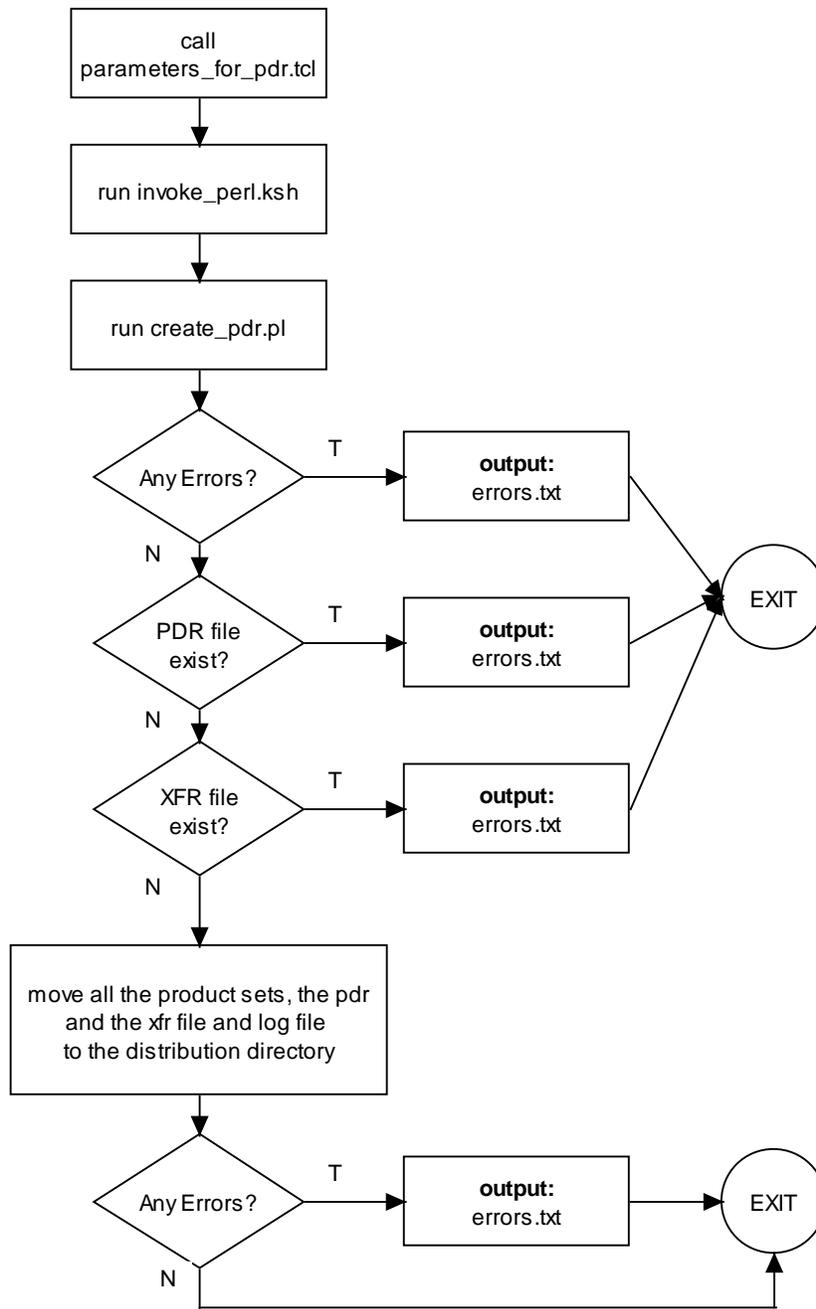


Fig 6.1

## 8.1 parameters\_for\_pdr.tcl

It prepares the control file that runs invoke\_perl.ksh .

### output file:

ctrl\_pdr.dat

It is a text file that consist of keywords with the parameters needed for creating the PDR files.

### Example of a control file:

```
REQUEST_ID = r_582;
TOTAL_FILE_COUNT = 3;
SUBDIR = /SCF/product_sets/logs;
DATA_TYPE = GLA05;
DATA_VERSION = 0;
FILE_ID = tzipi_Feb27_13:43:06_information.log;
FILE_TYPE = LOG;
SUBDIR = /SCF/product_sets/V2_inttest;
DATA_TYPE = GLA06;
DATA_VERSION = 0;
FILE_ID = GLA06_00010112_r_582.268_00;
FILE_TYPE = SCIENCE;
FILE_ID = BNA06_00010112_r_582.268_00;
FILE_TYPE = ANCILLARY;
```

More detailed information can be found in the *SCF Interface Software Detailed Design Document*.

### Error Handling:

The script return an error number that is not zero to the calling script. The error message is written to the global variable errorInfo.

- Can't open a file
- No bin directory was created
- No product file was created
- Can't delete the REQ files

### Description of Algorithm:

- Open the control file
- Find the number of files to deliver and write it in the control file
- Write the rest of the parameters into the control file
- Delete the REQ files from the directory

### Calling Subroutines:

None.

## 8.2 invoke\_perl.ksh

Defines the environment variables for the perl script and runs create\_pdr.pl.

## 8.3 create\_pdr.pl

This program reads a list of file names and creates a PDR file. It also creates an XFR file denoting the completion of the PDR file.

The format of the input file is keyword = value.

More detailed information can be found in the *SCF Interface Software Detailed Design Document*.

### output files:

PDR and XFR files.

The format and contents of the following files is in the *SCF Interface Software Detailed Design Document*.

### Example of PDR file:

```
PDR_TYPE = DISTRIBUTION;
ORIGINATING_SYSTEM = MSCF;
REQUEST_ID = r_583;
TOTAL_FILE_COUNT = 3;
TIME_STAMP = 2002-02-27T18:46:57Z;
OBJECT = FILE_GROUP;
  SUBDIR = /SCF/product_sets/logs;
  DATA_TYPE = GLA05;
  DATA_VERSION = 0;
  NODE_NAME = icesat0.gsfc.nasa.gov;
  OBJECT = FILE_SPEC;
    DIRECTORY_ID = /SCF/tmp/dir_ALT_25214;
    FILE_ID = tzipi_Feb27_13:46:55_information.log;
    FILE_TYPE = LOG;
    FILE_SIZE = 313;
    FILE_CHKSUM = 3636182780;
  END_OBJECT = FILE_SPEC;
END_OBJECT = FILE_GROUP;
OBJECT = FILE_GROUP;
  SUBDIR = /SCF/product_sets/V2_inttest;
  DATA_TYPE = GLA06;
  DATA_VERSION = 0;
  NODE_NAME = icesat0.gsfc.nasa.gov;
  OBJECT = FILE_SPEC;
    DIRECTORY_ID = /SCF/tmp/dir_ALT_25214;
```

```
FILE_ID = GLA06_00010112_0583.P0269_00;
FILE_TYPE = SCIENCE;
FILE_SIZE = 864960;
FILE_CHKSUM = 2478905258;
END_OBJECT = FILE_SPEC;
OBJECT = FILE_SPEC;
  DIRECTORY_ID = /SCF/tmp/dir_ALT_25214;
  FILE_ID = BNA06_00010112_0583.P0269_00;
  FILE_TYPE = ANCILLARY;
  FILE_SIZE = 384;
  FILE_CHKSUM = 312849633;
END_OBJECT = FILE_SPEC;
END_OBJECT = FILE_GROUP;
```

**Example of XFR file:**

Done writing /SCF/tmp/dir\_ALT\_25214/MSCF.r\_583.PDR

**Error Handling:**

The script return an error number that is not zero to the calling script. The error message is written to the global variable errorInfo.

- Can't open a file
- Can't create a file.

## 9 Submitting Product QA Updates to the I-SIPS

### 9.1 Invocation

The submit\_qa.tcl script is run by run\_submit\_qa.ksh.

### 9.2 Input Arguments

The following input arguments are required to run submit\_qa.tcl:

QAID= \$1	QA update ID from database
CCB_ACCEPT= \$2	CCB decision to accept QA update: Y (yes), N (no)

### 9.3 Environment Definitions

The following environmental variables are required to run submit\_qa.tcl:

```
export ICESATVIS_BIN=[location of the executable and script files]
export ISIPS_DIST =[location of I-SIPS distribution cache]
export DB_NAME=[name of mysql database]
export DB_USER=[username for mysql database]
export DB_PASSWD=[password for mysql database]
export LOCAL_NAME=[local host name]
export QAID =[input argument 1]
export CCB_ACCEPT =[input argument 2]
```

### 9.4 submit\_qa.tcl

#### Synopsis:

Submits product QA update to I-SIPS and updates database.

#### Assumptions:

- None

#### Output File:

- QA Update Form (QAUF) and XFR via create\_quaf.tcl

#### Error Handling:

The script terminates when an error occurs.

- Can't open or connect to the database
- Error from a subroutine.
- If the input arguments are not valid, the script exits with a usage statement.

#### Subroutines called:

- create\_quaf.tcl

### 9.4.1 create\_qauf.tcl

Creates a QA Update Form (QAUF) for I-SIPS.

#### Input Arguments:

- Output directory
- QA update ID
- Product
- Release
- Start date of data
- End date of data
- QA update (passed, failed, inferred\_passed)

#### Output Arguments:

- QAUF file name

#### Global Variables Used:

- errorInfo
- local\_name

#### Output File:

- QA Update Form (QAUF) and XFR

#### Example of a QA Update Form:

```
ORIGINATING_SYSTEM = MSCF;  
QA_UPDATE_ID = 1;  
TIME_STAMP = 2002-10-30T16:51:12Z;
```

```
PRODUCT = gla06;  
RELEASE = 2.2;  
START_DATE = 63115200;  
END_DATE = 92664000;  
QA_UPDATE = Passed;
```

#### Error Handling:

- Returns the error number to the calling script
- The error message goes to errorInfo and the main script will write it into the database and will terminate.
- Error opening the QAUF.
- Error touching the XFR

#### Subroutines called:

- None

## 10 Monitoring

There are several scripts that monitor the processing status and health of the system. They are run automatically at set time intervals via cron jobs and email SCF personnel if certain criteria are met. They include:

- check\_dist.tcl
- check\_ingest.tcl
- check\_ps.tcl
- check\_ps\_rscf.tcl

### 10.1 check\_dist.tcl

#### Synopsis:

Checks the directories in a list for files greater than the threshold number of hours. If a file is found an email is sent to SCF personnel. Excludes subdirectories in the directory.

#### Invocation:

The check\_dist.tcl script is run by run\_check\_dist.ksh on the main SCF (icesat0).  
The check\_dist.tcl script is run by run\_check\_dist\_rscf.ksh on the remote SCF's.

#### Environment Definitions:

```
export ICESATVIS_BIN=[location of the executable and script files]
export ICESATVIS_TMP =[location of temporary directory for cron log file]
export DIR_LIST =[list of directories to be checked separated by at least one space]
export OLD_HOURS =[threshold number of hours]
export PROC_FILE=[name of processing file to check for]
export PULL_FILE=[name of pull file to check for]
export PUSH_FILE=[name of push file to check for]
```

#### Subroutines called:

- send\_mail\_mscf.pl - Refer to section 3.1.14 for details

### 10.2 check\_ps.tcl

#### Synopsis:

Checks the number of “scf” processes currently running. If the number of processes is greater than the threshold number an email is sent to SCF personnel. Also added disk space check. If certain disks are more than x percent full an email is sent to SCF personnel

#### Invocation:

The check\_ps.tcl script is run by run\_check\_ps.ksh.

#### Environment Definitions:

```
export ICESATVIS_BIN=[location of the executable and script files]
export ICESATVIS_TMP =[location of temporary directory for cron log file]
export MAX_NUM_PROC =[threshold number of processes]
export DIR_LIST=[list of directories]
export MAX_PCT_FULL=[threshold percentage]
```

**Subroutines called:**

- send\_mail\_mscf.pl - Refer to section 3.1.14 for details

### 10.3check\_ps\_rscf.tcl

**Synopsis:**

Checks the number of “scf” processes currently running on rSCF’s. If the number of processes is greater than the threshold number an email is sent to SCF personnel. Also added disk space check. If certain disks on rSCF’s are more than x percent full an email is sent to SCF personnel. Also if full, a lock file is put in distribution cache.

**Invocation:**

The check\_ps\_rscf.tcl script is run by run\_check\_ps\_rscf.ksh.

