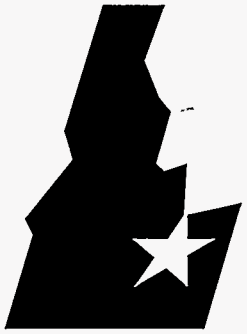

Idaho
National
Engineering
Laboratory

INEL/EXT-97-00005

February 1997

Automatic TLI Recognition System, Programmer's Guide

G. D. Lassahn

LOCKHEED MARTIN

DISCLAIMER

INEL/EXT-97-00005

Automatic TLI Recognition System, Programmer's Guide

G. D. Lassahn

HQ PROJECT MANAGER - Michael O'Connell
PROJECT NUMBER - ST474E

Published February 1997

Idaho National Engineering Laboratory
EG&G Idaho, Inc.
Idaho Falls, Idaho 83415

Prepared for the
U.S. Department of Energy
Office of Arms Control
under DOE Idaho Field Office
Contract DE-AC07-76ID01570

ABSTRACT

This report describes the software of an automatic target recognition system (version 14), from a programmer's point of view. The intent is to provide information that will help people who wish to modify the software. In separate volumes are a general description of the ATR system, *Automatic TLI Recognition System, General Description*, and a user's manual, *Automatic TLI Recognition System, User's Guide*.

CONTENTS

Automatic TLI Recognition System, Programmer's Guide

INTRODUCTION

This is the third of three volumes, a programmer's manual to help people who want to change the software of the automatic target recognition (ATR) system. It might be helpful for the reader of this volume to have some knowledge of the information presented in the other two volumes. The first volume[1] is a general description of the ATR system. The second volume[2] is a user's manual for people who do the hands-on image data analysis, giving instructions on how to use this ATR system. The software described here is version 14 of programs E, F, and G, and version 7 of TSCSI.

SOFTWARE OVERVIEW

We have three totally separate hardware systems: ATR1, ATR2, and ATR3. Each is a parallel processor system with several nodes serving different functions. All three systems use essentially the same ATR software, with minor variations to accommodate certain hardware differences. Each of the three hardware systems runs each of three applications, named E, F, and G. ATR1 runs a fourth application, TSCSI, which handles hardware that is not present in ATR2 or ATR3. Each application on each hardware system comprises several programs, one program for each hardware node in the system. The four programs used here are called MASTER, SLAVE, SCSI, and VIDEO. Table 1 indicates which program runs on which type of node for the three systems. Each application for hardware system ATR1, for example, runs the program named MASTER on the node named ROOT, a copy of the program named SLAVE on each DAISY node, the program named SCSI on the node named SCSI, and the program named VIDEO on the node named VIDEO. The several applications include programs with the same names, although the contents of the programs may be different for the different applications. The ATR1 and ATR2 host computers run a server program that is supplied by the vendor with the parallel processor hardware. This server is mostly transparent to the user and is regarded as system software, a kind of device driver that the host computer uses to communicate with the rest of the network. The server's primary functions are to load the programs onto the non-host nodes and to allow the root node program to access the host devices as if the root node program were running on the host. ATR3 has a server only in the very limited sense of loading the software onto the non-host nodes. In ATR3, the MASTER program runs on the host, so there is no need for a server to make the host resources available to a master program on another node. The SCSI and VIDEO nodes and programs exist only in the ATR1 system.

Table 1: Nodes Types and their Programs

system	node type	program
ATR1	HOST	(server)
	ROOT	MASTER
	DAISY	SLAVE
	SCSI	SCSI
	VIDEO	VIDEO
ATR2	HOST	(server)
	ROOT	MASTER
	DAISY	SLAVE
ATR3	HOST	MASTER
	DAISY	SLAVE

Much, but not all, of the source code is common to all the applications. Tables 2-4 indicate which source code files are used in which applications. Each of the source code files is the same for all programs and all applications. For example, there is only one file named

emntrp.c, and it is used in both applications E and F in all three hardware systems. The .lnk files are merely lists of object files that must be linked to produce a complete program. The .cfs files describe the hardware configuration, following procedures supplied by the compiler vendors. efg.h is a header file used by every .c file. comm.inc contains the inter-node communication functions, and is included in the file that contains the main program for each node.

All the source code is in the C language. ATR1 and ATR2 require cross-compilers appropriate to the T805 and T9000 transputers. In ATR3, the C compiler for the host computer is also used for the non-host node programs.

The SLAVE program does the bulk of the computational work, with a copy of SLAVE running on each DAISY node. MASTER controls and coordinates the several SLAVE programs, and also controls SCSI and VIDEO in ATR1. The SCSI and VIDEO programs handle the parallel processor network's communication with external mass storage devices on the SCSI bus and with the RGB monitor in ATR1.

Table 2: ATR1 Source Code File Usage

	E	F	G	TSCSI
MASTER:				
em.lnk	+			
fm.lnk		+		
gm.lnk			+	
emmain.c	+			
fmmain.c		+		
gmmain.c			+	
emntrp.c	+	+		
gmntrp.c			+	
emio.c	+	+	+	
eminfo.c	+	+	+	
emman.c	+	+	+	
emmath.c	+	+	+	
emfilt.c	+	+	+	
emconv.c	+	+	+	
emlines.c	+	+	+	
empeak1.c	+	+	+	
fmopt.c		+		
gmmask.c			+	
tscsim.lnk				+
tscsim.c				+
SLAVE:				
es.lnk	+			
fs.lnk		+		
gs.lnk			+	
esntrp.c	+	+	+	
esio.c	+	+	+	
esinfo.c	+	+	+	
esman.c	+	+	+	
esmath.c	+	+	+	
esfilt.c	+	+	+	
esconv.c	+	+	+	
eslines.c	+	+	+	
espeak1.c	+	+	+	
esdummy.c	+		+	
fsopt.c		+		
tscsis.lnk				+
tscsis.c				+

Table 2: ATR1 Source Code File Usage (continued)

VIDEO:				
video.lnk	+	+	+	
video.c	+	+	+	
tscsiv.lnk				+
tscsiv.c				+
SCSI:				
scsi7.lnk	+	+	+	+
scsilib.h	+	+	+	+
scsilib.c	+	+	+	+
scsi7.c	+	+	+	+
disk7.c	+	+	+	+
tape7.c	+	+	+	+
ALL (MASTER&SLAVE&VIDEO&SCSI):				
efg.h	+	+	+	+
comm.inc	+	+	+	+
alta.cfs	+	+	+	+

Table 3: ATR2 Source Code File Usage

	E	F	G
MASTER:			
em.lnk	+		
fm.lnk		+	
gm.lnk			+
emmain.c	+		
fmmain.c		+	
gmmain.c			+
emntrp.c	+	+	
gmntrp.c			+
emio.c	+	+	+
eminfo.c	+	+	+
emman.c	+	+	+
emmath.c	+	+	+
emfilt.c	+	+	+
emconv.c	+	+	+
emlines.c	+	+	+
empeak1.c	+	+	+
fmopt.c		+	
gmmask.c			+
SLAVE:			
es.lnk	+		
fs.lnk		+	
gs.lnk			+
esntrp.c	+	+	+
esio.c	+	+	+
esinfo.c	+	+	+
esman.c	+	+	+
esmath.c	+	+	+
esfilt.c	+	+	+
esconv.c	+	+	+
eslines.c	+	+	+
espeak1.c	+	+	+
esdummy.c	+		+
fsopt.c		+	
ALL (MASTER&SLAVE):			
efg.h	+	+	+
comm.inc	+	+	+
atr2.cfs	+	+	+

Table 4: ATR3 Source Code File Usage

	E	F	G	
MASTER:				
emmain.c	+			
fmmain.c		+		
gmmain.c			+	
emntrp.c	+	+		
gmntrp.c			+	
emio.c	+	+	+	
eminfo.c	+	+	+	
emman.c	+	+	+	
emmath.c	+	+	+	
emfilt.c	+	+	+	
emconv.c	+	+	+	
emlines.c	+	+	+	
empeak1.c	+	+	+	
fmopt.c		+		
gmmask.c			+	
SLAVE:				
esntrp.c	+	+	+	
esio.c	+	+	+	
esinfo.c	+	+	+	
esman.c	+	+	+	
esmath.c	+	+	+	
esfilt.c	+	+	+	
esconv.c	+	+	+	
eslines.c	+	+	+	
espeak1.c	+	+	+	
esdummy.c	+		+	
fsopt.c		+		
ALL (MASTER&SLAVE):				
efg.h	+	+	+	+
comm.inc	+	+	+	+

BUILD&LOAD&RUN PROCEDURES

This section describes the procedures for building program files, loading the programs onto the parallel processor network, and executing the programs. The examples are for application E; the procedures are the same for applications F and G.

The ATR systems require a separate program to run on each type of node. The several DAISY nodes all run copies of the same program, SLAVE. The program MASTER runs on the ROOT node (for ATR1 and ATR2) or on the HOST node (for ATR3). ATR1 requires two additional programs, SCSI and VIDEO, to run on special interface nodes.

The software build procedures are illustrated in the following diagrams, in which "em * " and "es * " represent an unspecified number of additional files. The symbols GDL_... must be defined at the beginnings of the indicated files, before the #include "efg.h" statement. Some files are used or created by explicit operations; for example, the icc operation uses a .c file and creates a .tco file. Other files are used as a result of being listed in a different file; .tco files are listed in and hence used with .lnk files, and .lku files are listed in the .cfs file.

The ATR1 procedures are diagrammed in Figure 1. In ATR1, there are 10 source code (.c) files, plus comm.inc and efg.h, for the MASTER program. These are compiled separately, using the compiler icc. The 10 .tco files, and also some library files, are referenced by name in the file em.lnk, which is used by ilink to produce the file master.lku. A similar procedure is used to produce .lku files for SLAVE, SCSI, and VIDEO. These four .lku files are referenced by name in the file alta.cfs, which also contains information about the parallel processor network configuration. The tools icconf and icollect use the network configuration information and combine the four .lku files into the file e.btl. This completes the build procedure for program E.

Program E is loaded and executed in one step, using the command "run e". run.bat is a simple batch file that merely invokes the server program with the appropriate arguments:

```
\atr1\bin\iserver  /sb %1.btl  /se
```

The processes for ATR2, diagrammed in Figure 2, are similar to those for ATR1. However, ATR2 does not have the SCSI and VIDEO nodes. Also, because the T9000 transputer HTRAMs in ATR2 are an early release, memory configuration for each type of node must be given explicitly, in the files qt9d.mem and b927.mem. The imem tool is not essential to the build process; it merely generates files containing information that may be interesting to the programmer.

For ATR2, the run.bat batch file is a little different:

```
@if %2x == x  goto DEFNIF
@rem  \qt9s\bin\t9server  /sb %1.btl  /sn %2.nif
c:\atr2\bin\t9server  /sb %1.btl  /sn %2.nif
goto END
:DEFNIF
@rem  \qt9s\bin\t9server  /sb %1.btl  /sn \atr2\atr2.nif
c:\atr2\bin\t9server  /sb %1.btl  /sn c:\atr2\atr2.nif
:END
```

Note that the file atr2.nif must be available at load&run time.

For ATR1 and ATR2, the programs for all the nodes are put into one .btl file. For ATR3, the MASTER and SLAVE programs are not put into the same file.

