SHARK

Technical Reference Manual

Prepared for:	ESA/ESRIN
Under contract:	9921/92/HGE-1
By:	David Warren, Howard Cambridge and Keith Muirhead Serco Space Limited Serco House Hayes Road Southall Middlesex, UK
Responsible Officers:	Giancarlo Pittella Alessandra Buongiorno, DPE/OT, Third Party Missions ESA/ESRIN via Galileo Galilei I-00044 Frascati Italy

Abstract

The SHARK Technical Reference Manual provides detailed information on the organisation and workings of the SHARK Software Environment and the hardware on which it runs.

SHARK is a standalone system that processes HRPT data transmitted from the TIROS-N/NOAA polar orbiting meteorological satellites.

The manual begins with a brief description of the basic hardware configuration required to run SHARK and continues with a step-by-step guide to installing the SHARK software, creating the necessary directory structure and setting up the shark user account.

The next three chapters contain a detailed description of the directory hierarchy and its contents, an explanation of the definitions and meaning of the environment variables used by the system and an outline of the function of the executable programs and scripts. Chapter 7 consists of outlines of the architecture of the more important SHARK tools. The final chapter of the manual describes the location and format of the various data-sets read and written by SHARK.

An appendix contains detailed information on the location, function, required input, and output of each executable program and shell script present in the SHARK Software Environment.

This manual should be read in conjunction with the "SHARK Users' Manual".

Document Status Sheet

Document Status Sheet

Document title: **SHARK Technical Reference Manual**

Issue	Version	Date	Reason for change
1	0	13-10-93	First Issue

Document Change Records made since last issue.

Table of Contents

Chapter 1 Introduction

1.1 Intended Readership

This manual provides a reference guide for those people who are responsible for setting up, maintaining and administering a SHARK system. It will also be useful for those responsible for operating SHARK who wish to understand how the software operates in more detail than is provided in the "SHARK Users' Manual".

1.2 Applicability Statement

This manual describes the facilities provided by version 3.2 of the SHARK Software Environment.

1.3 Purpose

SHARK is a complete system that reformats and archives data from the NOAA meteorological satellites. It automatically maintains a catalogue of the archived data and provides the means of retrieving archived data and distributing it to users.

The system reads raw HRPT data sets, calibrates and navigates the imagery contained in those data sets, and reformats them to generate SHARP products. It then archives the SHARP products to optical disk and automatically makes an appropriate entry in a catalogue of all archived data. It can also copy SHARP products to magnetic tape for distribution to users, either immediately before archiving or at some later time having first retrieved the required products from the optical disk archive.

1.4 How to use this Document

The manual begins with a description of the hardware components which make up a SHARK system.

A step-by-step guide to installing the SHARK software follows.

Chapters four, five, and six outline the various software components of a SHARK system. Chapter four is a guide to the contents of each subdirectory in SHARK's directory hierarchy and is followed, in chapter five, by a description of the environment variables used by SHARK. The executables and shell scripts which make up each of the SHARK tools are summarized in chapter six. Chapter seven continues with an outline of the stages involved in transforming raw HRPT data into SHARP products and a description of the SHARK tools used in each stage. The final chapter consists of a description of the SHARK data spaces.

Appendix 1 contains a description and notes on each executable program and shell script contained in SHARK.

1.5 Related Documents

A guide to using SHARK, "SHARK Users' Manual", is available from ESA/ESRIN. An altemative guide entitiled "Using SHARK version 3.2" is also available from the same source.

A description of the format of the raw HRPT data stream is contained in "NOAA Technical Memorandum 107", published by NOAA/NESDIS.

The SHARP format itself is described in two documents: "SHARP-1, Technical Specification of CCT Format", and "SHARP-2, Technical Specification of Format". There is also a user guide for Level 2 products, "SHARP Level 2 User Guide", that describes the geophysical quantities included in Level 2 products and the algorithms used to calculate them from the HRPT data. All three publications are available from ESA/ESRIN.

The SHARP format conforms to the Standard Family Tape Format specified by the Landsat Technical Working Group.

1.6 Conventions

Various textual conventions are used within this document to help clarify the procedures by identifying different types of object.

Unix file names and commands are printed as

filename

The names of windows, menus and buttons which make up the Graphical User Interface are printed as

widgetname

Text printed by a program on the screen is printed as

```
system output
```

The system prompt and required user input are printed as

```
[shark@nimbus](usr/tiros/EXEC)> user input
```

Descriptions of arguments in command lines are enclosed in angle brackets. So, for example, the command line to run *pass_info* might be represented as

```
pass_info <passage number>
```

to indicate that the program takes the number of the passage as its single argument. When typing a command the characters not included inside the angle brackets should appear exactly as printed in the text but the items within angle brackets should be replaced with whatever they

represent. This convention is also used to make clear how certain filenames are constructed.

Optional arguments in command lines are enclosed in brackets. For example, the command line for a program which takes one argument and up to two optional ones might be represented as,

```
<command> <arg 1> [ , <arg 2> [ , <arg 3> ] ]
```

Note that the commas are required in the command line.

A list of unspecified length is denoted by an ellipsis. For example,

```
<command> <argument 1> [, <argument> ...]
```

would indicate a command which accepts any number of arguments greater than one in a comma-separated list.

Unix conventions are used to denote filename patterns. A single character chosen from a set of characters is denoted by including the set within brackets, so that

```
[ab2]
```

represents one of the characters 'a', 'b', or '2'.
A set which consists of a range of characters can be denoted by a dash, so that a single lower case letter could be specified by

```
[a-z]
```

In addition the symbol '#' is used to represent an integer number of any length.

As an example, any of the following forms might be used to represent a set of filenames each consisting of the word TIROS followed by a two digit number less than 20:

```
TIROS<number>

TIROS[01][0-9]

TIROS#
```

The SHARK software is normally loaded into a directory hierarchy beneath the directory */usr/tiros*. In this manual pathnames are given relative to */usr/tiros* unless they begin with a slash '/' in which case they are absolute pathnames. So '*CODE/SRC*' refers to the directory */usr/tiros/CODE/SRC* while '*/home/remora*' refers to */home/remora*.

Chapter 2 Hardware

This chapter details the basic hardware configuration required to run a SHARK system. All the hardware components are standard; no modifications are required specifically for SHARK. They should be set up and connected according to the instructions supplied by the manufacturers.

2.1 Host Processor

A SHARK system's central hardware component is a Sun 4 workstation with at least 8 Mbytes of memory, although 32 Mbytes is more realistic. The system requires an 8-bit colour monitor and a large magnetic disk with a minimum capacity of 500 Mbytes.

SHARK runs under SunOS, Sun's version of the Unix operating system. Follow the manufacturers instructions to connect the workstation, disk, display, keyboard and mouse and install the operating system. This is all fairly straightforward.

Because some of SHARK's software tools are controlled from Graphical User Interfaces (GUIs) the workstation must also run SunView. Although Sun still support SunView applications OpenWindows is now the default windowing environment supplied as part of the Solaris environment.

The workstation must also have compilers for both C and FORTRAN. Sun's C compiler is no longer bundled with SunOS, it is an optional item that must be purchased separately. SHARK requires version 1.4 of Sun's FORTRAN compiler. The makefiles in each directory will have to be modified if a version later than 2.0.1 is used. The 'cg89' flag is not recognised by these more recent versions of the compiler.

2.2 Magnetic Tape Drive

SHARK will read data from and write it to magnetic tape if a drive is connected to the system. Two kinds of tape drive can be used; ½" reel-to-reel systems and 8mm cartridge, "Exabyte", systems. Both types of drive are likely to be connected to the workstation through the SCSI. Connect the drive in accordance with the instructions supplied with the machine.

A 2400 foot ½" tape recorded at 6250 bits per inch can hold a single HRPT passes or four SHARP scenes. By contrast an 8mm Exabyte cartridge can hold about 60 SHARP scenes. Apart from the differences in capacity both types of drive are treated in the same way by both Unix and SHARK.

No special software is required to interface to either type of drive.

2.3 Optical Disk

If a drive is available SHARK uses 12" optical disks to archive SHARP scenes. Two types of drive can be used; ATG GD 9001 or LMSI 1200.

Both types of drive are accessed through specialised software provided by Dorotech. The drive should be connected and the software installed according to the manufacturers instructions.

If a system has an optical disk drive and is to be used for archiving SHARP products it must also have access to a PostScript printer and a network connection. Each time SHARK archives a SHARP product it prints a "QuickLook" image of the scene and generates a catalogue entry which it sends to the central catalogue at ESRIN.

2.3.1 Switching on

Switch the unit on. The power switch is on the rear panel. There are four illuminated buttons on the panel at the front of the machine labelled 'START/STOP', 'WRITE PROTECT', 'CONTROL MODULE' and 'DEVICE ADDRESS'. Make sure all four buttons are not pressed in. The two green lights in the right-most two buttons ('CONTROL MODULE' and 'DEVICE ADDRESS') should remain on while the other two should be off.

2.3.2 Loading a disk

Optical disks have two sides only one of which is accessible at any one time. To read or write to the other side the disk has to be removed from the drive, turned over, and reloaded. When a disk is loaded the drive accesses its top side.

To load a disk open the front hatch by pressing the lip down. Slide a disk in making sure that the arrows on the top of the cartridge point towards the drive.

Close the hatch by pushing the lip upwards. If the disk side is write-protected, the orange light on the 'WRITE PROTECT' button will now light up.

Now place the unit on-line by pressing the 'START/STOP' button. The drive will click several times. The 'START/STOP' button should light up to indicate that the disk is ready for use. Sometimes the button will not light up until the disk is accessed.

The disk is now ready for use.

2.3.3 Unloading a disk

Take the unit off-line by pressing the 'START/STOP' button. The green light will go out. Open the load hatch by pressing the lip down and slide the disk out. Close the door to prevent dust getting in.

2.3.4 Write-enabling a disk

A write-protected disk can be write-enabled by rotating the white circular 'tab' on the cartridge so that the word 'WRITE' lines up with the arrow on body of the cartridge. You will need to insert a coin or screwdriver into the tab to turn it.

To prevent writing to the disk rotate the tab until the arrow points to the words 'WRITE PROTECT'.

2.4 Printer

If a suitable printer is available SHARK prints copies of the QuickLook from each SHARP scene that it archives. The printer must be capable of interpreting and printing PostScript files.

The "Transcript" software should be installed.

The printer's communication speed should be set to 9600 baud.

2.5 Network Connections

If the SHARK system is going to use an X.25 connection the software must be initialised each time the system is switched on. This can be done automatically by logging in as the user 'X.25'. Type 'x25' at the login prompt. The process automatically sets up the X.25 and X.29 software and then logs out.

If the initialisation is successful the process types the message 'packet level is up' just before it logs itself out. If it cannot make a connection it types the message 'CONNECTION TIMED OUT'.

If the initialisation fails check the physical connection from the workstation's serial port to the modem and that the host address entry in the configuration file, */etc/x25params*, matches the host address of the local DTE.

2.6 Powering Up SHARK

Switch on the workstation monitor, workstation and the peripherals. The workstation will go through its automatic boot-up procedure during which it will type a great many messages to the screen. Eventually the login prompt will appear indicating that the system is up and ready to run.

If necessary login as 'x25' to initialise the communications software. (See section 2.5).

Log in as 'shark' and enter the password. After a few seconds SunView will start up, the screen will turn grey and some icons will be positioned at the top of the screen. The system should now run.

Chapter 3 Software Installation

SHARK systems are normally distributed with the software already loaded and installed on the magnetic disk. There may be circumstances in which it becomes necessary to reinstall the software. This is a simple procedure but you must carry it out while you are logged in to a privileged user process.

The description which follows assumes that the installation is being done for the first time.

3.1 Preliminaries

Before you begin to load the software itself you should create the root of the directory hierarchy and set up a user account for SHARK processing.

3.1.1 Creating the File-system Root

The SHARK software is usually loaded under */usr/tiros*. If your system uses */home* as the user partition, install the software under */home/tiros* and make a symbolic link to it from */usr* with the following commands:

```
root% cd /usr
root% ln -s /home/tiros
```

In the following descriptions SHARK's root directory is assumed to be */usr/tiros*.

3.1.2 Creating the shark User Account

It is normal to create a user called 'shark' whose home directory is */usr/tiros/EXEC* and whose user id number is 99 (for tape copy compatibility). A typical entry in the */etc/passwd* file is:

```
shark::99:10:SHARK Operational Login:/usr/tiros/EXEC:/bin/csh
```

The script */usr/etc/install/add_user* creates a new user almost automatically given the name, user number and login directory. Read the Unix manual page for details of the arguments.

3.1.3 Disk Space Requirements

The software occupies about 80 Mbytes. This does not include SunView, the optical disk software, laser printer software, the raw data area or the processed data area.

3.2 Installation

The software is distributed as a single 'tar' file on 8 mm tape. It must first be unloaded from the tape and then a script, *install.shark*, must be executed to build the executables.

- Login as root
- Change directory to the partition in which the SHARK system will reside.

```
root% cd /usr
```

In the following SHARK is loaded into the */usr* partition. If the software is to be written into the */home* partition, a link from */usr/tiros* to */home/tiros* must already have been created before the tape is unloaded.

- Unload the tar file from the tape.

```
root% setenv TAPE /dev/nrst0
root% mt rew
root% tar xvbf 1024 $TAPE
```

In this example the tape drive is */dev/nrst0*. The unloading operation takes a little while. The program types out the name of each file it unloads.

- Make shark the owner of all the unloaded files.

```
root% chown -R shark tiros
```

The password file entry for user shark must already exist.

- Logout and login again as 'shark'.

```
root% logout
```

(Now login as shark...)

- Change directory to */usr/tiros*.

```
shark% cd /usr/tiros
```

- Run the installation script.

```
shark% install.shark
```

This script will ask you to provide information describing your setup:

```
>> Using Optical disks / Dorofile ? (y/n)
```

Obviously, only answer 'y' if the system has an optical disk

```
>> Using 1/2" magnetic tapes ? (y/n)
```

Answer 'y' to this if you plan to ingest raw HRPT data from ½" magnetic tapes or wish to read or write SHARP products using ½" magnetic tapes.

```
>> Will you be using Sunlink DNI to load images over DECnet ? (y/n)


>> Will you be using C-Kermit / fax transmission interface ? (y/n)


>> Install (n)ormal or (a)ntarctica QL program (n/a) ?
```

Choose the 'normal' option unless you will be processing data acquired at the Antarctic station. The source code for both versions will be loaded onto the system. This option simply determines which executable is built. If the system will be used to process data from the Antarctic station and another 'normal' station you can use the *avhrr_env* script to build the appropriate version each time *display* is started.

```
>> Enter HRPT record size - "
(D)undee (24576) or (E)SRIN (22528) (d/e): "
```

If your system will be processing HRPT data acquired at a station with a receiver built at Dundee choose 'D'. If the system is used to process HRPT data from a different station *install.shark* has to be run specifying the record size.

At this point the software build will begin. As each stage is completed a message is typed to the screen. When it reaches the 'general SHARP-formatting software' stage, building the executables in *CODE/SHARP*, the script will prompt you with the question:

```
>> Make clean (y/n) ?
```

If this is the first attempt to install, answer 'y'. If not, answer 'n'. This saves time if you are making a second attempt to build the system after it failed. If you answer 'y' all the SHARK executables and object files created during your first attempt will be deleted.

Finally when the software has been installed the script types a message reminding you to set the definitions of site specific environment variables in the shark user's *.login* file.

3.2.1 Error Recovery

While the install script runs it logs messages to a file named '*/tmp/sh_install.<PID>*'. <PID> represents the process identification number. It is added to the file name to create a unique filename. If, for any reason, the installation fails, the script prints out the full name of the log file. Examine this file and see if you can fix the problem.

Once the problem is fixed you can continue the installation at the point of failure by typing:

```
shark% install.shark -i
```

This allows you to choose the step in the installation at which to continue. The script types a menu:

```
Enter stage to continue from:

        a) General Support Software
        b) C/Fortran IO Interface Software
        c) Shark Parent Process
        d) Shark Tape-Handler Child Process
        e) Shark OD-Handler Child Process
        f) Shark QL-Handler Child Process
        g) QL-Generation Process
```

h) Monitor Process
i) Display Parent Process
j) Display SHARP-Handler Child Process
k) Display Tape-Handler Child Process
l) SHARP-Formatting Software
m) TOVS-Formatting Software
n) DECnet-Handling Software
o) Fax-Handling Software
p) CEOS-format Software
q) Making symbolic links to required files

>>

3.3 Data Spaces

Two distinct areas on magnetic disk are reserved to hold data waiting to be processed. One for raw data waiting to be reformatted into SHARP Level-1 products and another for SHARP products waiting to be further processed or archived.

3.3.1 Raw Data Area

The SHARK processing tools expect to read raw HRPT data from a reserved area on magnetic disk.

It takes a maximum of 16 minutes for one of the NOAA satellites to travel from horizon to horizon. During this time it transmits approximately 120 Mbytes of raw HRPT data. Space for data from a single pass is reserved as a Unix file or, on older systems, as a raw disk partition. Whichever is used the space is referred to as a passage.

The number of passages defined on any given system depends on the amount of available disk space. SHARK's software tools read data from one of the passages and generate SHARP products.

The file *RAWINFO/rawtable* contains a list of the names of the files or raw partitions that constitute each passage. For example,

```
/usr/big01
/dev/rxd0f
```

tells the software that there are two passages which use the file */usr/big01* and the raw partition */dev/rxd0f*. The rawtable file must be edited to reflect the organization of each system's disk.

The filenames in the *RAWINFO/rawtable* MUST be exactly ten characters long (e.g. */home/big1* or */usr/x/ff1)*. If the filenames cannot be shortened to ten characters, create symbolic links with ten-character names and insert the link names in *RAWINFO/rawtable*.

Up to nine passages can be made available to SHARK's tools at one time. If there is sufficient disk space to support more than nine passages they must be grouped into sets, each containing up to nine passages. A tool called *switch_set* defines which passage set is active.

3.3.2 Processed Data Area

When a SHARP product is created it is written to a directory referred to as a 'slot'. A single slot can contain just one SHARP product, which occupies approximately 34 Mbytes. Most systems have ten slots, although the software will work happily with fewer. The environment variable 'SLOTS_PARTITION' must be set to the name of the directory which contains the slots. It is usually set to '*/usr/tiros*'. Each slot is actually a subdirectory named $SLOTS_PARTITION/slot#, where '#' represents the slot number.

Normal SHARP processing uses slots one and above. A special slot, slot number 0, which must be created along with the others, is reserved for the *display* process. *display* writes '30-second' SHARP scenes into slot 0 while the image navigation is corrected.

Chapter 4 Directory Structure

The following table contains a brief description of the contents of each directory in the SHARK Software Environment. The presence of a directory in the table does not mean that it contains anything useful. Some directories contain older versions of code that are no longer used.

The directories are all subdirectories of */usr/tiros*. (See section 1.6, page 3).

Directory	Description
BIN	Location for executables and scripts.
BIN/OLD	Old versions of files in */BIN.*
BIN/SRC	C source for programs in */BIN.*
BIN/SRC/DB	Database program.
BIN/SRC/IP_MAIL	C source for *ip-mail.*
BIN/SRC/NOAA	C source for program that calls NOAA EBB and retrieves TBUS.
CODE	C source for *tiros* (*shark* executable).
CODE/CEOS/...	Code to generate output data for the 1-km project.
CODE/LEVEL2	FORTRAN source for generating SHARP 2 products. AVHRR count to radiance and count to brightness LUTS; AVHRR visible channel calibration data; threshold values for QuickLook classification, LUT for quantising SHARP level 2 products.
CODE/LEVEL2/QLOOK	FORTRAN source for generating classified QuickLook. Experimental version.
CODE/LEVEL2/METEOSAT	FORTRAN source to calibrate infra-red bands for comparison with Meteosat.
CODE/ODISC	C source for Optical Disk handler *odh.* Child process spawned by *display* and *shark* to take care of optical disk transfers.
CODE/ODISC/OLD	Optical disk handler code; old version.
CODE/ODISC/RODH	Remote optical disk handler.
CODE/QLOOK	C source for *ql_handler*, a child process spawned by *shark*, that calls *CODE/LEVEL2/level2_gen* to generate QuickLooks.