**Environment Definitions:**

```
export ICESATVIS_BIN=[location of the executable and script files]
export ICESATVIS_TMP =[location of temporary directory for cron log file]
export MAX_NUM_PROC =[threshold number of processes]
export SSH_DIR=[location of the secure shell executables]
export DIR_LIST=[list of directories]
export MAX_PCT_FULL=[threshold percentage]
export LOCK_FILE=[name of lock file]
export rscf_names=[rscf host names]
```

**Subroutines called:**

- send\_mail\_mscf.pl - Refer to section 3.1.14 for details
- ssh\_rscfs.tcl
- mv\_saved\_files.tcl

#### 10.4.1 ssh\_rscfs.tcl

Sets a ssh login on each rSCF and performs command. Just returns site if command is ""

**Input Arguments:**

- site host name
- command

**Output Arguments:**

- answer from command (list)

**Global Variables Used:**

- errorInfo
- bin\_path
- ssh\_dir

**Subroutines called:**

- None

**Comments:**

- Always returns 0
- Sometimes the answer contains the results from the command plus messages from the ssh.

### 10.4.2 mv\_saved\_files.tcl

Moves files from saved\_dist\_dir to site's dist dir.

**Input Arguments:**

- site abbreviation

**Output Arguments:**

- none

**Global Variables Used:**

- errorInfo
- bin\_path
- db\_name
- db\_user
- db\_passwd
- saved\_dist\_dir
- dist\_dir
- lock\_file

**Subroutines called:**

- None

## 11 Gap Checks

There are several scripts that are run manually after processing data to make sure that all files were ingested and processed. They include:

- check\_pending\_subs.tcl
- check\_pending\_ea.tcl
- file\_gap.tcl

There is one script that can be run manually during processing data to check on subscription and energy analysis processing status. It is:

- check\_subs\_ea.tcl

### 11.1 check\_pending\_subs.tcl

#### **Synopsis:**

Checks for bad, pending, or waiting subscriptions in database and sets them to “N” or “NF” based on input argument.

#### **Invocation:**

This script is run by test\_check\_pending\_subs.ksh.

Call with no input arguments to change P1, P2, P3, P4 and B to N

Call with input arguments 1, 2, 3, or 4 to change P1, P2, P3, or P4 to N

Call with input arguments 6, 7 to change W to N or to NF

#### **Environment Definitions:**

export ICESATVIS\_BIN=[location of the executable and script files]

export DB\_NAME=[mysql database name]

export DB\_USER=[mysql database user name]

export DB\_PASSWD= [mysql database password]

#### **Subroutines called:**

- None

### 11.2 check\_pending\_ea.tcl

#### **Synopsis:**

Checks for pending energy analysis runs in database and sets them to "N"

#### **Invocation:**

This script is run by test\_check\_pending\_ea.ksh.

#### **Environment Definitions:**

export ICESATVIS\_BIN=[location of the executable and script files]

```
export DB_NAME=[mysql database name]
export DB_USER=[mysql database user name]
export DB_PASSWD= [mysql database password]
```

**Subroutines called:**

- None

### 11.3file\_gap.tcl

**Synopsis:**

Checks for gaps in products in a directory

**Invocation:**

This script is run by run\_file\_gap.ksh.

**Environment Definitions:**

```
export ICESAT_PRODUCT_SET=[location of data products]
export ICESATVIS_BIN=[location of the executable and script files]
export DB_NAME=[mysql database name]
export DB_USER=[mysql database user name]
export DB_PASSWD= [mysql database password]
```

**Subroutines called:**

- zero\_out.tcl - Refer to section 3.3.5 for details
- find\_track.tcl - Refer to section 5.3.5 for details
- send\_mail\_mscf.pl - Refer to section 3.1.14 for details
- check\_laser\_op.tcl
- get\_file\_times.tcl

#### 11.2.1 check\_laser\_op.tcl

**Synopsis:**

Checks for gaps in products in a directory

**Input Arguments:**

- laser

**Output Arguments:**

- refid\_list
- start\_cycle\_list
- start\_track\_list
- start\_date\_list
- start\_time\_list
- end\_cycle\_list
- end\_track\_list
- end\_date\_list

- end\_time\_list

**Global Variables Used:**

- bin\_path

**Subroutines called:**

- None

### 11.2.2 get\_file\_times.tcl

**Synopsis:**

Gets starting/ending dates/times for product file (reads headers)

**Input Arguments:**

- input\_file

**Output Arguments:**

- start\_date
- start\_time
- end\_date
- end\_time

**Global Variables Used:**

- errorInfo
- bin\_path

**Subroutines called:**

- read\_keyword - Refer to section 2.3.5.17 for details
- read\_header\_val.f90 - Refer to section 4.1.26.1 for details

### 11.4check\_subs\_ea.tcl

**Synopsis:**

Checks subscriptions and energy analysis status in database for input laser period

**Invocation:**

This script is run by test\_check\_subs\_ea.ksh

**Input Arguments:**

- in\_laser

**Environment Definitions:**

export ICESATVIS\_BIN=[location of the executable and script files]  
export DB\_NAME=[mysql database name]

```
export DB_USER=[mysql database user name]
export DB_PASSWD= [mysql database password]
```

## 12 Clean-up

There are several scripts that monitor the processing status and health of the system. They are run automatically at set time intervals via cron jobs and email SCF personnel if certain criteria are met.

### 12.1 daily\_cleanup.tcl

#### Synopsis:

Checks product files in the data directory and subdirectories beginning with “L” to see if they should be used or not based on the INSTRUMENT\_UPDATES table (i.e. is the laser on). If not, then moves files to a "deleted files" directory along with associated UR and PS files and emails SCF personnel. Also checks if a newer version of the file exists. If so then the file is moved to a "deleted files" directory. Also reads a “deleted files” file to check if browse and QAP files need to be moved to the "deleted files" directory as well/

#### Invocation:

The daily\_cleanup.tcl script is run by run\_daily\_cleanup.ksh.

#### Environment Definitions:

```
export ICESATVIS_BIN=[location of the executable and script files]
export ICESATVIS_TMP =[location of temporary directory for cron log file]
export DATA_PATH=[location of data files and/or subdirectories with data files]
export BROWSE_PATH=[location of browse files]
export QAP_PATH=[location of QAP files]
export DELETED_DATA_PATH=[location of “deleted” files]
export DEL_FILE=[name of file containing deleted file names]
export DB_NAME=[mysql database name]
export DB_USER=[mysql database user name]
export DB_PASSWD= [mysql database password]
export PROC1=[loop over previous data path (1=yes/0=no)]
export PROC2=[loop over current data path (1=yes/0=no)]
export PROC3=[loop over deleted_files file (1=yes/0=no)]
export ACCESS_DAYS=[check directory if it has been accessed within this many days]
```

#### Subroutines called:

- send\_mail\_mscf.pl - Refer to section 3.1.14 for details
- get\_file\_j2000.tcl
- check\_inst\_update.tcl

### 12.1.2 get\_file\_j2000.tcl

Gets starting/ending times in J2000 seconds for product file (reads headers).

**Assumptions:**

Start and end dates/times are within first 500 header lines

**Input Arguments:**

- Product file

**Output Arguments:**

- Starting J2000 seconds
- Ending J2000 seconds

**Global Variables Used:**

- errorInfo
- bin\_path

**Error Handling:**

- Returns 2 if starting/ending J2000 seconds not found
- Errors if the following keywords are not in file header: RangeBeginningDate, RangeBeginningTime, RangeEndingDate, RangeEndingTime

**Subroutines called:**

- read\_header\_val.f90 - Refer to section 4.1.26.1 for details

### 12.1.3 get\_pid\_j2000.tcl

Gets starting/ending times in J2000 seconds from refID, cycle, track, and segment using rev file.

**Assumptions:**

- A rev is 5800 secs, a quarter rev is 1450 secs

**Input Arguments:**

- Reference orbit ID (prkk)
- Cycle (3 digit)
- Track (4 digit)
- Segment
- Rev file

**Output Arguments:**

- Starting J2000 seconds
- Ending J2000 seconds

**Global Variables Used:**

- errorInfo
- bin\_path

**Error Handling:**

- Returns 2 if starting/ending J2000 seconds not found

#### **12.1.4 check\_inst\_update.tcl**

Compares input time in J2000 seconds against INSTRUMENT\_UPDATES table and returns flag indicating whether to use the data (Y/N). Returns -2 if use\_flag could not be found, 0 if found.

##### **Assumptions:**

- Default is use\_flag=N if data end time before table start time
- If use\_flag=Y for any part of time span, then use\_flag=Y

##### **Input Arguments:**

- Start time in J2000 seconds
- End time in J2000 seconds

##### **Output Arguments:**

- Data use flag

##### **Global Variables Used:**

- errorInfo
- bin\_path
- db\_name
- db\_user
- db\_passwd

##### **Error Handling:**

- Returns error if problem accessing database

## 13 Instrument Updates

Instrument updates are sent by email to the scfistat account. Every new email that arrives will trigger the execution of the readIStatMail.ksh script that will parse the information parameters and insert them to the mysql database tables.

The readIStatMail.ksh script is triggered every time a new mail is received in the scfistat account because the full path name of the script is in the /etc/mail/forwards/scfistat.forward file.

### 13.1 Invocation and environment

The readIStatMail.ksh script automatically executes whenever a new email is received in the scfistat account. This script defines all the environment variables needed to process the email and executes the mail parsing script.

### 13.2 Environment Definitions

```
export PATH=`cat /etc/PATH`:$PATH    [Include the /etc/ directory to the path]
export ICESATVIS_TMP=                [temporary directory]
export MAIL_FILE= ICESATVIS_TMP/mail.txt [mail file]
export ICESATVIS_BIN=                [location of the executable program and scripts]
export TMP_DIRECTORY=$ICESATVIS_TMP/dir_$$ [the working directory]
export STATION_PATH=                [path to TOO station files]
export LOCAL_NAME=                  [mscf or rscf]
export DB_NAME=                      [mysql database name]
export DB_USER=                      [mysql database user name]
export DB_PASSWD=                    [mysql database password]
```

### 13.3 parse\_istat\_mail.tcl

This script parses the instrument update email, writes the parameters into a file and into the mysql database tables.

#### Input parameters:

Original email – from stdin

#### Output files:

Text file that the mail parameters are written to.

**Error Handling:** The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and will be written into the log file.

- Can't open a file
- Can't parse the file – not in expected format
- Can't populate the mysql database table

**Description of Script:**

- This script parses instrument update email sent to the scfistat account and populates either the TOO\_UPDATE, RTSCM\_UPDATE, or RTSCM\_POINTING database table.
- The email can be a TOO update, a real-time, saved command (RTSCM) update, or an RTSCM pointing command.
- If the email contains TOO information, a station file is created for TOO's on the same day.
- This code processes only one email at a time - the mail is in the stdin

**Subroutines called:**

- update\_error\_table.tcl

## 14 I-SIPS Distribution Monitoring

The I-SIPS creates a report from its Oracle database listing the product files completed by the I-SIPS that day. These I-SIPS distribution reports are sent by email to the scfstat account. Every new email that arrives triggers the execution of the readStatMail.ksh script that parses the information parameters, writes the parameters into a file, and checks to see if the files listed exist in the mSCF directories. Any files that are not found on the mSCF are listed in an email that is sent to mSCF personnel.

The readStatMail.ksh script is triggered every time a new mail is received in the scfstat account because the full path name of the script is in the /etc/mail/forwards/scfstat.forward file.

### 14.1 Invocation and environment

The readStatMail.ksh script automatically executes whenever a new email is received in the scfstat account. This script defines all the environment variables needed to process the email and executes the mail parsing script.

### 14.2 Environment Definitions

```
export PATH=`cat /etc/PATH`:$PATH    [Include the /etc/ directory to the path]
export ICESATVIS_TMP=                [temporary directory]
export ICESATVIS_BIN=                [location of the executable program and scripts]
export TMP_DIRECTORY=$ICESATVIS_TMP/dir_$$ [the working directory]
export DIR_LIST =[list of directories to be checked separated by at least one space]
export LOCAL_NAME=                  [mscf or rscf]
export DB_NAME=                      [mysql database name]
export DB_USER=                      [mysql database user name]
export export DB_PASSWD=             [mysql database password]
```

### 14.3 parse\_stat\_mail.tcl

This script parses the I-SIPS distribution email, writes the parameters into a file and checks to see if the files exist in the mSCF directories listed in an environmental variable. Any files that are not found on the mSCF are listed in an email that is sent to mSCF personnel.

