SSSC Implementation Version 1.0

J. Aloise, K. Leibrand, J. Suryanarayanan

SOI-TN-109

1 November 1993

1 Introduction

This document describes the Version 1.0 implementation of the Stanford SOI_MDI Science Support Center (SSSC). The broad SSSC functions are to receive SOI_MDI telemetry data, process it according to science goals, and maintain a catalog of data products for further internal processing or external distribution.

The SSSC consists of three major subsystems:

- Pipeline Scheduler (PS): Determines the data processing requirements and configures and executes Pipeline Execution tasks to perform them. This subsystem has not been implemented in SSSC Version 1.0 and is not discussed here. Its design is presented in "Preliminary SSSC Conceptual Design" SOI-TN-105.
- Pipeline Execution (PE): Performs the actual data processing in cooperation with DSDS, as directed by PS.
- Data Storage and Distribution System (DSDS): Receives, stores, catalogs, archives and distributes data products and assigns output storage for PE.

2 Overview

Figure 1 presents the SSSC Version 1.0 block diagram. Each box represents a Unix process with the exception of "oracle" which is a set of processes run by the ORACLE database program V6.0. Furthermore, with the exception of "oracle", the processes are run as Parallel Virtual Machine (PVM V3.0) tasks. PVM is a software package that allows a heterogeneous network of computers to appear as a single concurrent computational resource. The tasks exchange information via PVM messages in XDR encoding. The ORACLE interface is via SQL commands via SQL*Net. Figure 1 can be independently replicated for each Unix user id.

Table 1 presents a summary of the processes. The remainder of the document discusses each process in detail.

The SSSC Version 1.0 source and binary code are under /home/soi/beta/src/pipe.

FIGURE 1. SSSC Diagram

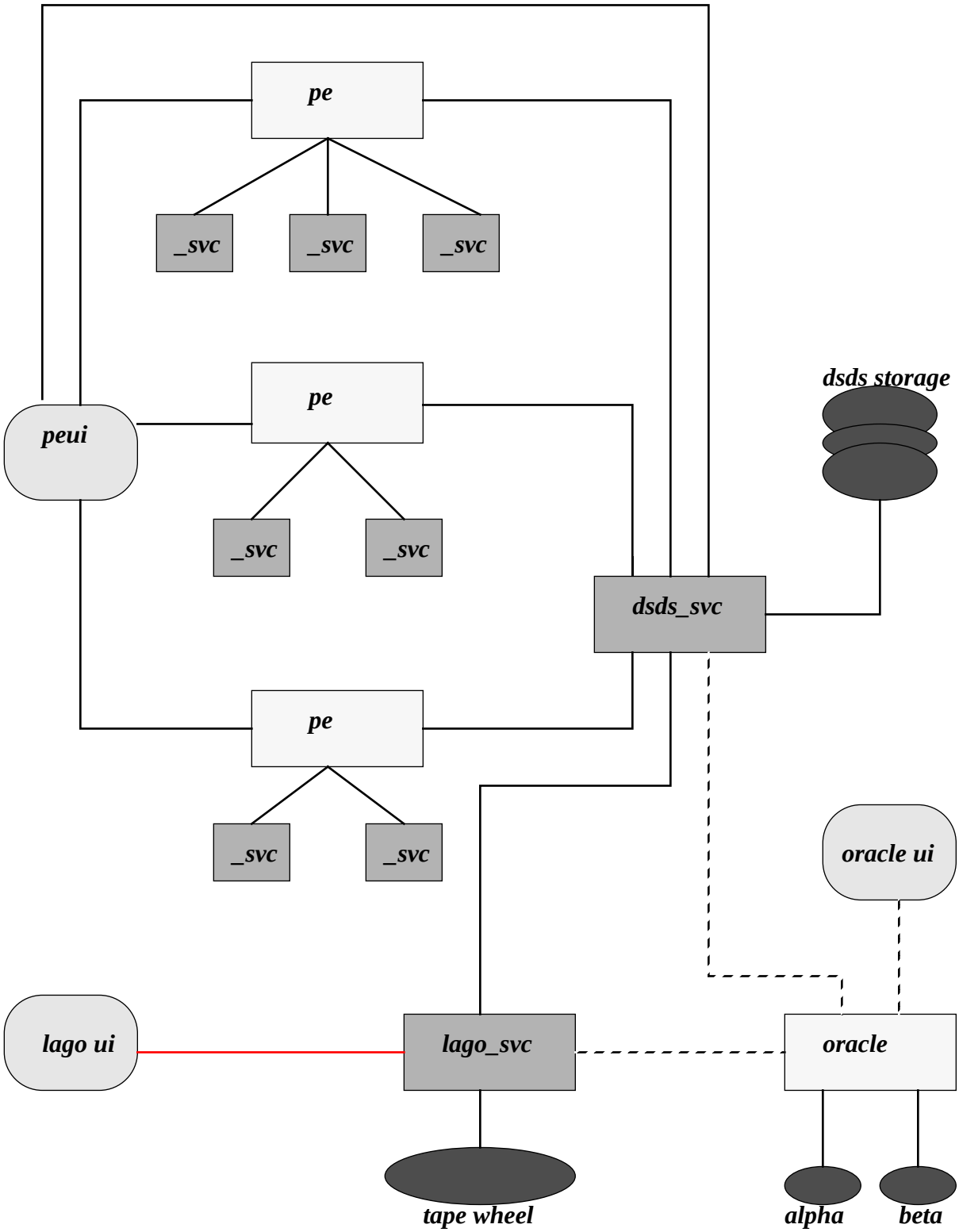


TABLE 1. SSSC Processes

peui	The pe user interface. Launches any number of pe processes and displays their ongoing statuses.
pe	Pipeline execution. Launches the computational servers and interfaces with the DSDS server.
_svc	Any number of computational processes serving a pe.
dsds_svc	Provides DSDS services to the pe clients.
lago_svc	Provides tape services on the Lago Datawheel to dsds_svc.
lago_ui	The lago_svc user interface. Not implemented in Version 1.0
oracle	The set of processes making up the ORACLE database.
oracle ui	A SQL*Forms process to view or update the database.

3 Pipeline Execution User Interface (peui)

This is a point and click TAE+ V5.1 program to give a user interface to pe. It will allow a selection of a map file which defines the pe processing, a display of this map file, and a launch of pe with this map file specified. It also allows for a dsds_svc status window to be displayed. Currently there are a maximum of 7 pe's that can be running simultaneously. Any attempt to run more will generate a modal error message. The number of pe's can be increased by a compilation parameter and defining additional pe status windows. Figure 2 presents the main window. Figure 3 presents the map file display window. Figure 4 presents the pe status window. Figure 5 presents the dsds_svc status window.

When pe is launched by peui it is given a "-t" argument which directs it to send its output to peui. Pe can be run stand-alone without the "-t" argument. When the Display dsds button is activated, a message is sent to dsds_svc that causes it to send its output to peui, in addition to its log file. When the dsds_svc status window is closed, a message is sent to dsds_svc to discontinue sending its output to peui. The Exit button will display a warning message if you attempt to exit while a pe is still active.

Peui is found in /home/soi/beta/src/pipe/tae/peui.