Directory	Description
CODE/QLOOK/FTN	FORTRAN source for generating QuickLooks. Special version for the Antarctic station. Rebuilt if local environment is changed to or from 'Antarctic'.
CODE/QLOOK/FTN/LIB	Library of C i/o routines that can be called from FORTRAN or C. Uses data pools for efficient disk transfer.
CODE/SHARP	C and FORTRAN source for SHARP code. Includes the *tir** programs*(HRPT ingestion)*, *tloc2aq* (calculates location data) and *shgrid** (SHARP lat/long grid generation).
CODE/SHARP/POOL_DEMO	FORTRAN and C source to demonstrate data pools.
CODE/SHARP/TOVS	FORTRAN source for extracting, calibrating and geolocating TOVS data.
CODE/SHARP/UTILS	FORTRAN and C source for SHARP utilities.
CODE/TAPE	C source for *tape* process (child process of shark). Reads site identification strings from *MISC/site_details.dat.*
DISPLAY	C source for *display.*
DISPLAY/SHARPHANDLER	C source for handling *sharp* child process.
DISPLAY/TAPEHANDLER	C source for handling *tape* child process.
DOCS	Assorted documentation. Handle with care; some of it may be out of date.
DOCS/CDB/... *DOCS/Frame_Templates* *DOCS/TOVS*	Documentation dated Nov '92. Incomplete, but reliable. *DOCS/CDB/print_all* gathers it all together and sends it to the printer.
EXEC	Working directory, many temporary files are written here. Contains many SHARK executables and data files. Also contains the coastline and state boundary data bases and TOVS calibration data files.
EXEC/DBP	Actual location of *dbp* binary. *EXEC/dbp* is really a link to *BIN/batch_menu*, a script which eventually executes *EXEC/DBP/dbp.*
EXEC/DBP.pre-ATG	Old version of *dbp.*
EXEC/pc	Used for DOS emulation.
FAX	Temporary location for files to be held while waiting to be faxed.

Directory	Description
ICONS	SHARK icons, thumbnail version of QuickLook, generated for each SHARP product and displayed by *shark* to aid identification.
LOGS	Log files produced by SHARK programs.
MISC	Two important files in here. *site_details.dat* contains a 4-character site name, 2-character site id, 1-character archiving site id, and site number (for LEDA catalogue). The same format is used in *PRODEF.DOC* and byte 148 of VDF file. *ATG_OD_LABELS* contains label of next optical disk to be formatted.
MONITOR	C source and binary for *monitor*.
NETWORK	Network transfer utilities.
NETWORK/AUTOXCAT	C source and binary for *sendcat*, *sendcat.ftp* and *findcat*.
NETWORK/FTR	Quick installation of X.25.
PRODEFS	One compressed *PRODEF.DOC* file for each site containing data to be copied to SHARP header.
RAWINFO	Space for headers for ingested HRPT data and a table, *rawtable*, which points to the files or raw partitions which contain the HRPT data.
RAWINFO/BATCH	Working directory for *dbp*. Requests to generate Level 1 products are placed in this directory from where *dbp* will pick them up and process them.
RAWINFO/header1 *RAWINFO/header2* *RAWINFO/header3* *RAWINFO/header4* *RAWINFO/header5* *RAWINFO/header6* *RAWINFO/header7* *RAWINFO/header8* *RAWINFO/header9*	If a passage is not empty then the corresponding header directory contains four files: *.id*, passage identifier *AQTIM.DAT*, passage start and stop times, and number of scan-lines, *XHEADER.SCR*, header information and geolocation data, *tbus.dat*, TBUS data.
SITES *SITES/MERGE* *SITES/antarctica* *SITES/cairo/...* *SITES/dlr/...* *SITES/dundee* *SITES/frascati* *SITES/ispra* *SITES/maspalomas/...* *SITES/tromsoe/...*	Site-specific code. One subdirectory for each site. Contains variants of code made to solve problems particular to each site. Duplicates parts of main directory tree.

Directory	Description
slot 0	Special slot used for creating small SHARP scenes while image navigation is corrected.
slot#	SHARP product directories. Each directory corresponds to a slot. There can be up to ten slots.
TBUS	TBUS data base. One file per satellite per day. Named N<nn><yy><mm><dd>. <nn> is the satellite's mission number and <yy><mm><dd> the date for which the TBUS data is valid.
ql_trans/...	Programs and source for preparing QuickLooks for transmission by fax.

Chapter 5 Environment Variables

The SHARK system refers to several Unix environment variables. These can be tailored to let the system run in various environments. They are usually defined in the shark user's *.login* file, but can be reset from the command prompt using the command *setenv*.

In the table the variables are grouped according to function: directory definitions first then SHARK processing parameters followed by variables that relate to the magnetic tape drive, the optical disk drive and the printer.

Variable name	Default value	Function
EXEC_DIR	*/home/tiros/EXEC*	Working directory for shark user. Contains coastline database and many executables.
SLOTS_PARTITION	*/home/tiros*	Directory that contains the SLOTS directories.
LEVEL2_DIR	*/home/tiros/code/LEVEL2*	Directory containing code for producing SHARP level 2 products.
HRPT_BLOCK_SIZE		Size of HRPT record in bytes 22528 for ESRIN format, 24576 for Dundee.
TOVS_ON	YES/NO	Produce TOVS data.
SHARK_QL_COPIES		Number of copies of archived QuickLooks to be printed.
TAPE	*/dev/nrst0*	Tape device name as used by *tar* and *mt*.
REMOTE_TAPE_HOST		Name of host to which remote tape device is attached.
TAPE_ORDER_CONT		Number of next scene to be written.
TAPE_ORDER_TOTAL		Number of scenes in complete order.

Variable name	Default value	Function
OHOME		Device name for optical disk.
OHOME1		Device name for optical disk.
OHOME_MNT		Mount point for optical disk.
OHOME1_MNT		Mount point for optical disk.
REMOTE_OD_HOST		Name of host to which remote optical disk is attached.
FORMAT_ODS	YES/NO	Allow host to format optical disk - set YES on only one machine in any installation.
OPID	" " (10 spaces)	Used by Dorofile software.
OPTIC_CHAR	"\!"	Used by Dorofile software.
PRINTER		Logical name for PostScript printer.

Chapter 6 SHARK Executables

Most of the executable and shell scripts that make up the SHARK Software Environment are stored in the directory */usr/tiros/BIN*. They are listed here with a brief description of their function. Further details of the environment variables they reference, their command line arguments, and their input and output files are given in Appendix I.

File	Function
avhrr_env	Sets up processing environment.
batch_menu	Front end for *dbp*.
checkcats	Examine latest SHARP and AVHRR catalogue record. Update log file entries.(Only if a system has access to catalogue MicroVax).
checkimage	Simple facility for looking at SHARP or raw image files.
clp	Clears a single passage.
clps	Clears all passages.
cls	Clears a single slot.
clss	Clears all slots.
create_id_file	Makes a *.id* file for a passage.
d	ASCII dump of all ASCII files in a directory.
disp_sharp	Create SHARP (and TOVS) from a passage.
disp_tirget	Download HRPT data from tape to a passage. Manager script for *dbtirget* when called from *display*.
exabyte_to_slot	Download a SHARP product (internal format) from magnetic tape to slot.
find_tbus	Finds nearest TBUS to specified date for specified satellite.
geodisp	Displays a single band at a time from a four band data file containing 480 lines each of 2048 bytes.
getcats	Read all LEDA catalogue entries from 12" optical disks.
load_slots	Download multiple (La Reunion) SHARP products from magnetic tape to slots.
makehdr	Builds a passages header, *XHEADER.SCR* and *.id*, from *AQTIM.DAT* and TBUS files.
odh.remote	Lets *shark* use a remote optical disk drive.
OD_ops	Menu driven optical disc utility.
pass_info	Examines HRPT header and data.
print_OD_ql	Send selected SHARP QuickLooks from optical disk to laser printer.

File	Function
project.silent	Quiet version of *project*. Called by *shark*.
projlan	Transverse Mercator projection program for NOAA AVHRR images (land).
projsea	Transverse Mercator projection program for NOAA AVHRR images (sea).
raw_backup	Manages HRPT data (passages) backups to magnetic tape (Used at Cairo DRC).
repair	Corrects faulty tie point flags in SHARP products.
repair_sharp	Called by repair.
rgrep	Recursive *grep*.
sendcat.ftp	Transfers groundstation catalogue file to EPOCAT.
shark	Front end for *tiros*.
sharp_exabyte.ops	Exabyte handling for SHARP products.
slot_in_window	Finds if centre of image is within area of interest.
slot_to_exabyte	Uploads a SHARP product (internal format) from magnetic disk to tape.
switch_header	Switches between sets of passages. Old version.
switch_set	Switches between sets of passages.
tape.remote	Run on remote host to let *shark* to use remote tape drive.
tbus_send	Sends latest TBUS to Internet for relay to Antarctica.
tbus_tape_archive	Copies entire TBUS archive to tape.
tirant	Read Antarctic HRPT files from tape to passage.
tirant.NEW	Read Antarctic HRPT files from tape to passage.
tireros.csh	Read EROS HRPT files from tape to passage.
tirnia	Read Nairobi HRPT files from tape to passage.
tirph.csh	Read Philipines HRPT files from tape to passage.
tirreun	Read La Reunion HRPT files from tape to passage.
tirsa	Read South Africa HRPT files from tape to passage.
tirscanz	Read Scanzano HRPT files from tape to passage.
tirtnb	Read Antarctic HRPT files from tape to passage.
upload	Sun-3 fix to upload SHARP products to magnetic disk.
x25_send	Interactive or batch file transfer facility.
xtree	Outputs listing for a directory and all its subdirectories.

Chapter 7 SHARP Processing

7.1 Ingestion

Once it has been received by a ground station the HRPT data stream is stored in some locally defined format. Ground stations define their own formats to suit local requirements. This means that there is no single format for ingesting data. As a result, several, very similar, post-acquisition, ingestion programs have been developed to rewrite the HRPT-like data into a common format that can be read by SHARK.

There are two stages to HRPT ingestion. The first is to transfer the HRPT data stream to a 'passage', a reserved area on the system's magnetic disk. The second is to generate the header files that describe the data set to the SHARK system.

7.1.1 Copying HRPT Data to a Passage

Copying the raw HRPT data stream to a passage is performed by one of three sets of software tools depending on whether the data is being read directly from the satellite receiving station, from magnetic tape, or over a network connection.

7.1.1.1 Real-time Ingestion

Some stations have their own real time acquisition capability, eg. Tromsoe, La Reunion. These systems transfer HRPT data from the acquisition system to SHARK by magnetic tape or a network connection. Other stations use customised software systems which interface directly to the SHARK Software Environment. These are the DLR 'albedo' and Dundee 'rec' systems.

7.1.1.2 Magnetic Tape

The term 'magnetic tape' includes both ½" CCT (computer compatible tape) and 8 mm 'Exabyte' tapes. Exabyte tapes are now becoming more common. HDDT systems will not be discussed here as they do not constitute a part of standard SHARK operations.

7.1.1.3 Network

Some sites ingest their data in real time on an acquisition system and then pass it directly to a SHARK system via an Ethernet LAN. For example:

CAIRO:	uses two Suns and a TCP/IP LAN connection.
NAIROBI:	plans to implement a DECnet link between the acquisition VAX/VMS host at Nairobi Met. Office and a SHARK system running DECnet emulation software at the RCMRSS using a microwave link to span the 20-odd mile distance.
ANTARCTICA:	acquires on a MicroVAX and HRPT data is subsequently passed to the Sun SHARK system using DECnet protocol.

Whether the data is transferred to the SSE by magnetic tape or over a LAN the raw HRPT data is converted from the local groundstation format into the SHARK internal format and written into the passage by one of a family of programs known collectively as the *tir** programs. Each of these is tailored to the requirements of a single groundstation format.

7.1.2 Header Generation

The second stage of HRPT data ingestion is the creation of the passage header. This begins with the creation of the file *AQTIM.DAT* in the passage header. The *tir** program generates the file at the same time as it creates the HRPT data file. *AQTIM.DAT* contains the satellite identification, the passage's start and stop times and the number of scan-lines in the passage.

Before the SSE can understand the raw data, it expects two files to exist in the header directory; the *XHEADER.SCR* file and the *.id* file. These are both generated using the *AQTIM.DAT* file left by the previous step.

The *XHEADER.SCR* contains all critical parameters relating to the pass including the start and stop dates and times, a flag to indicate whether the satellite crossed the equator from south to north (ascending) or from north to south (descending), and numerous fields which facilitate the pass navigation in later processing stages. The *XHEADER.SCR* file is created from the *AQTIM.DAT* file and the closest TBUS data by a program called *bach* that is executed by a script called *makehdr.*

XHEADER.SCR consists of 12 records of 512 bytes each. The first record contains the information held in *AQTIM.DAT*. The second contains the catalogue entry. The third record contains the mean orbital elements and the satellite's attitude. The fourth holds the original data from which the data in the third record were derived. This is usually the TBUS message but other formats are dealt with. The remaining eight records hold a single character for each scan line in the scene that denotes whether the time code for that scan is reliable or not.

The *.id* file is a small status file which must be present in a passage header to indicate that there is data in the passage. It contains a single line of ASCII text which identifies the satellite which created the image and the date and time at which the acquisition began. The text string is modified while the data is processed. The text strings from all existing *.id* files are displayed by the *monitor* tool. The most common use of this file is to display a meaningful identifier string describing the pass. The identifier string takes the form:

```
<sat>       <date>       <start time>
N11         120192       142212
```

Which identifies the scene as one acquired by NOAA-11 on 12th January 1992 beginning at 14:22:12 GMT.

All three fields are generated by *makehdr* (or sometimes a customisation of it, depending on the site's characteristics) which is always called immediately prior to ingestion of HRPT and the subsequent creation of the *AQTIM.DAT* file.

At some sites, *makehdr* may itself be bundled into another script which is run after ingestion and performs some other action as well, (For

example, in Cairo the operators run a script called *xfer* which builds a header for the newly acquired/ingested pass and then copies the header and the HRPT file to the processing SHARK system over an Ethernet connection). Once a header is complete standard SHARK operations can begin.

7.2 display

Once the raw data has been written into a passage, the next step in the operation is to define the four minute sections of the data which will be processed into SHARP products. The *display* subsystem provides an interactive means of selecting the scenes. Its main purpose is to generate the SHARP request file, *SHARPREQ.DOC*, that serves as input to the SHARP processing chain.

The *SHARPREQ.DOC* file consists of a single 80 byte record of the form:

```
<sat>, <date>, <start time>, <stop time>,
                    [<start time>, <stop time>...]
```

where <sat> is the satellite identifier, <date> is the date of the acquisition, and <start time> and <stop time> are scan-line times, measured as milliseconds since midnight, taken from the HRPT telemetry which together define a four-minute scene. The record is space-padded so as always to be 80 bytes long. The SHARP request file is written into */usr/tiros/EXEC* so only one can exist at any one time. Consequently only one instance of the SHARP processing chain can exist at a time.

7.2.1 AVHRR Environment Setup

As *display* is a pre-processor for SHARP processing, the correct AVHRR environment must be set up before it runs. One of the support files of the SHARP processing chain is the Product Definition file, *PRODEF.DOC*, a template for any SHARP product. It is a large ASCII file, located in *EXEC*, which describes the length and type of each record in a product. It also contains some ASCII text strings which uniquely identify the product's source. These strings are used to generate various identifiers, such as the two letter acquisition-station code and hence the name which is printed on QuickLooks.

EXEC/PRODEF.DOC may have to be tailored for a new site. The most important fields are 'Generating country' (entry 25, line 43), 'Generating agency' (entry 26, line 44) and 'Generating facility' (entry 27, line 45). These must be modified to reflect the local site. Each field is preceded by a FORTRAN-style format descriptor which defines the type of data and its length. (For example "A12" means that the next field takes up to 12 characters). Some of these fields may be repeated later on, so changes should be made globally, throughout the file.

The file */usr/tiros/MISC/site_details.dat* should be modified at the same time as *PRODEF.DOC*. It consists of a list of colon separated fields. The first field holds the site name. It must match the 'Generating facility' of *PRODEF.DOC*. It is followed by a unique 2-letter code to show where the data were acquired (e.g. MP for Maspalomas). In most cases the

data will be acquired and processed at the same site. If the site is to be an archiving station the third field must be a unique single-letter code to identify it. If not, the field contains a hyphen. The fourth field is an integer that identifies each site. This must be assigned by ESRIN.

Normally, a station needs to install its product definition file once only, on initial installation. But, there has to be a convenient mechanism for changing the AVHRR environment at any station that processes data from more than one acquisition station. For example, the Frascati site processes and archives data from several different acquisition stations (at present Nairobi, Niamey, Antarctica, S.Africa and Cairo). This is managed by a script called *avhrr_env* which is invoked automatically when *display* starts up. It types out the name of the current environment (eg. MASPALOMAS) and asks the operator whether or not to change it. If 'yes' is selected, the script types a menu of stations and can select another environment. Normally the other options will be disabled. The corresponding *PRODEF.DOC* file (stored as a compressed file in the PRODEFS directory) is then installed in *EXEC* as the current version.

The script then installs the appropriate QuickLook generation program for the selected site. Because ground stations can be as far apart as the Pole and the equator, it means that there is no single general-purpose classification algorithm currently in use. In fact, there are two - one for polar regions (Antarctica) and one for the rest of the world. So unless the site chosen is Antarctica, the latter is installed.

The script's final sction is to run *display* which presents the user with a GUI through which a single HRPT passage can be viewed, zoomed, contrast stretched, partially navigated and split into SHARP scenes. The selected SHARP scenes can be queued for batch processing or processed immediately. A full description of how to use *display* is contained in the "SHARK User's Guide".

7.3 shark

The process of transforming raw HRPT data into a SHARP Level 1 product is carried out by four FORTRAN programs, controlled by a shell script called *disp_sharp*.

The inputs to the SHARP processing chain are the HRPT data file and the SHARP request file. The chain outputs between one and four SHARP Level 1 volumes from each HRPT file which are written to slots on the magnetic disk.

7.3.1 SHARP Processing Chain

The first step is known as the header dump. It is carried out by a program *scrheadaq* that reads the header file, *EXEC/XHEADER.SCR* and types certain useful feedback to the workstation's screen.

Next the earth location program *tloc2aq* reads the orbital information and the passage start and stop times from the scratch header file, *EXEC/XHEADER.SCR* and uses that data to navigate the HRPT data. It creates a file containing the latitude and longitude of every 32nd pixel on every 16th scan of the pass. It also calculates the azimuth and elevation of both the sun and the satellite as viewed from the same set

of points. The grid of tie points is generated according to the specification contained in the product definition file *EXEC/PRODEF.DOC*. The tie points are written to a file named *EXEC/LOCATE.DAT* which forms part of the final SHARP product and is read by the third program in the processing chain to calculate the map overlays.