There are three fundamentally different ways to build, load, and execute the programs in ATR3. The DEC 2000 AXP (host) UNIX operating system and the AL/V66 (daisy node) VxWorks operating system allow the user to set up a separate host window for each daisy and use this window to rlogin to the daisy and monitor its operation while the ATR software is running, using the TCP/IP communication protocol between HOST and DAISY. This procedure can be very useful while debugging new software, but it is not the most convenient procedure for normal operation. The ATR3 build&load options are diagrammed in Figures 3 through 9, and some parallel explanation is given in the text following the figures.

In the diagrams of the ATR3 build&load&execute processes, some of the file names are not really the names of files but are links to other files; in these cases, the link name is followed by the real file name in parentheses. Some of the file names have the notations {AOS} or {TOS}, which indicate that the file includes the Alpha or Transputer operating system software.

Figure 3 shows the build procedure that is common to both procedures 1 and 2. This produces an executable file (em) for the MASTER program, and an object file (es.o) for the SLAVE program.

Load procedure 1 is shown in Figure 4. This procedure uses the object file (es.o) from the previous build procedure, along with some system software files. This procedure is a little tedious, requiring that several commands be typed for each daisy. Load procedure 2, in the following figure, uses a script file to accomplish the same load result much more efficiently. After the daisies are loaded, the user may at his discretion rlogin (or simply remain logged in at the end of the load process) to any or all of the daisies and monitor their operation and use the VxWorks diagnostic tools. These build&load procedures 1 and 2 (sometimes referred to as dynamic linking) preserve the names of global symbols in the SLAVE software, and the values associated with these symbols can be inspected with VxWorks. The ATR software is executed by typing "em" to the host. This execution can be repeated without reloading the daisies, unless there is some error that causes the daisy communications to become unsynchronized.

ATR3 build/load procedure 3 allows monitoring the daisies via the TCP/IP links, as in procedures 1 and 2. However, unlike procedures 1 and 2, procedure 3 does not preserve the global symbols in the SLAVE software (static linking, as opposed to dynamic linking), so the use of the VxWorks diagnostic tools is somewhat more limited.

Procedure 4 for ATR3 is convenient for routine use of the software. This procedure does not allow logging in to the daisies.

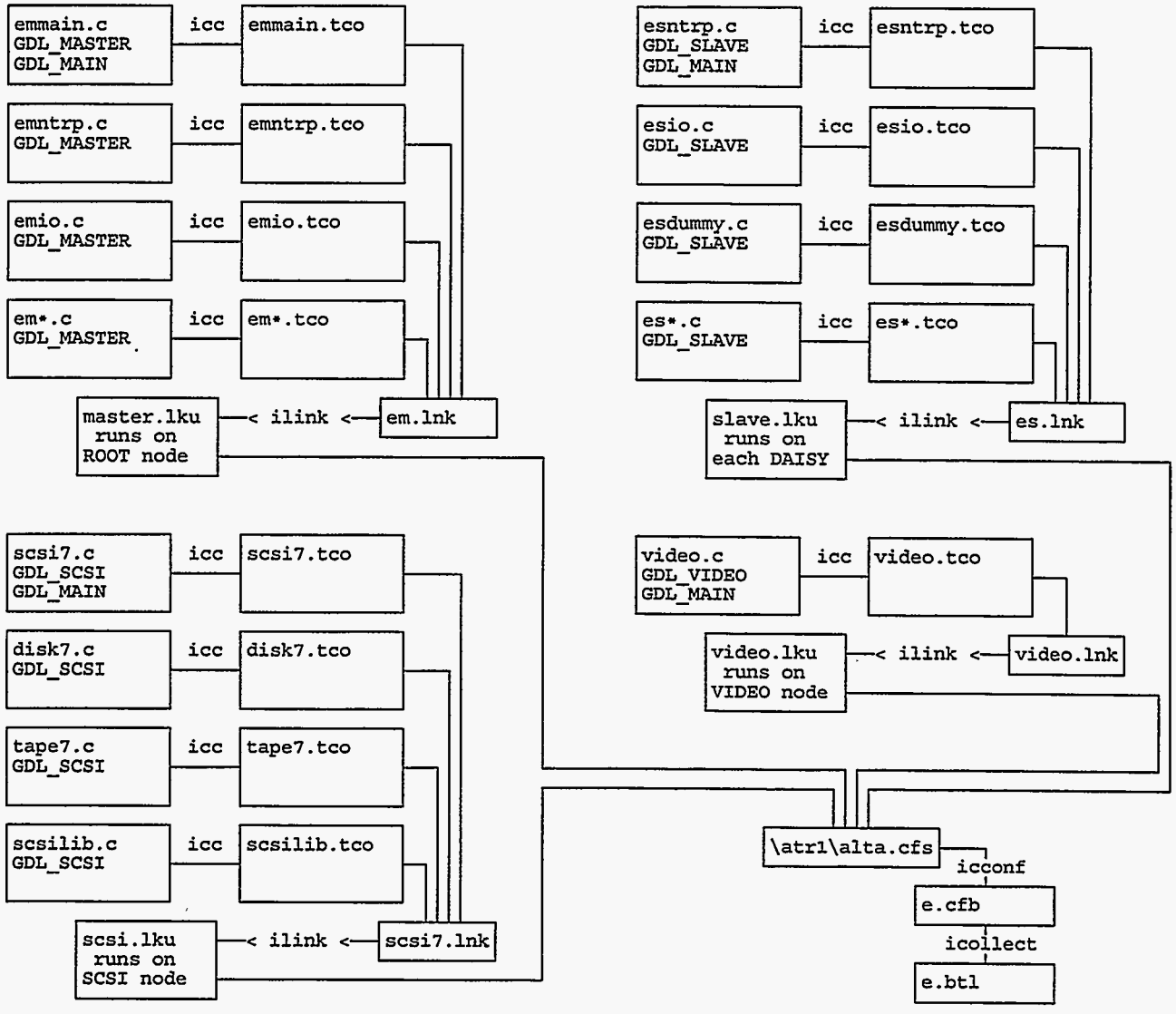

Figure 1: Software build procedure for ATR1.

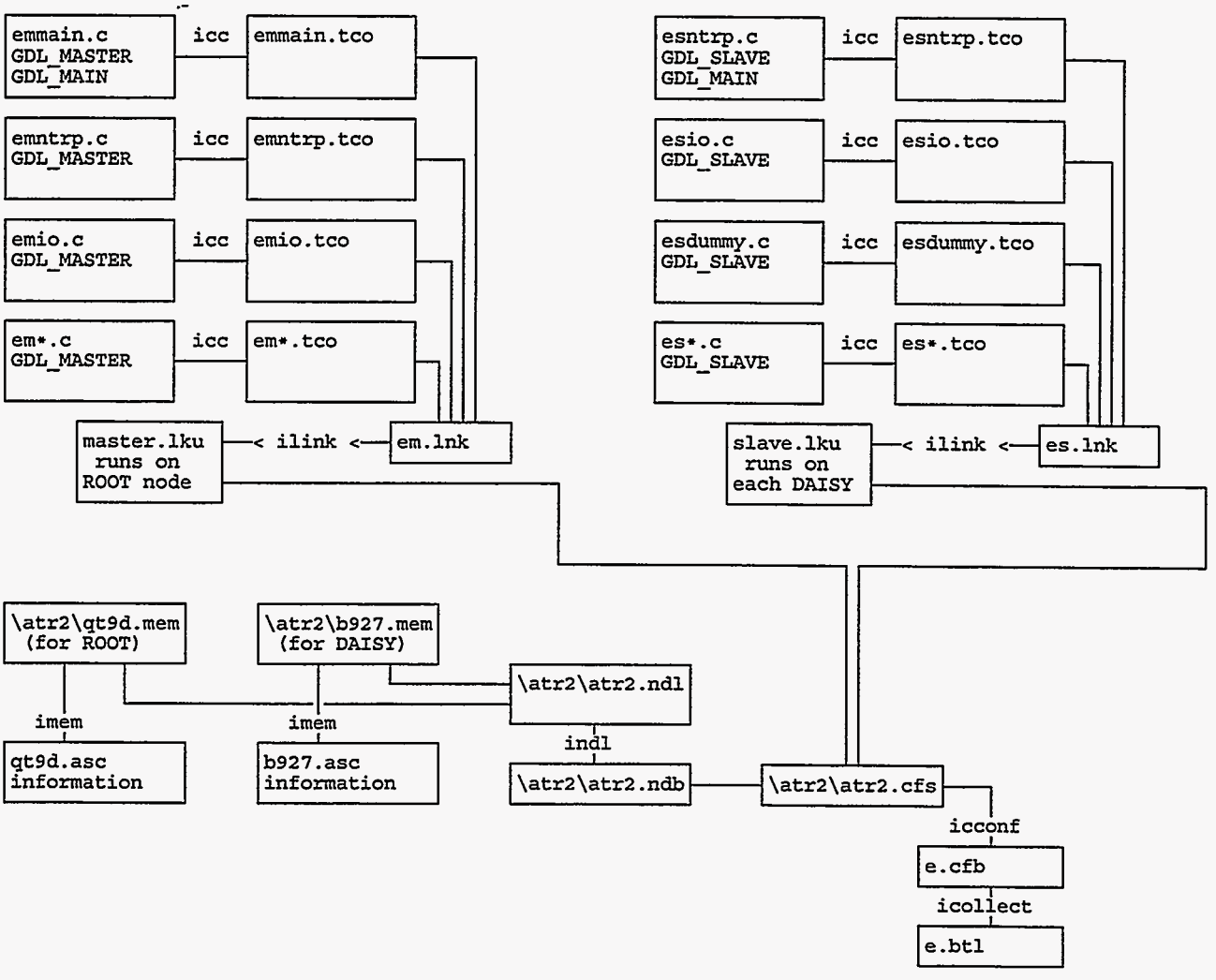

Figure 2: Software build procedure for ATR2.