FIGURE 2. **peui Main Window**

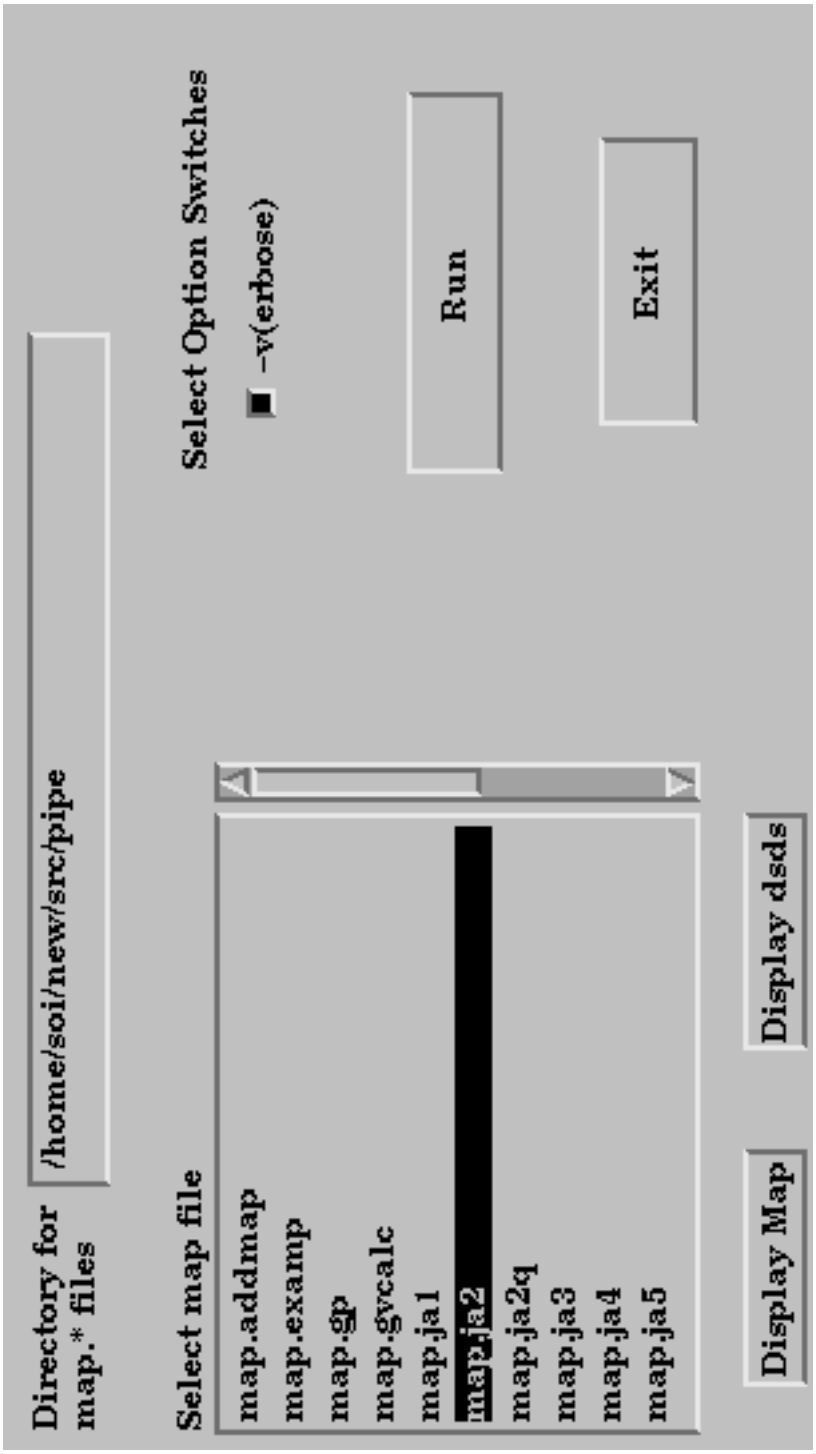


FIGURE 3. peui Map File Window

FIGURE 4. peui pe Status Window

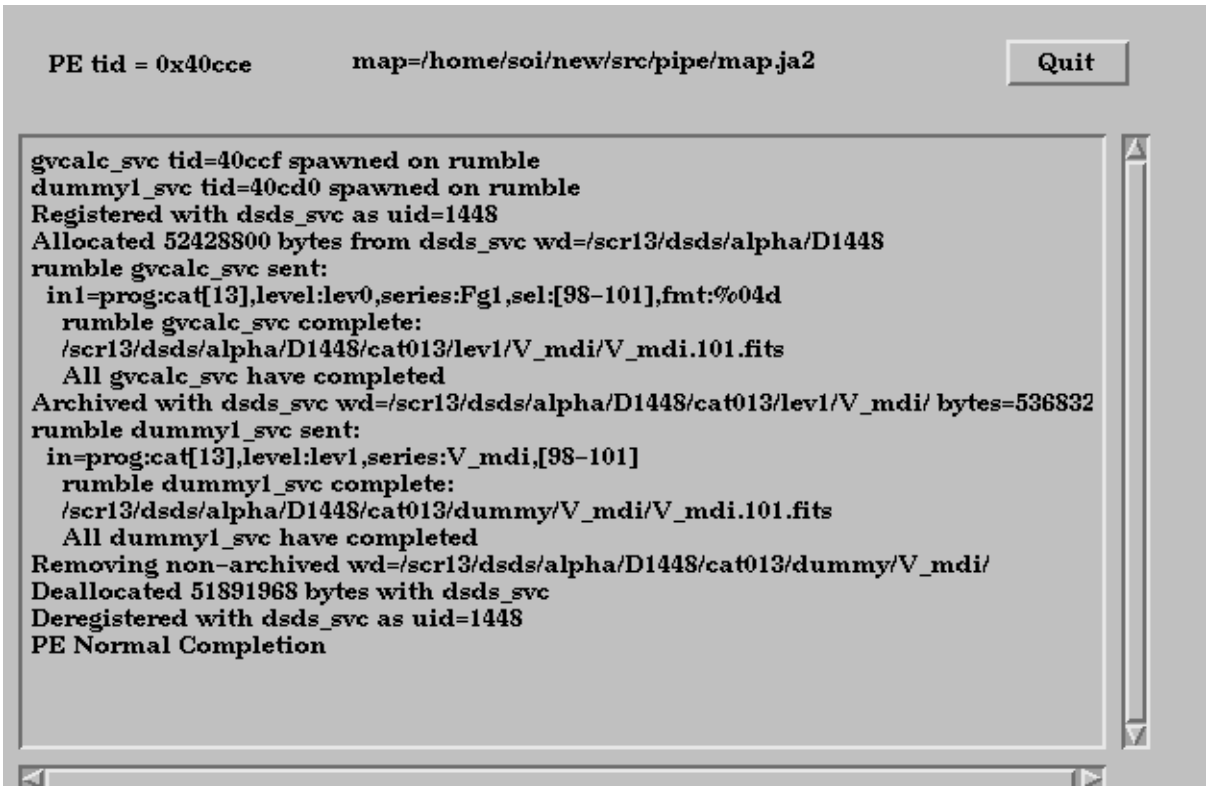
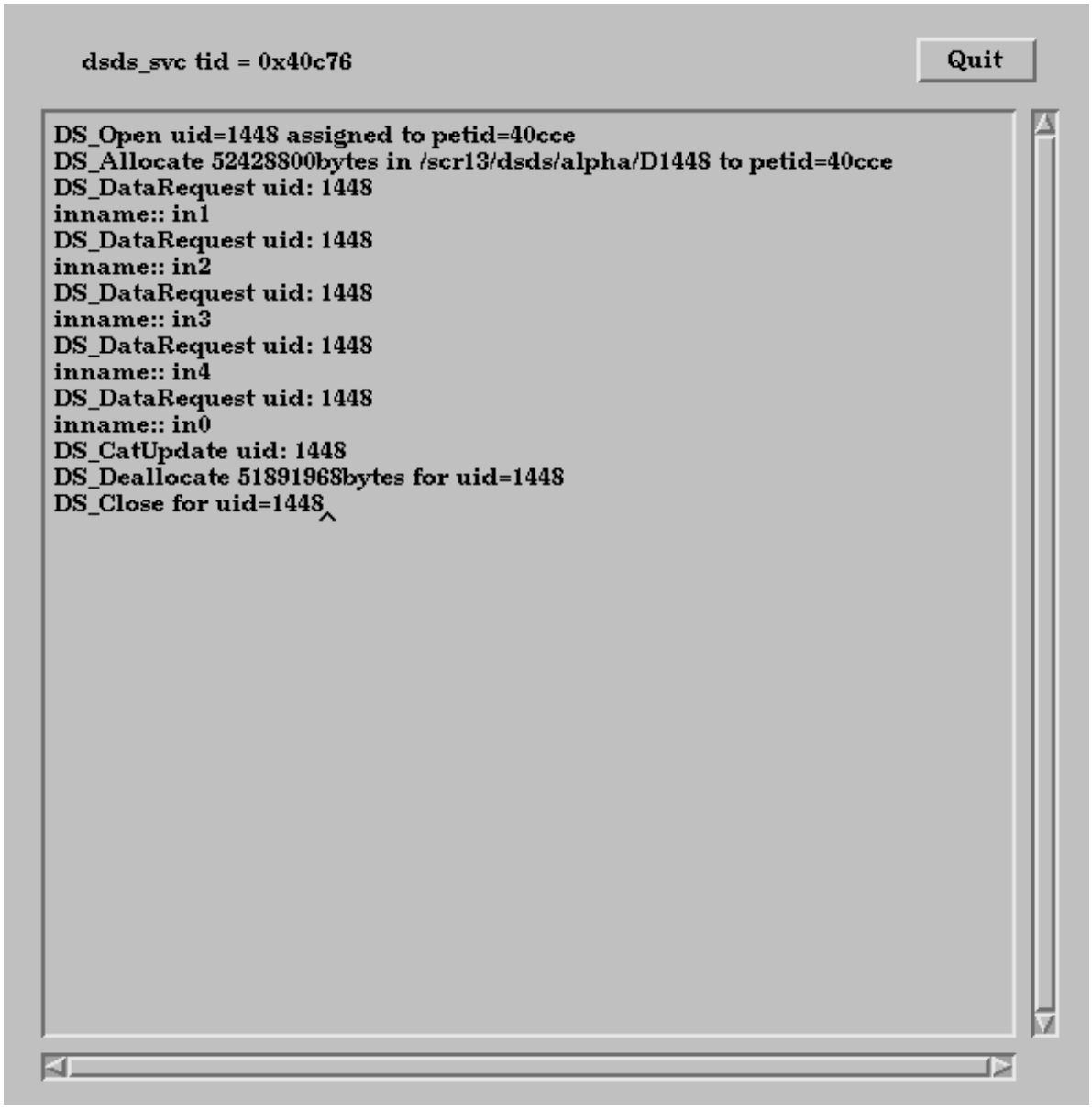


FIGURE 5. peui dsds_svc Status Window



4 Pipeline Execution (pe)

This is the process that manages a group of computation servers that take a set of input data and transforms it into various output products. The computational servers form a pipeline in that the output of one server is available as input to the next server on the pipeline. Pe sequences the data through the computational servers on the various host machines.

Pe provides input data location, output data storage and output data cataloging on behalf of the computational servers via pe's interface with dsds_svc. Pe can also run stand-alone without dsds_svc. In this case, input and output name resolution is accomplished through the use of environmental variables as described in 4.1.

4.1 Pe Map File

When any pe is launched, it is supplied the name of a map file that defines its processing. The map file specifies the computational servers to be run, the hosts they are to be run on, the sequence in which to run them, the datasets they are to process, and any DSDS support for dataset name resolution or data archiving. (For a discussion of dataset names see "Naming of Datasets for SOI" SOI-TN-104.)

Figure 6 shows the complete definition of a map file along with a sample specification.