The third step generates the coastline, state boundary and latitude/longitude grid overlays. The program is called *shgridsaq*. Worldwide datasets of coastlines and state boundaries derived from the WDBII are contained in the files *EXEC/CSTLATU.ISS* and *EXEC/CSTLATH.ISS* respectively. *shgridsaq* reads the information in *EXEC/XHEADER.SCR* and *EXEC/LOCATE.DAT* and generates three bitmaps, the same size as the HRPT passage and writes them to the files *EXEC/COASTLINE.DAT*, *EXEC/BOUNDARY.DAT* and *EXEC/LATLONG.DAT*.

The last step is to build the SHARP products and write them into the available slots. The program *wrsharpaq* reads the SHARP request file, the HRPT data file, the location file, the three overlays and the product definition file. These file are all in the EXEC directory. Up to four SHARP can be generated per passage, so an equivalent number of slots must be available for this output when *wrsharpaq* runs. If they are not, the program will fail.

Having read the data, and done an enormous amount of reformatting, *wrsharpaq* writes the SHARP files into the slot. There are four files in a standard SHARP product; the 'Volume Directory' file, 'Leader' file, 'Image' file and 'Trailer' file. The files contain the data, a complete description of its nature and origin and a description of the way it is organised within the Image file. Each file contains, in its first record, a description of its own contents. The files' formats are described in the document "SHARP-1, Technical Specification of CCT Format".

The Volume Directory File, *slot#/VDF*, consists of five records of 360 bytes each. It contains information about the contents of each of the other files in the SHARP volume.

The Leader File, *slot#/LEADER*, contains five records of 1800 bytes each. It includes information about the satellite, its orbit, the imaging instrument and the map projection of the image. It also contains an explanation of how to interpret the geolocation information provided for the image. The details of the satellite orbit are included in two forms, the original TBUS message, and the mean orbital elements and state vectors which are derived from the TBUS information by *tloc2aq*.

The image data is contained in the file *slot#/IMAGE*. The AVHRR image data values are written into the lowest 10 bits of the 16-bit words and the three overlays are added in the unused most-significant bits. The file also contains the earth location data, and the sun and satellite angles at the tie points.

The Trailer File contains the histograms for each of the five AVHRR bands.

7.3.2 Process Locking

While the chain is in operation, the presence of a lock file (*.sharp_lock*) in the passage header will cause *monitor* to display the legend "Locked for SHARP processing"; when it is over, the lock file will be deleted and the usual passage (as found in *.id*) will revert.

A file in EXEC called *sharp_lockfile* will also be maintained for the duration of the processing whose presence there indicates to other processes that the resources are in use.

As *wrsharpaq* prepares for output into slots, it calls a C function called *getslot()* which grabs the lowest available slot for its' use. *getslot()* claims the slot by creating a *.id* file in it containing the message 'locked'. Once the product is completely built in the slot, this message will be replaced by the normal product identification string.

7.3.3 QuickLook For Each Product

When the four SHARP files have been written into a slot, *wrsharpaq* forks a background process called *ql* that creates the QuickLook for the scene. The QuickLook is composed of two files, *QUICKLOOK.DAT* and *QUICKLOOK.PIC*. The first contains the classification summary data; the number of pixels assigned to each of the five classes, and the second contains the QuickLook image.

7.3.4 TOVS

The SHARK system is capable of extracting, calibrating and writing the TOVS data into a SHARP scene. TOVS, the TIROS Operational Vertical Sounder, consists of three multi-channel instruments which are sensitive in the visible, infra-red and microwave parts of the spectrum and which measure temperature and humidity profiles through the depth of the atmosphere.

disp_sharp will extract and generate the TOVS product at the same time as SHARP if the environment variable 'TOVS_ON' (set in *.login* or at the command line prior to processing) is set to 'YES'. The default value is 'NO'.

The TOVS processing chain is organised in a similar way to the SHARP chain. The FORTRAN programs are executed in sequence and their statuses checked for errors by *disp_sharp*. TOVS products are written to slots with their SHARP counterparts.

When the SHARP, QuickLook and TOVS file have been written to the slot *disp_sharp* exits with an appropriate message and status code.

7.4 dbp

Rather than generate SHARP products directly from *display*, it is possible, and usually more convenient, to use a background process, *dbp*, to create the products that have been defined in the interactive process.

Several HRPT passages can be queued for processing by *dbp* at the same time. To prevent the process running out of empty slots in which to write the SHARP scenes it can also archive the scenes it creates to

optical disk or copy them to magnetic tape. This is useful because there are only a limited number of slots on the magnetic disk and each passage can generate as many as four SHARP scenes.

It is also possible to instruct *dbp* to copy selected scenes to magnetic tape. The selection is based on the geographical location of the scene. A polygonal 'area of interest' is defined by writing the latitude and longitude of the corners of the polygon into a file, *EXEC/CURRENT_WINDOW.DAT*. *dbp* will copy a newly created SHARP scene to magnetic tape if the scene's centre is within the current area of interest.

The exact way in which dbp operates is determined by the name used to run the program. There are four links to *dbp* named *dbp.SHARP_only*, *dbp.OD*, *dbp.exabyte* and *dbp.OD+exabyte*. The first writes the new SHARP product into the first available slot. The second writes the product to the slot then archives it to optical disk and clears the slot. The third copies scenes whose centres fall within the 'area of interest' to magnetic tape and clears the slot. The fourth archives the scene to optical disk and copies it to tape if its centre is within the area of interest.

dbp begins by searching through the slot directories for subdirectories *slot#/BATCH*. These are created by *display* when a SHARP scene has been defined and queued for processing later. Using the creation time of the *BATCH* subdirectories, *dbp* orders the requests and processes them in the order they were requested.

Each *BATCH* subdirectory contains two files, the SHARP request file that contains the definition of the SHARP scene, and a file that contains the 'raw data quality indicator' for the scene. As it processes each HRPT passage *dbp* copies the SHARP request file into *EXEC* and then calls *disp_sharp* to initiate the SHARP processing chain.

dbp creates all of the SHARP scenes from a single passage before archiving to optical disk or copying to tape. This means that there should be at least four slots free before it runs. As the program writes each scene to magnetic disk it stores the slot number so that it knows which slots to copy to optical disk or magnetic tape.

If *dbp* was called as '*dbp.OD*' or '*dbp.OD+exabyte*' it archives the generated SHARP scenes once it has finished processing each passage. It calls *odh* to check the available space on the disk and if there is room, calls *tiros* to archive the scene, print the QuickLook and update the catalogue. It then updates the station log, *LOGS/station.log*, and, if the original call was to '*dbp.exabyte*' or '*dbp.OD+exabyte*', copies the scene to tape. Unless the original call was to '*dbp.SHARP_only*' it then clears the slot. Finally, once the passage has been processed and all the SHARP scenes generated and copied as required *dbp* deletes the *BATCH* subdirectory, effectively removing the batch request from the queue and freeing the passage for other processing.

Chapter 8 Data

8.1 HRPT Passages

The definition of HRPT passages is described in section 3.3.1. Each passage has associated with it a header directory which is used to hold all the data relating to a passage apart from the HRPT data itself. All the header directories are contained in the directory RAWINFO. If a passage has data the header directory contains three files, the scene identifier, *.id*, the acquisition file, *AQTIM.DAT* and the TBUS file, *tbus.dat*. In addition a subdirectory, *BATCH*, is created when the passage is queued for processing to Level 1 by *dbp*. Once the passage has been processed the *BATCH* directory is deleted.

8.2 SHARP Slots

The standard SHARP format, as defined in 'SHARP 1, Technical Specification of CCT Format', consists of a single logical volume containing four files;

VDF	five records of 360 bytes
LEADER	five records of 1,800 bytes
IMAGE	up to 1,441 records of 22,680 bytes
TRAILER	6 records of 4,140 bytes.

These four files, sometimes referred to as the 'core' SHARP files, and their contents are defined in the "Standard Family Tape Format Specification" and the "SHARP-1, Technical Specification of CCT Format".

SHARK writes four additional files into SHARP slots and archives them to optical disk. The files are the scene identifier, the QuickLook and the CAT (catalogue) record:

.id	ASCII scene identifier.
QUICKLOOK.DAT	ASCII QuickLook classification result.
QUICKLOOK.PIC	480 records of 1,024 bytes.
CAT	1 record of 101 bytes.

The extra files are archived to speed up the process of retrieving a SHARP scene from optical disk. When a standard SHARP product is downloaded into a SHARK system it must be read in using the *download* function of the *shark* GUI. This reads the four 'core' files into the first available slot, and then has to generate the *.id* file and call *ql* to generate the QuickLook files. It is quicker to store the QuickLook and scene identification on the disk and it does not significantly increase the space occupied by a single scene.

8.3 QuickLooks

Each SHARP product includes a 'QuickLook' image which is intended to give a guide to the geographical area covered by a particular scene and the amount of cloud cover. QuickLooks are single band, 16 bit images consisting of 480 lines of 512 pixels each. They cover the entire length of the original image but only the middle 75% of the swath. The outermost parts of each scan are distorted because of the AVHRR's viewing geometry and are, consequently, excluded from the QuickLooks.

Each QuickLook contains a classification of the original scene into land, sea, cloud, snow/ice and sun-glint. There is also an unclassified category. The classified image can be printed using only four grey levels so that it can be faxed without suffering significant degradation.

The top byte of each pixel identifies the class to which it belongs. The classes are represented by the following values: cloud - 255, ice/snow - 204, land - 153, sunglint - 102, water - 51, unclassified - 0. The lower byte contains three one-bit overlays, which mark coastlines, state boundaries and latitude/longitude grids. The remaining five bits contain an "intensity" value that represents either temperature or NDVI depending on class to which the pixel belongs.

QuickLooks generated from data acquired at night are necessarily slightly different. Instead of the classification the QuickLook shows the brightness temperature obtained from channel 4.

QuickLooks are generated from SHARP Level 1 scenes by a program *EXEC/ql*, which is usually called from either *display* or *shark* through the *ql_handler* process. It generates two files; *QUICKLOOK.PIC*, the classified image, and *QUICKLOOK.DAT*, the classification summary data (the number of pixels in each class), and writes them into the SHARP product's slot.

Although they are not part of the standard SHARP product QuickLooks are archived to optical disk to speed up searching through the archive. *ql* can also generate "full resolution" QuickLooks, which are not subsampled but which cover only a small part of the SHARP scene. These are written to files named *FQUICKLOOK.PIC* and *FQUICKLOOK.DAT* in the SHARP product's slot. Different names are used to ensure that the full resolution QuickLooks are not archived.

Versions of QuickLooks suitable for transmission by FAX are generated by *ql4*. They have only four grey levels; black, white and two shades of grey, and are written into the directory */usr/tiros/fax* as *QUICKLOOK.PIC* and *QUICKLOOK.ASC* to wait to be transmitted. The two files are alsoa copied into the quick look files in the product's slot.

8.4 Auxiliary Data

8.4.1 Satellite orbit parameters (TBUS)

In order to calculate the position of the tie points in a SHARP scene the SHARK software has to calculate the position and attitude of the satellite at the time each tie point was scanned. Some of the necessary information, the time each scan line started and the satellite's roll, pitch, and yaw angles, is included in the telemetry header at the start of each

HRPT scan-line. Information on the position of the satellite has to be calculated from a knowledge of the satellite's orbit. The orbit changes and cannot easily be predicted over long periods with the accuracy needed. The observed mean orbital elements (MOE) of each satellite are distributed by NOAA on an electronic bulletin board (EBB) in a message format known as TBUS. The TBUS messages are updated every day.

SHARK systems that can connect to OMNET are able to log in to the Bulletin Board and transfer these elements once a day. This is done automatically by a program, *get_periodic_tbus*, that runs in a background process started off by the *cron* job-scheduler. (See the tool description in Appendix 1 for *get_periodic_tbus* and *archive_tbus*).

The retrieved TBUS files are written into the */usr/tiros/TBUS* directory. Each file contains the TBUS message relevant to one particular satellite on a single day. The file names are constructed from elements representing the satellite mission number and the date of the TBUS message. For example,

n<sat><yymmdd>.tbu,
n11910523.tbu (NOAA 11, 23 May 1991)

where <sat> represents the mission number and <yymmdd> the date.

Stations without any network connections can be supplied with up-to-date TBUS information by fax from ESRIN. The contents of the message have to be retyped into the system. To reduce the likelihood of typing errors a special TBUS input tool, *tbus_input* is provided. It allows an operator to enter TBUS parameters into a screen template, checks the parameters as far as is possible, and writes the complete message into the local archive.

The normal way to retrieve a TBUS message from the archive is by using the tool *find_tbus*. Given the required satellite and date it locates the nearest, earlier TBUS to that date and returns its name. It then asks whether or not to install the message in a passage header for use in header generation. If so, it asks for the passage number and copies the TBUS data to the file *tbus.dat* in that passage's header directory.

find_tbus is called almost exclusively by scripts (eg. *tirnia* which perform both HRPT ingestion to passage and header generation). They call *find_tbus* using the parameters in the *AQTIM.DAT* file in the passage's header directory. Once the TBUS is available in the header it can be used by *makehdr* to compute the earth location data for the pass and write it into the scratch header file *XHEADER.SCR*.

If a TBUS is not available for a particular day, *find_tbus* returns the name of the closest prior TBUS in time. However, because of the difficulties of predicting how satellite orbits change with time, the greater the time difference between the TBUS message and the start of the pass the less reliable the earth location is likely to be.

8.4.2 Coast-line databases

All SHARP products include three one-bit graphical overlays of the coastline, state boundaries and a 5° latitude longitude grid. The coastline and state boundary overlays are created for each scene from two global data sets held in *EXEC*. The file *EXEC/CSTLATU.ISS* contains a line segment representation of the world's coastline at a resolution of 2.3 km. The file *EXEC/CSTLATH.ISS* contains similar data for the world's state boundaries. There are several other files in *EXEC* that contain similar information at higher resolution.

shgridsaq generates overlays for the current scene and writes them to bitmap files *COASTLIN.DAT*, *BOUNDARY.DAT* and *LATLONG.DAT.*

8.4.3 Area of Interest

The 'area of interest' or 'window' is used by *dbp* when deciding whether or not to copy a given scene to magnetic tape. The window is a polygonal area defined by the latitude and longitude of its vertices written in a file *EXEC/CURRENT_WINDOW.DAT*. The first record in the file contains a single integer value which defines the number of vertices in the polygon. Each of the remaining records contains the longitude and latitude (both in degrees) of a single vertex. The two values are separated by a colon.

8.4.4 Calibration data

AVHRR image data is transmitted from the satellite, and stored in SHARP scenes, as uncalibrated 10-bit integers. In order to be able to make sensible comparisons between scenes acquired by different satellites or at different times the instrument views two black-body targets as it completes each scan-line. One target is internal and its temperature is monitored, the other is deep space, its temperature is assumed to be constant. Given the temperatures of the two targets and the measured response of the detectors while viewing them it is possible to make a linear calibration of the instrument's raw data. The HRPT telemetry includes the temperature of the internal target and the responses of the three infra-red detectors. This data is used to calibrate the infra-red imagery when generating QuickLooks and SHARP Level 2 products.

Because the detectors and filters in each AVHRR differ one from another, the relationship between the infra-red radiance measured by the detector and the temperature of the target is satellite dependent. For this reason a set of Look-up Tables is provided from ESRIN for each satellite as it is launched. They are stored as sets of three files in the directory */usr/tiros/CODE/LEVEL2*, one set for each satellite. The files are named *BTA#*, *BTB#* and *BTCT#*, where # is the two-digit mission number. The first two contain radiance to temperature look-up-tables at 0.1C and 0.01C resolution respectively and the third contains the 'non-linear' correction.

8.4.5 Data Catalogue

Every time a SHARP scene is archived to optical disk the *odh* program generates a catalogue entry that details the date and time and geographical extent of the scene and adds it to the file *LOGS/catalogue*.

The newly created catalogue entries are transmitted every day to the LEDA catalogue at ESRIN. A script, *NETWORK/AUTOXCAT/sendcat.ftp*, is executed by the *cron* job-scheduler early every morning.

The output from background job is usually mailed to the system administrator, and should be checked every day to ensure that the transfers have happened. If the transmission does not succeed the unsent catalogue entries are retained for transmission the following day.

Glossary

ACQUISITION
The process by which the HRPT data stream transmitted by a NOAA satellite is received at a groundstation, decoded and stored on magnetic disk or an HDDT tape.

ATG
One of the two makes of optical disk drive used by SHARK systems.

AVHRR
Advanced **V**ery **H**igh **R**esolution **R**adiometer. The main imaging instrument carried on the polar orbiting NOAA meteorological satellites. The AVHRR is a four or five band instrument sensitive in the visible and infra-red parts of the spectrum. It produces imagery with a nominal ground resolution of 1 Km and a swath width of about 3000 Km.

CCT
Computer **C**ompatible **T**ape. Normally refers to ½" tape used for distributing data. See also HDDT and Exabyte.

DOWNLOAD
Copying a SHARP product from magnetic tape to a slot on magnetic disk.

EPOCAT
A computer at ESRIN on which the central catalogue of the SHARP archive is kept.

ESA
European **S**pace **A**gency.

Exabyte
A tape drive which uses a cartridge containing 8 mm tape in contrast to the older ½" reel to reel drives. Apart from the much greater capacity of the Exabyte tapes the two kinds of drive are treated identically by both SunOS and the SHARK system.

GMT
Greenwich **M**ean **T**ime.

GROUNDSTATION
A facility with equipment which can track satellites and receive the telemetry and data transmitted by them. In particular stations capable of receiving the transmissions made by the polar orbiting NOAA meteorological satellites.

GUI
Graphical **U**ser **I**nterface - the term given to graphical programs which run in a "window environment" where the user commands the program through the use of a mouse to select objects on the screen rather than with the keyboard. Some of the SSE's tools are GUIs.

HIRS
High **R**esolution **I**nfra-red **S**ounder. An instrument carried by the NOAA satellites which has 20 channels sensitive in the infra-red part of the spectrum. The data are used to calculate temperature profiles through the depth of the atmosphere.

HRPT
High **R**esolution **P**icture **T**ransmission - the name given to the data stream coming from the NOAA series of satellites. When stored on the SSE, this data is held in a file in the area known as the HRPT data space. The name of this file is made known to the rest of the SSE by being placed in the database file *RAWINFO/rawtable*. The database contains one entry for each HRPT file on the system. This means that if there is enough space for three HRPT files (each occupies about 115 Mbytes) on a system, there will be three 115 Mb disk areas reserved for them and three entries in the database file which point to them. There will also be three corresponding header directories - see Passage Header.

INGESTION
The process by which raw HRPT data is read into the SHARK Software Environment's 'HRPT data space'. This can be direct (i.e. from ACQUISITION) or indirect via an intermediate source such as a tape or a network.

LEDA
On-**L**ine **E**arthnet **D**ata **A**vailability. The on-line catalogue of all satellite data archived for ESRIN.

LMSI
One of the two types of optical disk drive used by SHARK systems.

LTWG
Landsat **T**echnical **W**orking **G**roup.

NDVI
Normalised **D**ifference **V**egetation **I**ndex.

MSU
Microwave **S**ounding **U**nit. A sounding instrument carried on the NOAA satellites. It is a four channel instrument sensitive in the microwave region of the spectrum.

NOAA
National **O**ceanic and **A**tmospheric **A**dministration.

OD
Optical **D**isk - this encompasses both the physical drive and the individual cartridges that can be loaded into it, but usually the implication is that of the filesystem on the currently-loaded cartridges (made available as a UNIX-like filesystem through the Dorofile software). Optical disks are used as a product archive medium by the SSE.

PASS
This refers to the movement of one of the NOAA satellites while being tracked by a groundstation. This usually lasts no longer than 16 minutes.