#### **Input parameters:**

Original email – from stdin

#### **Output files:**

Text file that the mail parameters are written to.

**Error Handling:** The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and will be written into the log file.

- Can't open mail file

- Can't parse mail file – not in expected format
- Can't open the Mysql database
- Can't send email

**Description of Script:**

- This script parses I-SIPS distribution email sent to the scfstat account
- The email can be an Oracle I-SIPS distribution email
- This code processes only one email at a time - the mail is in the stdin

**Subroutines called:**

- update\_error\_table.tcl

## 15 Daily Statistics Report

### 15.1 Invocation and environment

The stat\_report.tcl script is run by run\_stat\_report.ksh. It is run by a cron job once per day.

### 15.2 Environment Definitions

```
export ICESATVIS_TMP= [temporary directory]
export ICESATVIS_BIN= [location of the executable program and scripts]
export SIG_BLOCK= [location of the signature block file with SCF manager contact
information]
export DB_NAME= [mysql database name]
export DB_USER= [mysql database user name]
export DB_PASSWD= [mysql database password]
export CURRENT_LASER=[name of active laser campaign]
```

### 15.3 Stat\_report.tcl

This script writes a daily statistics report and emails it to SCF personnel. It calculates the number of hours of data processed from 4 pm yesterday to 4 pm today. If today is Monday, creates a report for the last 3 days. Does not create reports for Saturday and Sunday.

#### Input parameters:

- None

#### Output files:

- Report file

**Error Handling:** The script terminates when an error occurs. The error information from all the calling routines is saved in errorInfo and will be written into the log file.

- Can't open the Mysql database
- Can't open report file
- Can't send email

#### Subroutines called:

- update\_error\_table.tcl
- send\_mail\_report.pl
- get\_data\_directory.tcl: Refer to section 5.3.13.1 for details

#### 15.3.1 send\_mail\_report.pl

#### Synopsis:

Sends email to recipients of SCF status report.

**Invocation:**

send\_mail\_report.pl <file\_with\_message> <subject\_line>

## Appendix A - Mysql Database Tables

The mysql database tables are designed to track of the user's data request and distribution. The tables are on the mSCF system.

### User- Information

- User number – assigned by mSCF
- User name
- Institute name
- User contact info – email, phone, fax, address

### Subscription and Special Request input parameters

- Time/pass span
- Region latitude and longitude limits
- A unique subscription or special request ID, siiii or riiii respectively
- User number associated with the request
- Which GLAS standard data products are requested

### Distribution Information

- Special request or subscription unique ID
- PID span of the main SCF product sets used to fill the request (For subscriptions this will only be one number, for special requests several main SCF product sets could be used)
- Date and time request filled
- Time span of the data sent
- Name and size of each file sent (include output product set ID, PID)
- Names of I-SIPS files that went into each request file.

### Instrument Information

- Dates and times of important events and maneuvers with use flag to indicate if data should be used or not
- Real-time and saved instrument commands
- Target of Opportunity updates
- Dates of when the laser is operational
- Dates of when the laser is off

### Tables list:

- USER
- SPECIAL\_REQUEST\_USER
- SUBSCRIPTION\_USER
- ISIPS\_PRODUCT\_ID
- REQUEST\_PRODUCT\_SEG
- REQUEST\_TRACKS
- REQUEST\_CYCLES
- ISIPS\_SUBSCRIPTIONS
- CREATION

- SUBSCRIPTION\_INPUT\_FILES1
- SUBSCRIPTION\_INPUT\_FILES2
- DISTRIBUTION
- DISTRIBUTION\_FILES
- SPECIAL\_REQUEST\_PID
- [rSCF]\_PRODUCT\_ID
- QA\_PRODUCT\_UPDATE
- TOO\_UPDATE
- INSTRUMENT\_UPDATE
- RTSCM\_UPDATE
- RTSCM\_POINTING
- SUBSCRIPTION\_CYCLES
- SUBSCRIPTION\_TRACKS
- SUBSCRIPTION\_PRODUCT\_SEG
- REQUEST\_INPUT\_FILES

The software to create these tables is in /SCF/tables or use Navicat.

## A.1 Table Name: USER

This table has the information of all the users that are allowed to request data from the mSCF or the I-SIPS.

### Example

userId	userName	institute	email	phone	fax	address
10	Johna	ALT	johna@xxx.gsfc.nasa.gov	NULL	NULL	NULL

### Column Descriptions:

- userId: The assigned user ID of the person who submitted the request. Distinct userId is based on userName and institute.
- userName: The user's name
- institute: Site abbreviations for each rSCF as explained in the *SCF Interface Software Installation Guide*.
- email: User's email address (if submitted)
- phone: User's telephone number (if submitted)
- fax: User's fax number (if submitted)
- address: User's mailing address (if submitted)

### Column Properties

Field	Type	Null	Key	Default	Extra
userId	smallint (5) unsigned		PRI		auto_increment
userName	varchar(20)	YES		NULL	
Institute	varchar(60)	YES		NULL	
Email	varchar(60)	YES		NULL	
Phone	varchar(20)	YES		NULL	
Fax	varchar(20)	YES		NULL	
Address	varchar(60)	YES		NULL	

## A.2 Table Name: SPECIAL\_REQUEST\_USER

This table contains the information for each special request.

### Example

Rindx	requestId	userId	date	startDate	endDate	StartJ2000	EndJ2000
1	r0001	10	2000-07-18 13:08:57	2001-06-05 10:00:00	2001-12-31 00:00:00	0	189388800

### Table Continued

minLat	maxLat	minLon	maxLon	release	ql_flag	output_dir	browse
-90	-53	-180	180	007	0	/SCF/product_sets/Smith	N

### Column Descriptions

- rIndex: unique number that is assigned to every new special request
- requestId: unique ID to each request. The first letter stands for special request and the number is a unique number, rIndex.
- userId: The user ID of the person who sent the special request. The user ID must appear in the USER table.
- date: The date the request was sent
- startDate, endDate: The beginning and end times of the data.
- startJ2000, endJ2000: The beginning and end times of the data in J2000 sec.
- minLat, maxLat, minLon, maxLon: the area to get the data from.
- release: Data release number
- ql\_flag: Is this request quick-look? 0=no, 1=yes
- output\_dir : The directory to put the product sets in
- browse: Are browse products requested? Y=yes, N=no (default)

### Column Properties

Field	Type	Null	Key	Default	Extra
rIndex	smallint(5) varchar(5)		PRI	NULL	auto_increment
requestId	varchar(10)		PRI		
userId	smallint(5) unsigned		PRI	0	
date	datetime			0000-00-00 00:00:00	
startDate	datetime	YES		NULL	
endDate	datetime	YES		NULL	
startJ2000	int(20)	YES		NULL	
endJ2000	int(20)	YES		NULL	
minLat	float			0	
maxLat	float			0	
minLon	float			0	
maxLon	float			0	
release	int(3)	YES		NULL	
ql_flag	smallint(1) unsigned			0	
output_dir	varchar(100)	YES		NULL	
browse	Char(1)	YES		NULL	

### A.3 Table Name: SUBSCRIPTION\_USER

This table contains the information for each subscription request

#### Example

Rindx	requestId	userId	date	startDate	endDate	StartJ2000	EndJ2000
1	s0001	10	2000-07-18 13:08:57	2001-06-05 10:00:00	2001-12-31 00:00:00	0	189388800

Table Continued

minLat	maxLat	minLon	maxLon	ql_flag	output_dir	status	browse
-90	-53	-180	180	0	/SCF/product_sets/Smith	A	N

**Column Descriptions**

- rIndex: unique number that is assigned to every new subscription.
- requestId: unique ID to each request. the first letter stands for subscription and the number is a the unique number, rIndex.
- userId: The user ID of the person that sent the subscription. The user ID must appear in the USER table.
- date: The date the request was sent
- startDate, endDate: The beginning and end times of the data.
- StartJ2000, endJ2000: The beginning and end times of the data in J2000 sec.
- minLat, maxLat, minLon, maxLon: the area to get the data from.
- output\_dir : The directory to put the product sets in
- status: indicates whether the subscription is active (A) or not (N).
- browse: Are browse products requested? Y=yes, N=no (default)

**Column Properties**

Field	Type	Null	Key	Default	Extra
rIndex	smallint(5) varchar(5)		PRI	NULL	auto_increment
requestId	varchar(10)		PRI		
userId	smallint (5) unsigned		PRI	0	
date	datetime			0000-00-00 00:00:00	
startDate	datetime	YES		NULL	
endDate	datetime	YES		NULL	
startJ2000	int(20)	YES		NULL	
endJ2000	int(20)	YES		NULL	
minLat	float			0	
maxLat	float			0	
minLon	float			0	
maxLon	float			0	
ql_flag	Smallint(1) unsigned			0	
output_dir	varchar(100)	YES		NULL	
status	Char(1)	YES		NULL	
browse	Char(1)	YES		NULL	

**A.4 Table Name: ISIPS\_PRODUCT\_ID**

This table contains unique product ID and the passID parameters.

**Example:**

PID	refID	cycleID	TrackID	beginningDate
20	1101	1	15	03022100

**Column Descriptions:**

- PID : unique number for the product
- refID: the reference orbit ID
- cycleID: the cycle number
- trackID: the track number that is the beginning of 14 rev . (1,15,29, ect)
- beginningDate: the beginning date and time of the data in the yymmddhh format.  
The beginningDate is taken from the header of the first GLA01 of the product set.

**Column Properties**

Field	Type	Null	Key	Default	Extra
PID	smallint(5) unsigned		PRI	NULL	auto_increment
refID	int(11)			0	
cycleID	int(11)			0	
trackID	int(11)			0	
beginningDate	Varchar(8)	YES		NULL	

**A.5 Table Name: REQUEST\_PRODUCT\_SEG**

This table contains a list of GLAS products for a special request ID

**Example:**

index	requestId	product
1	r0001	GLA01
2	r0001	GLA05
3	r0001	GLA06

**Column Descriptions:**

- index: unique mysql record index
- requestId: unique special request ID
- product: requested product

**Column Properties:**

Field	Type	Null	Key	Default	Extra
index	int(10)	no	PRI		Auto_increment
requestId	varchar(10)				
product	varchar(5)				

## A.6 Table Name: REQUEST\_TRACKS

This table contains a list of tracks for a specific repeat cycle, for a special request ID.

### Example

lindex	requestId	begin_track	end_track	begin refTrack	end refTrack	refOrbitId
1	r0001	3	3	1	1	1
2	r0001	10	30	1	29	2

### Column Descriptions

- lindex: unique mysql record index
- requestId: unique special request ID
- begin\_track, end\_track: tracks span
- begin refTrack, end refTrack: span of the reference track.  
reference track is the beginning track of the 14 rev tracks.
- refOrbitId: 1 is for 8 days repeat cycles; 2 is for 91 days repeat cycles

### Column Properties

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(10)				
begin_track	int(11)			0	
end_track	int(11)			0	
begin refTrack	int(11)			0	
end refTrack	int(11)			0	
refOrbitId	int(11)			0	

## A.7 Table Name: REQUEST\_CYCLES

This table contains a list of cycles of a specific repeat cycle, for a special request ID.

### Example

lindex	requestId	begin_cycle	end_cycle	refOrbitId
1	r0001	1	1	1
2	r0001	3	3	1
3	r0001	2	2	2

### Column Descriptions

- lindex: unique mysql record index
- requestId: unique special request ID
- begin\_cycle, end\_cycle: Cycles span
- refOrbitId: 1 is for 8 day repeat cycle; 2 is for 91 day repeat cycle

### Column Properties

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment

requestId	varchar(10)				
begin_cycle	int(11)			0	
end_cycle	int(11)			0	
refOrbitId	int(11)			0	

### A.8 Table Name: ISIPS\_SUBSCRIPTIONS

This table gives a list of all available products and segments and indicates whether the mSCF has an active subscription to the I-SIPS for the product and if it is stored at the mSCF.

#### Example:

lindex	product	segment	store	status
1	GLA05	1	Y	A
2	GLA05	3	Y	A
3	GLA05	2	N	N

#### Column Descriptions:

- lindex: unique mysql record index
- product: product
- segment: product segment
- store: indicates whether the product files are stored at the mSCF (Y) or not (N).
- status: indicates whether the subscription to the I-SIPS is active (A) or not (N).