Figure 6 should be self describing except in its use of environmental variables. An environmental variable, of the name of the "prog:" specification in the dataset names, must be set to map the dataset names into a directory structure. This variable is used for input dataset name resolution when dsds_svc is not used for input name resolution, i.e d=1 is not specified. The variable is used for output name resolution in the case of both DSDS assigned output storage and local output storage.

In Figure 6, the environmental variables are: "setenv dbase /mnt/jim/data" and "setenv cat {dbase}/{prog}{#%03d#prog}/{level}/{series}/". The {} tokens are taken from the dataset name, except {dbase} which is specified, or resolved by dsds_svc when DSDS output is requested, i.e. when DSDSOUT=n is given. The request for DSDS output will override any specified {dbase}. The {dbase} is basically the root of the output directory tree.

For example, the following dataset names will resolve for the following cases:

in1=prog:cat[13],level:lev0,series:Fg1,sel:[98],fmt:%04d

out=prog:cat[13],level:lev1,series:V_mdi,[]

Local input: /mnt/jim/data/cat013/lev0/Fg1/Fg1.0098.fits

DSDS input: Wherever dsds_svc tells us it is.

Local output: /mnt/jim/data/cat013/lev1/V_mdi/V_mdi.98.fits

DSDS output: /scr13/dsds/alpha/D1417/cat013/lev1/V_mdi/V_mdi.98.fits

where {dbase} was resolved by dsds_svc to /scr13/dsds/alpha/D1417.

FIGURE 6. PE Map File

```
#
#This is a text file that specifies a pipeline to be run by PE.
#
#Control Statements:
#DSDSOUT=n
#       DSDS will be asked to allocate n megabytes for the output of this pipe.
#       The output directory that DSDS assigns can be used in the name template
#       env variable “prog:” by specifying {dbase}. Notice that if you do not
#       specify d=1 (see below) for a server, but did specify DSDSOUT then you
#       must set a default {dbase} so that the name template will also map the
#       input.
#ELBOW=1
#       Indicates that the next server is to wait for all previous servers to
#       complete before starting.
#ONESERVER=1
#       Indicates to only run one copy of the next server. This server will
#       handle all the input files itself. Note that this also implies
#       an ELBOW=1 because all the previous files must be available.
#!
#       A shell escape command. Any setenv is intercepted and interpreted
#       locally for the PE environment. No metachars for a setenv.
#Line Tokens:
#p=     The name of the server process to run.
#h=     The host names that all the servers are to run on. Multiple copies of
#       the servers can be run on the same host by specifying the host name
#       more than once.
#d=     A non-zero indicates that this servers input will be obtained from DSDS.
#       Otherwise a name template, found in the env variable of the name of the
#       “prog:” in the ds name, will be used to determine the input directory.
#a=      A non-zero indicates that this server’s output is to be archived.
#       This will be a nop unless DSDSOUT=n is specified.
#in=    The dataset name for the pipe’s input data. This isn’t necessarily
#       called “in”, but whatever the server’s input ds argument is called.
#arg=   Optional argument name and value to override any default argument
#       values for a server.
#
#Any control statement must be on a separate line.
#Any non-control line must start with p= ,and must be a separate line.
#The very first p= line must contain a h= .
#Any p= line may optionally contain an a=1.
#Any p= line may optionally contain an d=1.
#Any p= line may optionally contain any argument value for the server.
#Any line, except a “!” line, may be extended by ending it with “ \”.
!setenv dbase /scr2/jim/optest
```

```
!setenv cat {dbase}/{prog}{#%03d#prog}/{level}/{series}/
DSDSOUT=300
p=remap h=quake  \
        in=prog:cat[19],level:Image,series:V_mdi,sel:[8-107],fmt:%03d \
        out=prog:cat[19],level:Image,series:V_heliomap,sel:[],fmt:%03d
p=runmean  \
        in=prog:cat[19],level:Image,series:V_heliomap,sel:[],fmt:%03d \
        out=prog:cat[19],level:Image,series:V_runmean,sel:[],fmt:%03d
p=subtractmap \
        in=prog:cat[19],level:Image,series:V_heliomap,sel:[],fmt:%03d \
        out=prog:cat[19],level:Image,series:V_detrend,sel:[],fmt:%03d \
        runmean=prog:cat[19],level:Image,series:V_runmean,sel:[],fmt:%03d
p=shtfft  \
        in=prog:cat[19],level:Image,series:V_detrend,sel:[],fmt:%03d \
        out=prog:cat[19],level:Image,series:V_mlat,sel:[],fmt:%03d
p=dotprod a=1 \
        in=prog:cat[19],level:Image,series:V_mlat,sel:[],fmt:%03d \
        out=prog:cat[19],level:Image,series:V_dotprod,sel:[],fmt:%03d \
        masks=/scr2/jim/optest/cat019/Image/LM_masks.fits \
        MDEGEN=1 SN_BLOCK=100
p=tilefft_slice a=1 \
        in=prog:cat[19],level:Image,series:V_dotprod,sel:[],fmt:%03d \
        out=prog:cat[19],level:Image,series:V_spectrum,sel:[],fmt:%03d \
        L_BLOCK=10
```

4.2 Pe Implementation

Pe is spawned as a PVM task by peui and given a map file name argument. The map file is read in its entirety to configure the pe. Any syntax error will be reported and the pe will abort. The map file is not like a script file that is interpreted as you go, therefore any shell escape commands will all be executed at the start of pe. Pe will spawn all the given computational servers as PVM tasks on all the given host machines. Multiple servers of the same type can be spawned on the same host by specifying the host name multiple times.

If pe requires dsds_svc services, and one is not already running, pe will spawn it. The dsds_svc is spawned with the name of the database to connect to for this pe run. The database name is gotten from the DBNAME environmental variable that is passed to pe when it is spawned by the peui, or from the environmental variable or overriding db= switch on the command line when pe is run standalone.

Pe will register with dsds_svc via a PE_DSDS_REGISTER message (see Appendix A), if the map file indicates that there is input and/or output with DSDS. The PE_DSDS_REGISTER returns a unique id (uid) that is guaranteed to be unique for all runs of pe ever. All further requests to dsds_svc identify the requesting pe by this uid handle.

If output is to be to DSDS storage, then pe sends a PE_DSDS_ALLOCATE message with the number of bytes to allocate as given in the map file. Dsds_svc returns the required stor-

age from one of its dedicated storage partitions, appended with a directory name containing the uid. For example, for uid 57 the dsds_svc would return an output working directory such as /scr13/dsds/alpha/D57. This isolates any pe output from any other pe output. This directory is read/write for pe. As pe runs it will do any mkdir's, according to its output dataset names and "prog:" environmental variable, under this assigned wd.

If pe is to resolve any input dataset names, it will send a PE_DSDS_DATAREQ message with the dataset name and will be returned the wd and first serial number (fsn) and last serial number (lsn) for this data. This wd will be assigned to pe as read only. Pe will distribute this data to the next computational server in the pipe, dividing the data equally among the number of servers of this type running on the various hosts. When a server completes, its output data is available to the next server in the pipe. This server will be called on the same host with the same data range as the completing server. There are control directives for the map file to tell pe to wait until all servers of a type are complete if the next servers require access to a data range outside a previously completing server (ELBOW=1). Also there is the ability to run only one server of the next type if it requires exclusive access to all of the previous server's output (ONESERVER=1).

If indicated in the map file (a=1), when a computational server completes, pe will catalog its output dataset name with dsds_svc via a PE_DSDS_CATUPDATE message.

When pe sends a message to a server and does not get a response within a timeout , it will check that the server is still running. If the server has exited, pe will report the missing server's name, host and task id, and the pipe will be aborted . Upon pe abort, or normal completion, pe will delete all directories that it created that were not cataloged in DSDS (i.e. a=1 in map file). Pe will return all unused allocated storage via a PE_DSDS_DEALLOCATE request to dsds_svc. Pe will deregister with dsds_svc via a PE_DSDS_DEREGISTER. Pe will kill all of its spawned servers, except dsds_svc, and exit.

Pe is run standalone, without peui, as follows:

pe [-v] [db=database_name] map=map_file_name

Currently there are two database names, alpha and beta. Alpha is on aditya and beta is on quake. They are identically defined databases.

4.3 PVM Daemons

In order for pe to run, a PVM daemon must be running on each host machine specified in a map file. Pe will report on any missing daemons and abort. The daemons are started by the call:

>pvmd3 hostfile&

where hostfile has the host names and search path for the executable servers, for example:

quake ep=/home/soi/new/bin/_mips

To exucute pvmd3 your path must include /home0/jim/pvm3/lib/$ARCH, where $ARCH depends on the machine architecture. (See ~jim/.cshrc for the setting of $ARCH).