PASSAGE
A passage is a SSE-specific term which encompasses both the HRPT data file itself and the necessary supporting header files which the SSE needs to access it.

PASSAGE HEADER
This is the directory in which a passage's header files are kept. Passage headers (there are several, depending on how many HRPT files a system can accommodate) are located in the RAWINFO directory and are numbered sequentially - with the name *header[1-9]*. So the passage header which corresponds to HRPT file 1 (see "HRPT" definition above) will be called *RAWINFO/header1*.

PRODUCT
A SHARP Level-1B volume unless explicitly stated otherwise.

QL
Quick**L**ook. The output of a classification process performed on a SHARP product that is made immediately subsequent to the output of a product into the slot - either by the SHARP processing chain or when downloaded there from magnetic tape. A QuickLook consists of two files, named *QUICKLOOK.PIC* and *QUICKLOOK.DAT*; the former is a picture file consisting of a displayable mini (sub-sampled) picture of the product in classified form (cloud, land sea etc.), the latter the results of the classification as counts of each pixel type.

SHARK SOFTWARE ENVIRONMENT (SSE)
A tree-structured hierarchy of SHARK-specific directories containing a complex family of data files, UNIX scripts, executables, and their C and Fortran source files. These directories reside under a parent directory, */home/tiros*, (TIROS being the first of the NOAA satellites launched in 1978). For the sake of brevity, all subdirectories when mentioned in this document are positioned relative to it, rather than to the true filesystem root */*). For example, if the document makes reference to the *EXEC* directory, it means */home/tiros/EXEC*. Similarly, references to *RAWINFO/header[1-9]* mean */home/tiros/RAWINFO/header[1-9]*.

SHARK SYSTEM
Station **H**RPT **A**rchiving and **R**eprocessing **K**ernel. The software tools needed to generate and manipulate SHARP products and the hardware on which the software runs.

SHARP
Standard Family **H**RPT **A**rchive and **R**equest **P**roduct. The product family that the SSE produces from HRPT data, as defined by the ESRIN document "SHARP-1 TECHNICAL SPECIFICATION OF CCT FORMAT RELEASE 1.1 (Feb '89)" There are at present three levels of SHARP defined, (also 2A & 2B), but in the scope of this document it will be taken to mean LEVEL 1 (also known as "LEVEL 1B") unless otherwise stated.

SHARP Product Space
The disk space reserved for SHARP products in the directories *RAWINFO/slot[1-9].*

SLOT
A directory in which SHARP products are placed by SHARP processing. There are between one and ten slots on a SHARK system and they are located directly under */usr/tiros* and are known as 'slotN' where N is a number between one and ten.

SSE
SHARK **S**oftware **E**nvironment.

SST
Sea **S**urface **T**emperature

SSU
Stratospheric **S**ounding **U**nit. A infra-red temperature sounding instrument carried on the NOAA satellites.

TAPE
Magnetic tape. Either a ½" computer compatible tape (CCT) recorded at 6250 bpi or a low-density 8 mm tape. Both are handled identically at a system level.

TBUS
A small ASCII file containing the orbital elements for a single satellite on a certain day; these are necessary for the correct processing of SHARP.

TIP
TIROS **I**nformation **P**rocessor

TIROS
Television and **I**nfra-**R**ed **O**bservation **S**atellite. One of the earliest series of operational meteorological satellites.

TIROS-N
The name of the first of the current series of operational meteorological satellites. They are often referred to as the NOAA/TIROS-N series.

TM
Transverse **M**ercator. A map projection.

TOVS
TIROS **O**perational **V**ertical **S**ounder. The temperature and humidity sounding system carried on the NOAA/TIROS-N satellites. The system is composed of three individual instruments; the HIRS, MSU and SSU.

UPLOAD
Copying a SHARP product from slot on magnetic disk to magnetic tape.

Appendix 1: shark tool descriptions

Each of the SSE tools will be described in what follows.

All pathnames are given relative to the SSE file tree root */usr/tiros*, unless they begin with a slash '/', in which case they are full pathnames relative to the system tree root (/).

CODE/SHARP	=	/usr/tiros/CODE/SHARP
/usr/noaa	=	/usr/noaa

Each tool is assigned to one of the following processing phases.

HRPT-INGESTION
This covers all the numerous magnetic tape reading programs and program managers which handle the ingestion of HRPT-like data from remote stations onto local disk in common SHARK format.

HEADER-GENERATION
The step immediately prior to ingestion; the creation of a standard header for the new HRPT data file.

TBUS
All tools related to the archive and retrieval of TBUS in the local archive.

DISPLAY
This is the *display* GUI and its associated child processes which allow visual display of HRPT passages, coastline adjustment (navigation correction), interactive SHARP product selection and either immediate or queued SHARP product generation.

SHARP
This includes the four processes that make up the production chain for SHARP products and their manager script.

SHARK
This covers the *shark* GUI which is used for the handling of SHARP products (levels 1, 2A and 2B). Functions include; display classified QuickLook product, display SHARP product, magnetic tape upload and download, optical disk archive and retrieval, generate several "extra" products and print QuickLook.

BATCH
This is the set of tools which have been written to allow the existing SHARP production and *shark* archiving tools to work together as a single batch processing chain to streamline operations.

UTILS
All additional support tools for irregular ("off-line") operations.

NAME: archive_tbus

PROCESSING PHASE:
TBUS

DESCRIPTION:
Get latest TBUS from NOAA EBB to local archive via X.25 by mimicking an interactive login session.

SOURCE TYPE:
C program

LOCATION: *BIN/SRC/NOAA*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
BIN/SRC/NOAA/NOAA_EBB_input.dat

OUTPUT FILES:
TBUS/n#<YYMMDD>.tbu
/tmp/moe#.dat

NOTES: The process starts by reading the contents of the file *NOAA_EBB_input.dat* into an internal data structure. The file contains basic information on how to navigate around the NOAA EBB.
Next the program attempts to connect to the NOAA EBB via X.25. If the connection fails, it exits immediately with an error status. Otherwise, once connected to the NOAA EBB, it mimics an interactive login, stepping through successive menus until it reaches the top level, 'Schedules', menu. Here it first searches each message header for the keyword "TBUS". It then checks for the pattern "TBUS mm/dd/yy". Headers which contain this pattern are taken to refer to valid TBUS messages. The program keeps note of the message number and the date. The date is compared with that of any valid TBUS message found later. Only the more recent of the two is kept. In this way, only the very latest TBUS message is marked for transfer.

After all headers have been scanned the code transfers the message whose header it has noted. It scans through the message searching for keywords, isolating and reading valid TBUS messages for each of the operational satellites and writing them to appropriately-named files in the TBUS archive directory. Some of the parameters (MOEs) are written into small files in */tmp* which are later copied to the EPOCAT MicroVAX by the script *get_periodic_tbus*, which usually calls *archive_tbus*.

Once the process is completed it returns its status to the screen (usually redirected to a file by *get_periodic_tbus*) and then control is passed back to the main module which tidies up and exits.

SAMPLE OUTPUT:

(From shark's mail)

```
From shark@plod Wed Nov 18 09:51:12 1992
Return-Path: <shark@plod>
Received: from plod.esrin.esa.it by master.esa.it (4.1/SMI-4.1)
        id AA03400; Wed, 18 Nov 92 09:51:11 GMT
Received: from nimbus.esa.it by plod.esrin.esa.it (4.1/SMI-4.1)
        id AA16536; Wed, 18 Nov 92 09:57:56 GMT
Date: Wed, 18 Nov 92 09:57:56 GMT
From: shark@plod (Shark Operational Account)
Message-Id: <9211180957.AA16536@plod.esrin.esa.it>
To: shark@plod
Status: R

Program archive_tbus exited normally. It generated the following output:

>> Connected to DTE 0311090900406 <<

Got login prompt..
Got password prompt...
Login successful.
Got OMNET command prompt...
Got first access message...
Got top menu prompt...
The latest TBUS message (# 41) was written on: 20/11/92

TBUS extracted OK for NOAA 12
1991 032A 07892 325008860338 921120001245533 0624584
01012602 01013128 00134961 10133745 35262829 09868008
25893975 07193410 P071415257 M009239456 M000000001
M00161862 M01110633 P07355523 004027832 130124015 9449
0000500000 M00265360 P00098407 P00511948 SPARESPARE

>> Archive file /usr/tiros/TBUS/n12921120.tbu successfully written. <<

TBUS extracted OK for NOAA 11
1988 089A 21414 325041349049 921120005932558 0741864
01019271 01019833 00118558 30945399 29102943 09910382
05056570 07225222 P025928470 M067442441 M000000001
M01097357 M00420502 P07338322 003473223 130124015 9449
0000500000 M00280614 P00101618 P00508599 SPARESPARE

>> Archive file /usr/tiros/TBUS/n11921120.tbu successfully written. <<

TBUS extracted OK for NOAA 9
1984 123A 40923 325046888454 921120010731162 0761860
01018780 01019343 00150754 03923330 35534133 09912734
32100018 07222929 P071963203 M005864312 M000000000
M00110873 M01173727 P07342061 003339505 130124015 9449
0000500000 M00281381 P00101983 P00508844 SPARESPARE

>> Archive file /usr/tiros/TBUS/n09921120.tbu successfully written. <<

TBUS extracted OK for NOAA 10
1986 073A 32084 325004999840 921120000711986 0610648
```

```
01010718 01011297 00133087 20415221 34013049 09852689
15591152 07184723 P067706589 M024468659 M000000001
M00378499 M01035155 P07355926 001577242 130124015 9449
0000500000 M00293830 P00097105 P00512903 SPARESPARE

>> Archive file /usr/tiros/TBUS/n10921120.tbu successfully written. <<
Got EBB top menu prompt...

>> Logging off NOAA EBB <<
Got OMNET command prompt...

>> Logging off OMNET <<
```

SEE ALSO: get_periodic_tbus

DEBUGGING TIPS:

If the process hangs (*get_periodic_tbus* will report this in the *cron* mail), it is usually because the staff at NOAA responsible for input of the data have mistyped something in one of the TBUS headers for that day. e.g. the expected string "TBUS1/2/3/4" may appear as "TBUS1/2?3/4" or something similar. If the mistake is not rectified after a day or so NOAA should be informed.

The program is fragile in this way because it looks for exact matches to patterns; any deviation from the norm causes it to fail. There is a debugging facility available. Rebuild the program with CFLAGS set to "-g -DDEBUG_PKT" (see the makefile) and run it again. All incoming packets are dumped to standard output. This can be used either in conjunction with the debugger or a parallel real login to compare what the human and the program do to get the TBUS.

NAME: avhrr_env

PROCESSING PHASE:
UTILS

DESCRIPTION:
Set up the correct processing environment for a station.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
Executes in $(EXEC_DIR)

INPUT FILES:
EXEC/PRODEF.DOC

OUTPUT FILES:
[*EXEC/PRODEF.DOC*]
[*CODE/QLOOK/FTN/ql*]

NOTES: Called automatically at the start of *display*. Used to change environments to process data acquired at different sites.

SAMPLE OUTPUT:

```
          [shark@nimbus](usr/tiros/EXEC)> avhrr_env


Current processing environment is: MANILA

Do you want to change station environment (y/n) [n] ? y

          Current environments:

          1. TNB STATION (ANTARCTICA)
          2. FRASCATI
          3. MASPALOMAS
          4. NAIROBI
          5. NIAMEY
          6. DLR
          7. TROMSOE
          8. DUNDEE
          9. C. PAULISTA
          0. PRETORIA
          a. CAIRO
          b. MANILA
          c. LA REUNION

          Enter choice: 6
```

```
Installing /usr/tiros/PRODEFS/PRODEF.DOC.DLR as current PRODEF.DOC ...
Building QL ... (please wait a short time) ...
rm -f *.o ql core
f77 -misalign -Bstatic -w -cg89 -sun4 -c qlsun.f
qlsun.f:
        MAIN:
        slope:
        tielines:
        glintlim:
        dayornight:
        setsun:
        calib:
f77 -misalign -Bstatic -w -cg89 qlsun.o -o ql
/home/tiros/CODE/QLOOK/FTN/LIB/Fortran_C_IO.a

avhrr_env exiting OK at Thu Nov 19 17:48:26 GMT 1992
```

SEE ALSO: display

NAME: bach

PROCESSING PHASE:
HEADER-GENERATION

DESCRIPTION:
Build acquisition header file. Creates the *XHEADER.SCR* file in a passage header from *AQTIM.DAT* and *tbus.dat* files.

SOURCE TYPE:
FORTRAN

LOCATION: *EXEC*

INPUT PARAMETERS:
One command-line argument: passage number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
RAWINFO/header[1-9]/AQTIM.DAT
RAWINFO/header[1-9]/tbus.dat

OUTPUT FILES:
RAWINFO/header[1-9]/XHEADER.SCR

NOTES: Normally run by the script *makehdr* immediately after data ingestion.

SAMPLE OUTPUT:

```
        [shark@nimbus](usr/tiros/EXEC)> bach 1

* ARGUMENTS FOR ORBIT CALCULATIONS ARE:

        * EPOCH ORBIT = 6940
        * EPOCH DATE AND TIME
        92 9 14 0 41 25 108

        * MEAN ORBITAL ELEMENTS
        7193.5680000000 1.2530700000000D-03 98.687500000000
286.66721000000
        293.70249000000 66.291660000000

        * ACQUISITION START DATE AND TIME
        92 9 15 6 8 30 423

        * RESULTS OF ORBIT CALCULATIONS:

        * EQUATOR CROSSING DATE AND TIME
        92 9 15 6 14 27 49
        * ACQUISITION ORBIT = 6958
        * LONGITUDE = 19.809388398450
        * A/D INDICATOR = D
```

SEE ALSO: find_tbus, create_id_file, makehdr

NAME: batch_menu

PROCESSING PHASE:
BATCH

DESCRIPTION:
Front-end menu for SHARP batch-processor (*dbp*). presents user with choice of mode in which to run *dbp*.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
EXEC_DIR

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: *dbp* generates SHARP Level 1 products. In addition it can perform varying actions on the products it has generated. The menu presented by *batch_menu* allows the user to decide which of the alternative actions to take.

Choice 1 simply writes the SHARP products to slots on magnetic disk. However, there are no more than ten slots available on a disk and, usually more than two SHARP scenes are extracted from each passage. This severely restricts the number of passages which can be processed by *dbp* and copied directly to slots.

Choice 2 instructs *dbp* to append each SHARP slot whose centre point falls within the polygonal geographical area defined in *EXEC/CURRENT_WINDOW.DAT* to the current tape ($TAPE) using the utility *BIN/slot_to_exabyte* and then delete it from the slot.

Choice 3 archives each SHARP product to the optical disk ($OHOME).

Choice 4 is a combination of 2 and 3. Each product is archived to optical disk and, in addition, each product whose centre lies within the current window is copied to the tape.

dbp lives in *EXEC/DBP* and is actually one C-shell script with four symbolic links to it:

```
dbp
dbp.OD -> dbp
dbp.OD+exabyte -> dbp
dbp.SHARP_only -> dbp
dbp.exabyte -> dbp
```

batch_menu calls one of the four symbolic links, depending on which option is selected. *dbp* runs whichever link is used to initiate it. *dbp* examines its argv[0] variable (execution name) and uses this to decide what actions to take.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> dbp

BATCH SHARP PRODUCTION/ARCHIVE MENU

Please choose the action you require:

1) Batch process SHARP only (no archive)
2) Batch process SHARP & archive to exabyte tape only
3) Batch process SHARP & archive to optical disk only
4) Batch process SHARP & archive to optical disk and
        (selected products) to exabyte tape
5) quit.

Choice ?
```

SEE ALSO: dbp

NAME: checkcats

PROCESSING PHASE:
UTILS

DESCRIPTION:
Examines the latest SHARP and AVHRR catalogue record and updates the log file entries.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
epocat::disk$epo_cat_1:[cats.update]avhrr_cat_ceos.log
epocat::disk$epo_cat_1:[cats.update]sharp_cat_ceos.log

OUTPUT FILES:
<NONE>

NOTES: To check the day-to-day status of the SHARP catalogue and apply updates from both remote groundstations and the local SHARK systems use *checkcats* which is located in */home/tiros/BIN* (which is in *shark*'s path).

checkcats uses DNI (Sun's DECnet product) to copy over the 'transfer status logfile' from EPOCAT and then types out the last 24 lines (about one screenful). You can see at a glance if there have been file transfer problems over the previous few days.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> checkcats

Show AVHRR log ? (y/n) n
Show SHARP log ? (y/n) y
10-NOV-1992: Tromsoe   : Number of records input :  34
10-NOV-1992: DLR       : Number of records input :  17
10-NOV-1992: Frascati  : No catalogue activity since last transmission. 10-NOV-1992:
Dundee    : Number of records input :  50
11-NOV-1992: Maspalomas: Number of records input :  11
11-NOV-1992: Tromsoe   : Number of records input :  12
11-NOV-1992: DLR       : Number of records input :  10
11-NOV-1992: Frascati  : Number of records input :  22
11-NOV-1992: Dundee    : Number of records input :  50
12-NOV-1992: Maspalomas: Number of records input :  17
12-NOV-1992: Tromsoe   : Number of records input :  12
12-NOV-1992: DLR       : Number of records input :  12
12-NOV-1992: Frascati  : Number of records input :  46
12-NOV-1992: Dundee    : Number of records input :  23
```

```
13-NOV-1992: Maspalomas: << WARNING ! >> Network transmission problem.
13-NOV-1992: Tromsoe   : << WARNING ! >> Network transmission problem.
13-NOV-1992: DLR       : << WARNING ! >> Network transmission problem.
13-NOV-1992: Frascati  : << WARNING ! >> Network transmission problem.
13-NOV-1992: Dundee    : << WARNING ! >> Network transmission problem.
14-NOV-1992: Maspalomas: Number of records input :  23
14-NOV-1992: Tromsoe   : Number of records input :  25
14-NOV-1992: Frascati  : Number of records input :  36
14-NOV-1992: Dundee    : Number of records input :  75
14-NOV-1992: Dundee    : Number of records input :  27
```

In this example the user chose to see only the SHARP log, not the AVHRR log. The output indicates that there was a problem on the 13th November - in fact the EPOCAT machine was down. The unsent records were sent on the following day.

NOTES: The log file names on the VAX are hard-coded into *checkcats*. If these are changed for any reason, the *checkcats* script will have to be altered.

NAME: clp

PROCESSING PHASE:
UTILS

DESCRIPTION:
Clears a passage - Deletes a selected HRPT passage header including the *BATCH* subdirectory.

SOURCE TYPE:
Bourne Shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument: passage number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Only deletes the passage header, not the HRPT file, which will be overwritten anyway as a result of the *.id* file being removed.

monitor DOES delete the HRPT file (except at DLR).