#### Column Properties

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
product	varchar(5)				
segment	int(11)	YES		NULL	
store	char(1)				
status	char(1)				

### A.9 Table Name: CREATION

This table lists the granules files from the I-SIPS.

This table is populated every time a new product set is sent by the I-SIPS to the mSCF.

#### Example

lindex	ISIPS_file	data_dir	PID
1	GLA01_001_1101_003_0030_2_01_01.P0001	/SCF/product_sets/current/L1A	1

laser	date	data_start_j2000	data_end_j2000	subs_run	ea_run
L1A	2002-02-28 10:34:18	140524988	150524988	N	Y

#### Column Descriptions

- lindex: unique mysql record index
- ISIPS\_file: the granule file from ISIPS after been renamed to the MSCF file convention.

- data\_dir: directory where ISIPS\_file resides.
- PID: the product ID. The same as the one in table ISIPS\_PRODUCT\_ID.
- laser: laser campaign of ISIPS\_file.
- date: the date the I-SIPS file was copied to the mSCF
- data\_start\_j2000: the start date of the data in the I-SIPS file in J2000 secs
- data\_end\_j2000: the end date of the data in the I-SIPS file in J2000 secs
- subs\_run: Subscription indicator: N (needs to be run), Y (has been run), D (don't run), P (in process), B (bad), W (waiting), NF (needs to be run – force)
- ea\_run: Energy analysis indicator: N (needs to be run), Y (has been run), D (don't run), P (in process)

### Column Properties

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
ISIPS_file	varchar(100)				
data_dir	varchar(100)	Y			
PID	smallint(5) unsigned			0	
laser	char(3)	Y			
date	datetime			0000-00-00 00:00:00	
data_start_j2000	int				
data_end_j2000	int				
subs_run	char(2)	Y			
ea_run	char(1)	Y			

### A.10 Table Name: SUBSCRIPTION\_INPUT\_FILES1

This table contains a list of I-SIPS files that were used to fulfill a subscription.

#### Example:

lindex	requestId	inputProduct	findex
1	s0094	GLA10_019_2103_002_0351_0_01_0001.P0305	10

#### Column Descriptions:

- lindex: unique mysql record index
- requestId: subscription request ID
- Input Product: the I-SIPS file that went into the subscription output file
- findex: file index corresponding to subscription output file

#### Column Properties:

Field	type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(5)				
inputProduct	varchar(100)				
findex	Int(10)				

**A.11 Table Name: SUBSCRIPTION\_INPUT\_FILES2**

This table contains a list of subscription output files for a subscription ID.

**Example:**

findex	requestId	fileName	date	laser
10	s0094	GLA10_03111407_s0094_L2_2103.P0305_01_02	1/27/2005 10:51:52	L2A

**Column Descriptions:**

- findex: file index corresponding to subscription output file
- requestId: subscription request ID
- filename: name of subscription output file
- date: the date and time that the subscription was filled
- laser: the laser campaign

**Column Properties:**

Field	type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(5)				
Filename	varchar(100)				
date	datetime			0000-00-00 00:00:00	
laser	Varchar(3)				

**A.12 Table Name: DISTRIBUTION**

The table contains the relevant information related to the end of the data request processing

**Example**

lindex	requestId	beginPID	lastPID	date
1	r0001	1	3	2001-07-23 12:38:31

**Column Descriptions**

- lindex: unique mysql record index
- requestId: request ID.
- beginPID, lastPID: process ID span of the mSCF product sets used to fill the request
- date: the date and time that the request was filled

**Column Properties**

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar (10)		PRI		
BeginPID	smallint(5) unsigned			0	
LastPID	smallint(5) unsigned			0	
date	datetime			0000-00-00 00:00:00	

### A.13 Table Name: DISTRIBUTION\_FILES

This table contains the name and the size of each file sent after the process finished.

#### Example

index	requestId	filename	fileSize	date
1	r0001	GLA02_1101_001_0030_0_01_01.P0003	213094464	2001-07-23 12:38:31
2	r0001	BNL_1101_001_0029.P0003_00	238608	2001-07-26 11:39:10

#### Column Descriptions

- index: unique mysql record index
- requestId: request ID
- filename: name of the file sent
- fileSize: size of the file in bytes
- date: the date and time that the request was filled

#### Column Properties

Field	Type	Null	Key	Default	Extra
index	int(10)	no	PRI		Auto_increment
requestId	varchar (10)		PRI		
filename	varchar(100)				
fileSize	int(20)			0	
date	datetime			0000-00-00 00:00:00	

### A.14 Table Name: [rSCF]\_PRODUCT\_ID

This table contains a unique product id that has been assigned to each special\_request. There is a different table for each remote site.

#### Example

product_id	Requested	date
1	r0001	2001-06-31 00:00:00
2	r0002	2001-08-11 00:00:00

#### Column Descriptions

- product\_id: a unique number for each special request.
- date: The date and time the product id was assigned.

#### Column Properties

Field	Type	Null	Key	Default	Extra
product_id	smallint(5) unsigned		PRI	NULL	auto_increment
requestId	varchar(5)	no			
date	datetime	no		0000-00-00 00:00:00	

### A.15 Table Name: SPECIAL\_REQUEST\_PID

This table contains the pid span of each special request.

#### Example

index	requestId	beginPID	lastPID
1	r0001	1	3

#### Column Descriptions

- index: unique mysql record index
- requestId: a unique number for each special request.
- beginPID,endPID: the PID span

#### Column Properties

Field	Type	Null	Key	Default	Extra
index	int(10)	no	PRI		Auto_increment
requestId	varchar(5)				
beginPID	int(11)			0	
lastPID	int(11)			0	

### A.16 Table Name: QA\_PRODUCT\_UPDATE

This table contains product QA update information submitted by the science team.

#### Example

qaID	product	release	date1	date2	user	site	qa_update
1	gla08	2.2	6311520	634564800	tzipi	ALT	Failed

#### Table continued

description	date	ccb_accept	ccb_date
This is just really bad data.	2002-10-29 10:10:31	Y	2002-10-30 12:10:31

#### Column Descriptions

- qaID: QA update ID
- product: Product type
- release: GSAS software release from product header
- date1: Start date of data in J2000 sec
- date2: End date of data in J2000 sec
- user: User name
- site: Site name
- qa\_update: QA update flag indicator: passed, inferred passed, or failed
- description: Justification of why data passed or failed which can be verified
- date: Date and time of QA update submission
- ccb\_accept: Decision of the CCB to accept the QA update: Y (yes), N (no), U (undecided)
- ccb\_date: Date and time that code was run to accept or reject QA update in database

#### Column Properties

Field	Type	Null	Key	Default	Extra
qaID	smallint(5) unsigned		PRI	NULL	auto_increment

product	varchar(5)				
release	float			0	
date1	int(20)			0	
date2	int(20)			0	
user	varchar(20)				
site	varchar(10)				
qa_update	varchar(20)				
description	varchar(255)	Yes		NULL	
date	datetime			0000-00-00 00:00:00	
ccb_accept	char(1)	Yes		NULL	
ccb_date	datetime	Yes		NULL	

### A.17 Table Name: TOO\_UPDATE

This table contains Target of Opportunity information submitted by the instrument team.

#### Example

index	location	status	date	doy	time	rev
1	Mt. Erebus, Antarctica	ACCEPTED	2003-07-17	198	22:20	2769

Table continued

start_lat	stop_lat	start_lon	stop_lon	Station_flag
35.23451	36.55111	140.20781	140.03385	Y

#### Column Descriptions

- index: unique mysql record index
- location: Location of proposed target
- status: Accepted, rejected, or pending as determined by the instrument team
- date: Date of proposed target
- doy: Day of year of proposed target
- time: Time of proposed target
- rev: Rev of proposed target
- start\_lat: Starting latitude of proposed target
- stop\_lat: Ending latitude of proposed target
- start\_lon: Starting longitude of proposed target
- stop\_lon: Ending longitude of proposed target
- station\_flag: Flag indicating whether the station file has been updated (Y/N)

#### Column Properties

Field	Type	Null	Key	Default	Extra
index	int(10)	no	PRI		Auto_increment
location	varchar(30)	YES		NULL	
status	varchar(20)	YES		NULL	
date	varchar(10)	YES		NULL	
doy	int(3)	YES		NULL	
time	varchar(5)	YES		00:00	
rev	int(5)	YES		NULL	
start_lat	varchar(12)	YES		NULL	
stop_lat	varchar(12)	YES		NULL	
start_lon	varchar(12)	YES		NULL	

stop_lon	varchar(12)	YES		NULL	
station_flag	char(1)	YES		NULL	

### A.18 Table Name: INSTRUMENT\_UPDATE

This table contains instrument events and maneuvers submitted by the instrument team.

#### Example

index	description	laser	date	time	J2000sec	doy	refID	cycle	track	use_flag
5	Enters sun acq mode	L1A	2003-03-26	11:41:00.0	101950860	85	1102	5	95	N
6	Leaves sun acq mode	L1A	2003-03-27	00:57:29.0	101998649	86	1102	5	104	Y

#### Column Descriptions

- index: unique index for each record in table
- description: Description of instrument event/maneuver
- laser: Laser campaign
- date: Date of event/maneuver
- time: Time of event/maneuver
- j2000sec: Time of event/maneuver in J2000 seconds
- doy: Day of year of event/maneuver
- refID: Reference orbit ID of event/maneuver
- cycle: Cycle of event/maneuver
- track: Track of event/maneuver
- use\_flag: Use flag – use data? (Y/N)

#### Column Properties

Field	Type	Null	Key	Default	Extra
index	Int(3)	NO	PRI		Auto_increment
description	varchar(100)	YES		NULL	
laser	varchar(3)	YES		NULL	
date	varchar(10)	YES		NULL	
time	varchar(10)	YES		NULL	
J2000sec	Int(20)	YES		NULL	
doy	int(3)	YES		NULL	
refID	Int(4)	YES		NULL	
cycle	int(3)	YES		NULL	
track	int(4)	YES		NULL	
use_flag	char(1)	YES		NULL	

### A.19 Table Name: RTSCM\_UPDATE

This table contains real-time and saved instrument commands submitted by the instrument team.

#### Example

	index	description	date	time	doy
	1	(RT)AD Enable/Disable auto-gain settings (enable)	2003-03-26	11:41:00.0	85
	2	ADPOINT with TARGTYPE fixed, TARGID 285	2003-03-27	00:57:29.0	86

### Column Descriptions

- index: unique mysql record index
- description: Description of command
- date: Date of command
- time: Time of command
- doy: Day of year of command

### Column Properties

Field	Type	Null	Key	Default	Extra
index	int(10)	no	PRI		Auto_increment
description	varchar(100)	YES		NULL	
date	varchar(10)	YES		NULL	
time	varchar(10)	YES		NULL	
doy	int(3)	YES		NULL	

### A.20 Table Name: RTSCM\_POINTING

This table contains real-time and saved pointing commands submitted by the instrument team.

### Example

index	description	date	time	doy	Point_flag
1	2006/032-09:07:24.00 CMD ICE ADPOINT with TARGTYPE fixed, TARGID 851 ;	2003-03-26	11:41:00.0	85	2
	2006/033-14:23:45.00 CMD ICE CBMEXE with INDEX 40	2003-03-27	00:57:29.0	86	3

### Column Descriptions

- index: unique mysql record index
- description: Description of command
- date: Date of command
- time: Time of command
- doy: Day of year of command
- point\_flag: Type of pointing indicator:
  - 0 - no pointing
  - 1 - reference track pointing
  - 2 - PATH target pointing
  - 3 - scan (ocean or ATW)

### Column Properties

Field	Type	Null	Key	Default	Extra
index	int(10)	no	PRI		Auto_increment
description	varchar(100)	YES		NULL	
date	varchar(10)	YES		NULL	
time	varchar(10)	YES		NULL	
doy	int(3)	YES		NULL	
point_flag	Int(1)	YES		NULL	

### A.21 Table Name: SUBSCRIPTION\_PRODUCT\_SEG

This table contains a list of GLAS products for a subscription ID

#### Example:

lindex	requestId	product
1	s0001	GLA01
2	s0001	GLA05
3	s0001	GLA06

#### Column Descriptions:

- lindex: Unique mysql record index
- requestId: unique subscription ID
- product: requested product

#### Column Properties:

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(10)				
product	varchar(5)				

### A.22 Table Name: SUBSCRIPTION\_TRACKS

This table contains a list of tracks for a specific repeat cycle, for a subscription ID.