You must also have a pvm3 directory under your home directory with the following links:

bin -> /home/jim/pvm3/bin

include -> /home/jim/pvm3/include

lib -> /home/jim/pvm3/lib

There is also a PVM console program that supplies information and control of the PVM daemons. It is executed by typing “pvm” at the unix command line. Typical commands are:

conf	Shows all the hosts running PVM daemons
ps[-a]	Shows [all] the PVM tasks running for each daemon
reset	Kills all the PVM tasks
halt	Kills all tasks and daemons
quit	Exits the PVM console program

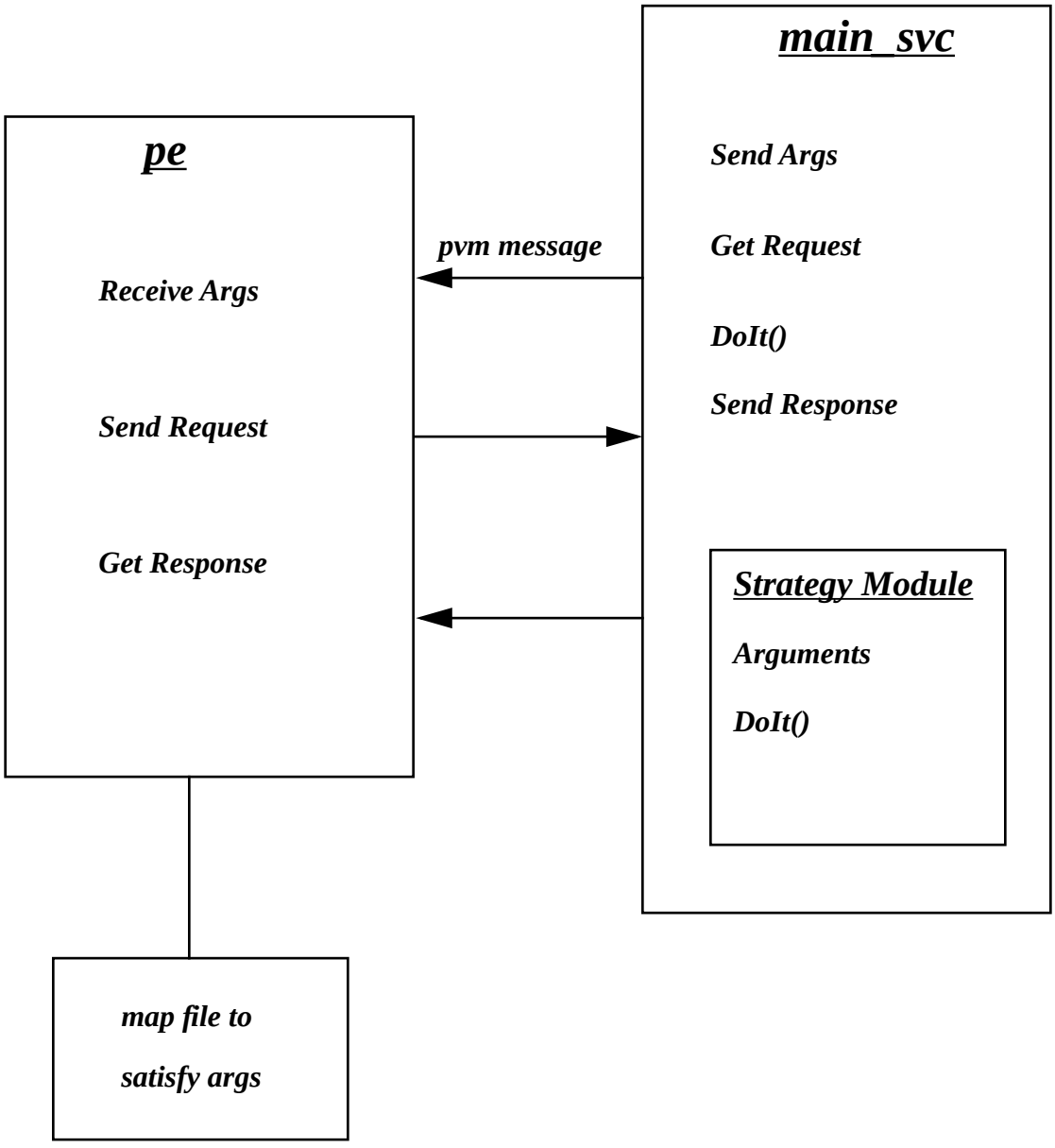
4.4 Pe Version 1.0 Restrictions

1. Pe assumes that a computational server can have only one output dataset, named “out”, in its argument list. If servers are built in the future that have more than one output dataset name then a new argument type, e.g ARG_DATASET_OUT, must identify them and pe must be modified to accept this.
2. The strategy level modules do not have a version number available yet. Pe will default a version 0 when cataloging with DSDS.
3. Pe request dataset name resolution from dsds_svc only in terms of program name, series name and level name. In later versions, pe and dsds_svc will handle any number of dataset classes.

5 Computational Servers (_svc)

In order to interface a strategy layer module with the pipeline, it is only necessary to link the strategy module with the pipeline interface module, main_svc (see Appendix B). This forms a Parallel Virtual Machine (PVM) task which can be spawned by and communicate with pe. The strategy module becomes a pipeline server to the PE client. For example, the remap strategy level module becomes remap_svc when wrapped. See Figure 7 for the relationship of a strategy level module to main_svc and pe. See “Programming in the SOI Analysis Environment” SOI-TN-107 for a discussion of strategy level modules.

FIGURE 7. Strategy Level Module as a PVM Server



Pe will first request the strategy module's argument list. It will satisfy this list from parameters specified in the pipeline map file, or use the default values given in the argument list. Any dataset names are resolved via environmental variables defined in the map file, or by SQL requests to the Data Storage and Distribution System (DSDS) server, dsds_svc, as specified in the map file. The resolved arguments are passed to the strategy server as a keylist in the same form as done in the command interface. A keylist is a linked list of name, value, datatype structures. The input keylist fully specifies the processing to be done by a strategy server. When a strategy server is complete it outputs a status integer and a results keylist. The status integer and output keylist are then passed back to pe.

Each computation server writes its stdout and stderr to a log file in /tmp. The log file name consists of the server's name appended with the PVM task id of the server, so each log file has a unique name for each computation server running on each host node. Occasionally old log files should be removed from /tmp.

5.1 Computational Server Version 1.0 Restrictions

1. The computational servers do not output any information in their results keylist other than the name of the last file processed. This mechanism is available for potential use to pass back information to pe which can be made available to other computational servers in the pipeline.
2. The servers do not yet write their version number as a parameter to their output files. This will enable data to be associated with the generating server version.

6 DSDS Server (dsds_svc)

Dsds_svc is a PVM task that handles dataset name resolution, dataset name cataloging and data storage allocation for any number of pe tasks. It is basically a main_svc program (Appendix B) that has been modified to call DSDS library routines instead of a strategy level module. A request number issued by pe determines which library routine is called. All information between dsds_svc and pe is in terms of keylists. The library routines detail the dsds_svc functionality.

There are four main database and in memory tables that dsds_svc operates on.

1. Partition Available (partn_avail). This table lists all the disk partitions that are dedicated to DSDS storage. An entry consists of the partition name, the total number of bytes in the partition, and the number of bytes not yet assigned . It is assumed that DSDS has exclusive access to all of these partitions. The table is initially set up by hand to contain the partition names, and total number of bytes. The available bytes is initially set equal to the total bytes.
2. Partition Allocation (partn_alloc). This is a dynamic table that keeps track of portions of the various DSDS partitions that are allocated to all the pe's. Each entry consists of the full working directory name that is allocated, the status of this wd, the number of bytes of this wd, and the uid that it is assigned to. The status of a wd can be read-write, read-only, archive-pending and delete-pending.

3. Open Pipeline Executions (open_pe). This table has one entry for every pe that has registered with dsds_svc and has not yet deregistered. An entry consist of the pe PVM task id and the uid assigned by dsds_svc when pe registers.
4. Dataset Catalog (dsds_test). This table has an entry for each dataset that has been saved in DSDS. The entry is basically the dataset name and location.

6.1 DS_Allocate

This is called by pe sending a REQALLOC request number. This is a request from a pe for output storage. This routine will scan the partn_avail table for available storage of the requested size. A directory is made under the found partition of the form /Dn. Where n is the uid of the requestor. This wd is then assigned as read-write in the partn_alloc table for this pe and the wd is returned to the requesting pe. If no partition has available storage, then the partn_alloc table is scanned for all delete-pending entries and each of these wd's are deleted. If there is still not adequate storage an error return is given to pe.

6.2 DS_CatUpdate

This is called by pe sending a REQUPDATE request number. This presents a dataset name and location that is to be cataloged in the dsds_test table. If a duplicate dataset name and version number is already in the dsds_test table then it is updated, otherwise an entry is inserted.

6.3 DS_Close

This is called by pe sending a REQCLOSE request number. This deregisters the pe with dsds_svc. The pe entry is removed from open_pe and the read-only partitions are deleted from partn_alloc for this pe.

6.4 DS_DataRequest

This is called by pe sending a REQREQ request number. This finds the given dataset name in the dsds_test table and returns the wd, fsn and lsn to the pe. A read-only entry is made in partn_alloc for this wd and pe.

6.5 DS_Deallocate

This is called by pe sending a REQDEALLOC request number. This returns the given number of bytes to the DSDS partition in partn_avail corresponding to the read-write wd of this pe in the partn_alloc table. The read-write entry is deleted from partn_alloc.

6.6 DS_Init

This is a routine called whenever dsds_svc first starts up. It reads the partn_avail, partn_alloc and open_pe tables from the database into memory. This initializes the new dsds_svc to the current state of the DSDS storage management . There should be no pe's registered at this point. Any pe that is still registered with DSDS will have all of its read-only partitions removed from partn_alloc. The read-write partitions for these pe's are handled by the sclean utility (see below) that will retrieve any lost storage.

6.7 DS_Open

This is called by pe sending a REQOPEN request number. This registers the calling pe with dsds_svc. An entry is made in open_pe and a uid is assigned and returned to the caller.

6.8 DS_OpenPeRequest

This is called by DS_Init to read the open_pe table into memory.

6.9 DS_PaUpdate

This is a routine to update the partn_alloc table in the database. It is called whenever the memory resident partn_alloc table is changed.