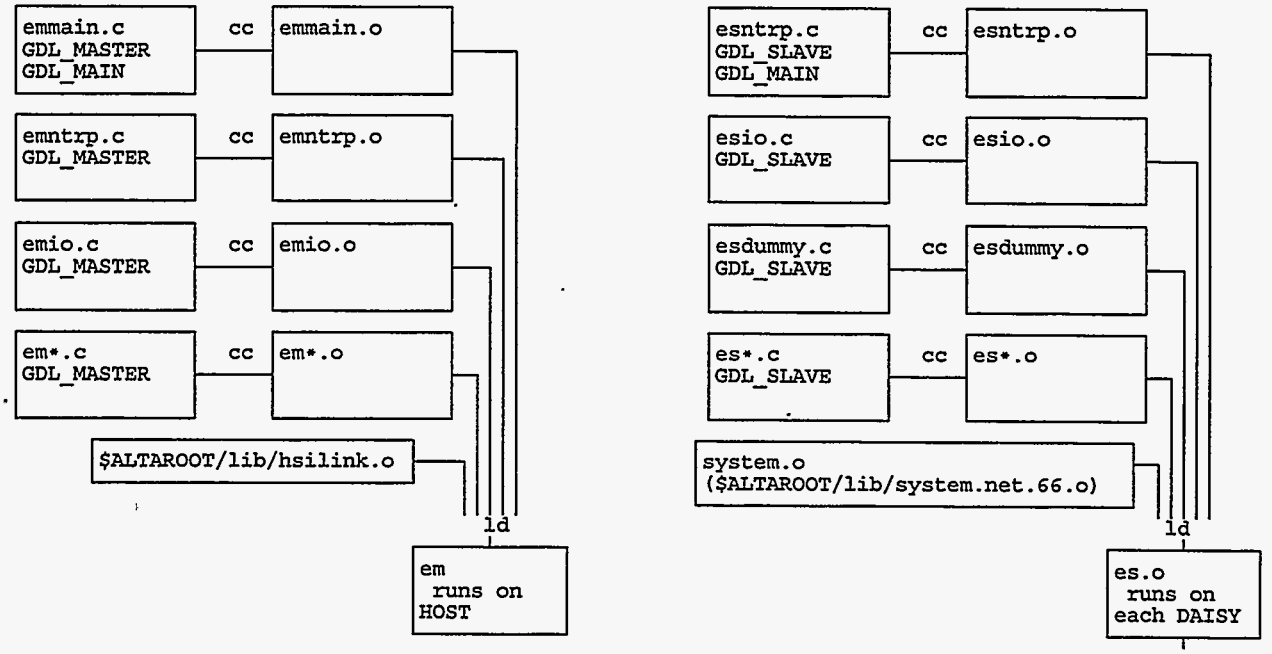

Figure 3: ATR3 build procedures 1 and 2.

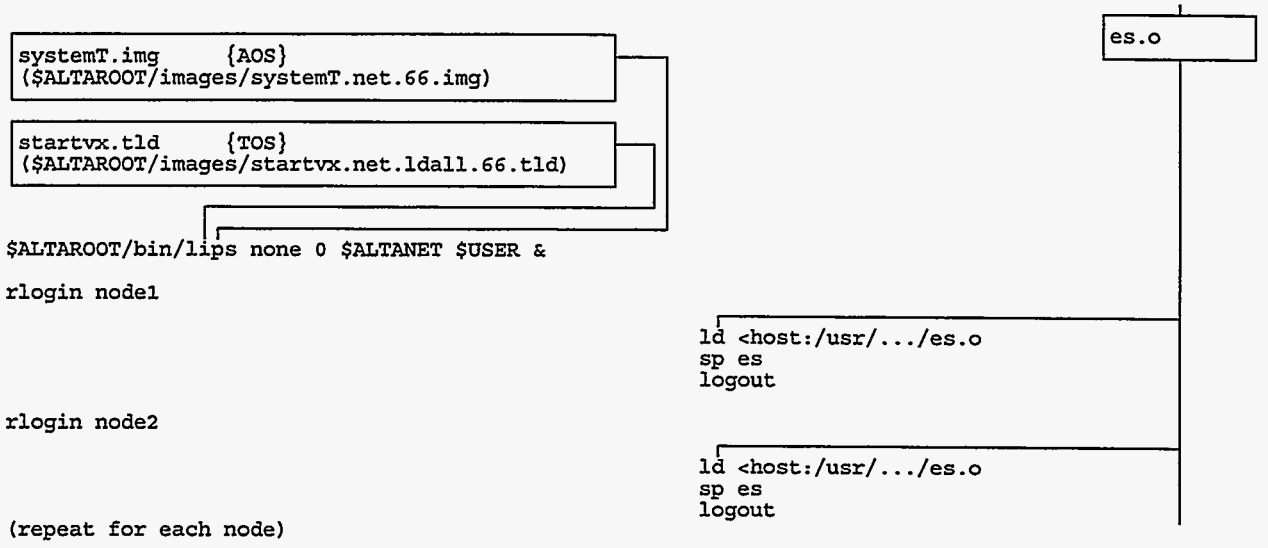

Figure 4: ATR3 load&run procedure 1.

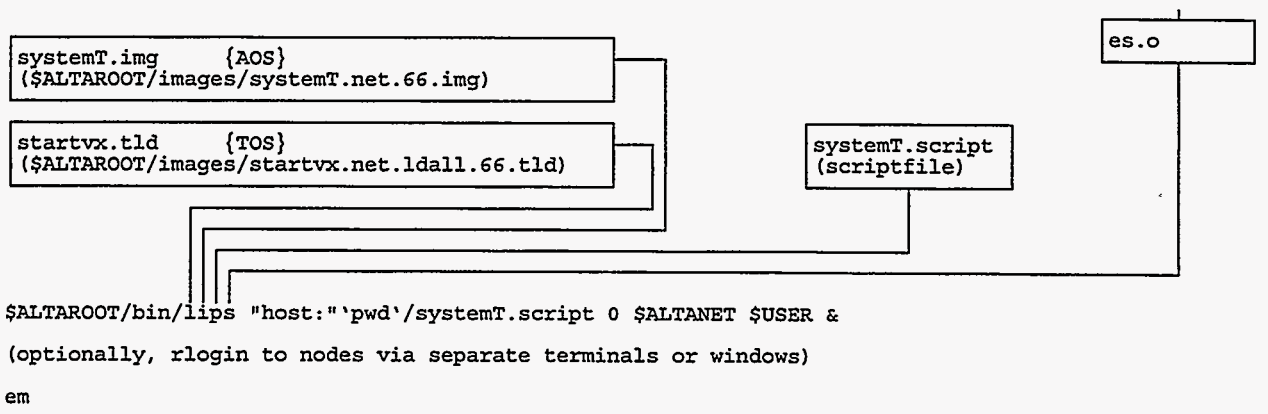

Figure 5: ATR3 load&run procedure 2.

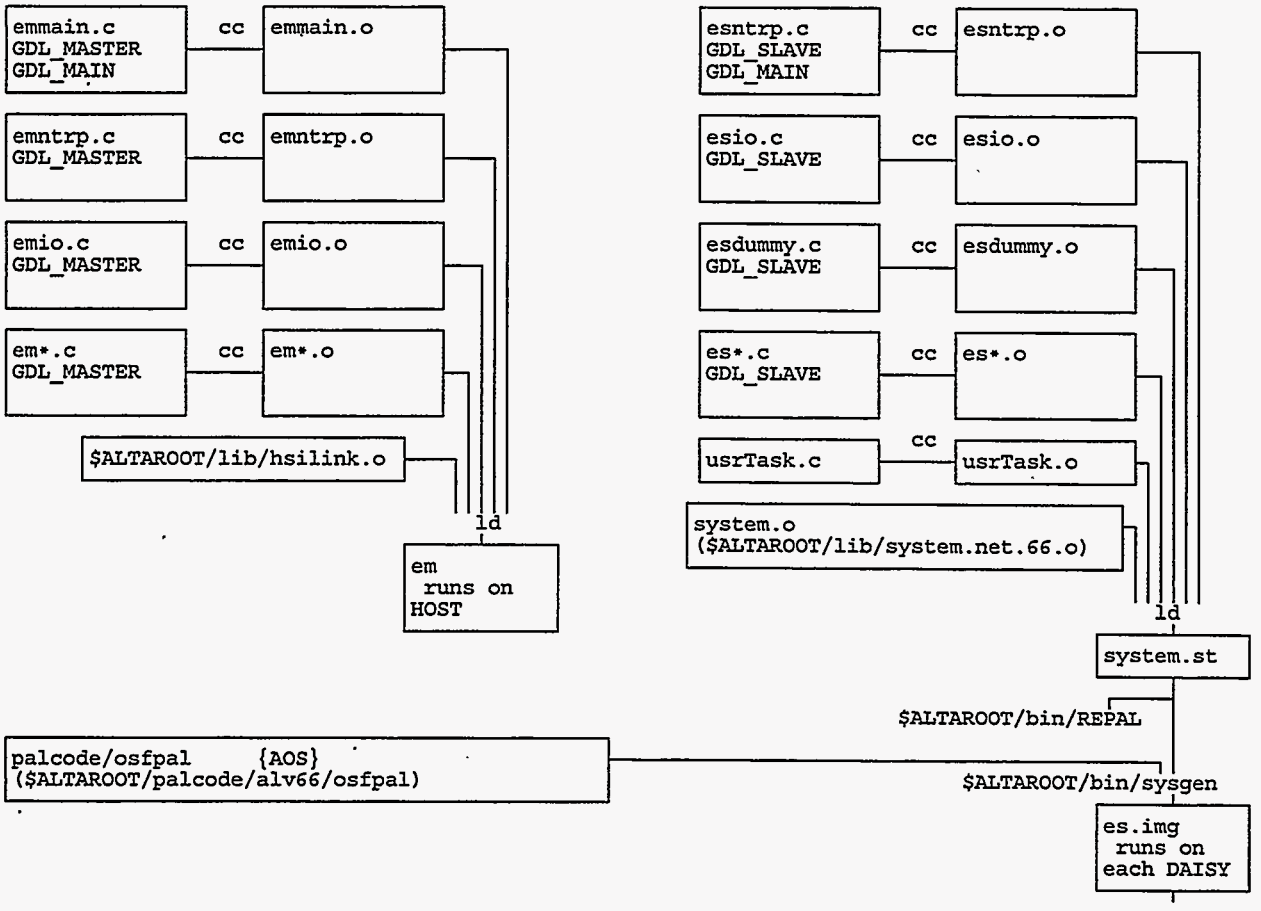

Figure 6: ATR3 build procedure 3.

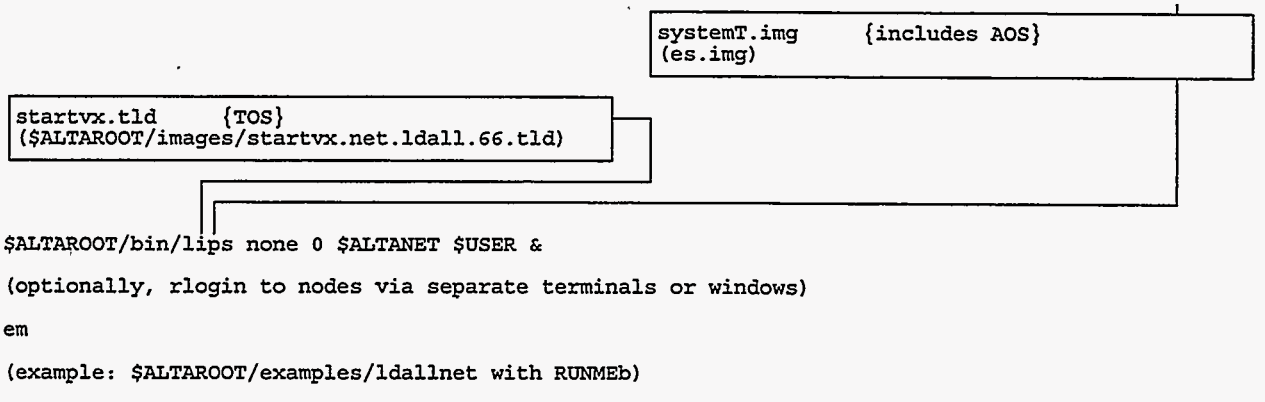

Figure 7: ATR3 load&run procedure 3.