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> clp 2

Really clear passage header #2 (y/n) ? y

Passage slot #2 cleared.
```

SEE ALSO: monitor, clps

NAME: clps

PROCESSING PHASE:
UTILS

DESCRIPTION:
Clears all passages - Delete all HRPT passage headers including the BATCH subdirectories.

SOURCE TYPE:
Bourne Shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: See clp.

SAMPLE OUTPUT:

```
        [shark@nimbus](usr/tiros/EXEC)> clps

Really clear all passage headers (y/n) ? y

All passage headers cleared.
```

SEE ALSO: monitor, clp

NAME: cls

PROCESSING PHASE:
UTILS

DESCRIPTION:
Clears a slot - Delete a selected SHARP product from magnetic disk.

SOURCE TYPE:
Bourne Shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument: slot number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Recursively removes all subdirectories i.e. LEVEL-2 products.

SAMPLE OUTPUT:

```
        [shark@nimbus](usr/tiros/EXEC)> cls 3

Really delete SHARP product in slot #3 (y/n) ? y

Scene slot #3 cleared.
```

SEE ALSO: monitor, clss

NAME: clss

PROCESSING PHASE:
UTILS

DESCRIPTION:
Clear slots - Delete all SHARP products in slots

SOURCE TYPE:
Bourne Shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: See cls.

SAMPLE OUTPUT:

```
        [shark@nimbus](usr/tiros/EXEC)> clss

Really delete all SHARP product in slots (y/n) ? y

Scene slots [1 - 10] cleared.
```

SEE ALSO: cls, monitor

NAME: create_id_file

PROCESSING PHASE:
HEADER-GENERATION

DESCRIPTION:
Used to make *.id* file for a passage.

SOURCE TYPE:
C program

LOCATION: *BIN*

INPUT PARAMETERS:
Two command-line arguments:
1) full path name of *.id* file to create
2) full path name of tape device or *XHEADER.SCR* file

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
tape device or *RAWINFO/header[1-9]/XHEADER.SCR* file

OUTPUT FILES:
RAWINFO/header[1-9]/.id
/tmp/ddfile

NOTES: Copies *RAWINFO/header[1-9]/XHEADER.SCR* file into */tmp/ddfile*. Reads some useful strings, (Satellite identifier, date (YYMMDD) and scene start time (HHMMSS)), from */tmp/ddfile*.
Writes strings to output file (*RAWINFO/header[1-9]/.id*) to provide quick identification of passage.

SAMPLE OUTPUT:
```
[shark@nimbus](usr/tiros/EXEC)>
create_id_file  /home/tiros/RAWINFO/header1/.id \
                  /home/tiros/RAWINFO/header1/XHEADER.SCR

2+0 records in
2+0 records out
```

SEE ALSO: makehdr

NAME: dbp

PROCESSING PHASE:
BATCH

DESCRIPTION:
SHARP batch processing and archiving engine. *dbp* finds each passage with a *BATCH* subdirectory, installs the SHARP request file found there in *EXEC* and calls *disp_sharp* to generate it. After the last QuickLook has been created in the last slot, *dbp* usually archives all the newly-generated slots to optical disk using *tiros* in a batch mode. Then the slots are cleared and *dbp* starts the next passage. In this way a lot of data can be processed and archived with no operator intervention.

If *dbp* is called as *dbp.exabyte* instead of archiving products to optical disk it appends selected products (see "batch_menu") to magnetic tape. Alternatively, *dbp* can be instructed to do both optical disk and magnetic tape operations at the same time.

SOURCE TYPE:
C-shell script

LOCATION: *EXEC/DBP*

INPUT PARAMETERS:
NONE - switches mode via argv[0] variable - see "batch_menu" for details.

ENVIRONMENT VARIABLES:
EXEC_DIR - *dbp* executes here.
TAPE - device name of tape drive.
OHOME - device name of optical disk drive.
OHOME_MNT - mount point used by optical disk drive.The drive is mounted using Dorofile's *womount.*
SHARK_QL_NCOPIES - During non-batch archiving, the number of QL printouts generated per product. However, when working in batch mode *dbp* temporarily modifies it so that *tiros* creates one extra copy.

INPUT FILES:
<NONE>

OUTPUT FILES:
Log files:
LOGS/dbp.log.1
LOGS/dbp.log.2
LOGS/dbp.log.3
LOGS/dbp.log.4
LOGS/dbp.log.5

tiros activity log file:
LOGS/station_log

Temporary files (deleted on successful exit):
/usr/tmp/dbp.<PID>
/usr/tmp/tempfile.<PID>

/usr/tmp/tempfile1.<PID>
/usr/tmp/odh_dir.<PID>
/usr/tmp/odh_write.<PID>

NOTES: LOCK FILES
While running, *dbp* maintains the following lock files which may require manual deletion if execution is interrupted:

EXEC/sharp_lockfile
RAWINFO/header[1-9]/.sharp_lock

BATCH REQUEST DIRECTORY
A BATCH header subdirectory, */RAWINFO/header[1-9]/BATCH*, contains two files.

The SHARP request file for the passage, *SHARPREQ.DOC*, which is copied into *EXEC* before *disp_sharp* executes.

A small file containing the raw-data quality indicator set by the operator while running *display*. This is passed to *tiros* as a command-line argument, and the quality indicator is added to the catalogue record.

If a batch operation is successful, the *BATCH* subdirectory is deleted afterwards. While it is present, *monitor* adds a "@" symbol to the end of the corresponding passage id string.

LOG FILE
dbp is very verbose - is logs everything to a file, *LOGS/dbp.log*, and keeps up to five previous versions for reference.

WORK FILES
If *dbp* fails, internal work files (see 'Temporary files' above) are not deleted, to give the users more chance to understand what happened. These can be read with *cat*(1) or *more*(1). <PID> represents the Unix process id of this instance of *dbp*. It is typed out by *dbp* when it starts.

ADDING GEOGRAPHICALLY-SELECTED PRODUCTS TO TAPE
Before beginning to append geographically-selected products to magnetic tape, *dbp* attempts to ensure that there is enough room on the tape. Obviously, this is not an easy calculation to make as the number of products actually selected in any given run can vary from none to all. So a broad safety margin is used. If there are more than 35 products on the current tape (as found in the file *EXEC/exabyte.toc*), *dbp* refuses to continue until the tape is 'shipped' (using *sharp_exabyte.ops*) and a fresh one made ready.

When a product is written to tape, *dbp* records the fact by updating the *tiros* activity log file *LOGS/station_log*.

FAILURES
If *dbp* fails to process a passage, the *BATCH* subdirectory is not deleted, an appropriate message generated and the program moves on to the next passage.

If *dbp* fails to archive a slot, the slot is not deleted, an appropriate message generated and the program moves on to the next slot.

The reason for a failure may be found by looking in the *wrsharpaq* error log file, *EXEC/SHARPLOG.DOC.*

A common cause of failure is that there are no more slots available. *wrsharpaq* does NOT handle this condition correctly - it tries to output data on slot -1 (-1 being the internal error code passed by a C routine called *getslot()* to show "all slots full").

CHECKING SPACE ON OD
Before attempting to archive to optical disk, *dbp* calls *odh* in batch mode to give it a directory listing of the current OD side.
dbp checks entry #2 and if there is no more space available it stops.

NAME: dbtirget

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
Double buffered magnetic tape read program for HRPT data in 'Maspalomas' format.

SOURCE TYPE:
C program

LOCATION: *EXEC*

INPUT PARAMETERS:
Two command-line arguments:
1) tape unit number to read from
2) passage number to write to

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<Tape device>
RAWINFO/rawtable

OUTPUT FILES:
<HRPT passage file>
RAWINFO/header[1-9]/XHEADER.SCR

NOTES: Uses System V IPC facilities and runs a fast double buffering system between two communicating processes; one reads the tape, the other writes the data to disk.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> dbtirget /dev/nrst20 3

* dbtirget:start date= 230191
* dbtirget:start time= 112314011
* dbtirget:stop date= 230191
* dbtirget:stop time= 114231221
* dbtirget:n. of lines= 4278
dbtirget reader: satellite = NOAA-10
* dbtirget:n. of lines= 4278
```

SEE ALSO: disp_tirget, display

NAME: display

PROCESSING PHASE:
DISPLAY

DESCRIPTION:
GUI tool which lets an operator interactively define the extent of SHARP products to be extracted from an HRPT passage. It also provides an interactive tool for correcting the poor navigation of a pass caused by old TBUS data.

SOURCE TYPE:
C program

LOCATION: *DISPLAY*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
TAPE - magnetic tape device name.
SLOTS_PARTITION - UNIX disk partition in which the SHARP slots reside.

INPUT FILES:
RAWINFO/header[1-9]/.id
RAWINFO/header[1-9]/XHEADER.SCR
RAWINFO/rawtable
HRPT Data files (as defined in *RAWINFO/rawtable*)

OUTPUT FILES:
EXEC/tape_lockfile
EXEC/sharp_lockfile
RAWINFO/header[1-9]/.sharp_lock
RAWINFO/header[1-9]/.coast_lock
RAWINFO/header[1-9]/.net_lock
EXEC/SHARPREQ.DOC
RAWINFO/header[1-9]/BATCH/SHARPREQ.DOC
RAWINFO/header[1-9]/BATCH/image.quality

NOTES: Tape Operations
There used to be a greater requirement for *display* to be able to ingest CCT data. Of the entire suite of *tir** (HRPT ingest) tape programs, only two have ever been configured to work in *display*: *dbtirget* for Maspalomas and *dbtirtro* for Tromsoe. There is no reason why others cannot be adapted to work from *display*.

If the ingest facility is used, (i.e. if the environment variable 'TAPE' is set when *display* starts up), a child process called *tapehandler* is forked from the *display* process. The role of the child process is to act as an asynchronous background handler for the HRPT ingest program (*disp_tirget*), so that *display* need not wait for tape operations to complete. Instead, when the user requests a tape download, *display* sends a message to *tapehandler* and then continues servicing user inputs. Meanwhile *tapehandler* forks a process which runs the HRPT ingest program and waits for it to finish.

The result is then sent to *display* in the form of an interrupt which causes it to momentarily break off from whatever it is doing, read the result, act accordingly (send a status message to the operator) and then continue where it left off.

While HRPT ingestion is underway, a lockfile called *EXEC/tape_lockfile* is created by *display* to prevent other processes trying to use the tape drive. However, this is a redundant mechanism which in the past was used to lock the system's only tape drive. On current systems it is not uncommon to have three or four tape devices all of which can be accessed simultaneously.

SHARP Operations
SHARP operations are really what *display* was designed for, and its major output is the SHARP request file (*SHARPREQ*) which contains the information that the SHARP processing chain needs to generate the requested products.

The SHARP generation (for full SHARP only) is handled by a forked child process called *sharphandler*, whose behaviour is almost identical to the *tapehandler*'s (see above). It calls *disp_sharp* and creates two lock files, *EXEC/sharp_lockfile* and *RAWINFO/header[1-9]/.sharp_lock*.

Coastline adjustment works in a slightly different way. In this case *display* itself forks *disp_sharp*, waits for it to finish, and interprets the returned status as an indication of success or failure, rather than a message from a pipe. It is done synchronously because the whole operation is very short and the operator can wait for it.

A lock file called "*.coast-lock*" is created in the passage header while *disp_sharp* is running, to signal to external processes (e.g. *monitor*) what is happening to the passage.

When the user invokes coastline adjustment, the following 'behind-the-scenes' steps occur:-

- Firstly, the passage's *XHEADER.SCR* file is copied to *XHEADER.SCR.full* and the original is rewritten to contain the extent of the 30 second selected area, rather than the whole pass - this takes the form of start/stop times (in milliseconds) and the number of lines in 30 seconds (180). It is done in this way because the SHARP processing chain reads the *XHEADER.SCR* file to get the extent of the pass.

- Next, a SHARP request file for the 30 seconds is produced in *EXEC*.

- The GUI now updates to display the message "-MAKING 30-SECOND SHARP-" and the passage name button changes to show "LOCKED BY COAST".

- The *sharphandler* is shut down to prevent attempts to process two SHARP scenes at once.

- disp_sharp is now forked and the calling process waits for it to complete; it is invoked with the parameter 'I' (for 'intermediate' product) which instructs the chain to compute coastlines only and to output the mini-SHARP into a special disk area called *slot0.*
- The file *XHEADER.SCR.full* is copied back into *XHEADER.SCR.*
- sharphandler is restarted, the GUI updated and the mini-SHARP data read in from slot0.

If any part of this operation fails, it may be necessary for the operator to manually rename the *XHEADER.SCR.full* file back to *XHEADER.SCR*, but this is uncommon because when *display* starts up it automatically attempts to correct any occurrence of this it encounters.

Batch Operations
When a SHARP request is registered for batch processing, the request file and raw data quality flag are stored away in a subdirectory of the passage header called "BATCH". The presence of the subdirectory is enough to signal to both *monitor* and *dbp* that there is a request pending. *monitor* shows this fact by displaying a "@" symbol next to the passage id string.

The */BATCH* directory is created by *display* and removed by *dbp* after successful processing has taken place.

Integrity Checks
Because of noise or data loss on acquisition, the time stamp at the beginning of a scan-line may be corrupted or missing. If such a scan-line is chosen as the start or stop line of a SHARP product, it causes problems for processing, as these times are used as reference for the rest of the lines in the scene. To prevent this, checks are made to prevent lines with unrealistic times from being used as start or stop lines in a product. The checks are:

1) the start and stop times of the chosen area are compared with each other to make sure that the stop time is later than the start.

2) the difference between start and stop (duration) is compared with the known length of a typical product of that type (e.g. four mins) - if greater than it, the line is rejected.

A further check is performed each time a passage is selected for the first time:

If the duration of the entire pass (stop time - start time) is less than one minute or greater than 20 minutes a warning is generated which informs the user of the problem.

The scan-line time information can be viewed by choosing *display*'s "Select" option and then by moving the 4-minute window frame slowly up and down the displayed image. The start time of the frame is read continuously from file and output to the screen. Negative values or sudden large changes indicate corruption of the timing information.

SEE ALSO: tapehandler, sharphandler, disp_sharp, disp_tirget

NAME: disp_sharp

PROCESSING PHASE:
SHARP

DESCRIPTION:
SHARP processing chain controller script - Produce SHARP (and optionally TOVS) for a supplied passage

SOURCE TYPE:
Bourne Shell script

LOCATION: *BIN*

INPUT PARAMETERS:
Three command-line arguments:
1) A flag (Y/N) which informs the script whether to delete the *.id* and *XHEADER.SCR* files (and hence the passage) after SHARP processing.
2) The number of the header directory in which the above files exist.
3) An OPTIONAL flag (I) which tells the script whether or not the product is an intermediate one - if so, the flag is passed to *wrsharp* and *shgrids* to indicate special treatment for the data. (Used for grids adjustment only.)

ENVIRONMENT VARIABLES:
EXEC_DIR - where to run
TOVS_ON - (YES/NO) - optional TOVS production switch

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Batch script to produce SHARP (and optionally TOVS) from a given passage - it assumes that the file *EXEC/SHARPREQ.DOC* is the current SHARP request file for the passage. Usually run as a child process of the display GUI.

The exit status will be either 0, if successful, or 1, if the program failed.

SEE ALSO: dbp, display, SHARP processing chain

NAME: disp_tirget

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
Download HRPT data from tape to a passage (manager script for the tool *dbtirget* when called from the *display* tool GUI).

SOURCE TYPE:
Bourne Shell script

LOCATION: *BIN*

INPUT PARAMETERS:
Two command-line arguments:
1) tape drive device name
2) passage number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
RAWINFO/header[1-9]/XHEADER.SCR
RAWINFO/header[1-9]/.id
<HRPT file>

NOTES: Used to be used for backlog processing at Maspalomas and Tromsoe. Should be modified to use any HRPT ingestion process, not just *dbtirget.*

SAMPLE OUTPUT:
See dbtirget

SEE ALSO: create_id_file, dbtirget, display

NAME: exabyte_to_slot

PROCESSING PHASE:
UTILS

DESCRIPTION:
Download a SHARP product (internal format) from mag tape to slot.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument: <slot number>

ENVIRONMENT VARIABLES:
[TAPE]
[REMOTE_TAPE_HOST]

INPUT FILES:
<Tape device>

OUTPUT FILES:
slot#/.id
slot#/LEADER
slot#/IMAGE
slot#/TRAILER
slot#/VDF
slot#/QUICKLOOK.DAT
slot#/QUICKLOOK.PIC

NOTES: Script to read the contents of a single slot (*.id*, *LEADER*, *IMAGE*, *TRAILER*, *VDF*, *QUICKLOOK.DAT* and *QUICKLOOK.PIC* file) from tape into a specified slot. There is no *CAT* file as the scene is not yet archived. Usually the slot will have been written with the sister program *slot_from_exabyte*.

The script assumes that the tape has been positioned correctly and that there is enough room on the tape for the slot. It does not check either condition.

If the environment variable TAPE is not set, the tape */dev/nrst1* is used. If the environment variable REMOTE_TAPE_HOST is set, the script uses the tape device on that machine using *rsh*(1). This works best when the two SHARK systems share file-systems through NFS, they can then share a tape device.

SEE ALSO: slot_to_exabyte, dbp, sharp_exabyte.ops

NAME: find_tbus

PROCESSING PHASE:
TBUS

DESCRIPTION:
Locate (and optionally install in a header) a TBUS message from specified parameters.

SOURCE TYPE:
C program

LOCATION: *BIN*

INPUT PARAMETERS:
Two command-line arguments:
1) Satellite's mission number (13 will be launched in late 1993)
2) Date, as YYMMDD

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
*TBUS/**

OUTPUT FILES:
[*header[1-9]/tbus.dat*]

NOTES: Using the specified parameters, the program builds the filename of the target TBUS message as *TBUS/n<MN><YYMMDD>.tbu* (where <MN> represents the satellite's two-digit mission number and <YYMMDD> the date). Next, it searches for all files in the TBUS directory which match the supplied satellite identifier using the Unix *ls*(1) command (e.g. "ls n12*"). It then reads each filename in turn, looking for an exact match to the complete target filename. As it reads through the file list it keeps the name of the TBUS message which is has an earlier date than the target and which is closest in time to it. If it reaches the end of the list, or finds an exact match, it ends the search and passes the name of the best match to the standard output.

Next, it asks whether or not to copy the TBUS message into the a slot, (confusingly referred to as a "passage slot"). If so, it calls the *cp*(1) command to copy the TBUS file to *RAWINFO/header[1-9]/tbus.dat.*

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> find_tbus 9 921120

The closest TBUS found is n09921120.tbu (0 days difference).