6.10 DS_PaUpdate_WR

Same as DS_PaUpdate except it is called by functions that have not already connected to the database.

6.11 DS_PallocRequest

This is called by DS_Init to read the partn_alloc table into memory.

6.12 DS_PavailRequest

This is called by DS_Init to read the partn_avail table into memory.

6.13 DS_PavailUpdate_WR

This is a routine to update the partn_avail table in the database. It is called whenever the memory resident partn_avail table is changed. It does an explicit connect to the database.

6.14 DS_PeUpdate_WR

This is a routine to update the open_pe table in the database. It is called whenever the memory resident open_pe table is changed. It does an explicit connect to the database.

6.15 DS_StatArchive

This is a routine to update the archive status in a dsds_test database table entry. It normally supplies the tape location of the given dataset.

6.16 DS_StatOnline

This is a routine to update the online status in a dsds_test database table entry. It supplies the online or offline status of the given dataset.

6.17 Dsds_svc Version 1.0 Restrictions

1. The DS_DataRequest does not currently handle the retrieving of a dataset from tape when it is no longer online. It returns a data not online status along with the tape and file number location of the data. Pe echoes this info and aborts.

7 Lago Server (lago_svc)

The Lago Server (lago_svc) is a Unix process which runs continuously and responds to messages from the DSDS server (dsds_svc) or from the stand-alone Lago Archiver (lagoarc) to provide services to archive and retrieve data. It can be started by either pe or by lagoarc.

The DSDS is designed to archive data on 8mm tape. The LS/380L DataWheel for Lago Systems is a mass storage device which can hold and manipulate these tapes. The DataWheel includes two 8mm cartridge tape drives, a robotic cartridge loader with a removable 54 slot carousel, a controller for the robotics, and a barcode reader. The robotics can be controlled using either a SCSI or serial command set. Quick Restore software from Workstation Solutions uses the SCSI command set and provides our software interface to the DataWheel. Quick Restore may also be used in a stand-alone mode for maintenance or debugging.

At start-up lago_svc takes an inventory of the contents of the Lago DataWheel. It removes tapes from the drives and returns them to their slots if it can determine where they belong. It also ensures that all tapes in the DataWheel have barcodes which are known to the Oracle TAPE database. After a successful start-up lago_svc's internal data structures, which are described later, are consistent with the states of the Quick Restore Software and the DataWheel itself. Start-up can fail when lago_svc detects inconsistencies which it cannot resolve. For example there may be a tape in the DataWheel which does not have a barcode or which has a barcode that is unknown to the database. In these cases the tape is ejected from the DataWheel if it is mechanically possible to do so. Operator intervention is then required to remove the tape from the DataWheel door and correct the situation. All possible error status returns are enumerated in soi_error.h. Those which are specific to lago_svc have a LAGO_ prefix.

lago_svc maintains two internal arrays. One is the tapes array which has an entry for each slot in the DataWheel. The other is the drives array which has an entry for each tape drive in the DataWheel. Each entry in the tapes array contains the tape's barcode (or an integer which indicates no barcode), the index (file number) of the most recently written archive file on the tape, and an estimate of the number of available blocks on the tape. Each entry in the drives array contains the user id of the process which has reserved the drive (or an indication that the drive is not reserved), a pointer to a datastructure which describes the tape in the drive (NULL if none), and the names used by Quick Restore for the drive.

After start-up lago_svc responds to the following messages:

DSDS_LAGO_OPENARCHIVE, DSDS_LAGO_WRITE, DSDS_LAGO_RETRIEVE, DSDS_LAGO_READ, and DSDS_LAGO_CLOSE. In response to DSDS_LAGO_OPENARCHIVE, lago_svc finds a tape drive which is not being used and

reserves it for the requesting process. It also locates a tape that can store the requested number of bytes. If the tape is not already in the drive lago_svc loads the tape into the drive. lago_svc is then ready to respond to a series of DSDS_LAGO_WRITE messages. Upon receiving a DSDS_LAGO_WRITE message lago_svc archives the requested directory onto the requestor's reserved tape and updates its estimate of the available blocks on the tape. A DSDS_LAGO_RETRIEVE message causes lago_svc to find and if necessary load the requested tape in preparation for a series of DSDS_LAGO_READ messages. A DSDS_LAGO_READ message causes the requested archive to be read into the requested directory. At the end of a series of DSDS_LAGO_WRITE or DSDS_LAGO_READ messages, the requesting process should send lago_svc a DSDS_LAGO_CLOSE message, thereby freeing the reserved drive for other requests. In general the archival and retrieval strategy followed by lago_svc is to leave tapes in drives on the theory that it is likely that they will soon be used again. Tapes are returned to slots in the DataWheel only when necessary to respond to a request. Appendix A contains detailed definitions of messages. DataWheel commands are effected by creating strings containing commands in the format required by Quick Restore and then using the system call to execute the command string. An API when it becomes available from Quick Restore or elsewhere could be substituted at this level.

7.1 Lago Server Version 1.0 Restrictions

A limitation of the current implementation is that it only operates synchronously using one tape drive at a time. A planned improvement is to allow asynchronous operation using both drives simultaneously. Another limitation is that it does not take advantage of Quick Restore's ability to quickly seek to a given archive. To change this it will be necessary to maintain seek locations (indices) in the database.

8 Lago User Interface (lagoui)

Not implemented in SSSC Version 1.0

9 Oracle Database (oracle)

This is a key element of DSDS which provides a catalog of all data received by DSDS through a set of tables stored in Oracle RDBMS V6.0. Currently there are two catalogs, alpha and beta, on aditya and quake respectively. These two databases have identical table structures for the catalog but the actual catalog elements may vary depending on which pe connects with which database. They are accessed from pe through dsds_svc which in turn connects to them using embedded SQL functions and SQL*NET TCPIP drivers. The dsds_svc connects to different databases according to a calling argument that is passed on from pe or peui from the user environment variable DBNAME.

Following are the table descriptions of the catalog tables used. In general the column names are designed to be self explanatory such as PROG_NAME for Program Name etc. However comments are added to some columns for further clarification.

TABLE 2. Dataset catalog (Table name: dsds_test)

Column Name	Null?	Type	Comments
PROG_NAME	NOT NULL	CHAR(20)	Program Name
PROG_NUM	NOT NULL	NUMBER(8)	Program Number
LEVEL_NAME	NOT NULL	CHAR(20)	Level Name
SERIES_NAME	NOT NULL	CHAR(20)	Series Name
SERIES_NUM	NOT NULL	NUMBER(8)	Series Number
LEVEL_NUM		NUMBER(8)	Level Number
S_SERIAL_NUM		NUMBER(8)	Start Serial Number of Image file
E_SERIAL_NUM		NUMBER(8)	End Serial Number of Image file
ONLINE_STATUS		CHAR(5)	“Y” if the data on disk, “N” if not
ONLINE_LOC		CHAR(80)	Online Location on disk
ARCHIVE_STATUS		CHAR(5)	“Y” if the data is archived, “N” if not
ARCH_TAPE_ID		NUMBER(5)	Tape id of archive tape
WORK_TAPE_ID		NUMBER(5)	Tape id of working tape
SAFE_TAPE_ID		NUMBER(5)	Tape id of safe tape in offsite location
HISTORY_COMMENT		CHAR(60)	Comments and notes
BYTES		NUMBER(20)	Space in bytes used by the dataset
CREATE_UID		NUMBER(38)	UID of the PE which created the dataset
SVC_VERSION		NUMBER(10)	Software version
SVC_NAME		CHAR(20)	Software name
CREAT_DATE		DATE	Creation date
TAPE_FILE_NUM		NUMBER(5)	File number of archive file on tape

TABLE 3. Open Pipeline Executions(Table name: open_pe)

Column Name	Null?	Type	Comments
U_ID	NOT NULL	NUMBER(38)	UID of PE
PE_ID	NOT NULL	NUMBER(20)	PE ID as assigned by PVM

TABLE 4. Partition Available(Table name: partn_avail)

Column Name	Null?	Type	Comments
PARTN_NAME	NOT NULL	CHAR(80)	Partition name on disk
TOTAL_BYTES	NOT NULL	NUMBER(20)	Total space available in bytes
AVAIL_BYTES	NOT NULL	NUMBER(20)	Currently available space in bytes

TABLE 5. Partition Allocation(Table name: partn_alloc)

Column Name	Null?	Type	Comments
WD	NOT NULL	CHAR(80)	Working Directory
U_ID	NOT NULL	NUMBER(38)	PE UID
STATUS	NOT NULL	NUMBER(5)	PE status
BYTES		NUMBER(20)	Space used in bytes

TABLE 6. Tape (Table name: tape)

Note: This table is available only on aditya since it is attached to the Lago data-wheel and used by the lago_svc.

Column Name	Null?	Type	Comments
BARCODE	NOT NULL	NUMBER(20)	Barcode
AVAIL_BLOCKS	NOT NULL	NUMBER(20)	Total block available
LAST_FILE_NUM	NOT NULL	NUMBER(20)	File number of last archive file

10 Oracle Database User Interface (oracle ui)

The Oracle user interface provides a querying tool which can be used in both character mode and graphical mode using Oracle's SQL*FORMS V3.0. Currently the graphical mode is using X-windows with either OpenLook or Motif. However this gui can be easily ported to other windowing systems supported by Oracle. In the character this has been tested using VT100, VT220, Sun keyboards.