#### Example

lindex	RequestId	begin_track	end_track	begin refTrack	end refTrack	refOrbitId
1	s0001	3	3	1	1	1
2	s0001	10	30	1	29	2

#### Column Descriptions

- lindex: unique mysql record index
- requestId: unique subscription ID
- begin\_track, end\_track: tracks span
- begin refTrack, end refTRack: span of the reference track.  
reference track is the beginning track of the 14 rev tracks.
- refOrbitId: 1 is for 8 days repeat cycles; 2 is for 91 days repeat cycles

#### Column Properties

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(10)				
begin_track	int(11)			0	

end_track	int(11)			0	
begin refTrack	int(11)			0	
end refTrack	int(11)			0	
refOrbitId	int(11)			0	

### A.23 Table Name: SUBSCRIPTION \_CYCLES

This table contains a list of cycles of a specific repeat cycle, for a subscription ID.

#### Example

lindex	RequestId	begin_cycle	end_cycle	refOrbitId
1	s0001	1	1	1
2	s0001	3	3	1
3	s0001	2	2	2

#### Column Descriptions

- lindex: unique mysql record index
- requestId: unique subscription ID
- begin\_cycle, end\_cycle: Cycles span
- refOrbitId: 1 is for 8 day repeat cycle; 2 is for 91 day repeat cycle

#### Column Properties

Field	Type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(10)				
begin_cycle	int(11)			0	
end_cycle	int(11)			0	
refOrbitId	int(11)			0	

### A.24 Table Name: REQUEST\_INPUT\_FILES

The table contains a list of granules that were used to run a fulfill a special request.

#### Example:

lindex	requestId	inputProduct	fileName	date	laser
1	r0094	GLA10_019_2103_002_0351_0_01_0001. P0305	GLA10_03111407_r0094_L2_2103. P0305_01_02	1/27/2005 10:51:52	L2A

#### Column Descriptions:

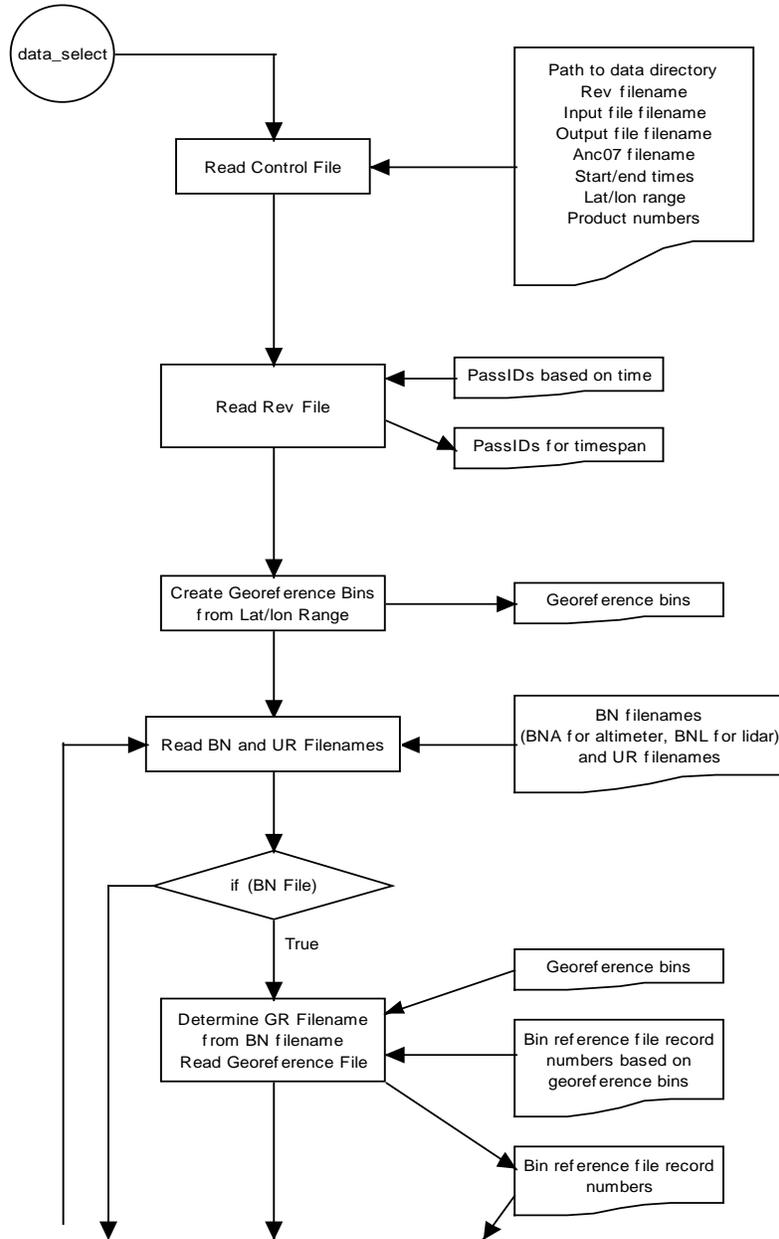
- lindex: unique mysql record index
- requestId: request ID.
- Input Product: the I-SIPS file that went into fulfilling the request
- Filename: name of the file sent to the remote sites
- date: the date and time that the request was filled
- laser: the laser campaign

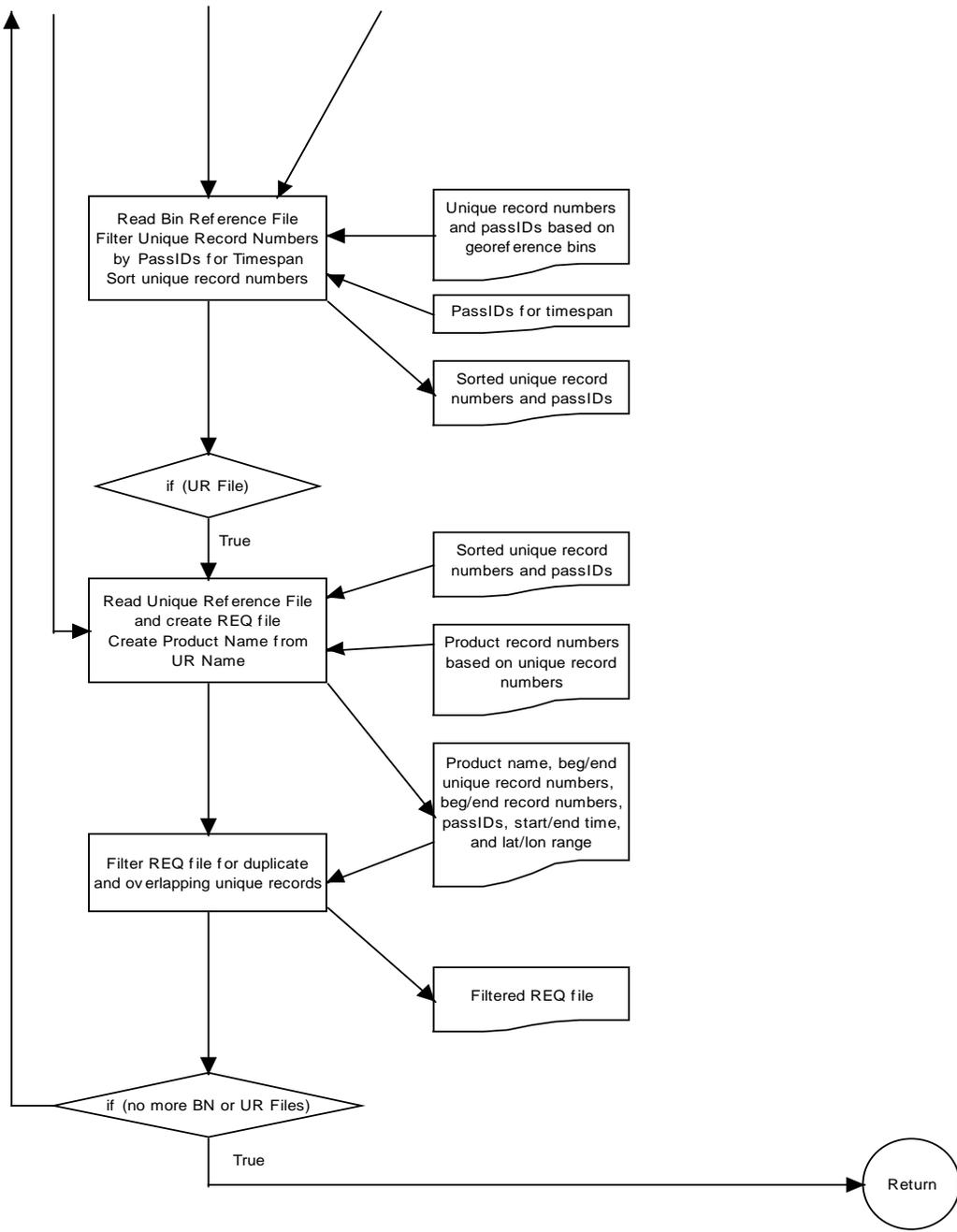
**Column Properties:**

Field	type	Null	Key	Default	Extra
lindex	int(10)	no	PRI		Auto_increment
requestId	varchar(5)				
inputProduct	varchar(100)				
Filename	varchar(100)				
date	datetime			0000-00-00 00:00:00	
laser	Varchar(3)				