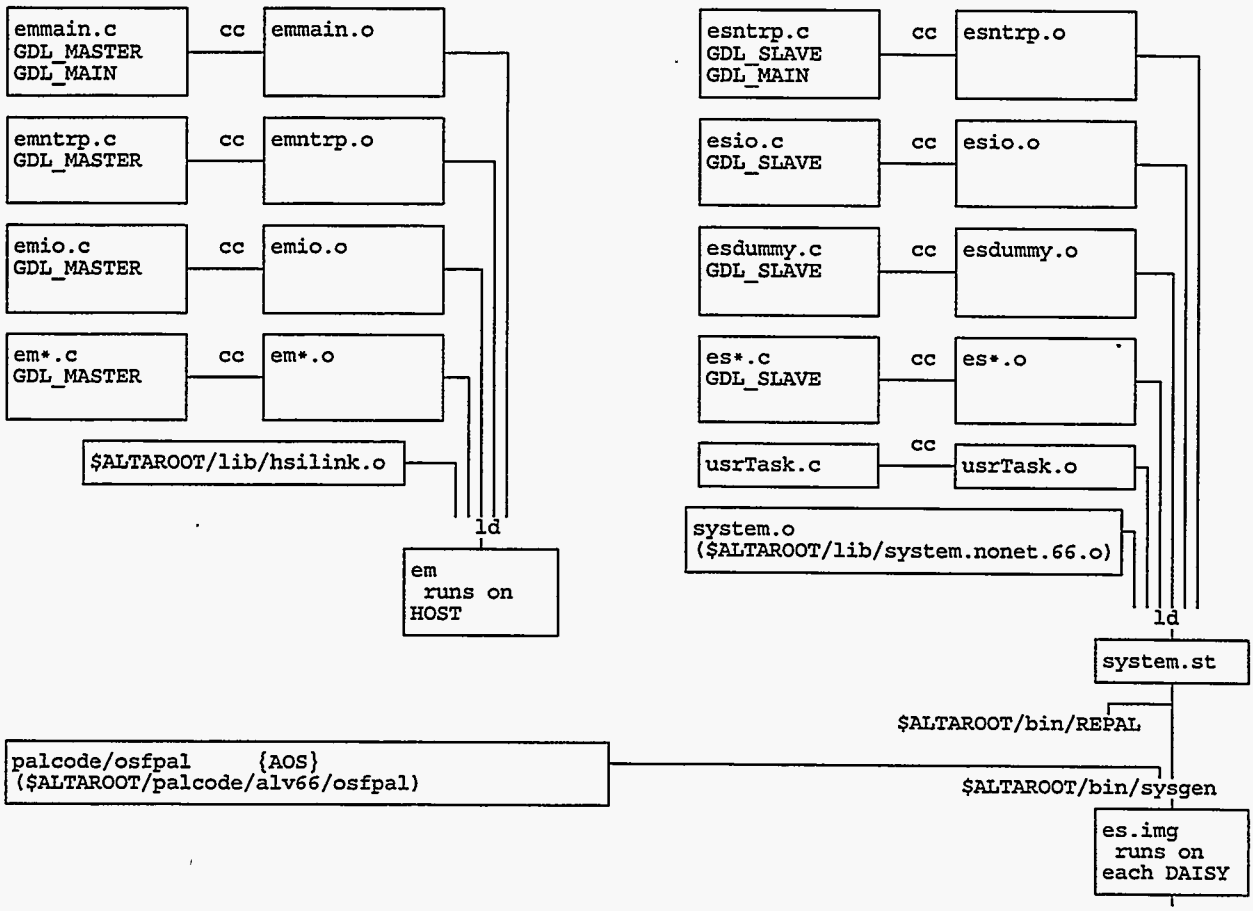

Figure 8: ATR3 build procedure 4.

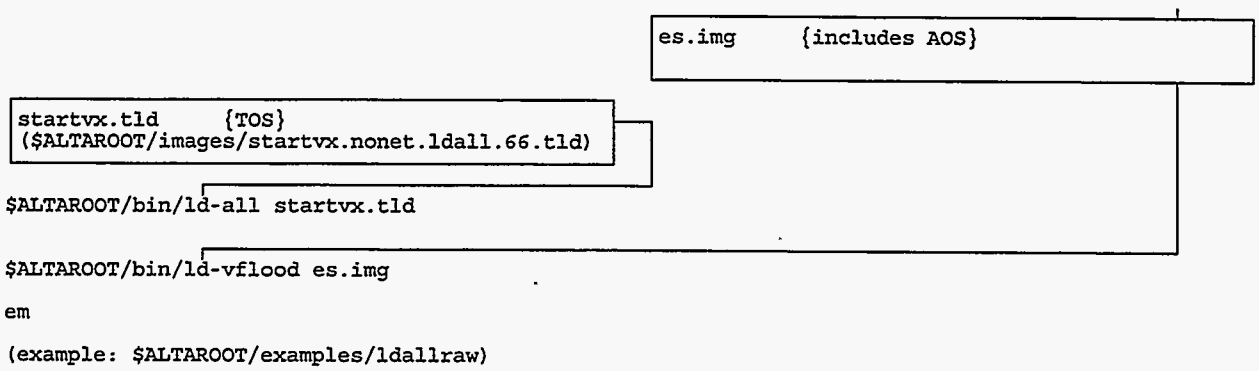

Figure 9: ATR3 load&run procedure 4.

PROCEDURE 1: net support, dynamic link, manual load
Assume links are set up for TCP/IP service.
Assume that the cbld program has been run.
Assume that the lips program is not already running.
1. Use "$ALTAROOT/examples/AlphaConfig 66 net ldall".
2. Use cc to make user code in the form of object (.o) files.
3. Use ld to combine the several object files into one executable file (em, for example) for MASTER, and one large object file (es.o, for example) for SLAVE.
4. Use "$ALTAROOT/bin/lips none 0 $ALTANET $USER &" to start TCP/IP service. This assumes that
startvx.tld --> $ALTAROOT/images/startvx.net.ldall.66.tld and
systemT.img --> $ALTAROOT/images/systemT.net.66.img,
as is done by AlphaConfig with the "net" parameter.
5. For each node: Log in to the node ("rlogin nodeN"). Use a command like
"ld <host:/usr/users/gdl/alta6/examples/ldallnet/proc1/es.o" (the path may be different)
to load your user code object files onto the node. The current software has a bug, which requires that this load be done in two steps:
(1) "ld <host:/usr/.../dlo.o", and
(2) dlo("usr","/usr/users/gdl/alta6/examples/ldallnet/proc1","es.o").
Start your code running by typing the name of your main function (which should not be named "main") or, preferably, by typing the spawn command such as "sp es" where "es" is the name of your "main" function. Optionally, logout of the node.
6. Start your host program running by typing the name of the MASTER executable (em, for example).

PROCEDURE 2: net support, dynamic link, script load
Assume links are set up for TCP/IP service.
Assume that the cbld program has been run.
Assume that the lips program is not already running.
1. Use "ALTAROOT/examples/AlphaConfig 66 net ldall".
2. Use cc to make user code in the form of object (.o) files.
3. Use ld to combine the several object files into one executable file (em, for example) for MASTER, and one large object file (es.o, for example) for SLAVE.
4. Create a script file to run in the VxWorks system on the Alpha nodes. This file should contain the commands to load and start execution of your program, like the ld and sp commands used in Procedure 1. You can name the script file whatever you like; assume for this example that it is "scriptfile".
5. Use "$ALTAROOT/bin/lips "host:"`pwd`/scriptfile 0 $ALTANET $USER &"
to start TCP/IP service and load and execute your program. This assumes that
startvx.tld --> $ALTAROOT/images/startvx.net.ldall.66.tld and
systemT.img --> $ALTAROOT/images/systemT.net.66.img,
as is done by AlphaConfig with the "net" parameter. If you wish, you can rlogin to a node at this time and verify that your program is running.
6. Start your host program running.

PROCEDURE 3: net support, static link, auto load
Assume links are set up for TCP/IP service.
Assume that the cbld program has been run.
Assume that the lips program is not already running.
1. Use "ALTAROOT/examples/AlphaConfig 66 net ldall".
2. Use cc to make user code in the form of object (.o) files. Your "main" SLAVE function must be named "void usrTask (void)".
3. Use ld to create an executable file (em, for example) from your MASTER object files, and to combine your SLAVE object files with $ALTAROOT/lib/system.net.66.o (or simply **system.o**, which is a link set up by AlphaConfig) and with $ALTAROOT/lib/system.a to create a file called system.st.
4. Use $ALTAROOT/bin/REPAL and $ALTAROOT/bin/sysgen. This assumes that the **palcode** link has been set up, as by AlphaConfig. Assume for this example that the output file from sysgen is called

es.img (system.img is a more common choice).
5. Use "rm systemT.img" and "ln -s es.img systemT.img"; we must have the name "systemT.img" refer to the file created by sysgen in step 4.
6. Use "$ALTAROOT/bin/lips none 0 $ALTANET $USER &" to start TCP/IP service. This assumes that
startvx.tld --> $ALTAROOT/images/startvx.net.ldall.66.tld and
systemT.img --> the file created by sysgen in step 4.
If you wish, you can rlogin to a node at this time and verify that your program is running.
7. Start your host program running.

PROCEDURE 4: no net support, static link, auto load
Assume that the cbld program has been run.
Assume that the lips program is not running.
1. Use "ALTAROOT/examples/AlphaConfig 66 nonet ldall".
2. Use cc to make user code in the form of object (.o) files. Your SLAVE "main" function must be named "void usrTask (void)".
3. Use ld to combine your SLAVE object files with $ALTAROOT/lib/system.nonet.66.o (or simply **system.o**, which is a link set up by AlphaConfig), and your MASTER object files with $ALTAROOT/lib/system.a.
4. Use $ALTAROOT/bin/REPAL and $ALTAROOT/bin/sysgen. This assumes that the **palcode** link has been set up, as by AlphaConfig. Assume for this example that the output file from sysgen is called es.img.
5. Use "$ALTAROOT/bin/ld-all startvx.tld" to load the transputers. This assumes that
startvx.tld --> $ALTAROOT/images/startvx.nonet.ldall.66.tld,
which is done by AlphaConfig with the nonet option.
6. Use "$ALTAROOT/bin/ld-vflood es.img" to load the Alphas. Your SLAVE program is now running on the parallel processors.
7. Start your MASTER program running on the host, by typing "em" to the host.
Note that the lips program is never used in this procedure, and this procedure will not work if the lips program was already running. You cannot monitor the parallel processor nodes with this procedure.

INTERNAL CHANNELS

On each node, there are at least three threads (tasks, processes) running concurrently: the main thread, which does most of the work; a recv thread, which monitors internode links for incomimg messages; and a send thread, which sends messages over internode links. Some communication is necessary among the threads. When the recv thread receives data from a link, it must notify the main thread that data is ready to be used in this node, or it must notify the send thread that data is ready to be relayed on to the next node, or both. The main thread must notify the recv thread when it is finished with data that recv passed to main. And, the main thread must notify the send thread when main has prepared a message that must be sent to another node. The easiest way to do these inter-thread communications is to associate a simple flag (an integer variable) with each data buffer, with one thread setting the flag when it has filled the buffer and each of the other threads executing a loop that reads the flag until the flag is set appropriately for that thread to use the data buffer. A disadvantage of this method appears in ATR3 when the user wishes to monitor a daisy using the TCP/IP communication system. The loop, which a thread executes while waiting for a data buffer to become available, can tie up the processor in that node and exclude or slow other processes such as TCP/IP communication. This suggests using a different procedure, in which no thread simply loops while waiting for a flag to be set, but rather messages are sent on internal channels between threads when a data buffer is ready. With this procedure, the thread that is waiting for the data buffer is blocked -- not using the central processor -- until it gets the internal channel message indicating that the data buffer is ready. This second procedure is used in the ATR software if the symbol GDL_SHARE is defined in file efg.h. The internal channels used in this case are indicated in Figures 10 and 11. The four internal channels are named recv2main, main2recv, main2send, and recv2send. The diagram indicates which functions in each thread use these channels, and the direction of information flow for each channel. The actual data that is sent via these four channels is not used; the only significance is that a message is sent over the channel. The linkup and linkdn threads in ATR3 are very simple processes that do nothing more then relay incoming data from a link to the recv thread. They are used because the current system software has a bug that prevents the ProcAlt function from working with external channels. These threads use two elements of the Channel * inn[] array as two additional internal channels for data transfer.