The display provides various pages of information. Page 1 provides DATASETS information, Page 2 ONLINE information, Page 3 ARCHIVE, TAPE information and Page 4 PIPELINE_EXECUTION, SPACE_USED and SPACE_AVAILABLE. This is now avail-

able on aditya and quake, but without the TAPE information on quake. The oracle ui can be run by any user at this time as follows:

0. Before running the oracleui check the ~soi/new/src/pipe/doc/.orauisetup and modify the keyboard definition if needed to match the keyboard being used.
1. source ~soi/new/src/pipe/doc/.orauisetup
2. At the system prompt say oracleui

Now you will be in the SQL*FORMS screen. You can perform queries and navigate between different screens using the keys avaiable. To find which keys are available and their functions at any time hit Crtl K. For example F3 on a sun keyboard is “execute query”. Current configuration is set up to run with VT100 keyboards or emulatators. If more information is needed refer to Oracle Installation and User’s Guide to V6.0.3x, SQL*FORMS Under Ultrix(Chap 8), Supported terminals. Or check with Jayasree.

The following figures show sample screens.

FIGURE 8. DATASETS

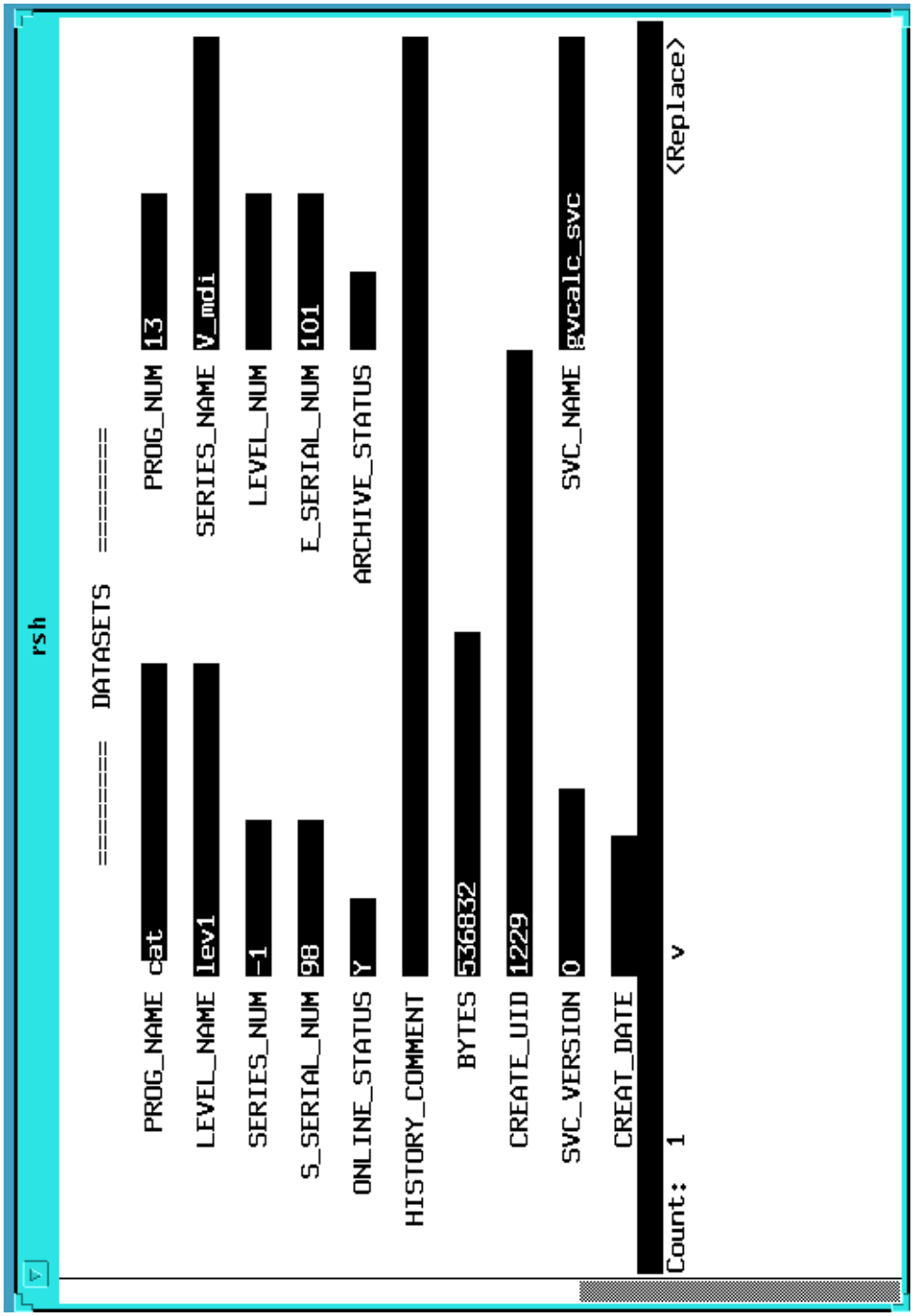


FIGURE 9. ONLINE_INFO

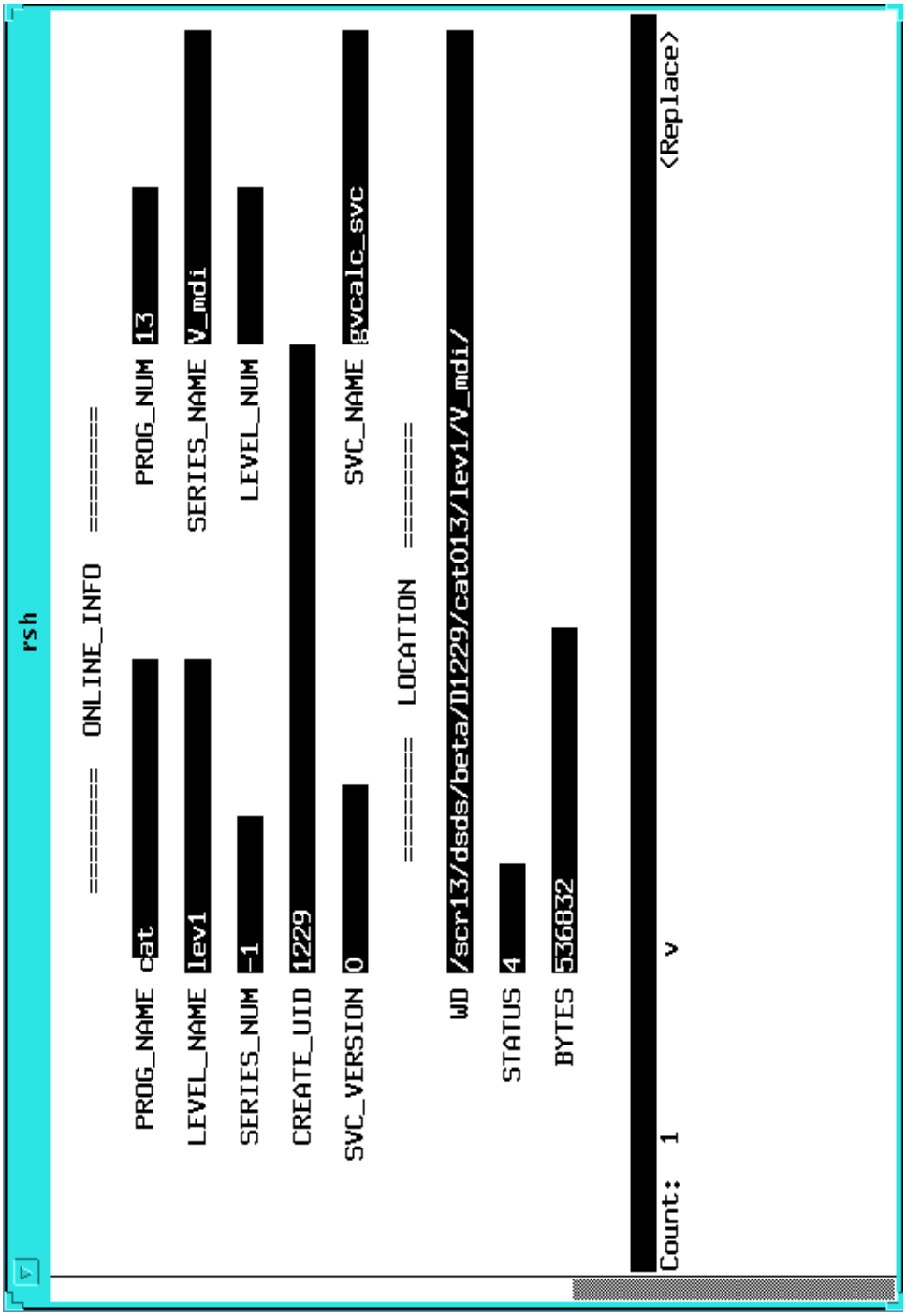


FIGURE 10. ARCHIVE_INFO

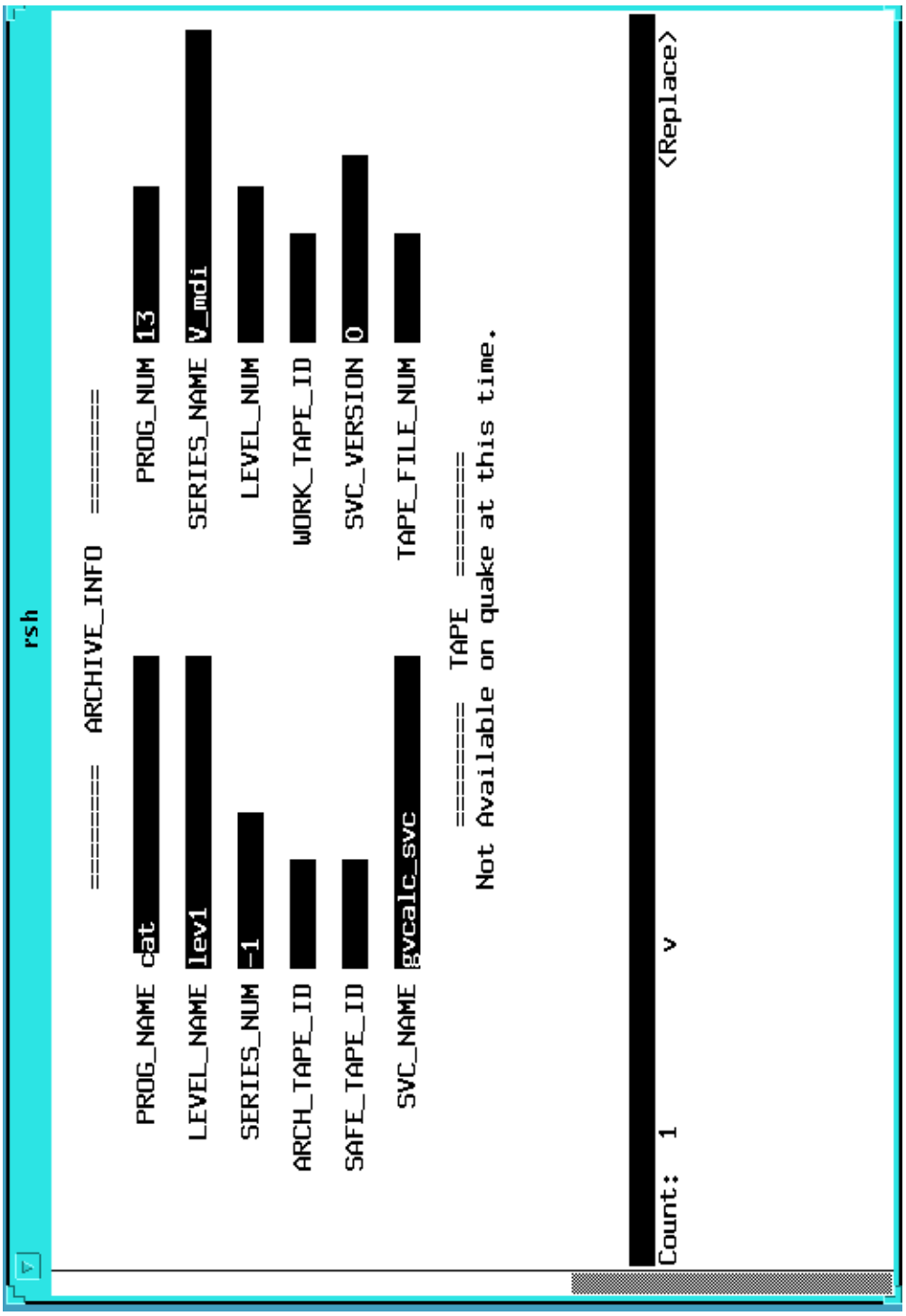
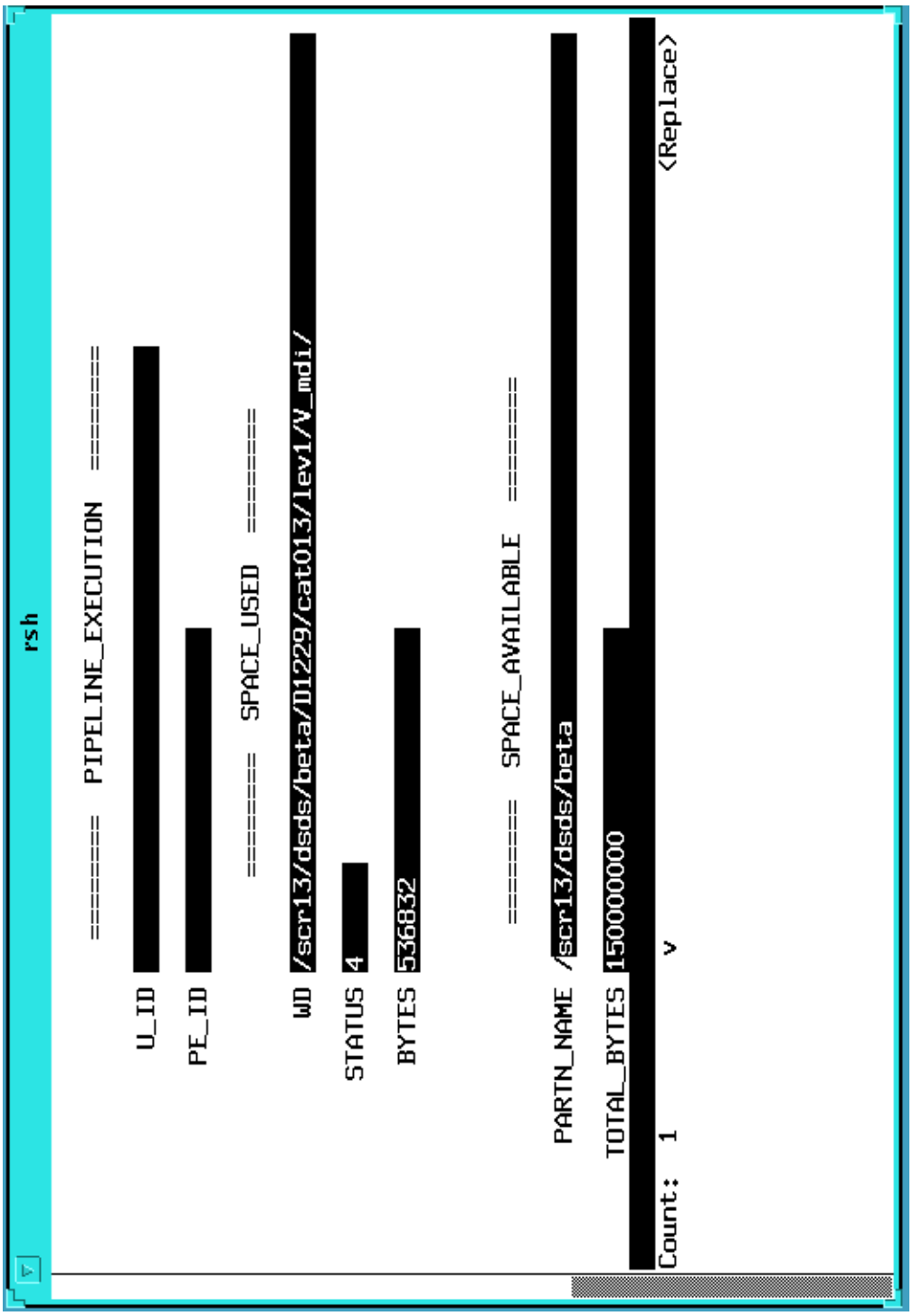


FIGURE 11. PIPELINE_EXECUTION etc.



11 Utility Programs

These programs operate on the DSDS database to provide various services. They only run offline, that is, when dsds_svc is not active. They will detect if a dsds_svc is running and exit in that event.

11.1 Storage Cleanup (sclean)

This is a stand alone program that is called on occasion to clean up any dangling DSDS storage. Dangling storage can occur when a pe allocates DSDS storage but never deallocates what was unused, i.e. the pe or dsds_svc crashed.

Sclean will get all the entries in the open_pe database table. Each of these entries represent a pe that opened with dsds_svc but never closed. For each entry, sclean will find the read/write partition assigned to this uid from the partn_alloc database table. It will then scan the wd for all subdirectories that are not archived and remove them and return the freed storage back to the appropriate partition in the partn_avail table.