Install it in a passage slot (y/n) ? y

Enter passage slot number [1 - 9]: 5
```

SEE ALSO: makehdr, tirnia

NAME: getcats

PROCESSING PHASE:
UTILS

DESCRIPTION:
Read all LEDA catalogue entries from 12-inch optical disks.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
Prompts for:
1) Acquisition site name
2) Optical disk unit number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
MISC/site_details.dat
Optical disk

OUTPUT FILES:
/usr/tmp/getcats.<PID>
/usr/tmp/cats.<PID>

NOTES: Interactive script to read all CAT files from a set of optical disks. All disks MUST contain data acquired from the same site. The script prompts only once for the acquisition site name and this is the written to each record's standard header as a site code.

The script reads the CAT files and writes their contents to */usr/tmp/getcats.<PID>* in LEDA format. It prompts the operator to mount each new disk. When they have all been read, it uses *awk*(1) to read the file, prepend a string identifying the acquisition site to each record and write the output to */usr/tmp/cats.<PID>* in *LOGS/catalogue* format.

SEE ALSO: tiros

NAME: get_periodic_tbus

PROCESSING PHASE:
TBUS

DESCRIPTION:
Routinely seed local TBUS archive from NOAA EBB. (program manager for *archive_tbus*).

SOURCE TYPE:
C-shell script

LOCATION: *BIN/SRC/NOAA*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
Temporary files (deleted on exit):

/tmp/tempfile.<PID>
/tmp/outfile.<PID>
/tmp/mailfile.<PID>

Files copied to EPOCAT VAX using DNI:

```
DISK$EPO_CAT_1:[CATS.AVHRR.SETUPS]
                                        MOE10.DAT
                                        MOE11.DAT
                                        MOE12.DAT
DISK$EPO_CAT_1:[CAT_NETWORK.UPDATES]
                                        N09<YYMMDD>.TBU
                                        N10<YYMMDD>.TBU
                                        N11<YYMMDD>.TBU
                                        N12<YYMMDD>.TBU
```

NOTES: Designed to be run by *cron*(1) silently. All output is sent to a log file. Runs the process *archive_tbus* and monitors it. If *archive_tbus* hangs the script kills it gracefully and mails the user *shark* the output file with an appropriate error message.
If all goes well, the output file is mailed to *shark* with a success message.

SAMPLE OUTPUT:

From shark's mail:

```
From root@plod Sat Nov 21 07:04:20 1992
Return-Path: <root@plod>
Received: from plod.esrin.esa.it by bungle.esa.it (4.1/SMI-4.1) id AA13471; Sat, 21 Nov
92 07:04:19 GMT
Received: from nimbus.esa.it by plod.esrin.esa.it (4.1/SMI-4.1) id AA26949; Sat, 21 Nov
92 07:04:13 GMT
Date: Sat, 21 Nov 92 07:04:13 GMT
From: root@plod (System Administrator (Victor))
Message-Id: <9211210704.AA26949@plod.esrin.esa.it>
To: shark@plod
Subject: Output from "cron" command
Status: R

Your "cron" job

/home/tiros/BIN/SRC/NOAA/get_periodic_tbus

produced the following output:

[1] 3702
[1] + Done ( ./archive_tbus >& $outputfile )
usr/tiros/TBUS/n12921123.tbu copied to
epocat::disk$epo_cat_1:[cat_network.updates]
/usr/tiros/TBUS/n11921123.tbu copied to
epocat::disk$epo_cat_1:[cat_network.updates]
/usr/tiros/TBUS/n09921123.tbu copied to
epocat::disk$epo_cat_1:[cat_network.updates]
/usr/tiros/TBUS/n10921123.tbu copied to
epocat::disk$epo_cat_1:[cat_network.updates]
```

SEE ALSO: archive_tbus

NAME: install.shark

PROCESSING PHASE:
UTILS

DESCRIPTION:
SSE (SHARK SOFTWARE ENVIRONMENT) installation. Post-*tar*(1) operation to build executables, set up links and so forth. Also advises the system administrator on use of the *.login* file for the *shark* operational account.

SOURCE TYPE:
C-shell script

LOCATION: */home/tiros*

INPUT PARAMETERS:
One optional command-line argument: -i

ENVIRONMENT VARIABLES:
(Used with 'make -e')
WINLIBS
CFLAGS
FFLAGS

INPUT FILES:
The newly *tar*'d SSE.

OUTPUT FILES:
/tmp/sh_install.<PID> (log file from *make*(1))

NOTES: To run the instal script:

```
[shark@nimbus](usr/tiros/EXEC)> cd /home/tiros
[shark@nimbus](usr/tiros/EXEC)> install.shark
```

The script will ask you to provide information describing your setup:

When it reaches the 'general SHARP-formatting software' stage, it will prompt you with the question:

```
        [shark@nimbus](usr/tiros/EXEC)>
Make clean (y/n) ?
```

If this is the first attempt to install, answer 'y'. If not, answer 'n'. This is only a time-saving device.

After performing *make*(1) in all code directories, this script uses *ln -s* to make required links for the SSE. (NOTE if you ever need to know if a link is really needed, look in this section of the script.)

If all has gone well the script will print the following message:

```
=================================================================
>>> *DON'T* FORGET TO TAILOR THE 'shark' OPERATIONAL ENVIRONMENT  <<<
>>> BY EDITING THE .login FILE & SETTING SITE-SPECIFIC PARAMETERS <<<
>>>             USING setenv(1) AND unsetenv(1).             <<<
=================================================================
```

Ensure the following line is in your *.login*:

```
setenv HRPT_BLOCK_SIZE 22528
```

At this point the system administrator should very carefully tailor the file *EXEC/.login* to reflect exactly the setup at the site. There is a section in this file clearly marked with the following line.

```
## Shark parameters - set with environment variables in the C-shell
##
```

All parameters (environment variables) in this section have an effect on some part of the SSE; carefully study the accompanying text before altering their settings.

NAME: load_slots

PROCESSING PHASE:
UTILS

DESCRIPTION:
Download multiple (La Reunion) SHARP products from tape to slots.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
TAPE

INPUT FILES:
<Tape device>

OUTPUT FILES:
slot#/.id
slot#/LEADER
slot#/IMAGE
slot#/TRAILER
slot#/VDF
slot#/QUICKLOOK.DAT
slot#/QUICKLOOK.PIC

Temporary (deleted on exit):
/tmp/load_slots.<PID>
/tmp/load_slots1.<PID>

NOTES: Script to read multiple (La Reunion) SHARP products from tape to slots. Continues to attempt download until the operator types Ctrl/C. Sits in wait state until a free slot is available.

Uses the *tiros* tools *tape* and *ql* to do the hard work.

It expects the tape to be positioned correctly at start of run.

It expects only the four "core" SHARP files per product (*LEADER*, *IMAGE*, *TRAILER* and *VDF*), the rest it generates.

.id: initially this serves as a lock while downloading (contains the string "LOADING FROM TAPE"), but afterwards, it extracts the true product id from a temporary file and writes that instead.

*QUICKLOOK.**: generates these in the usual way with *ql*.

NAME: makehdr

PROCESSING PHASE:
HEADER-GENERATION

DESCRIPTION:
Post-ingestion build of a passage's header from *RAWINFO/header[1-9]/AQTIM.DAT* and tbus files

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One optional command-line argument: passage number

ENVIRONMENT VARIABLES:
EXEC_DIR (see *.login*)

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Calls tools *bach* and *create_id_file*.

SAMPLE OUTPUT:

```
          [shark@nimbus](usr/tiros/EXEC)> makehdr 9

* ARGUMENTS FOR ORBIT CALCULATIONS ARE:
 * EPOCH ORBIT = 20129
* EPOCH DATE AND TIME
92 8 21 0 48 27 317

* MEAN ORBITAL ELEMENTS
7225.3780000000 1.1391600000000D-03 99.087960000000 198.60400000000
201.93103000000 158.14503000000

* ACQUISITION START DATE AND TIME
92 8 21 9 35 23 320

* RESULTS OF ORBIT CALCULATIONS:
 * EQUATOR CROSSING DATE AND TIME
92 8 21 9 18 22 27
* ACQUISITION ORBIT = 20134
* LONGITUDE = 89.439776866949
* A/D INDICATOR = A

2+0 records in
2+0 records out
Mon Nov 16 15:56:15 GMT 1992
--- *** ---
```

SEE ALSO: bach, create_id_file

NAME: make_sa_hrpt

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
The South African station produces HRPT-like data in its own format. AVHRR image data from each band are extracted from the raw data stream and written to separate files. The images are always aligned with the northernmost scan at the beginning of the file.

make_sa_hrpt reconstitutes the original HRPT format. The program's manager script (*tirsa*) copies all five files to disk and then calls *make_sa_hrpt* to read them all simultaneously, re-merging the pixels back into single records and, if the original pass was an ascending one, rebuilding the original aspect (bottom -> top, right -> left). The reconstituted data is finally written into a passage.

SOURCE TYPE:
C program

LOCATION: *EXEC*

INPUT PARAMETERS:
Six command-line arguments:
1-5) The names of the five single band image files (input)
6) passage number (output)

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<Single band image file>
RAWINFO/rawtable

OUTPUT FILES:
RAWINFO/header[1-9]/AQTIM.DAT
<HRPT file>

NOTES:

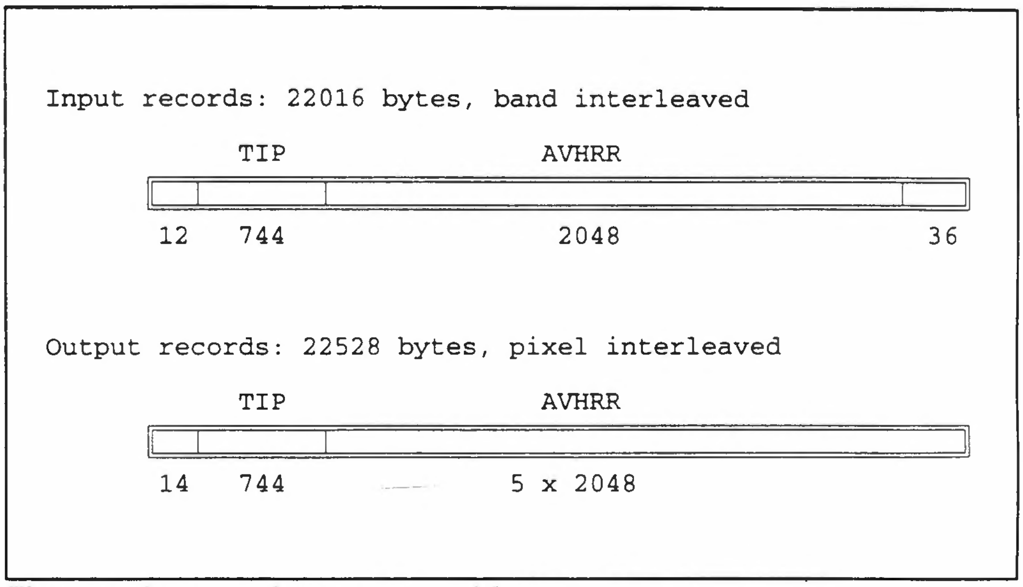
Input records: 22016 bytes, band interleaved
TIP
AVHRR
12
744
2048
36
Output records: 22528 bytes, pixel interleaved
TIP
AVHRR
14
744
5 x 2048

Figure 1 Input and output record formats.

For faster sequential disk access, *make_sa_hrpt* uses 'pooling'. This is a data-caching technique which is used by various FORTRAN processes - most notably, *wrsharpaq* and *ql*. The program must be linked with the library *CODE/QLOOK/FTN/LIB/Fortran_C_IO.a*.

SEE ALSO: */home/tiros/CODE/QLOOK/FTN/LIB/Fortran_C_IO.doc*, tirsa

NAME: make_aqtim

PROCESSING PHASE:
HEADER-GENERATION

DESCRIPTION:
Builds *RAWINFO/header[1-9]/AQTIM.DAT* from a newly-ingested HRPT file knowing only the number of lines read.
(Non-standard.) It expects the number of lines to have been written as the first field in *RAWINFO/header[1-9]/.id.*

SOURCE TYPE:
C program

LOCATION: EXEC

INPUT PARAMETERS:
One command-line argument: passage number

ENVIRONMENT VARIABLES:
HRPT_BLOCK_SIZE (see *.login*)

INPUT FILES:
RAWINFO/header[1-9]/.id
<HRPT file>
RAWINFO/rawtable

OUTPUT FILES:
RAWINFO/header[1-9]/AQTIM.DAT

NOTES: It reads the first and last lines of the HRPT passage file (hence the need to know the number of lines) and extracts from these the satellite identifier and the start/stop dates/times.

SAMPLE OUTPUT:

```
        [shark@nimbus](usr/tiros/EXEC)> make_aqtim 3

sat id = 11
date = 251291
make_aqtim: 5002 good lines in raw pass /home/big3.
File /home/tiros/RAWINFO/header3/AQTIM.DAT successfully written.
```

NAME: monitor

PROCESSING PHASE:
UTILS

DESCRIPTION:
GUI tool to constantly show the state of SSE (SHARK SOFTWARE ENVIRONMENT) data space - both HRPT passages and SHARP products.
Operators can also delete data with it.

SOURCE TYPE:
C program

LOCATION: *MONITOR*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: *monitor* shows the state of HRPT passages and SHARP products on a SHARK system. It updates every ten seconds, checking the appropriate header or slot directories and changing its output as the disk state changes. When not required it can be closed down to a SunView icon which rests in the top-right corner of the screen.

monitor is divided into three "panels":

1) The left hand display entitled "RAW HRPT DATA (PASSAGES)" which shows the state of HRPT data at any moment. If a passage is empty the corresponding information string for that passage is simply marked "< EMPTY >". If occupied by a passage, it displays whatever message is found in the ".id" file unless a 'lock' file is also present in the header. These lock files mark the passage as busy in some way and the presence of any of them will cause a different message to be displayed instead of the id string:

File name	Status	Activity
.sharp_lock	LOCKED BY SHARP	HRPT being SHARP processed
.coast_lock	LOCKED BY COAST	HRPT being 'coast adjusted'
.net_lock	LOCKED BY NETWORK	HRPT ingestion to passage from LAN

Some tools expand this facility by creating a *.id* file in a passage header containing their own message for two reasons:

1) to reserve the passage for their private use, eg while downloading data from tape to a passage. (It is always the presence of a *.id* file in a passage or slot which marks it as used).

2) to broadcast (via *monitor*) the on-going activity to users. A typical message might be "LOADING FROM TAPE".

So, there are two ways that *monitor* displays messages to users - by the presence of lock files certain messages are printed or, in their absence, by reading the ".id" file contents. Lock files are more specific to *display*, the *.id* files are a more general mechanism.

2) If a BATCH (see *dbp*) subdirectory exists in a passage header, the "@" symbol is displayed alongside the passage id message to show there is a batch request waiting to be serviced.
For example, if passage 2, called "N11 920330 060829", has a batch request pending, *monitor* will display:

```
Passage 2: N11 920330 060829 @
```

When the batch request has been serviced and the directory *RAWINFO/BATCH* deleted, the display will revert to:

```
Passage 2: N11 920330 060829
```

2) The right-hand display, entitled "SHARP PRODUCTS (SLOTS)", shows the state of the SHARP slots. If a slot is empty the corresponding information string is simply marked "< EMPTY >". If a slot contains a product, the program types out whatever message is found in the ".id" file. Although lock-files in slots do not affect the behaviour of *monitor* (unlike passages, see above), files named ".busy" are created from time-to-time which contain useful activity information.

If the slot contains a TOVS product as well as a SHARP one (detected by the presence of a TOVS file called *ANCILLARY*, the standard SHARP identifier string has the legend "+T" appended to it. Similarly, if a LEVEL 2 product is present in a slot (either 2A or 2B - detected by the presence of subdirectories LEVEL2A and/or LEVEL 2B), the string "+2A" or "+2B" is appended to the identifier string.

E.g.:

```
Slot 1: RE119109221326 (SHARP 1 only)
Slot 2: NY109108140613 +T (SHARP 1, TOVS)
Slot 3: DF109008140644 +2A (SHARP 1, 2A)
Slot 4: DL129209281607 +2B (SHARP 1, 2B)
Slot 5: MN119203300619 +T +2A +2B (SHARP 1, 2A, 2B,
TOVS)
```

3) The top (long) display which contains the SHARK icon and three buttons. These buttons are:

(QUIT!) - Kills *monitor* tool

(CLOSE) - Iconises *monitor* tool
(DELETE DATA) - When pressed, gives a menu of two options:

HRPT =>
SHARP =>

If the mouse pointer is moved onto one of the two small arrows (=>), a vertical list of numbers is presented to the user representing the numbers of the data sets (either HRPT or SHARP, depending which was chosen) present on the system. By clicking on one of these numbers the operator signals *monitor* that he/she wishes to delete that data set and all its subdirectories.

SEE ALSO: clp, clps, cls, clss, Ingest tools, shark, display

NAME: odh

PROCESSING PHASE:
SHARK

DESCRIPTION:
Asynchronous optical disk handler process for *shark.*

SOURCE TYPE:
C program

LOCATION: *CODE/ODISC*

INPUT PARAMETERS:
One optional command-line argument: [-debug]

ENVIRONMENT VARIABLES:
OHOME
OHOME_MNT
[OHOME1]
[OHOME_MNT1]

INPUT FILES:
<OD>/<PRODUCT-ID>/.verif
<OD>/<PRODUCT-ID>/.icon
<OD>/<PRODUCT-ID>/CAT
<OD>/<PRODUCT-ID>/LEADER
<OD>/<PRODUCT-ID>/IMAGE
<OD>/<PRODUCT-ID>/TRAILER
<OD>/<PRODUCT-ID>/VDF
<OD>/<PRODUCT-ID>/QUICKLOOK.DAT
<OD>/<PRODUCT-ID>/QUICKLOOK.PIC
slot#/.id
slot#/.icon
slot#/CAT
slot#/LEADER
slot#/IMAGE
slot#/TRAILER
slot#/VDF
slot#/QUICKLOOK.DAT
slot#/QUICKLOOK.PIC

OUTPUT FILES:
<Directory filename specified by GUI>
<OD>/<PRODUCT-ID>/.id
<OD>/<PRODUCT-ID>/.icon
<OD>/<PRODUCT-ID>/CAT
<OD>/<PRODUCT-ID>/LEADER
<OD>/<PRODUCT-ID>/IMAGE
<OD>/<PRODUCT-ID>/TRAILER
<OD>/<PRODUCT-ID>/VDF
<OD>/<PRODUCT-ID>/QUICKLOOK.DAT
<OD>/<PRODUCT-ID>/QUICKLOOK.PIC
<OD>/<PRODUCT-ID>/<TOVS FILES...>
slot#/IMAGE
slot#/TRAILER

slot#/VDF
slot#/QUICKLOOK.DAT
slot#/QUICKLOOK.PICslot#/.id
slot#/temp.icon
slot#/CAT
slot#/LEADER

/tmp/opdo<PID>
/tmp/opde<PID>
slot#/<TOVS FILES...>

NOTES: Contains the code which handles the optical disc. The main process, *shark* forks a child process to run *odh*. The two processes communicate through two pipes from the parent connected to the child's *stdin* and *stdout.*

The model is as follows. The operator initiates optical disk requests on *shark*'s menus which are then passed to *odh*. *odh* handles the requests silently and passes the results back to *shark* (using the pipes). *shark* then informs the operator of the success or failure of the operation.
Unfortunately, when a "Format New OD" request is given, *odh* needs to prompt the operator to turn the disk over so that it can check and format the other side. This means that *odh* includes a few SunView routines to create the necessary popup windows and dialogues.

odh recognises the following operation codes and data requests from its parent process. If a request is satisfied, the request string is sent back to the parent. If not, an 'F' (fail) is pre-pended to it, an error message appended to it and the whole lot sent to the parent. (NB All operations are performed on drive $OHOME unless specified).

Code	Data	Function	Example
P	NONE	Partition (format) the current OD	P
C	Drive names	Use named device for operations	C/dev/donU1,0 C/dev/donU1,0 /dev/donU2,0
X	NONE	Exit	X
S	slot name	Archive slot to OD	S/home/tiros/slot4
Q	OD dir disk dir	Copy a QL from OD dir to disk dir	QDL119205291254,/home/tiros
R	slot name OD dir	Retrieve a product from OD to slot	R/usr/tiros/slot3,DF119004261205
D	filename	List contents of OD to a file	D/home/tiros/.dir
I	NONE	Create any missing icon files on OD	I
O	OD dir		ODF119004261205

If it is called with the 'D' command code *odh* writes a summary of the space on the optical disk to the specified file. This listing has three entries in it, the OD label, the free space on the disk and a list of the identifiers already on the disk.