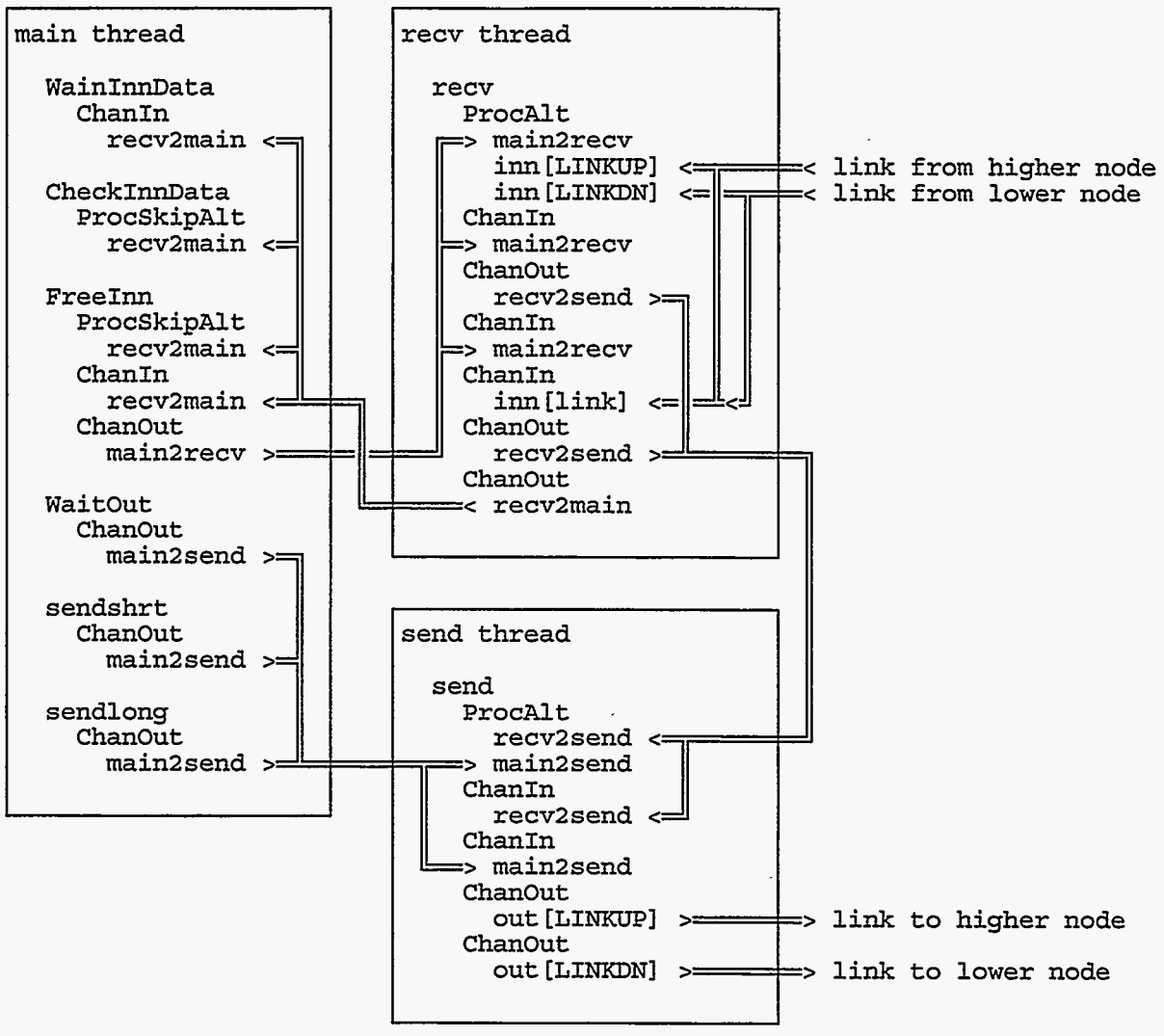

Figure 10: Internal channels in ATR2 software.

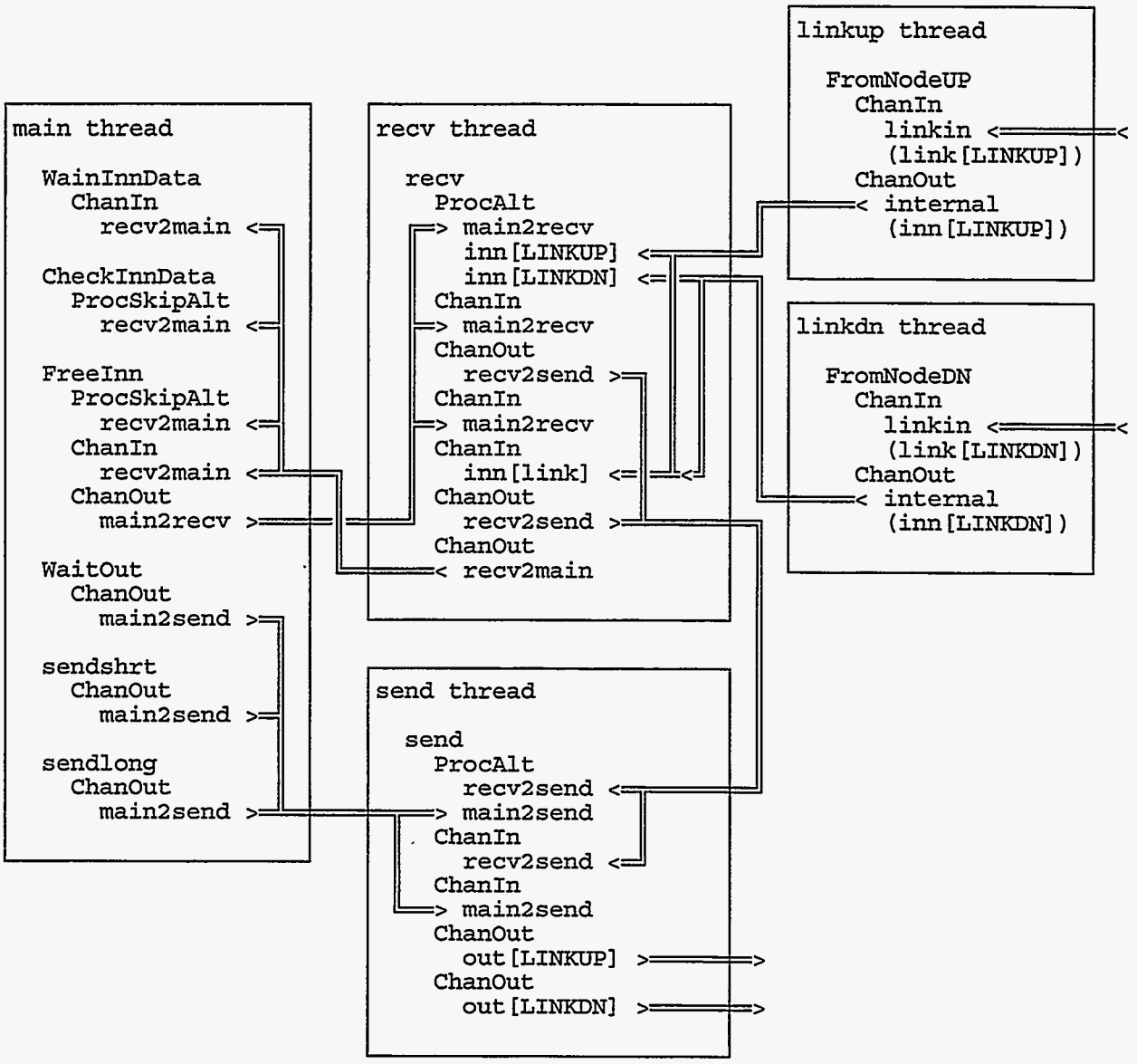

Figure 11: Internal channels in ATR3 software.

MASTER and SLAVE SOFTWARE

DEFINES

This section describes some of the important symbols that appear in #define ... statements.

I32 — the type of a 32-bit integer

F32 — the type of a 32-bit floating point value

GDL_ATR1, GDL_ATR2, GDL_ATR3
: one of these three sysbols must be defined at the beginning of the file efg.h, to indicate which hardware system the software is for

GDL_MASTER, GDL_SLAVE, GDL_SCSI, GDL_VIDEO
: one of these four symbols should be defined at the beginning of each source code (.c) file, before "#include "efg.h"" or "#include "comm.inc"", to indicate which type of node the software is to run on. MASTER indicates the controlling, coordinating program, which runs on ROOT or HOST. SLAVE indicates a DAISY node; the DAISY nodes share the bulk of the computational work. SCSI and VIDEO indicate speciallized SCSI bus interface and RGB display interface nodes that exist only in ATR1. These symbols should not be defined in file comm.inc (which is used each node's main or equivalent function file) or in file efg.h (which is used in all .c files).

GDL_MAIN — this symbol should be defined at the beginning, before "#include "efg.h"" or "#include "comm.inc"", of the file that includes the function main, or the equivalent function by some other name, for any of the nodes.

GDL_SHARE — See the section on INTERNAL CHANNELS

GLOBAL VARIABLES

This section describes some of the important symbols that are defined as global variables, in file efg.h.

I32 bufinni[MIOWRD]	bufinni is a buffer used to receive messages from other nodes and to relay messages on to the next node as appropriate. bufinni[2] - bufinni[9] are always received in any communication. If the bufinni[9] value is greater than 0, that number of additional bytes is received in the next data transfer as part of the same message. bufinni[0] and bufinni[1] are never received from other nodes, but are set internally by each node.
bufinni[0]	buffer status code: 0 --> this buffer is ready for a new use 1 --> other threads don't touch this buffer 2 --> main thread should use this data 4 --> this data should be output to another node
bufinni[1]	number of the link by which this data was received
bufinni[2]	number of the node that sent this data
bufinni[3]	lowest number node to receive this data
bufinni[4]	highest number node to receive this data; this data will go to all nodes with numbers between bufinni[3] and bufinni[4] inclusive
bufinni[5]	command code
bufinni[6]	command-specific data
bufinni[7]	command-specific data
bufinni[8]	command-specific data
bufinni[9]	the number of additional bytes to be received in the next communication, as a part of this message

F32 * bufinnf=(F32 *)bufinni
char * bufinnc=(char *)(bufinni+10)
bufinnf and bufinnc are F32 and char type pointers to the bufinni array. They facilitate transferring F32 and char data between nodes.