11.2 Largo Archiver (lagoarc)

This is a stand alone program that will archive all the archive pending datasets to the Lago DataWheel. It gets all the archive pending working directories from the DSDS partn_alloc table and communicates with the lago_svc to write them to tape. When the write is complete the information in the dsds_test table is updated to give the tape and file number location of the dataset. The status of the dataset in the partn_alloc table is changed from archive pending to delete pending.

APPENDIX A. PVM Messages by Initiator

Name: DSDS_LAGO_CLOSE
Intent: Closes a retrieve or archive operation.
Freq: After a DSDS_OPEN_ARCHIVE followed by a series of DSDS_LAGO_WRITE or after a DSDS_RETRIEVE followed by a series of DSDS_LAGO_READ
Cont: An integer request type = TAPEDONE followd by a keylist.
Keys: dsds_tid = task id of dsds_svc
dsds_uid = unique id for this close
Action: Frees the tape which had been reserved for the given uid.
Return: Status int.

Name: DSDS_LAGO_OPENARCHIVE
Intent: Informs lago_svc of our intent to archive data.
Freq: When dsds_svc has one or more datasets to archive.
Cont: An integer request type = TAPEARCH followed by a keylist.
Keys: dsds_tid = task id of dsds_svc
dsds_uid = unique id for this open
arch_bytes = the number of bytes to be archived
Action: lago_svc will find a tape that can store the given # of bytes and load it into a drive in preparation for writing.
Return: Status int & keylist:
dsds_tape = barcode of loaded tape
dsds_drive = drive where requested tape has been loaded

Name: DSDS_LAGO_READ
Intent: Reads data from a tape loaded from the DSDS_LAGO_RETRIEVE
Freq: Once for each dataset to be read.
Cont: An integer request type = TAPERD followd by a keylist.
Keys: dsds_tid = task id of dsds_svc
dsds_uid = unique id for this open
dsds_wd = the online location to read the data into
tape_file = the file number on the tape
Action: Reads the file from the tape.
Return: Status int.