```
NYF92273S11B          #1:       OD label
10                    #2:       Space free on OD, in products
NY119209081411  -|
NY119209081415   |
NY119209081418   |
NY119209081550   |
NY119209081554   |
NY119209081558   |-- #3:       list of product ids on OD
NY119209161556   |
NY119209161600   |
NY119209161602   |
NY119209171401   |
NY119209171405   |
NY119209171410  _|
```

SAMPLE OUTPUT:
NOT AVAILABLE

SEE ALSO: shark, odh.remote

NAME: OD_ops

PROCESSING PHASE:
UTILS

DESCRIPTION:
Command-line menu of optical disk operations (using Dorofile)

SOURCE TYPE:
Bourne Shell script

LOCATION: BIN

INPUT PARAMETERS:
One command-line argument: optical disk's unit number.

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
[*/tmp/<OD-label>*]

NOTES: All options are fairly self-explanatory. Option 8 runs the Dorofile *testbios* program which performs numerous low-level tests on the optical disk. This should not be used on a disk which contains data. However, it is sometimes useful when configuring a new OD drive to see if it is responding.

Options 2 and 10 are useful for listing the contents of a disk. Option 7 reports the remaining free space in terms of the number of SHARP products that will fit onto the disk.

Option 5 can be used to retry to archive a SHARP product on OD - the original attempt may have failed because of a system crash.

Option 11 is the most useful operationally as it allows operators to access the QLs on an OD directly.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> OD_ops 1  (refers to /dev/donU1,0)

Please select operation:

1) Mount volume
2) Directory listing
3) Disk info
4) Disk usage
5) Manually archive a TIROS scene
6) Change a TIROS OD label
7) Show remaining space as TIROS slots
8) Run test program
9) Unmount volume
10) List SHARP details to disk file
11) Print QLs on OD
q) QUIT

Choice:
```

NAME: pass_info

PROCESSING PHASE:
UTILS

DESCRIPTION:
Examine HRPT header and data

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
A variable length list of command-line arguments which specifies the passages to examine. For example:

```
[shark@nimbus](usr/tiros/EXEC)> pass_info 1
[shark@nimbus](usr/tiros/EXEC)> pass_info 3 4 8
```

ENVIRONMENT VARIABLES:
EXEC_DIR - *pass_info* executes here.

INPUT FILES:
RAWINFO/rawtable

OUTPUT FILES:
<NONE>

NOTES: For each requested passage number, *pass_info* runs *scrheadaq* to dump header data, then gets the corresponding HRPT file name from the rawtable file and asks the user if they wish to examine the data. If they reply 'y', it calls the program *raw* to do the work.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> pass_info 1

==============================================================
=
PASSAGE 1:
N11 920330 060829
Contents of (rec. 1 of) File /usr/tiros/RAWINFO/header1/XHEADER.SCR

Satellite ID NOAA-11
Date of start of reception 1992/03/30
Time of start of reception 6:08:29.653
Date of end of reception 1992/03/30
Time of end of reception 6:21:25.987
No. of image lines received 4658
Clock indicator

11
920330
060829653
920330
062125987
```

```
4658
Examine raw data in /home/big1 (y/n) ?
 y

0
enter rec,i1,i2
1,1,10
*** satellite ***
dat0= 258 msec= 58672153
date= 258 time= 161752153
*** station ***
dat0= 11535099 msec= 8192292
date= 115350409 time= 21632292
0 10985data: 149 779 3 516 695 977 25 851 847 185 enter rec,i1,i2
0,0,0
=================================================================
=
```

SEE ALSO: scrheadaq, raw

NAME: print_OD_ql

PROCESSING PHASE:
UTILS

DESCRIPTION:
Selects SHARP QuickLooks on optical disk and send them to the laser printer.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument: optical disk's unit number. For example,
[shark@nimbus](usr/tiros/EXEC)> print_OD_ql 1 refers to /dev/donU1,0

ENVIRONMENT VARIABLES:
SHARK_QL_NCOPIES - number of copies of each QuickLook to print. The variable is modified by *print_OD_ql* and read by *tiros.*

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Uses Dorofile *womount* to mount the current OD on UNIX directory "/mnt". Next, it loops through the list of all valid products on the OD asking the operator whether or not to print the product's QL. If "yes", the software finds an unoccupied slot (one with no *.id* file) and reserves it by creating a *.id* file and writing the product's name into it. It prompts the operator to enter how many copies of the QL to print and resets the environment variable SHARK_QL_NCOPIES to the given value. The next step is to create a soft link in the slot for each product files on the OD and to call *tiros* with the following command;

tiros -batchprint <slot number>

which instructs *tiros* to do nothing more than print SHARK_QL_NCOPIES of the QL found in slot <slot number>.

Finally, the links and the .id file are deleted to leave the slot as it was before.
When all the products have been processed in this way, the script *woumount*s the disk and exits.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> print_OD-ql 2

Mounting OD /dev/donU2,0 on UNIX directory /mnt ..... done.
OD id: DLD91325S11A
Print QL of DL109111200735 (y/n/q) ? n
Print QL of DL109111200739 (y/n/q) ? n
Print QL of DL109111200743 (y/n/q) ? n
```

```
Print QL of DL109111210712 (y/n/q) ? n
Print QL of DL109111210716 (y/n/q) ? n
Print QL of DL109111210720 (y/n/q) ? n
Print QL of DL109111220649 (y/n/q) ? y
Searching for a free slot ... using slot10
How many QL copies [2] ? 1
/usr/tiros/slot10/QUICKLOOK.PS spooled to printer sparc at Fri Nov 20 14:57:39 1992
Print QL of DL109111220653 (y/n/q) ? n
Print QL of DL109111220657 (y/n/q) ? n
Print QL of DL119111201204 (y/n/q) ? n
Print QL of DL119111201208 (y/n/q) ? y
Searching for a free slot ... using slot10
How many QL copies [1] ? 3
/usr/tiros/slot10/QUICKLOOK.PS spooled to printer sparc at Fri Nov 20 14:58:11 1992
Print QL of DL119111201212 (y/n/q) ? q
Unmounting OD...
print_OD_ql: exiting at 14:58:32 on 20/11/92
```

SEE ALSO: tiros.

NAME: ql

PROCESSING PHASE:
SHARK

DESCRIPTION:
Generates QuickLook images.

SOURCE TYPE:
FORTRAN

LOCATION: *CODE/QLOOK/FTN*

INPUT PARAMETERS:
Two optional command line parameters:
1) Pixel number of the centre of the required extract
2) Scan-line number of the centre of the required extract.

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<HRPT data file>
EXEC/COASTLIN.DAT
EXEC/BOUNDARY.DAT
EXEC/LATLONG.DAT

OUTPUT FILES:
slot#/QUICKLOOK.DAT
slot#/QUICKLOOK.PIC
[*slot#/FQUICKLOOK.DAT*]
[*slot#/FQUICKLOOK.PIC*]

NOTES: *ql* generates a QuickLook from the image data in the current working directory.

If the program is called with two command line parameters it generates a full resolution QuickLook taking only a 512 pixel by 480 scan-line extract. The extract is centred on the point defined by the command line arguments. If a full resolution version is generated the alternative filenames are used.

SEE ALSO: disp_sharp, wrsharpaq, tape, ql_handler, shark, tiros dbp

NAME: ql_handler

PROCESSING PHASE:
SHARK

DESCRIPTION:
Asynchronous quicklook-process handler for *shark* GUI.

SOURCE TYPE:
C program

LOCATION: *CODE/QLOOK*

INPUT PARAMETERS:
One optional command-line argument: [-debug]

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Uses IPC pipes in exactly the same way as *odh*

Child of the *tiros* parent process. When a SHARP scene (4 files) has been successfully downloaded from magnetic tape *ql_handler* is called to create a Quicklook from the IMAGE file.
It requires two arguments from the parent:

1) the slot name from which to read the scene
2) the unique identification string created from the station name, satellite identifier, date and time of creation.

It can also accept two additional, optional arguments:

3) x (pixel) position of the centre of the area of interest
4) y (line) position of the centre of the area of interest

These last two arguments need only provided if the user wants a full-resolution QL of an extract of the current product. The central point of the extract is defined by the image coordinates (x,y).

Depending on the code/data passed down the IPC pipe, this process has very different behaviour; it can call a number of different QL generation processes. First it performs a "cd" to the slot in question, then it calls the correct QL process with the supplied parameters.

Code	Data	Function	Example
M	NONE	Create default QL	M
M	<x> <y>	Create full resolution QL	M 147 200
X	NONE	Exit	X
T	<x> <y>	Create TM QL	T 147 200
2	0	Create a Level 2A product	2 0
2	1	Create a Level 2B product	2 1
2	2	Create Level 2A and 2B products	2 2

'T' calls process "project.silent"
'2' calls process "level2_gen"

SEE ALSO: shark, tiros, ql. odh

NAME: raw_backup

PROCESSING PHASE:
UTILS

DESCRIPTION:
Manage HRPT data (passage) backups to tape.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
[TAPE] [HRPT_BLOCK_SIZE] (this is set in the .login)

INPUT FILES:
RAWINFO/rawtable
<Tape device>
[*RAWINFO/header[1-9]/XHEADER.SCR*]
[<HRPT file>]

OUTPUT FILES:
[*RAWINFO/header[1-9]/XHEADER.SCR*]
[<HRPT file>]

NOTES: Runs from a command-line menu that gives HRPT volume archive/retrieve and basic tape handling capabilities. Very straight forward.

Cairo has its own version of this called *hrpt_backup*.

SAMPLE OUTPUT: (Menu only)

```
Using /dev/nrst1 as device ($TAPE)

Please select operation:

        1) Dump a raw volume to Exabyte
        2) Read a raw volume from Exabyte
        3) Forward skip a raw volume on Exabyte
        4) Backward skip a raw volume on Exabyte
        5) Rewind Exabyte
        6) Take Exabyte offline
        q) Quit

        Choice ? q

Continue quit ? [n] y

raw_backup ended at Fri Nov 20 15:03:23 GMT 1992
```

NAME: scrheadaq

PROCESSING PHASE:
SHARP

DESCRIPTION:
Reads the first record of the scratch header file *EXEC/XHEADER.SCR* and lists the satellite identifier, the passage's start and stop dates and times, the number of line in the passage and the clock flag to the screen.

SOURCE TYPE:
FORTRAN program

LOCATION: *EXEC*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
EXEC/XHEADER.SCR

OUTPUT FILES:
<NONE>

NOTES: Called by *disp_sharp* as the first progam in the SHARP processing chain.

SEE ALSO: tloc2aq, wrsharpaq, shgridsaq, display, disp_sharp, dbp

NAME: sendcat.ftp

PROCESSING PHASE:
UTILS

DESCRIPTION:
Daily file transfer agent for LEDA catalogue files to reach EPOCAT from groundstations.

SOURCE TYPE:
Bourne shell script

LOCATION: *NETWORK/AUTOXCAT*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
LOGS/catalogue
[*LOGS/avhrrcat*]
[*LOGS/ispracat*]
[*LOGS/news*]

OUTPUT FILES:
EPOCAT::DISK$EPO_CAT_1:[CAT_NETWORK]
<STN><YYMMDD>.SHA
<STN><YYMMDD>.AVH
<STN><YYMMDD>.FFG

NOTES: Run nightly by *cron*(1) in the 'shark' account at all archive sites.

Transfers 'catalogue' files by *ftp*(1) from the *LOGS* directory to the EPOCAT machine to be inserted into the LEDA catalogue there. (old versions e.g. Maspalomas still use older X.25 file transfer programs)

Makes three attempts to copy the files before giving up if it cannot connect.

The following variables have to be set within the script. Their values will differ from site to site.

The variable 'STN' identifies the station the catalogue is being sent from. It must be set to the 2-letter code for that station (used in naming files - see EPOCAT Oracle administrator if you are unsure). eg Maspalomas would read,

```
STN="MP"
```

'FILE_TYPES' contains a space-separated list of the names of the files that are to be transmitted. For example, if a station wants to send the files *catalogue* and *avhrrcat* files from its *LOGS* directory to EPOCAT, then FILE_TYPES should be defined as,

```
FILE_TYPES="catalogue avhrrcat"
```

Lastly, if a site has more than one archiving SHARK (a SHARK that has its own OD drive and does archiving, and therefore has its own 'catalogue' file), then the hostnames of all the remote machines should appear in the variable 'REMOTE'. For example, at ESRIN there are two archiving SHARKs, "nimbus" and "zippy" and REMOTE is set as

```
REMOTE="zippy"
```

The machine with the network connection (where the *cron*(1) job is running) - in this case "nimbus", should not be included in the list, only the names of the remote machines.

SAMPLE OUTPUT: Mail output from *cron*

```
Date: Sat, 21 Nov 92 01:15:06 GMT
From: root@plod (System Administrator (Victor))
Message-Id: <9211210115.AA26916@plod.esrin.esa.it>
To: shark@plod
Subject: Output from "cron" command
Status: R

Your "cron" job

/home/tiros/NETWORK/AUTOXCAT/sendcat.ftp

produced the following output:

Checking host(s) zippy zorro for catalogue records..
rcp: /home/tiros/LOGS/catalogue: No such file or directory
No records found from zippy
rcp: /home/tiros/LOGS/catalogue: No such file or directory
No records found from zorro

Connection attempt #1 at 01:15:03 on 21/11/92
Connected to epocat.
220 LOCALHOST FTP server (UCX Version 1.3) ready.
331 Password required for KINGS.
230 User logged in
cd disk$epo_cat_1:[cat_network]
250 CWD command okay.
put no_updates RM211192.flg
200 PORT command okay.
150 Opening data connection for RM211192.flg (192.106.252.171,1058)
226 Transfer complete.
local: no_updates remote: RM211192.flg
93 bytes sent in 0.032 seconds (2.9 Kbytes/s)
quit
221 Goodbye.
ftp: Success..
<< Successful zorro to Frascati flag transfer at 01:15:08 on 21/11/92 >>
```

NAME: shark

PROCESSING PHASE:
SHARK

DESCRIPTION:
Front-end to *tiros* executable (SHARK GUI).

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
TAPE, PRINTER, OHOME, OHOME1

INPUT FILES:
LOGS/station_log

OUTPUT FILES:
<NONE>

NOTES: <NONE>

SEE ALSO: tiros, "SHARK Users' Manual"

NAME: shgridsaq

PROCESSING PHASE:
SHARP

DESCRIPTION:
Plots three overlays that show the coastlines, state boundaries and latitude/longitude grid. Each overlay is plotted as a bitmap the same size as the passage and written into a separate file.

SOURCE TYPE:
FORTRAN program

LOCATION: *EXEC*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
EXEC/PRODEF.DOC
EXEC/MISSION.DOC
EXEC/XHEADER.SCR
EXEC/LOCATE.DAT
EXEC/CSTLATU.ISS
EXEC/CSTLATH.ISS

OUTPUT FILES:
EXEC/BOUNDARY.DAT
EXEC/COASTLIN.DAT
EXEC/LATLONG.DAT
EXEC/GRIDLOG.DOC

NOTES: The third step in the SHARP processing chain. The program reads the details of the satellite, instrument and the bitmaps from the product definition file PRODEF.DOC. It reads the position of the image data from the file LOCATE.DAT. The overlays are written initially to tiled files named TILEGRID.DAT and then converted to raster format and written into the final output files.

SEE ALSO: tloc2aq, scrheadaq, wrsharpaq, display, disp_sharp, dbp

NAME: sharphandler

PROCESSING PHASE:
DISPLAY

DESCRIPTION:
Handler process to allow asynchronous SHARP operations from *display* GUI.
Communicates with the parent process through a standard 2-way read/write IPC pipe.

SOURCE TYPE:
C program

LOCATION: *DISPLAY/SHARPHANDLER*

INPUT PARAMETERS:
Two command-line arguments:
1) The file number of the IPC pipe read channel
2) The file number of the IPC pipe write channel

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: The function of this process is described in the section that describes *display*.

display passes the following data messages to *sharphandler*:

P<request>:	means 'Process SHARP' using arguments in <request>
X:	means 'Exit on receipt of this message'.

SAMPLE OUTPUT:
<NONE>

SEE ALSO: display, disp_sharp, tapehandler

NAME: sharp_exabyte.ops

PROCESSING PHASE:
UTILS

DESCRIPTION:
Exabyte handling for SHARP products.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
<NONE>

ENVIRONMENT VARIABLES:
[TAPE]

INPUT FILES:
[<Tape device>]
[*slot#/.id*]
[*slot#/VDF*]
[*slot#/LEADER*]
[*slot#/IMAGE*]
[*slot#/TRAILER*]
[*slot#/QUICKLOOK.DAT*]
[*slot#/QUICKLOOK.PIC*]

OUTPUT FILES:
[<Tape device>]
[*slot#/.id*]
[*slot#/VDF*]
[slot#/LEADER]
[slot#/IMAGE]
[slot#/TRAILER]
[slot#/QUICKLOOK.DAT]
[slot#/QUICKLOOK.PIC]

NOTES: The script *sharp_exabyte.ops* manages SHARP products archived on Exabyte tapes. It expects a SHARP product to consist of seven files. Therefore, if you choose to skip three products, the system skips 21 files.

This script keeps track of what products are on the current tape in a disk file called the TOC (table-of-contents) file. The following describes the use of this tool, the TOC file and some additional points of interest.

When *sharp_exabyte.ops* starts, it checks the file *~shark/exabyte.toc* which contains the scene identifiers of all products written to the current exabyte tape (in the drive) so far. If there are more than 35 products on the current tape, the script issues a warning advising that the tape should be "shipped".
When a tape is shipped, the script prints out the "table-of-contents" sheet and zeroes *~shark/exabyte.toc.* A copy of the TOC file is kept in

another directory until it has been copied to ESRIN by an automatic *cron*(1) process.

The only way that the table-of-contents file can accurately reflect the current exabyte contents is through maximum care and attention on the part of the operator(s) when dealing with exabytes. For instance, if the tape is positioned correctly to append the next product and the system goes down unexpectedly (and hence rewinds the tape), the onus is on the operator to re-portion the tape correctly. (There is a simple menu script to handle SHARP exabytes, explained later.)

sharp_exabyte.ops allows the following functions:

1) Forward skip SHARP product(s) on Exabyte
This allows you to input a number of products to forward skip.

2) Backward skip SHARP product(s) on Exabyte
This allows you to input a number of products to backward skip.

3) Position Exabyte at start of a SHARP product
Input the absolute product number from BOT and it will position the tape at the start of it. (Useful if position is lost for any reason.)

4) Rewind Exabyte to BOT
Rewinds tape to start.

5) Rewind and unload Exabyte
Ejects rewound tape.

6) Scan Exabyte for directory contents
Scans the entire tape, outputting the contents of each id file found. As it uses the mt(1) command to skip, there is no way to detect if it is skipping nothing (after the last product) - for this reason it is interruptible with Control/ C when it becomes apparent that this is occurring. This function is only useful if for any reason the table-of- contents disk file is incorrect and needs to be corrected by hand.

7) Copy SHARP slot from disk to Exabyte
Asks for the slot number on disk to copy from, then writes the product found therein to the current position on the tape.

8) Copy SHARP slot from Exabyte to disk
Asks for the slot number on disk to copy to, then reads the product from the current position on the tape into it.

9) List current Exabyte contents
Outputs the contents of the current table-of-contents (TOC) disk file.

0) Ship current Exabyte tape
This function takes the tape drive offline, prints out the current table-of-contents (TOC) disk file (to accompany the tape),

makes a copy for transmission to ESRIN and finally, resets the TOC file to 0 bytes.