I32 bufouti[MIOWRD] bufouti is very similar to bufinni, except that bufouti is used to send data from the main thread of this node to another node. The first 10 words of bufouti have the same meanings as the corresponding words of bufinni, except for bufouti[1] which is not used.

F32 * bufoutf=(F32 *)bufouti
char * bufoutc=(char *)(bufouti+10)
bufoutf and bufoutc are F32 and char type pointers to the bufouti array. They facilitate transferring F32 and char data between nodes.

char * hfabuf=(char *)NULL
hfabuf is a scratch buffer used by several functions for reading HFA files.

I32 hfabufsiz=0 hfabufsiz is the size of the buffer hfabuf.

struct HFAobject hfaobject
hfaobject is the only instance of the structure HFAobject. It is used when reading images from HFA files; see the description of the function inn8HFA.

I32 np	the number of parameters in the user command line; this count does not include the command itself
F32 pf[MP]	pf[k] is the F32 representation of user command parameter k, if such a representation is possible
I32 pi[MP]	if user command parameter k is a single value (not a whole array) that can be represented as an I32 value (perhaps after truncation of a fractional part), pi[k] is that I32 value; or, if parameter k is a user-accessible array (not just one element of an array), pi[k] is the number of the user-accessible variable corresponding to that array
I32 pn[MP]	pn[k] is the value of the array index of user command parameter k, or -1 if no index brackets are part of parameter k, or -2 if parameter k includes empty index brackets
char pr[MP][VARLEN]	pr[k][] is a copy of user command parameter k as a character string
char pq[MP][VARLEN]	pq[k][] is a copy of user command parameter k as a character string, with any array index or subscript removed
char * ps[MP]	ps[k] is a pointer to the character string representation of user command parameter k
I32 pt[MP]	pt[k] is a code indicating the type of user command parameter k

STRUCTURES

This section describes some of the important global structures that are defined in file efg.h.

* *

STRUCTURE TYPE: BANDR, MASTER & SCSI

STRUCTURE INSTANCES: rband[MBAND], MASTER

STRUCTURE MEMBERS:

I32 img	the number of the memory image to receive this band, or 0 if this band is not to be used for a memory image
I32 byt0	the number of bytes from the beginning of the file image composite row to the beginning of the row for this band
I32 ncol	the number of pixels in one row for this band
I32 bpp	the number of bytes per pixel in this band

rband is an array of MBAND structures, each of which describes one band of a multi-band image file. This array of structures is used when reading the file.

* * * * *.* *

STRUCTURE TYPE: HFAobject

STRUCTURE INSTANCES: hfaobject

STRUCTURE MEMBERS:

I32 n	indicates the n-th occurrence of the HFA file object with name and type matching the .name and .type members of this structure
char name[NAMELEN]	the name of the sought HFA file object
char type[NAMELEN]	the type of the sought HFA file object

This hfaobject structure is used by functions like inn8HFA, which scan an HFA file until they find the hfaobject.n-th occurrence of an object with name and type that match hfaobject.name and hfaobject.type. hfaobject.name or hfaobject.type may be " * ", in which case any name or type is construed as matching.

* *

STRUCTURE INSTANCES: imtab See TABIMG

* *

STRUCTURE INSTANCES: imtyp See TYPIMG

* *

STRUCTURE INSTANCES: konop See TABOPR

* *

STRUCTURE INSTANCES: rband See BANDR

* *

STRUCTURE TYPE: TABIMG, MASTER & SLAVE

STRUCTURE INSTANCES: imtab[MIMG], MASTER & SLAVE

STRUCTURE MEMBERS, MASTER:

I32 img	this image number if this image exists, otherwise -1
I32 ncol	number of columns in this image
I32 nrow	number of rows in this image
I32 novl	number of overlap rows in this image
F32 a	A in (this image) = A + B * (other image)
F32 b	B in (this image) = A + B * (other image)

STRUCTURE MEMBERS, SLAVE:

I32 img	this image number if this image exists, otherwise -1
F32 *addr	memory address of this node's part of this image
I32 ncol	number of columns in this image
I32 nrow	number of rows in this image
I32 novl	number of overlap rows in this image
I32 rlo	lowest row number of this image in this node, including overlap rows
I32 rhi	highest row number of this image in this node, including overlap rows
I32 alo	DISUSED; lowest row number to be analyzed in this node
I32 ahi	DISUSED; highest row number to be analyzed in this node
I32 slo	lowest row number of this image in this node, excluding overlap rows
I32 shi	highest row number of this image in this node, excluding overlap rows

imtab is an array of structures, each of which describes one image in memory. The structure is different for MASTER and SLAVE nodes, because MASTER, not SLAVE, maintains the A and B values for each image and because SLAVE, not MASTER, stores image data in memory.

* *

STRUCTURE TYPE: TABOPR, MASTER & SLAVE

STRUCTURE INSTANCES: konop[MOPR], MASTER & SLAVE

STRUCTURE MEMBERS, MASTER & SLAVE:

I32 opr	this kernel number if this kernel is defined, or -1
F32 *addr	address of this kernel in memory
I32 mini	lowest i (column) number in this kernel's domain, relative to the origin pixel
I32 maxi	highest i (column) number in this kernel's domain, ...
I32 minj	lowest j (row) number in this kernel's domain, ...
I32 maxj	highest j (row) number in this kernel's domain, ...

konop is an array of MOPR structures, each of which describes one kernel in memory. The kernel comprises (maxi-mini+1) * (maxj-minj+1) F32 values. Kernel 0 is a dummy, whose domain is only the origin pixel.

* *

STRUCTURE TYPE: TYPIMG, MASTER

STRUCTURE INSTANCES: imtyp[MIMGTYP], MASTER

STRUCTURE MEMBERS, MASTER:

I32 nhead	number of bytes in the image file header
I32 col0	number of the header byte that contains the less significant part of the number of columns in the image
I32 col1	number of the header byte that contains the more significant part of the number of columns in the image
I32 row0	number of the header byte that contains the less significant part of the number of rows in the image
I32 row1	number of the header byte that contains the more significant part of the number of rows in the image
I32 bpp	bytes per pixel

imtyp is an array of MIMGTYP structures, each of which contains data on image structure and header contents for one image file format.

* *

STRUCTURE TYPE: (none)

STRUCTURE INSTANCES: var[MUAV]

STRUCTURE MEMBERS:

char nam[VARLEN]	name of variable
void * addr	address of variable in memory
I32 typ	variable type code
I32 dim	number of array dimensions; 0 for a scalar variable, 1 for a one-dimensional array
I32 siz	number of elements in this array; 1 for a scalar

var[k] is a structure that contains information about the k-th user-accessible variable.

* *

FUNCTIONS

This section describes all the functions in both the MASTER and SLAVE programs. After the function name, at the top of each entry, is an indication of the file or files in which the function occurs, and an indication of whether the function occurs in the MASTER or the SLAVE program. In many cases, two different functions with the same name, and often with the same argument list, occur in two different files, one for the MASTER program and one for the SLAVE program. A few functions occur in only one of the E, F, or G programs; if there is no indication to the contrary, the function is used in all three of E, F, and G.

* *

void abort0 (mess) file emmain.c, fmmain.c, & gmmain.c, MASTER

char * mess	message to be printed

abort0 prints the specified error message and stops program execution.

* *

void abort2 (mess, c) file emmain.c, fmmain.c, & gmmain.c, MASTER

char * mess	message to be printed
I32 * c	communication buffer to be partially printed

abort2 prints "UNEXPECTED MESSAGE FROM SLAVE" along with the message supplied as the first argument and part of the communication buffer specified as the second argument, and stops program execution.

* *

void abort8 (msg) file comm.inc, SLAVE

char * msg	message to be displayed

abort8 is used by a slave node that does not have direct access to the host devices, to print an error message and stop program execution.

* *

void abort9 (c) file emmain.c, fmmain.c, & gmmain.c, MASTER

I32 * c	communication buffer with message from slave

abort9 is used in response to a message from a slave, printing the "SLAVE RECEIVED BAD COMMAND" message and part of that bad command and then stopping program execution.

* *

I32 alist (psrc, pv, pi, jhi, listlen, inc) file esfilt.c, SLAVE

F32 * psrc	an array of values that may be put into the list; the source image data

F32 *pv	an array of values that are in the list
I32 *pi	an array of the psrc indices for the values that are in the list
I32 jhi	the highest psrc index value for which the psrc value should be added to the list
I32 listlen	length of existing list
I32 inc	the increment between successive psrc index values used in this function

alist is used by the one-dimensional median filter functions med1x and med1y. alist adds pixel values to the list that comprises all the pixel values from the current local region, adding pixels with index less than or equal to jhi. alist returns the new list length.

* *

F32 asig (sig, n) file emfilt.c, MASTER

F32 sig	total standard deviation of smoothing function
I32 n	number of smoothing passes

asig is used by smthx and smthy. These functions do multi-pass exponential smoothing. asig calculates the witdth of the smoothing function required for each separate pass, in order to achieve the desired total smoothing function width. asig returns the value of a constant related to the single-pass width, which constant is used by the slave functions smthx and smthy.

* *

void asums (void) file fmopt.c, MASTER, F only

asums accumulates certain sums of feature image pixel values and their products, for use in the training process. Each call to asums accumulates the values for one training scene. zsums should be called before the first call to asums for the first of a sequence of scenes, and fsums should be called after the last call to asums. asums uses sums2.

* *

F32 *brent2 (func, fparm, iparm, ax, bx, cx, atol, rtol, itmax) file emman.c, MASTER

F32 *func(...)	represents the single-variable function to be minimized
F32 *fparm	an array of F32 parameters
I32 *iparm	an array of I32 parameters
F32 ax	the minimum allowable value of x, the function argument
F32 bx	an intermediate value of x, the function argument
F32 cx	the maximum allowable value of x, the function argument
F32 atol	fraction-of-range convergence criterion
F32 rtol	fraction-of-value convergence criterion
I32 itmax	maximum allowed number of iterations

brent2 is not used in the current version of the software; it is replaced by brent3.

This function is adapted from William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, "Numerical Recipes in C, Second Edition", Cambridge University Press, 1992. This function brent2 finds a minimum in the single-variable function represented by func(x,...), using Brent's procedure, with the function minimum position bounded by ax and cx. A test has been added to make sure the parabolic extremum is a minimum. This allows this function brent2 to work even if the initial middle point bx does not have a function value lower than those of the two end points ax and cx. The result from such a case may be one of the end points, which is a minimum in the constrained domain, instead of a true unconstrained minimum.

func represents a function of one primary variable x, perhaps with some indirect dependence on fparm and iparm. func (see reg2c, for example) must be of a particular form:

* The second and third arguments of func are fparm and iparm, the same as the second and third arguments of brent2. The first argument of func is an F32 value, call it x.
* the function is to be minimized with respect x.
* func returns a pointer to an F32 array, with array element 0 set to the function value, array element 1 set to the x value used in the function evaluation (which may be different from the x value supplied as an argument, depending on how func is programmed), and array elements 2 and 3 reserved for future use as a condition code and a not-yet-specified use.

ax, bx, and cx are three values of x, with ax < bx < cx, usually (but not necessarily in this modified version of brent2) such that func(bx) is less than both func(ax) and func(cx).

atol is a fraction-of-range convergence criterion. Convergence is assumed complete when the x-range that includes the minimum point is smaller than atol * (cx-ax). If a negative value is given for atol, the value is changed to 0.000001 (a rather arbitrarily selected value). The use of atol as a convergence criterion can effectively be suppressed by supplying a value of 0.0 for atol.

rtol is a fraction-of-value convergence criterion. Convergence is assumed complete when the x-range that includes the minimum point is smaller than rtol * x, where x is the current estimate of the minimum point. If a value less than 0.00025 is given for rtol, the value is changed to 0.00025 at the beginning of brent2 (this value is appropriate for F32, single-precision floating point, calculations). Convergence is assumed if either the atol or the rtol criterion is satisfied.

itmax is the maximum number of iterations allowed. If the value 0 is given for itmax, there is no limit on the number of iterations.