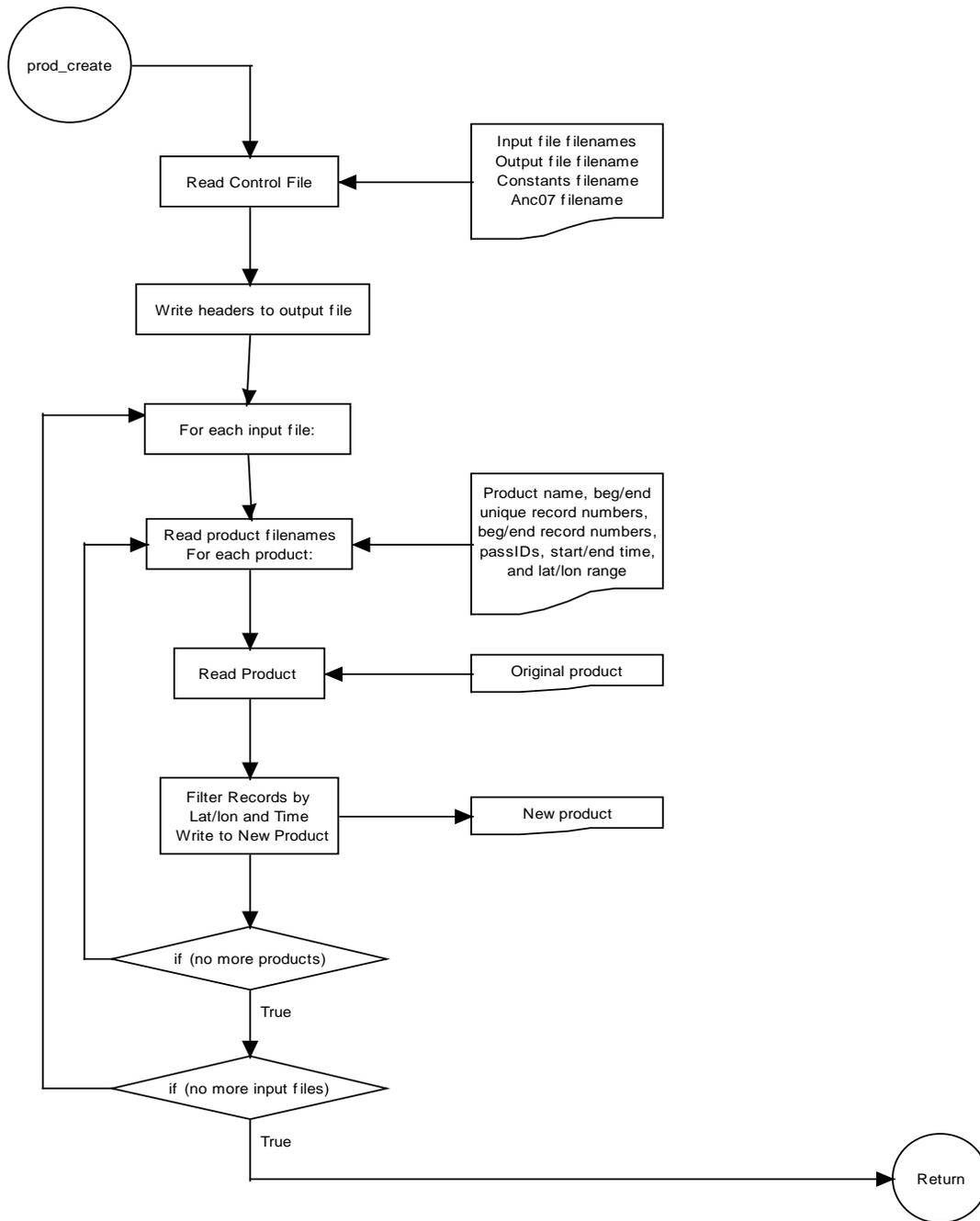
# Appendix B - Flowcharts

## B.1 Flowchart for data\_select.f90

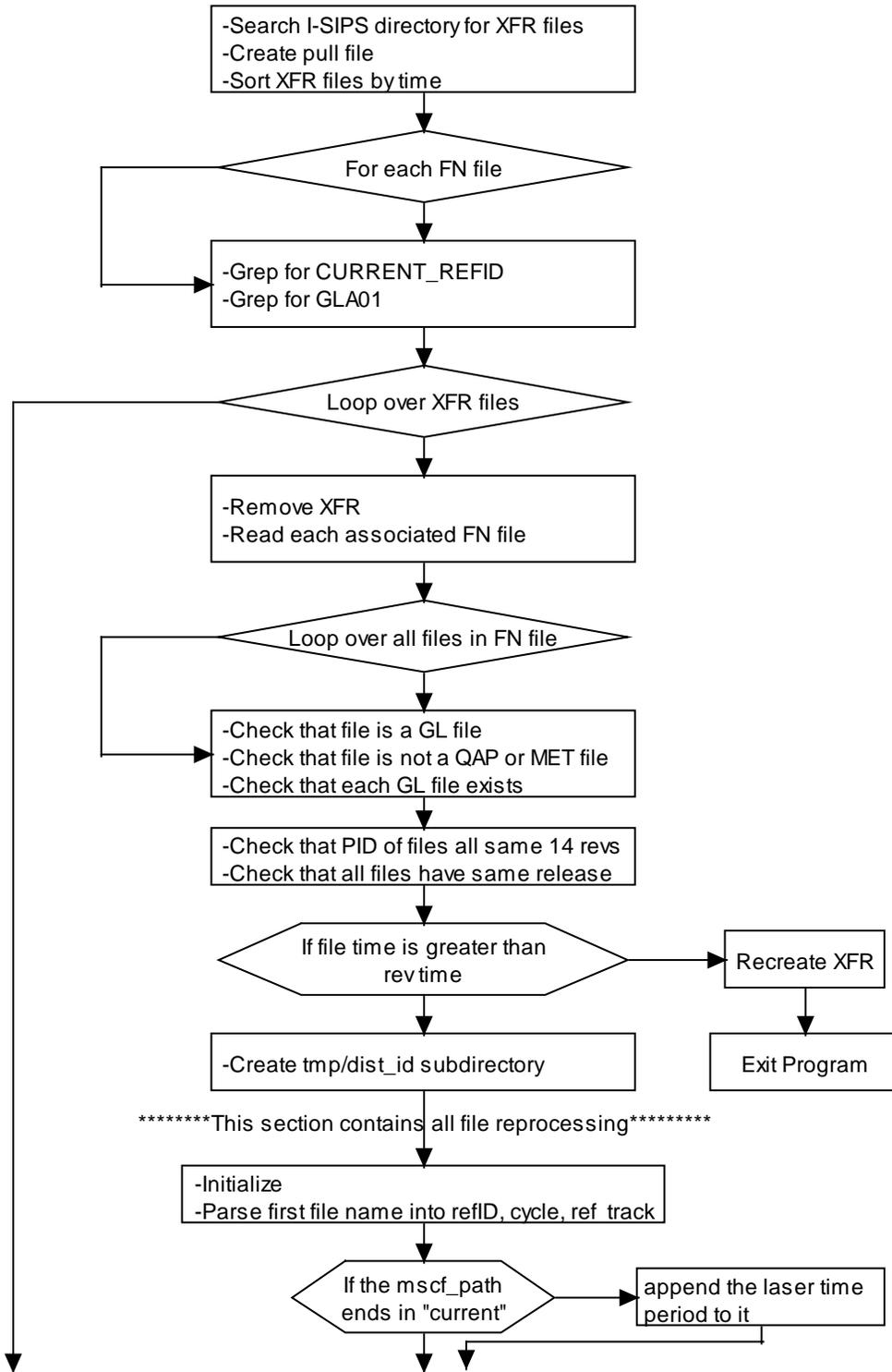


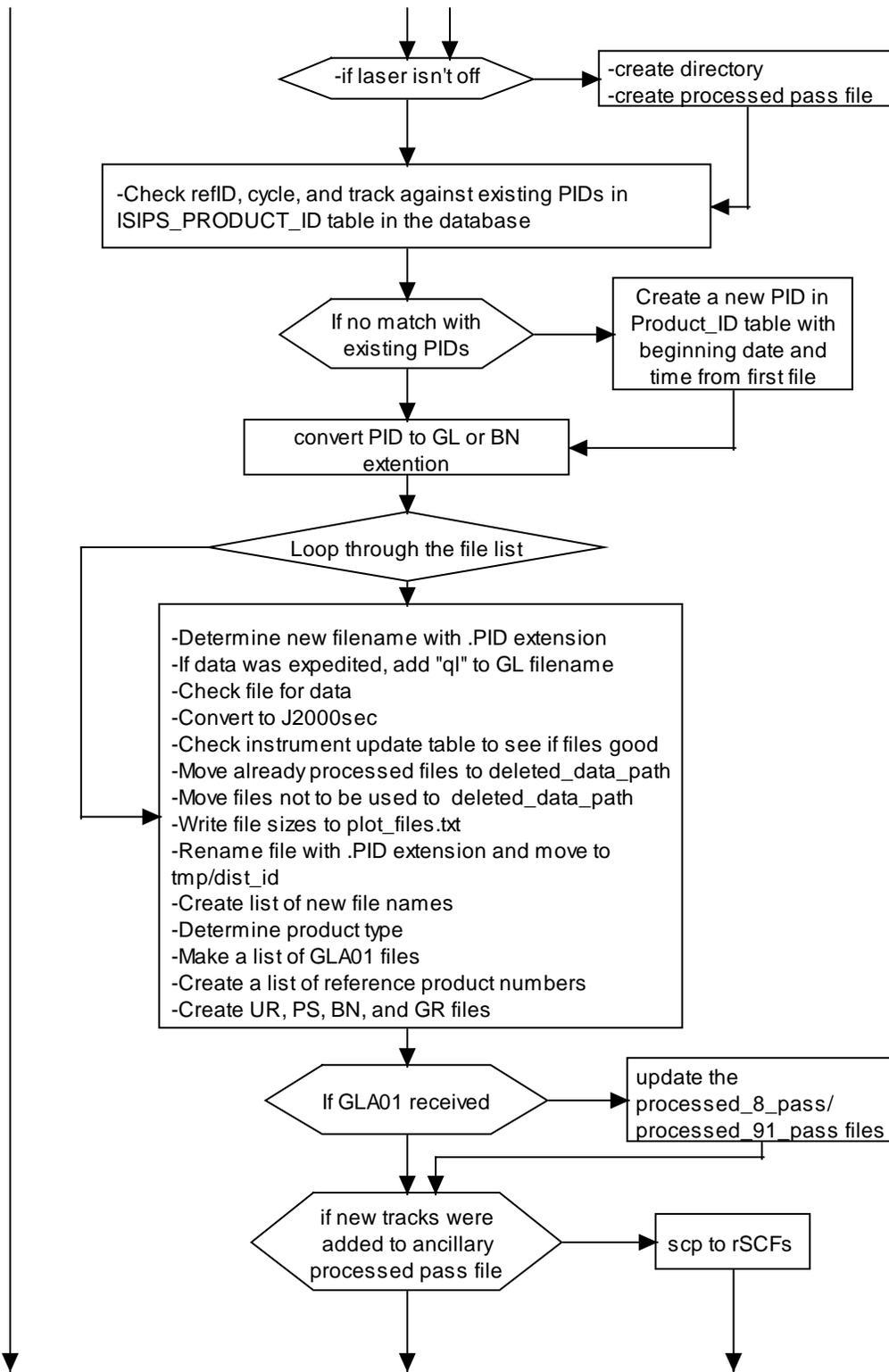


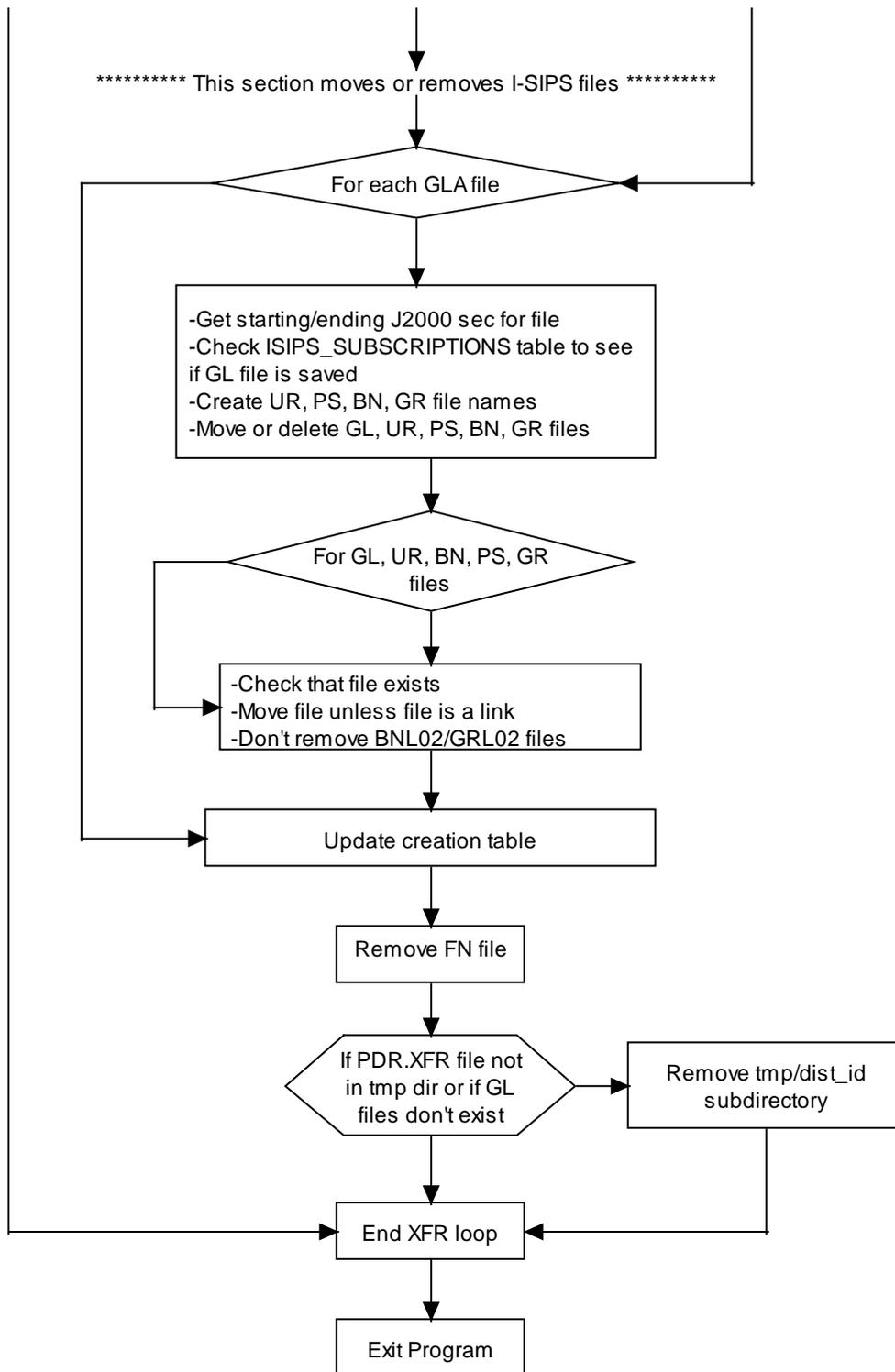
## B.2 Flowchart for prod\_create.f90



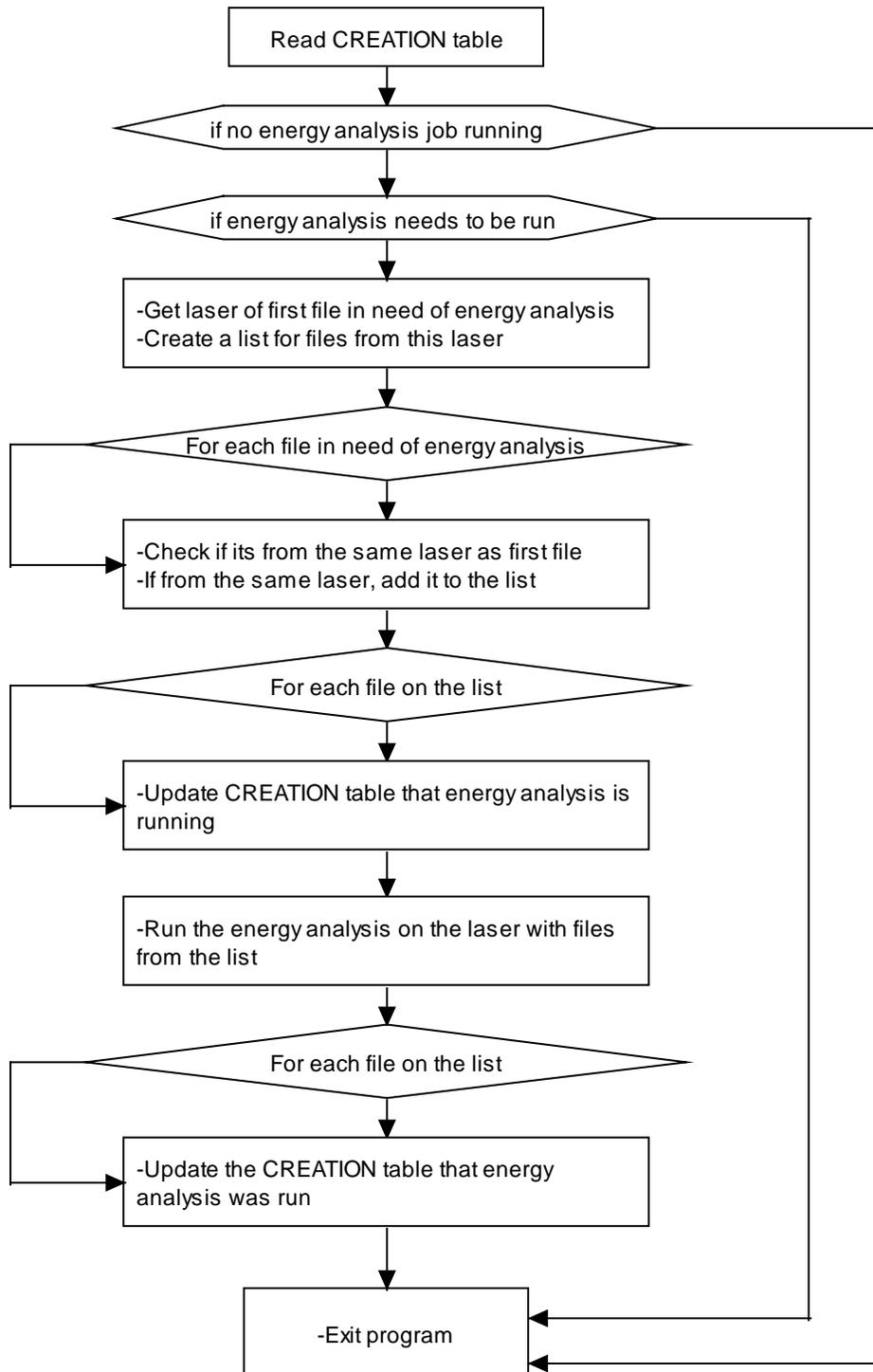
### B.3 Flowchart for process\_data.tcl



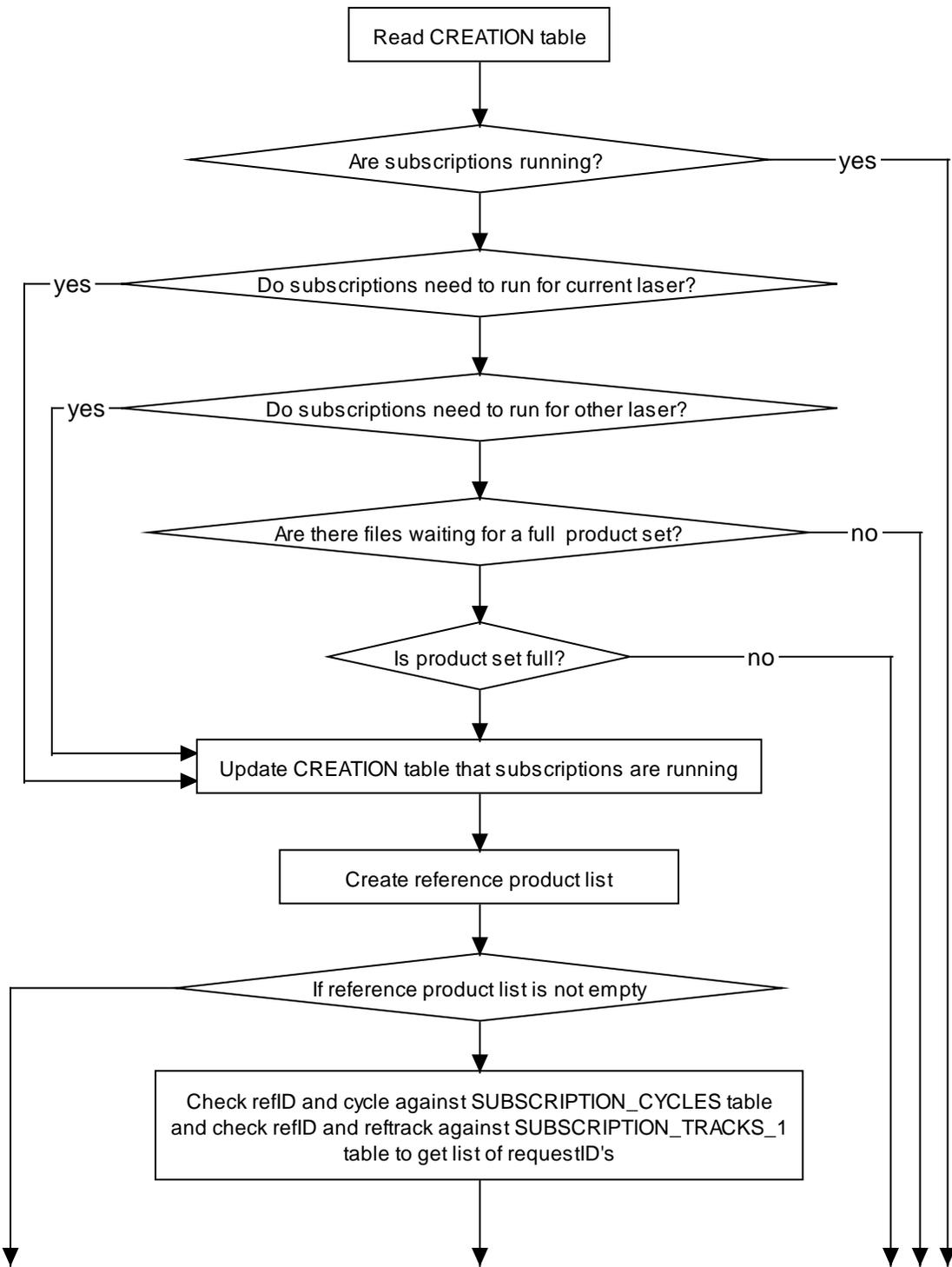


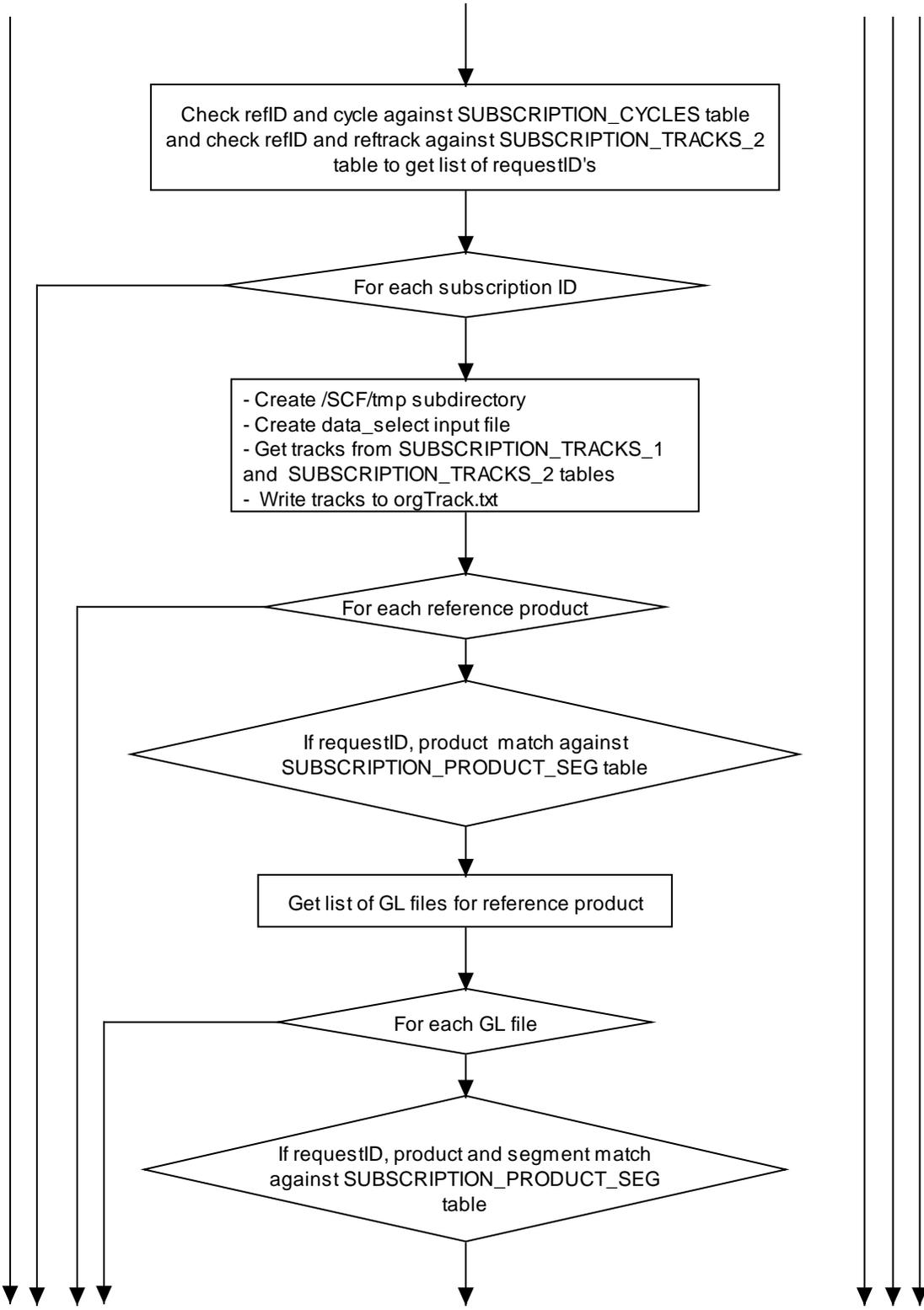


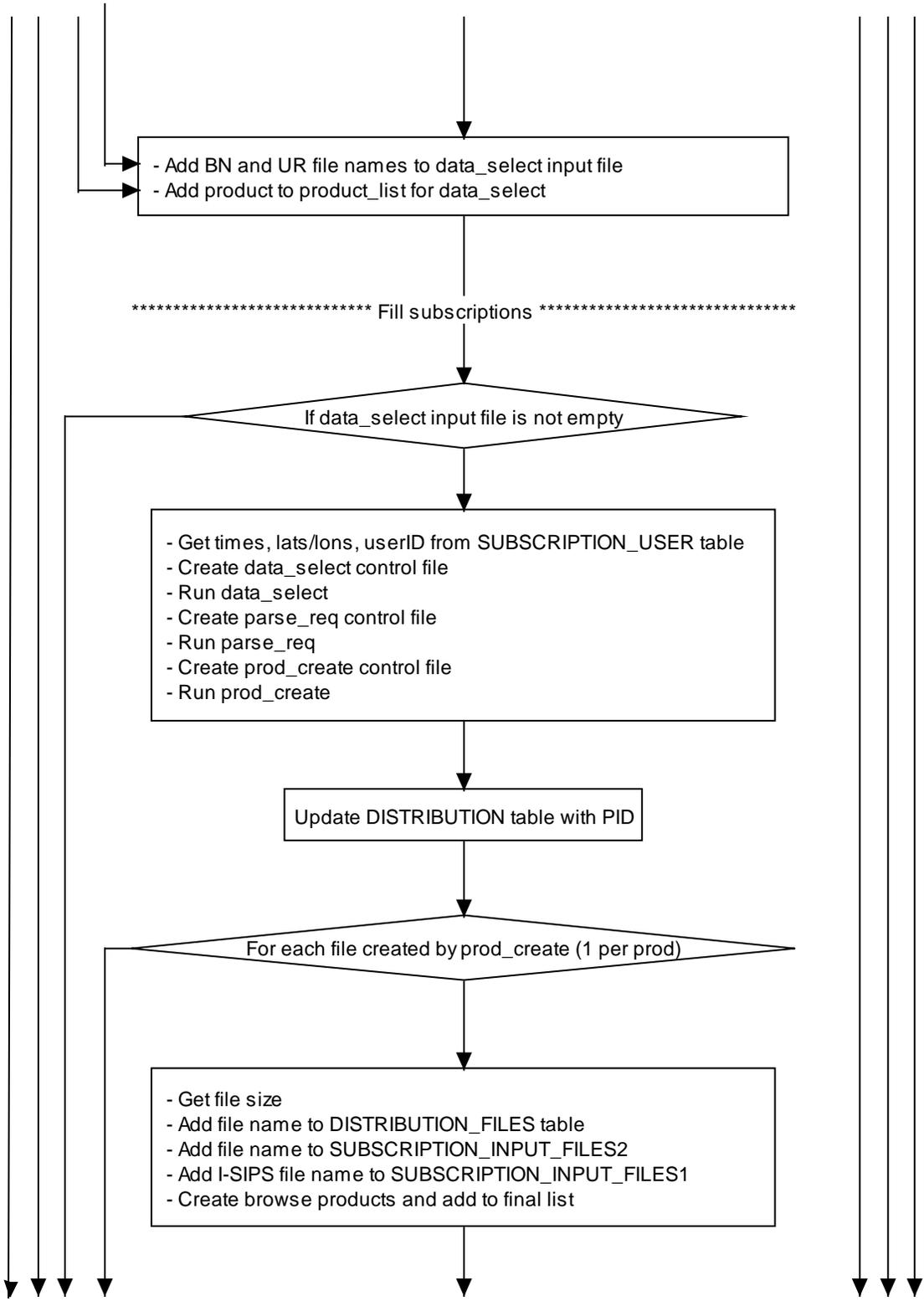
## B.4 Flowchart for process\_ea.tcl

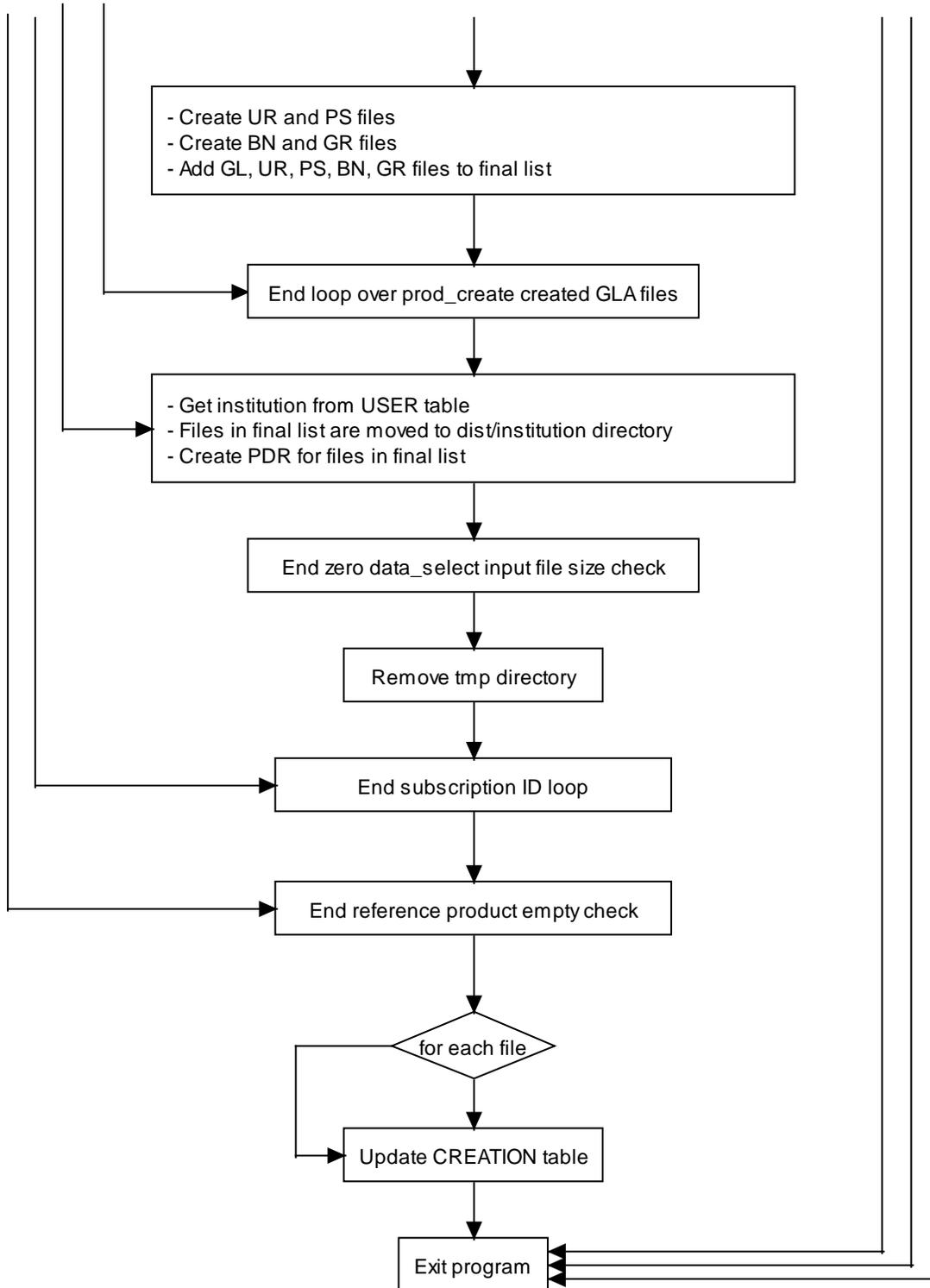


## B.5 Flowchart for process\_subs.tcl









## B.6 Flowchart for data\_select\_req.tcl

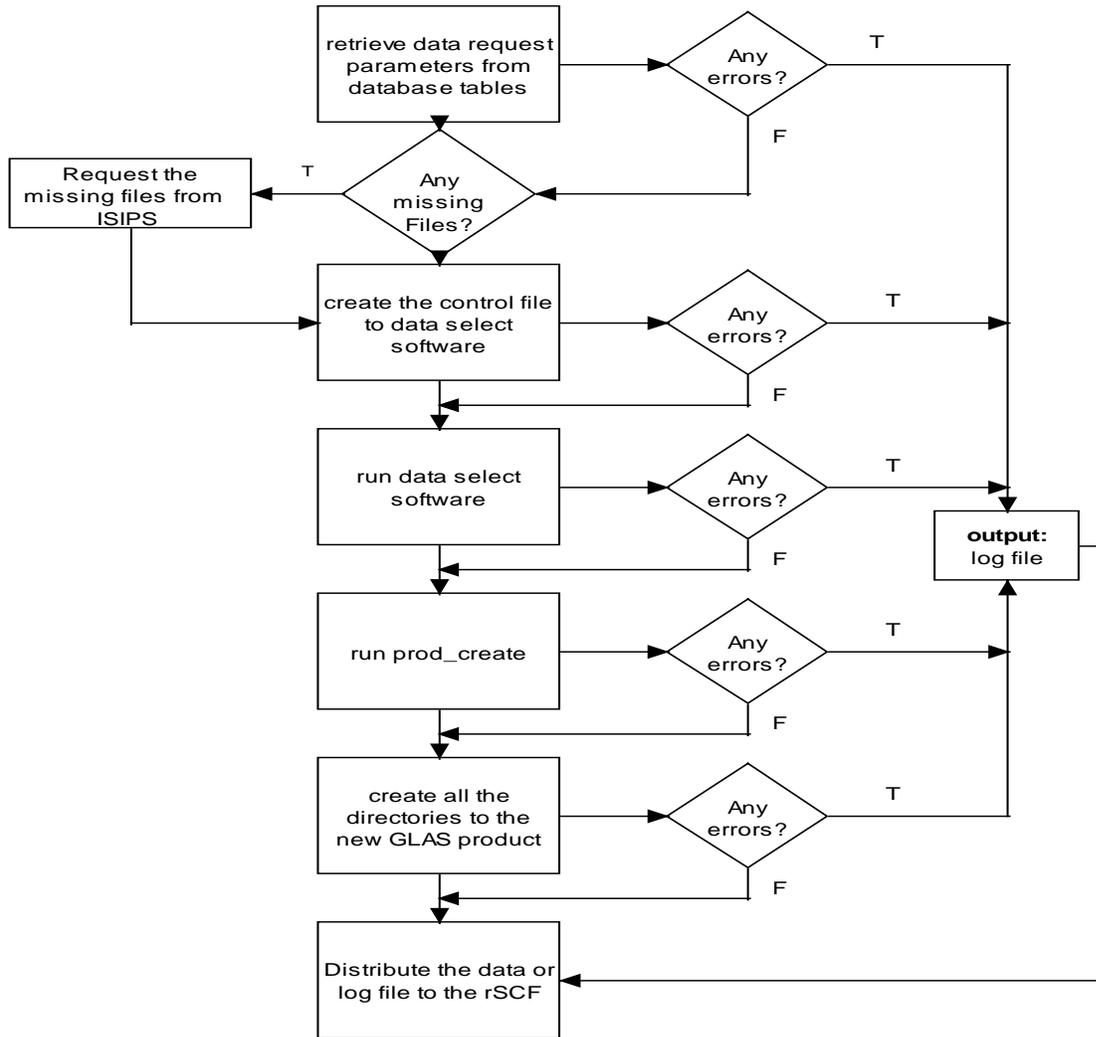


Fig 4-1

## Appendix C – File formats

### C.1 REQ File Format

Below is the format of the REQ file created by data\_select.f90. It is a direct access, binary file of record length 288 bytes. It contains information for creating or visualizing subsetting or supersetting products adhering to certain temporal and geographic criteria using data from the original I-SIPS product files. Each record contains the original product file name, starting and ending unique record numbers, corresponding starting and ending physical record numbers, pass ID, and mode for GLA01. There is one REQ file created per product type unless the file would exceed 2 GB; then it is separated into multiple files.

The first 3 records contain ASCII headers:

- First header is the record length.
- Second header is the number of headers.
- Third header is the start and end time of the data request, and the requested area longitude and latitude.

Following the headers, the data are in the following format:

Bytes	Type of variable	Description
1-255	Char*255	GLA product file name
256	Char*1	Spare byte so next integer will start on 4 byte boundary
257-260	I*4	Unique record index of first record
261-264	I*4	Unique record index of last consecutive record
265-268	I*4	Physical number of first record
269-272	I*4	Physical number of last consecutive record
273-283	Char*11	Pass ID (prkccccttt)
284	Char*1	Spare byte so next integer will start on 4 byte boundary
285-288	I*4	Mode for GLA01 only