SAMPLE OUTPUT: (Main menu only)

```
shp_exabyte.ops started at 15:03:57 on 21/11/92

                         SHARP EXABYTE HANDLING
                         ----------------------

        Please select operation:

                1) Forward skip SHARP product(s) on Exabyte
                2) Backward skip SHARP product(s) on Exabyte
                3) Position Exabyte at start of a SHARP product
                4) Rewind Exabyte to BOT
                5) Rewind and unload Exabyte
                6) Scan Exabyte for directory contents
                7) Copy SHARP slot from disk to Exabyte
                8) Copy SHARP slot from Exabyte to disk
                9) List current exabyte contents
                0) Ship current Exabyte tape
                q) Quit

        Choice ? q


Really quit ? [y]

sharp_exabyte.ops ended at 15:05:08 on 21/11/92
```

SEE ALSO: slot_to_exabyte, exabyte_to_slot, slot_in_window

NAME: slot_to_exabyte

PROCESSING PHASE:
UTILS

DESCRIPTION:
Upload a SHARP product (internal format) from magnetic disk to tape.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument: slot number

ENVIRONMENT VARIABLES:
[TAPE]
[REMOTE_TAPE_HOST]

INPUT FILES:
slot#/.id
slot#/LEADER
slot#/IMAGE
slot#/TRAILER
slot#/VDF
slot#/QUICKLOOK.DAT
slot#/QUICKLOOK.PIC

OUTPUT FILES:
<Tape device>

NOTES: Script to write the contents of a single slot (the files: *.id*, *LEADER*, *IMAGE*, *TRAILER*, *VDF*, *QUICKLOOK.DAT* and *QUICKLOOK.PIC*) tomagnetic tape, */dev/nrst1*. There is no *CAT* file as the scene is not yet archived.

The script assumes that the tape has been positioned correctly and that there is enough room on the tape for the slot. It does not check either condition.

If the environment variable TAPE is not set, the tape */dev/nrst1* is used. If the environment variable REMOTE_TAPE_HOST is set, the script uses the tape device on that machine using *rsh*(1). This works best when the two SHARK systems share filesystems through NFS, they can then share a tape device.

slot_to_exabyte is called by menu script *sharp_exabyte.ops*.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> slot_to_exabyte <slot number>
```

SEE ALSO: exabyte_to_slot, dbp, sharp_exabyte.ops

NAME: slot_in_window

PROCESSING PHASE:
UTILS

DESCRIPTION:
Checks if a SHARP product's centre point lies within a geographical area of interest.

SOURCE TYPE:
C program

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument: slot number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
EXEC/CURRENT_WINDOW.DAT
slot#/LEADER

OUTPUT FILES:
NONE.

NOTES: This tool identifies products whose centre points fall into a specific geographical region. In the course of standard batch processing/archiving operations *dbp* writes such products to magnetic tape by calling *slot_to_exabyte*.

slot_in_window expects the geographic area of interest to have been defined in the file *EXEC/CURRENT_WINDOW.DAT*. It compares the slot's centre point (found in the *LEADER* file) with each edge of the defined polygon to see on which side of the edge the point lies. If the slot's centre point is found to be inside the polygon the program returns the value 0, otherwise it returns a value 1.

The latitude and longitude (in degrees) of a slots's centre point are held as ASCII character strings in record two of the *LEADER* file starting at byte 53. Both occupy 16 bytes with eight places of decimals. Latitude is stored first.

The current 'area of interest' is defined as a polygon in the file *EXEC/CURRENT_WINDOW.DAT*. The file structure is as follows:

Line 1:	<integer: number of points in polygon>
Line 2:	<float: longitude, point 1>:<float: latitude, point 1>
Line 3:	<float: longitude, point 2>:<float: latitude, point 2>
Line n+1:	<float: longitude, point n>:<float: latitude, point n>

For example:

```
5
20.0:20.0
```

30.0:0.0
40.0:0.0
50.0:10.0
40.0:30.0

Exit status:

-1 System error
-2 Incorrect usage
 0 Slots centre point in window
 1 Slots centre outside window
 2 Window covers polar area.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)>  slot_in_window 3

/home/tiros/slot3/LEADER: No such file or directory


[shark@nimbus](usr/tiros/EXEC)> slot_in_window 10

Slot 10's centre point lies at:
Lat  =     33.03401947
Long =      27.19404221

[shark@nimbus](usr/tiros/EXEC)> echo $status

1
```

SEE ALSO: dbp

NAME: tape

PROCESSING PHASE:
SHARK

DESCRIPTION:
Magnetic tape handler. *tiros* child process.

SOURCE TYPE:
C program

LOCATION: *CODE/TAPE*

INPUT PARAMETERS:
One optional command-line argument: [-debug]

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: Uses IPC pipes in exactly the same way as *odh.*

SEE ALSO: ql_handler, odh, tiros, shark

NAME: tape.remote

PROCESSING PHASE:
SHARK

DESCRIPTION:
Allows *shark* to use another host's tape drive. Both hosts MUST share */home/tiros* through NFS.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One or two command-line arguments:
1) tape drive
2) [-debug]

ENVIRONMENT VARIABLES:
TAPE
REMOTE_TAPE_HOST

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: A device for allowing the *shark* GUI software to access another host's tape drive on the LAN. The two environment variables must be set on the calling host.

If the environment variable REMOTE_TAPE_HOST is set to the name of a networked host, *shark* runs *tape.remote* as a remote shell (*rsh*(1)) on the remote machine. In this case TAPE should be set to the name of the tape drive on the remote host.

SEE ALSO: shark, odh, tape, tiros

NAME: tbus_input

PROCESSING PHASE:
TBUS

DESCRIPTION:
Assists with entering TBUS messages into the local archive. Screen-based template tool for manual TBUS entry.

SOURCE TYPE:
C program

LOCATION: *BIN*

INPUT PARAMETERS:
Two command-line arguments:
1) Satellite Identification Number (SI)
2) Passage Number (PN)

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
TBUS/n<SI>YYMMDD.tbu
RAWINFO/header<PN>/tbus.dat

NOTES: The program draws a screen template into which the TBUS parameters can be typed. The form of the template reflects the normal layout of a TBUS message. The operator enters the appropriate information at the cursor. When one block is filled the cursor automatically skips to the start of the next. Where possible the program checks that the input is within the expected range, and if it is not, refuses to accept it.

Mistakes which are accepted by the program can be corrected once all the data has been entered. At that point the software prompts:

```
All fields OK (y/n) ?
```

If the reply is "n" the operator will be given the opportunity to edit any field.

If, at any stage, the screen becomes unclear, the operator should press the <Delete> key which causes the entire display to be redrawn.

SEE ALSO: find_tbus

NAME: tbus_tape_archive

PROCESSING PHASE:
TBUS

DESCRIPTION:
Copies entire TBUS archive to tape in OS-independent format.
Afterwards prints out a report of its actions.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
Prompts for:
1) Tape device
2) Satellite identification numbers

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:

[*TBUS/<SID><YYMMDD>.tbus*]	<SID> is the satellite identifier, N07, or N11, for example

OUTPUT FILES:
<Tape device>
/tmp/report.<PID> : spooled to printer, then deleted.
Numerous temporary files which are deleted on exit.

NOTES: A utility for saving all the TBUS details for any of the NOAA satellites onto tape for transfer to other machines running different operating systems.

Two files, a header and a data file, are written to the tape for each satellite specified in the command-line. The program first copies the names of all TBUS files for a single satellite into one file. This becomes the 'header' file, padded to 80-byte records.
Next all the files listed in the header file are merged into a single file with a blank line (80 spaces) separating each TBUS message. This is the data file.

Neither file contains any carriage control characters but, when they are written to tape, the output block size is set to 80, so an 'End of Record' marker is written after each 80 bytes.

NAME: tirant

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
Reads Antarctica HRPT files (passages) from local tape drive. Antarctica HRPT are in the standard format expected by SHARK, so that no reformatting is needed on ingestion.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument:
<tape device> (For example, st20, the script adds "/dev/nr").

ENVIRONMENT VARIABLES:
HRPT_BLOCK_SIZE

INPUT FILES:
<Tape device>
RAWINFO/rawtable

OUTPUT FILES:
RAWINFO/header[1-9]/XHEADER.SCR
RAWINFO/header[1-9]/.id
<HRPT file>

NOTES: Script that downloads raw HRPT data written by the Antarctic SHARK on magnetic tape. The script first finds a free passage on the magnetic disk, locks it and uses *dd*(1) to copy the header file from tape to disk. It then copies the raw data into the passage and makes the *.id* file. Finally the program prompts the operator whether to download a further data set or to quit.

SEE ALSO: dd(1)

NAME: tirnairobi

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
Download HRPT data from Niamey or Nairobi to passage. It is becoming the model for all new *tir** tape/network ingest programs. Nairobi/Niamey HRPT records come as (logical) 22016 bytes and (physical) 11008 bytes. This makes two reads per logical record - slow. Then it must be 'massaged' into the standard format and expanded to fill the 22528 byte buffer size.

SOURCE TYPE:
C program

LOCATION: *EXEC*

INPUT PARAMETERS:
Two command-line arguments:
1) <tape device>
2) <passage number>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<tape device>

OUTPUT FILES:
RAWINFO/header[1-9]/AQTIM.DAT
<HRPT file>

NOTES: Method:

```
Get year of pass from user
Loop
    Read in 22016 bytes from tape in two reads of 11008
    Swap all byte pairs
    Write station time field as satellite time
    Compute and inject calibration data
    Replace the first five I*2s (10 bytes) with six I*2s (12
    bytes) (effectively shift data up by one I*2)
    Write 11264 I*2s (22528 bytes) to raw partition
Until input EOF
Write AQTIM.DAT file into passage header area.
```

When it starts, *tirnairobi* opens the tape device and then keeps reading until it finds a record of the correct size.

A logical read consists of two physical tape accesses and some data pre-processing to get it into a standard format:

```
swap all byte pairs
sort it from band- to pixel-interleaved
shift the pixels to a different position
```

Now it searches for a healthy scan-line time to use as a first (reference) line. It does this by reading a line, holding it, reading the next and comparing the times. As soon as a sequence of two lines yields a time delta of 166/7 milliseconds it can start to write output data. This is done because often data at the start of a pass are noisy and cannot be processed.

From here on it reads the raw data, calibrates it and writes it out to disk.

Additionally, it keeps hold of the previous line's time for constant checking with the current one and watches the number of bad lines encountered. If more than 16 are encountered sequentially, the program complains and exits, writing the last good time found to the *AQTIM* FILE (i.e. the one before the bad sequence). In this way, the last line written to disk is guaranteed to be good, a safeguard against processing problems later on.

As Nairobi and Niamey data often come as multiple HRPT volumes on a single tape, some tape handling has been built in to cope with the following problem. If the ingest of HRPT data finishes prematurely (see discussion above about bad line sequences), the read/write head will not be positioned at the end of file proper. This causes the next *tirnairobi* (called in a loop by *tirnia*) to fail as it expects to be file-mark aligned.

If it is reading from magnetic tape the program automatically skips to the end of file before exiting.

SAMPLE OUTPUT:

```
[shark@nimbus](usr/tiros/EXEC)> tirnairobi st20 3

Enter acquisition year as YYYY:
1991

First good line search...
First good line search...
jcondate = 1991102
sat_id = 11
3) Unacceptable delta 1789 - skipping line
26) Unacceptable delta -333 - skipping line
4902) Unacceptable delta -623787 - skipping line
4902) Unacceptable delta -3789 - skipping line
4902) Unacceptable delta -345763 - skipping line
4902) Unacceptable delta -623787 - skipping line
4902) Unacceptable delta -367880 - skipping line
4902) Unacceptable delta 36873873 - skipping line
4902) Unacceptable delta -623787 - skipping line
4902) Unacceptable delta -623787 - skipping line
4902) Unacceptable delta -623787 - skipping line
tirnairobi: 4891 lines copied to raw pass.

... Tape is EXABYTE, so skipping to end of file (please wait) ...
```

SEE ALSO: tirnia

NAME: tirnia

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
Program manager for *tirnairobi*. Downloads Niamey/Nairobi HRPT files (passages) from local tape to magnetic disk.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument:
<tape device> (For example, st20, the script adds "/dev/nr").

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<Tape device>
RAWINFO/rawtable

OUTPUT FILES:
RAWINFO/header[1-9]/XHEADER.SCR
RAWINFO/header[1-9]/.id
<HRPT file>

NOTES: Simple loop script for *tirnairobi*.
For each ingestion, it locks the lowest available passage header, ingests the data (with *tirnairobi*) into the corresponding HRPT file, creates a header for it (see *makehdr*) and then prompts whether to repeat the cycle or not.

SAMPLE OUTPUT:
<NONE>

SEE ALSO: tirnairobi, makehdr, find_tbus

NAME: tiros

PROCESSING PHASE:
SHARK

DESCRIPTION:
SHARK GUI process

SOURCE TYPE:
C program

LOCATION: *CODE*

INPUT PARAMETERS:
Three optional command-line arguments:
[-debug] - for verbose activity logging to stdout
[-batch] - for non-GUI, batch archiving
[-batchprint] - for non-GUI, batch printing

ENVIRONMENT VARIABLES:
OHOME, OHOME1, OHOME_MNT, OHOME1_MNT, TAPE, PRINTER, SHARK_QL_NCOPIES, EXEC_DIR, SLOTS_PARTITION, TAPE_ORDER_CONT, TAPE_ORDER_TOTAL, FORMAT_ODS, LEVEL2_DIR, REMOTE_TAPE_HOST, REMOTE_OD_HOST

INPUT FILES:
Too numerous to list..

OUTPUT FILES:
Too numerous to list..

NOTES: On startup, *tiros* forks three child processes which it uses to perform asynchronous operations on its behalf. Once forked, 2-way pipes are established between the parent and each child for IPC communications. These child processes are used to run three handler programs:

odh -	the optical disk handler,
tape -	the magnetic tape handler, and
ql_handler -	the QuickLook handler.

The first two are peripheral handlers that, in response to user requests from the GUI (*tiros*), perform appropriate operations on the device in question. For example, list the contents of a directory on the optical disk, or copy a product from a slot to the magnetic tape

The third provides a 'wrapper' for the execution of one of several QL-generation programs - this wrapper receives the request from the parent, changes to the appropriate directory, forks the program, waits for it and catches its exit status. This is then interpreted and an appropriate message sent back to the parent.

All three child processes can only execute a single request at a time, so requests are simply queued in the child process's input pipe until such time as the process is ready to accept them. The parent provides an additional locking mechanism that allows only one request at a time for the *odh* and *tape* processes.

Having started the child processes, *tiros* presents the operator with a full GUI (unless running in either 'batch' or 'batchprint' mode in which case all communication is provided through command-line arguments, *stdin/stdout* and exit status) through which the operator can process SHARP products.

If started with the optional '-debug' flag, *tiros* dumps large amounts of tracing information to *stdout*; it also passes the same flag to the child processes to command them to do the same.

If run with the optional '-batch' flag, *tiros* runs in batch archiving mode which can only be used to archive a single product. This mode is used by *dbp* to archive single slots to the optical disk. To assist it in this task, *odh* is forked to do the actual archiving, while the parent controls it and, if it is successful, also prints the required number of QLs and updates the catalogue file. Three extra command-line arguments must be supplied if '-batch' is used:

slot# -	slot in which to find product.
OD label -	label of current OD. Used in catalogue update.
Quality flag -	an integer which describes the user's rating of the raw data during the *display* process; also used in catalogue update.

Once the slot is successfully archived, *tiros* updates the catalogue file, prints the QLs, signals *odh* to exit and then exits with status 0. If is fails to archive, no update is made to the catalogue file, no QLs are printed and *tiros* exits with a non-zero status.

The '-batchprint' flag is only used to make QL printouts of a slot; the requirement for this comes in batch operations when a product is geographically selected and then copied to tape - in this case, an extra copy of the QL is deemed necessary. A second argument, 'slot#' is also passed in this case to signal to *tiros* which slot the product is in.

OTHER MODES
SHARKs on a LAN can share resources through NFS and by setting the following environment variables:

REMOTE_TAPE_HOST - tells a SHARK system that another, connected SHARK has a magnetic tape drive to share. The device name of the remote tape drive is defined by the environment variable 'TAPE'. If 'REMOTE_TAPE_HOST' is defined *tiros* forks the *tape.remote* process rather than *tape.*

REMOTE_OD_HOST - tells a SHARK system that another, connected SHARK has an OD drive to share. The device name and mount point of the remote optical disk drive are defined by the environment variables OHOME and OHOME_MNT. If 'REMOTE_OD_HOST' is defined *tiros* forks the *odh.remote* process rather than *odh.*

SEE ALSO: "SHARK Users' Guide", odh, shark, odh.remote, tape, tape.remote, ql_handler, ql, dbp, display, print_OD_ql

NAME: tirsa

PROCESSING PHASE:
HRPT-INGESTION

DESCRIPTION:
Manager script for program *make_sa_hrpt*. Downloads South African station HRPT on CCT into a passage in standard format.

SOURCE TYPE:
C-shell script

LOCATION: *BIN*

INPUT PARAMETERS:
One command-line argument:
<tape device> (For example, st20, the script adds "/dev/nr").

ENVIRONMENT VARIABLES:
TAPE

INPUT FILES:
<tape device>
RAWINFO/header[1-9]/.id

OUTPUT FILES:
./band[1-5]
RAWINFO/header[1-9]/.id
RAWINFO/header[1-9]/AQTIM.DAT
<temporary file> (deleted on exit)

NOTES: Removes the five AVHRR band files after *make_sa_hrpt* has read them.

SEE ALSO: find_tbus, makehdr, make_sa_hrpt, dd(1)

NAME: tloc2aq

PROCESSING PHASE:
SHARP

DESCRIPTION:
Creates the earth location data file for a passage. Calculates the latitude and longitude of tie points at every 32nd pixel on every 16th scan-line across the passage. At the same time it calculates the sun and satellite pointing angles at each of the tie-points.

SOURCE TYPE:
FORTRAN program

LOCATION: *EXEC*

INPUT PARAMETERS:
One command line parameter: passage number

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
EXEC/PRODEF.DOC
EXEC/XHEADER.SCR
EXEC/SHARPREQ.DOC

OUTPUT FILES:
EXEC/LOCATE.DAT

NOTES: <NONE>

SEE ALSO: wrsharpaq, scrheadaq, shgridsaq, display, disp_sharp, dbp

NAME: tapehandler

PROCESSING PHASE:
DISPLAY

DESCRIPTION:
Handler process to allow asynchronous SHARP operations from *display* GUI. Communicates with the parent process through a standard 2-way read/write IPC pipe.

SOURCE TYPE:
C program

LOCATION: *DISPLAY/TAPEHANDLER*

INPUT PARAMETERS:
Three command-line arguments:
1) file number of the IPC pipe read channel
2) file number of the IPC pipe write channel
3) <tape device>

ENVIRONMENT VARIABLES:
<NONE>

INPUT FILES:
<NONE>

OUTPUT FILES:
<NONE>

NOTES: The behaviour of this process is mostly described in *display*.

display passes the following data messages to *sharphandler*.

D<request>:	means 'Download HRPT' using arguments in <request>
X:	means 'Exit on receipt of this message'.

SEE ALSO: display, disp_sharp, sharphandler

NAME: wrsharpaq

PROCESSING PHASE:
SHARP

DESCRIPTION:
Writes a SHARP and, if required, a TOVS logical volume to a slot.

SOURCE TYPE:
FORTRAN program

LOCATION: *EXEC*

INPUT PARAMETERS:
One command line parameter: passage number

ENVIRONMENT VARIABLES:
TOVS_ON

INPUT FILES:
EXEC/PRODEF.DOC
EXEC/MISSION.DOC
EXEC/XHEADER.SCR
EXEC/SHARPREQ.DOC
EXEC.LOCATE.DAT
EXEC/BOUNDARY.DAT
EXEC/COASTLIN.DAT
EXEC/LATLONG.DAT

OUTPUT FILES:
slot#/VDF
slot#/LEADER
slot#/IMAGE
slot#/TRAILER
slot#/SHARPLOG.DOC
[<TOVS FILES>]

NOTES: Performs the final stage of the SHARP processing chain.

If the environment variable TOVS_ON is set to 'YES' it also writes the TOVS files to the slot.

SEE ALSO: tloc2aq, scrheadaq, shgridsaq, display, disp_sharp, dbp

Index