On entry to brent2, fparm[0] must be the function value at bx. brent2 returns a pointer to an F32 array. Element 0 of the array is the minimum function value; element 1 is the value of x at the function's minimum point; element 2 is a status code: 0 indicates normal completion, 1 indicates some error that does not allow normal completion, 2 warns that the minimum point is one of the range end points, 3 indicates that no minimum was found, and 4 indicates that too many iterations were used; element 3 is the number of iterations done.

* *

F32 * brent2a (func, fparm, iparm, atol, rtol, itmax) file emman.c, MASTER

F32 * func(...)	represents the multi-variate function to be minimized
F32 * fparm	an array of F32 parameters for func and brent2a
I32 * iparm	an array of I32 parameters for func and brent2a
F32 atol	fraction-of-range convergence criterion for brent2
F32 rtol	fraction-of-value convergence criterion for brent2
I32 itmax	maximum allowed number of iterations for brent2

brent2a is not used in the current version of the software; it is replaced by brent3a.

brent2a is an example of the linmin function used by function powell. brent2a finds the minimum of a multivariate function along a line in the multi-dimensional variable space. brent2a uses the single-variable function minimization procedure brent2.

The multivariate function is represented by func (see reg2c, for example), which must be of a particular form:

* The second and third arguments of func are fparm and iparm, the same as the second and third arguments of brent2a. The first argument of func is an F32 value, call it x.
* the function is to be minimized with respect to n variables, fparm[1], fparm[2], ... fparm[n], where n = iparm[0].
* func evaluates the function at the point fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n].
* func returns a pointer to an F32 array, with array element 0 set to the function value, array element 1 set to the x value used in the function evaluation (which may be different from the x value supplied as an argument, depending on how func is programmed), and array elements 2 and 3 reserved for future use as a condition code and a not-yet-specified use.

On entry to brent2a:

* fparm[1], fparm[2], ... fparm[n] must be a point on the line along which the function is to be minimized, and fparm[0] must be the function value at that point.
* fparm[1+n], fparm[2+n], ... fparm[n+n] must specify the direction of the line along which the function is to be minimized. That is, points on the line are specified by fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n], with x being a measure of position along the line.
* fparm[1+2n], fparm[2+2n], ... fparm[n+2n] must be lower limits for the values of the n function variables, and fparm[1+3n], fparm[2+3n], ... fparm[n+3n] must be upper limits for the values of the n function variables. The combination fparm[i+2n]=1.0 and fparm[i+3n]=-1.0 can be used to indicate that variable i has no explicit limits. For the whole set of n variables, at least 2 limits must be imposed such that there is an implied limit on x values, an implied limit on the length of the line along which the function is to be minimized.

brent2a is arranged so that it simply returns the first guess input point if the parameter limits are both the same as the first guess parameter value, if the parameter range has zero width. This facilitates using powell with brent2a to minimize multivariate functions with some of the variables constrained to constant values.

brent2a returns a pointer to an F32 array. Array element 0 is the minimized function value; element 1 is the value of x at the minimum; element 2 is a status code: 0 indicates normal completion, 1 indicates some error that does not allow normal completion, 2 warns that the minimum point is one of the range end points, 3 indicates that no minimum was found, 4 indicates that too many iterations were used in brent2, -1 indicates failure because of unbounded parameters, -2 indicates that the parameter range has zero width (sometimes an indication that the function was already minimized), and -3 indicates failure because the initial guess was not in the acceptable parameter range; and element 3 is the number of iterations done in brent2 (brent2a does not iterate). Also, on return from brent2a, fparm[1], fparm[2], ... fparm[n] indicates the location of the minimum of the function along the specified line, and fparm[0] is the value of the function at the minimum.

* *

F32 * brent3 (func, fparm, iparm, ax, bx, cx, atol, rtol, itmax) file emman.c, MASTER

F32 * func(...)	represents the single-variable function to be minimized
F32 * fparm	an array of F32 parameters
I32 * iparm	an array of I32 parameters
F32 ax	the minimum allowable value of x, the function argument
F32 bx	an intermediate value of x, the function argument
F32 cx	the maximum allowable value of x, the function argument
F32 atol	fraction-of-range convergence criterion
F32 rtol	fraction-of-value convergence criterion
I32 itmax	maximum allowed number of iterations

This function is adapted from William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, "Numerical Recipes in C, Second Edition", Cambridge University Press, 1992. This function brent3 finds a minimum in the single-variable function represented by func(x,...), using Brent's procedure, with the function minimum position bounded by ax and cx. For this brent3 version, the function func is presumed to represent 1-correlation for some correlation process, so that the normal function values range from 0.0 to 2.0.

func represents a function of one primary variable x, perhaps with some indirect dependence on fparm and iparm. func (see reg3c, for example) must be of a particular form:

* The second and third arguments of func are fparm and iparm, the same as the second and third arguments of brent3. The first argument of func is an F32 value, call it x.
* the function is to be minimized with respect x.
* func returns a pointer to an F32 array. Array element 0 is set to the function value, which must be 1-correlation for some correlation process; element 1 is set to the x value used in the function evaluation (which may be different from the x value supplied as an argument, depending on how

func is programmed); element 2 is a status code: 0 for normal return, 1 for an unspecified error, 2 if the function argument is out of the allowable range; and, element 3 is the number of degrees of freedom associated with the correlation calculation.

ax, bx, and cx are three values of x, with ax < bx < cx, usually (but not necessarily in this modified version of brent3) such that func(bx) is less than both func(ax) and func(cx).

atol is a fraction-of-range convergence criterion. Convergence is assumed complete when the x-interval that includes the minimum point is smaller than atol * (cx-ax). If a negative value is given for atol, the value is changed to 0.000001 (a rather arbitrarily selected value). The use of atol as a convergence criterion can effectively be suppressed by supplying a value of 0.0 for atol.

rtol is a fraction-of-value convergence criterion. Convergence is assumed complete when the x-interval that includes the minimum point is smaller than rtol * x, where x is the current estimate of the minimum point. If a value less than 0.00025 is given for rtol, the value is changed to 0.00025 at the beginning of brent3 (this value is appropriate for F32, single-precision floating point, calculations). Convergence is assumed if either the atol or the rtol criterion is satisfied.

itmax is the maximum number of iterations allowed. If the value 0 is given for itmax, there is no limit on the number of iterations. If a value greater than 0 is specified for itmax, it must be at least 3 or brent3 will never indicate a successful minimization.

brent3 returns a pointer to an F32 array. Element 0 of the array is the minimized function value; element 1 is the value of x at the function's minimum point; element 2 is a status code: 0 indicates normal completion, 1 indicates that no acceptable minimum was found, and 2 indicates that the values that were input to brent3 represented an already-converged condition (this is normally not an error); and, element 3 is the number of iterations done. brent3 does not directly change any of the fparm or iparm values, although brent3 does call func which may change these values.

This version of brent3 requires that the function have a minimum (the correlation have a maximum) within the specified interval. brent3 then uses all the function values it has calculated in finding this minimum, and estimates the uncertainty (standard deviation) in the position of the assumed-parabolic correlation peak. If the peak position uncertainty is greater than half of the initial x-interval cx-ax, the result is declared unacceptable and brent3 returns the status code 1.

* *

F32 * brent3a (func, fparm, iparm, atol, rtol, itmax) file emman.c, MASTER

F32 * func(...)	represents the multi-variate function to be minimized
F32 * fparm	an array of F32 parameters for func and brent3a
I32 * iparm	an array of I32 parameters for func and brent3a
F32 atol	fraction-of-range convergence criterion for brent3
F32 rtol	fraction-of-value convergence criterion for brent3
I32 itmax	maximum allowed number of iterations for brent3

brent3a is an example of the linmin function used by function powell. brent3a finds the minimum of a multivariate function along a line in the multi-dimensional variable space. brent3a uses the single-variable function minimization procedure brent3.

The multivariate function is represented by func (see reg3c, for example), which must be of a particular form:

* The second and third arguments of func are fparm and iparm, the same as the second and third arguments of brent3a. The first argument of func is an F32 value, call it x.
* the function is to be minimized with respect to n variables, fparm[1], fparm[2], ... fparm[n], where n = iparm[0].
* func evaluates the function at the point fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n].
* func returns a pointer to an F32 array. Array element 0 is set to the function value, which must be 1-correlation for some correlation process; element 1 is set to the x value used in the function evaluation (which may be different from the x value supplied as an argument, depending on how

func is programmed); element 2 is a status code: 0 for normal return, 1 for an unspecified error, 2 if the function argument is out of the allowable range; and, element 3 is the number of degrees of freedom associated with the correlation calculation.

brent3a does not directly require that the function func represent a 1-correlation value, but brent3a uses brent3 which does make that requirement.

On entry to brent3a:

* fparm[1], fparm[2], ... fparm[n] must be a point on the line along which the function is to be minimized.
* fparm[1+n], fparm[2+n], ... fparm[n+n] must specify the direction of the line along which the function is to be minimized. That is, points on the line are specified by fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n], with x being a measure of position along the line.
* fparm[1+2n], fparm[2+2n], ... fparm[n+2n] must be lower limits for the values of the n function variables, and fparm[1+3n], fparm[2+3n], ... fparm[n+3n] must be upper limits for the values of the n function variables. The combination fparm[i+2n]=1.0 and fparm[i+3n]=-1.0 can be used to indicate that variable i has no explicit limits. For the whole set of n variables, at least 2 limits must be imposed such that there is an implied limit on x values, an implied limit on the length of the line along which the function is to be minimized.

brent3a is arranged so that it simply returns the first guess input point if the parameter limits are both the same as the first guess parameter value, if the parameter range has zero width. This facilitates using powell with brent3a to minimize multivariate functions with some of the variables constrained to constant values.

brent3a returns a pointer to an F32 array. Array element 0 is the minimized function value; element 1 is the value of x at the minimum; element 2 is a status code: 0 indicates normal completion, 1 indicates that no acceptable minimum was found, and 2 indicates that the parameter values input to brent3a represented an already-converged condition (this is normally not an error); and element 3 is the number of iterations done in brent3 (brent3a does not iterate). Also, on return from brent3a, fparm[1], fparm[2], ... fparm[n] indicates the location of the minimum of the function along the specified line, and fparm[0] is the value of the function at the minimum, if an acceptable minimum was found. If no acceptable minimum was found, fparm[0] ... fparm[n] are not changed by brent3a.