Name: DSDS_LAGO_RETRIEVE
Intent: Opens a tape in order to retrieve data for dsds_svc
Freq: Whenever dsds_svc needs a dataset that is not online.
Cont: An integer request type = TAPERET followed by a keylist.

Keys: dsds_tid = task id of dsds_svc
dsds_uid = unique id for this open
dsds_tape = the tape number to open
Action: Loads the tape # into a drive.
Return: Status int.

Name: DSDS_LAGO_WRITE
Intent: Writes data to a tape loaded from the DSDS_LAGO_OPENARCHIVE
Freq: Once for each dataset to be written.
Cont: An integer request type = TAPEWRT followed by a keylist.
Keys: dsds_tid = task id of dsds_svc
dsds_uid = unique id for this open
dsds_wd = location of the data to be saved
Action: Writes the dsds_wd to the tape.
Return: Status int & keylist:
dsds_fn = the file/archive number on the tape

Name: DSDS_PEUI_STATUS
Intent: Informs peui of ongoing dsds_svc status
Freq: Whenever dsds_svc outputs a status message
Cont: Text of status
Keys: N/A
Action: Status message is displayed in the corresponding peui window
Return: None

Name: PEUI_DSDS_CLOSE
Intent: Disables dsds_svc status reporting to peui
Freq: Whenever the peui exits
Cont: An integer request type = REQTAEOFF
Keys: N/A
Action: dsds_svc no longer sends status messages to peui
Return: None

Name: PEUI_DSDS_OPEN
Intent: Enables dsds_svc status reporting to peui
Freq: Whenever the peui DSDS Display button is activated
Cont: An integer request type = REQTAEON
Keys: N/A
Action: dsds_svc now sends status messages to peui
Return: None

Name: PE_DSDS_ALLOCATE
Intent: Allocates dsds storage for a pe.
Freq: When pe starts and sees a DSDSOUT=n specified in the map file.
Cont: An integer request type = REQALLOC followed by a keylist.
Keys: pe_tid = task id of the calling pe
dsds_uid = the uid returned in the PE_DSDS_REGISTER
alloc_bytes = the number of bytes to allocate
Action: Finds available storage in one of the DSDS partitions. Will delete any delete-pending directories if required.
Return: Status int & keylist:
dsds_wd = the root directory under which pe can use storage.

Name: PE_DSDS_CATUPDATE
Intent: To update the data catalog with a new pe data product.
Freq: Whenever pe has been told in the map file to catalog a servers output.
Cont: An integer request type = REQUPDATE followed by a keylist.
Keys: pe_tid = task id of the calling pe
lago_tid = task id of the lago_svc
dsds_uid = the uid returned in the PE_DSDS_REGISTER
fsn = the first serial number of the data
lsn = the last serial number of the data
wd = the location of the data
prog = the program name
series = the series name
prog_sn = the program number
series_sn = the series number
level = the level name
dsds_bytes = the number of bytes for this data
Action: Updates the data catalog.
Return: Status int.

Name: PE_DSDS_DATAREQ
Intent: Tells pe the location of input data.
Freq: Whenever pe needs to resolve input dataset names.
Cont: An integer request type = REQREQ followed by a keylist.
Keys: pe_tid = task id of the calling pe
dsds_uid = the uid returned in the PE_DSDS_REGISTER
inname = the name of the key prefix (xx) for the following keys
xx_prog = the dataset program name

xx_series = the series name
xx_level = the level name

Action: Queries the data catalog to resolve the dataset name.

Return: Status int and keylist:
xx_fsn = first serial number of dataset
xx_lsn = last serial number
xx_wd = the data location
xx_prog = the program name
xx_series = the series name
xx_level = the level name

Name: PE_DSDS_DEALLOCATE

Intent: To return storage to DSDS that pe did not use.

Freq: When pe has a normal completion and previously did a PE_DSDS_ALLOCATE

Cont: An integer request type = REQDEALLOC followed by a keylist.

Keys: pe_tid = task id of the calling pe
dsds_uid = the uid returned in the PE_DSDS_REGISTER
free_bytes = the number of bytes to return to DSDS

Action: Returns the number of bytes to the partition from which it was gotten.
Clears the entry assigning the storage as r/w for this pe.

Return: Status int.

Name: PE_DSDS_DEREGISTER

Intent: Closes a pe session with dsds_svc.

Freq: Whenever pe has a normal completion and prev did a PE_DSDS_REGISTER.

Cont: An integer request type = REQCLOSE followed by a keylist.

Keys: pe_tid = task id of the calling pe
dsds_uid = the uid returned in the PE_DSDS_REGISTER

Action: Clears all the read-only partitions assigned to this uid.

Return: Status integer.

Name: PE_DSDS_REGISTER

Intent: Open a dsds_svc session for a pe.

Freq: Whenever a pe starts up and needs any input or output with DSDS.

Cont: An integer request type = REQOPEN followed by a keylist.

Keys: pe_tid = task id of the calling pe.

Action: dsds_svc gets a unique id (uid) for this session and associates it with the given pe_tid.

Return: Status integer followed by keylist:
dsds_uid = the uid assigned to this pe.

Name: PE_PEUI_STATUS
Intent: Informs peui of ongoing pe status
Freq: Whenever pe outputs a status message
Cont: Text of status
Keys: N/A
Action: Status message is displayed in the corresponding peui window
Return: None

Name: PE_STRATEGY_ARG_REQUEST
Intent: Tell a strategy level module to send its argument list to pe.
Freq: Whenever a strategy level module is started.
Cont: None
Keys: N/A
Action: The strategy level module sends its arguments to the requesting pe
Return: A keylist of strings for each argument

Name: PE_STRATEGY_COMMAND_LINE
Intent: Provides a keylist of arguments to the strategy module for execution.
Freq: Once for any entry in the pe map file specifying this strategy module.
Cont: An integer flag (currently not used) followed by a keylist.
Keys: Depends on the arguments sent by the strategy level module in reply to the PE_STRATEGY_ARG_REQUEST.
Action: The strategy module processes the data by forming the input file names, output file names and parameters from the keylist.
Return: An integer status, an integer tid of the module, a host name string and the last file name string output by the strategy module. (Note: Eventually the strategy module will output a complete keylist that will be returned to pe.)

Name: PE_STRATEGY_MESSAGE_ID
Intent: Inform a strategy level module which message id pe will use to communicate with it.
Freq: Whenever a strategy level module is first started
Cont: An integer message id and the verbose value that pe was started with.
Keys: N/A
Action: The strategy level module will listen for pe messages with this message id.
Return: None

APPENDIX B. Pipeline Interface Module main_svc

```
/*-------------------------------------------------------------------------
 * /home/soi/new/src/pipe/main_svc.c   PVM3 Version
 *-------------------------------------------------------------------------
 *
 * INPUTS:Messages of various types from Pipeline Execution
 * OUTPUTS: Files according to the strategy level module that it's linked with.
 * A log file /tmp/"strategy_name"_svc.log.
 * RETURNS: Response messages back to PE
 * EXTERNALLY READ: strategy_name from the strategy level module
 * EXTERNALLY MODIFIED: None
 *
 * DESCRIPTION:
 * Wrapper to make pipe interface level server when linked to a strategy
 * level module.
 * Each server is started by PE as a pvm task.
 * PE and each server exchange messages to effect processing.
 * Each server type must have a unique message number that PE uses to
 * identify that server type. This message number is given to the server in
 * the first msg that PE sends.
 * The major action is to receive a keylist from PE and call the strategy level
 * module with this keylist which will then transform the input files to the
 * output files.
 *
*/
main()
{
    int parent, msgtag, status;
    KEY *list;
    KEY *outlist = newkeylist();

    parent=setup();                       /* preliminaries & reg with pvm */

    while(1) {
         busy = 0;
         if((msgtag=getmsg(parent)) == -1)/* get any message */
                 continue;                /* error, loop again */
    busy = 1;
    alarm(SLAVE_TIMEOUT);                 /* reset the idle timeout */

    if(msgtag == msgid) {                 /* normal file processing msg */
         if(!(list=unpackmsg(list)))      /* get the key list sent */
                 continue;                /* top of while loop */

         status=DoIt(list, &outlist, write_log, write_log); /* strategy code */

         if(verbose) {
                 write_log("The keylist output by %s_svc is:\n",strategy_name);
                 keyiterate(logkey, outlist);
```

```
        }
        sendresult(parent, status, outlist);  /* send the results to PE */
    }
    else if(msgtag == MSGARGS) {              /* send the arg list */
        sendargs(parent);
    }
    else if(msgtag == MSGMSG) {             /* msg # for normal file processing */
        pvm_upkint(&msgid, 1, 1);
        pvm_upkint(&verbose, 1, 1);
    }
    else {
        write_log("Illegal msg type %d sent to %s_svc\n",msgtag,strategy_name);
    }
}                                             /* end while(1) */
}
```

APPENDIX C. References

1. "Preliminary SSSC Conceptual Design" SOI-TN-105
2. "Naming of Datasets for SOI" SOI-TN-104
3. "Programming in the SOI Analysis Environment" SOI-TN-107
4. "PVM 3.0 User's Guide and Reference Manual" ORNL/TM-12187
5. "tae+ User Interface Developer's Guide Version 5.1"
6. "tae+ C Programmer's Manual Version 5.1" Volume 1 and 2
7. LS/380L Datawheel 8mm Tape Library Systen Users Guide
8. Quick Restore User's Guide - Workstation Solutions, Inc.
9. Oracle Installation & User's Guide, v. 6.0.30
10. SQL*FORMS Operator's Guide v. 3.0