* *

void c2I32 (c, I)

char * c	pointer to a location to receive 2 characters
I32 I	an I32 value

c2I32 copies the least significant byte of the I32 value to the first byte at location c, and the second least significant byte to the second byte at c.

* *

void c4I32 (c, I)

char * c	pointer to a location to receive 4 characters
I32 I	an I32 value

c4I32 copies the least significant byte of the I32 value to the first byte at location c, the second least significant byte to the second byte at c, etc. for all 4 bytes.

* *

I32 CheckInnData (void) file comm.inc, MASTER & SLAVE

CheckInnData returns 1 if bufinni contains data for the main thread, 0 otherwise. CheckInnData does not wait for bufinni to receive data; it returns promptly. Data is put into bufinni by function rcv running in a separate thread.

* *

(macro) chknp(n) file emntrp.c, MASTER

I32 n	minimum acceptable number of parameters

chknp checks whether np (the number of parameters in the user command line) is at least as large as n, and prints a warning message if not. For programs E and F, chknp also stops execution if np is less than n.

* *

void clrmem (void) file emman.c, MASTER & esman.c, SLAVE

clrmem clears the tables of defined images, kernels, and user-defined variables, and de-allocates memory that was allocated for these items; defines kernel 0; and initiallizes rband, setting all the .img members to 0 and all the other members to -1. clrmem implements the CLEAR user command.

* *

void contour (dst, src, sval, dval) file esmath.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 sval	the source image pixel value for which the contour is to be drawn
F32 dval	the destination image pixel value with which the contour is to be drawn

contour draws one contour curve in destination image dst, representing a curve of constant intensity in source image src. If a pixel in src has intensity at least as great as the floating point value sval, and at least one of its four nearest neighbors has intensity less than sval, then the corresponding pixel in dst is set equal to the floating point value dval. Otherwise, dst pixel values are left unchanged. src should have at least 1 overlap row. dst does not need to be the same size as src. contour implements the CONTOUR user command.

* *

void convlv (dst, src, opr) file esconv.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel which is to be convolved with the source image

convlv sets image dst equal to the convolution of image src with kernel opr. That is, set each dst pixel equal to the sum of {src * opr}, summed over the domain of the kernel opr with the kernel's origin pixel positioned on the src pixel that corresponds to the dst pixel. The pixels of image dst in the excluded edge region, which is defined by the domain of the kernel opr, are set to zero. src should have enough overlap rows to accommodate kernel opr. The number of overlap rows should be at least as large as the larger of the absolute value of jmax or the absolute value of jmin for kernel opr. The image dst does not need to be the same size as the image src, but the two images do need to be distributed among the slaves in a manner compatible with the RESAMPLE algorithms. This is not strictly a proper convolution calculation, which would require reversing the signs of the two indexes in the kernel. convlv implements

the CONVOLVE user command.

* *

void copedg (dst, src, opr) file esman.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel that defines the edge region to be copied

copedg copies the values of the pixels at the edges of image src to the corresponding pixels of image dst. The edge pixels are specified by the domain of the kernel opr. That is, if the origin pixel of kernel opr can be aligned with a certain pixel of image src without the domain of the kernel extending off the image, then that certain pixel is NOT an edge pixel and it will not be copied. Images dst and src should be the same size. copedg implements the COPYEDGES user command.

* *

I32 copid (dst, src, novl) file emman.c, MASTER
void copid (dst, src, novl) file esman.c, SLAVE

I32 dst	number of the new image
I32 src	number of the old image
I32 novl	number of overlap rows for the new image

copid defines space for previously undefined image dst so that it is compatible with the previously defined image src. That is, the total sizes are the same for the two images, and their primary rows are distributed the same way among the several slaves, so that the two images are compatible for 2- or 3-image operations. The new image dst will have novl overlap rows regardless of how many overlap rows the old image src had. copid does not set any pixel values. copid implements the COPDEF user command.

* *

void cw1 (void) file fmopt.c (and emdummy.c), MASTER & fsopt.c (and esdummy.c), SLAVE, F only

cw1 uses the sums from fsums or rsums, and implements the training process by optimizing the coefficients for all the best subsets of feature images. MASTER cw1 uses cw1c, setfex, and incfex. SLAVE cw1 uses cw1b.

* *

void cw1b (void) file fsopt.c, SLAVE, F only

cw1b is used by SLAVE cw1, to optimize the coefficients for one subset of feature images in the training process. cw1b uses hdist, hdistx, hdist1, hdist2, and eint2.

* *

I32 cw1c (kfr, cfr, port) file fmopt.c, MASTER, F only

I32 * kfr
F32 * cfr
F32 * port

cw1c is used by MASTER cw1, to receive from a slave node the results of optimizing the coefficients for a particular set of feature images in the training process.

* *

void deci (dst, src, opr, xf, yf, novld) file emman.c, MASTER

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the exclusion kernel
F32 xf	x-direction (horizontal) decimation factor
F32 yf	y-direction (vertical) decimation factor
I32 novld	number of overlap rows for destination image

deci undersamples image src and puts the result in image dst. If image dst does not already exist, it is created with novld overlap rows. Its x and y dimensions will be equal to the src image x and y dimensions, excluding the edge region specified by kernel opr, divided by xf and yf respectively. If image dst already exists, it is used as is. deci simply uses the xf and yf values to calculate the size of the image dst, and then calls function resam1. deci implements the UNDERSAMPLE user command.

* *

I32 defimg (img, ncol, nrow, novl) file emman.c, MASTER
void defimg (img, ncol, nrow, novl) file esman.c, SLAVE

I32 img	number of the image being defined
I32 ncol	number of columns
I32 nrow	number of rows
I32 novl	number of overlap rows

The action of the function defimg depends on the value of the global variable np, the number of parameters in the user command line.

If np is zero, defimg lists on the screen the image number, number of rows, number of columns, and number of overlap rows for each defined image.

If np is one, defimg sets the user-accessible variables $NCOL, $NROW, and $N (internal variables _NCOL, _NROW, and _NUM) to the number of columns, number of rows, and number of overlap rows in image img. If echo is on, these values are printed on the screen.

If np is greater than one, defimg reserves memory in each slave to hold part of image img, and enters image description data in the appropriate tables. novl is the number of overlap rows. The full image comprises ncol columns by nrow rows. defimg sets the scale factors A and B for image img to 0.0 and 1.0. defimg does not set pixel values. If either ncol or nrow is 0, the image img is un-defined; its memory space is deallocated and any image data is lost.

The operator does not have control over which part of the image is held by which slave. Two images of the same size may be distributed differently among the several slaves, if the two images do not have the same number of overlap rows or if they were created by different processes; such image pairs are not compatible for most multiple-image operations. If two (or more) images are defined by this defimg function with the same values of all 3 parameters ncol, nrow, and novl, they will be compatible for two-image operations. The copid function (the COPDEF user command) can be used to ensure that two or more images will be compatible, even if they have different numbers of overlap rows.

defimg implements the DEFIMG user command.

* *

void defopr (opr, imin, imax, jmin, jmax) file emman.c, MASTER & esman.c, SLAVE

I32 opr	number of the kernel to be defined
I32 imin	minimum column in the kernel domain, relative to the destination pixel column
I32 imax	maximum column in the kernel domain, ...
I32 jmin	minimum row in the kernel domain, ...
I32 jmax	maximum row in the kernel domain, ...

The action of the function defopr depends on the value of the global variable np, the number of parameters in the user command line.

If np is zero, defopr lists on the screen the kernel number and the domain limits for each kernel that is defined.

If np is one, defopr sets the user-accessible variables $MINI, $MAXI, $MINJ, and $MAXJ (internal variables _MINI, _MAXI, _MINJ, and _MAXJ) equal to the domain limits of kernel opr. If echo is on, these values are listed on the screen.

If np is at least 5, defopr reserves memory in MASTER and in each SLAVE to hold all of kernel opr. The parameters imin, imax, jmin, jmax define the domain of this kernel, in the horizontal (positive to the right) and the vertical (positive downward) directions respectively, relative to the "origin" pixel. The origin pixel does not need to be in the two-dimensional domain specified by imin, imax, jmin, jmax. The origin pixel is merely the pixel with kernel domain coordinates i=0 and j=0, which is often, but not necessarily, in the center of the domain. opr must be greater than 0. Kernel 0 is always automatically defined with imin, imax, jmin, jmax all equal to zero; this is useful as a dummy kernel with a one-pixel domain. Using kernel 0 to specify an excluded edge region implies that none of the image is excluded.

defopr implements the DEFKERN user command.

* *

void defudv (typ) file emmath.c, MASTER

I32 typ	a user-accessible variable type code

defudv defines new user-accessible variables of type typ, one for each of the np parameters in the command line. Variable names are limited to 30 characters. The first character of a variable name must be an alphabetic character, not a numeral or a special character. You must not try to re-define an already defined variable. This operation does not assign values to the newly-defined variables. defudv uses defudv1 and implements the SDEFF32 and SDEFI32 user commands.

* *

I32 defudv1 (typ, k) file emmath.c, MASTER

I32 typ	a user-accessible variable type code
I32 k	the number of a parameter in the command line

defudv1 defines a new user-accessible variable with the name specified by command line parameter k, with type typ. This definition includes memory allocation for the variable's value (or values, if an array is implied by the variable pn for this parameter). defudv1 returns the index of the newly-defined variable.

* *

void dispinit (argc, argv) file emio.c, MASTER

int argc	argument count for the command line that invoked the master program
char * *argv	an array of pointers to command line arguments

dispinit is a very short function which calls the correct machine-specific function -- qt9dinit for ATR2

or xwininit for ATR3 (the ATR1 system in initiallized in the separate VIDEO program) -- for initiallizing the high-resolution RGB display software. dispinit is called once, when the program starts running.

* *

void dispres (img, row0, col0, q, caption, res) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen pixel row for top of image
I32 col0	screen pixel column for left edge of image
F32 q	green/red pixel color criterion
char * caption	caption to be printed under the image being displayed
I32 res	number of the image to be used for red/green pixel color decision

dispres is a short function which calls the correct machine-specific function -- ttg3res, qt9dres, or xwinres if res is not zero, or ttg3, qt9d, or xwin if res is zero -- for displaying an image on the high-resolution RGB display. dispres implements the user commands DISP and DISPRES.

A special case is img=-257, for which dispres does nothing except set the user-accessible variables _MAXI ($MAXI) and _MAXJ ($MAXJ) to the width and height of the RGB display screen, and _MINI ($MINI) and _MINJ ($MINJ) to the width and height of an alphanumeric character on the RGB display.

* *

I32 dlist (pv, pi, ilo, listlen) file esfilt.c, SLAVE

F32 * pv	an array containing the values in the list
I32 * pi	an array containing the indices associated with the values in the list
I32 ilo	the index value such that any values with lower index values should be deleted from the list
I32 listlen	length of the existing list

dlist is used with the one-dimensional median filter functions med1x and med1y. dlist removes entries from the list that is to contain all the values in the one-dimensional filter window, removing all entries with index value less than ilo. dlist returns the new list length.

* *

void drawln (nv, color) file gmmask.c, MASTER, G only

I32 nv	number of vertices in the user-drawn polygon
I32 color (for ATR1)	
or unsigned long int color (for ATR3)	numeric value for polygon color

drawln draws the user-drawn polygon on the screen, during the process of constructing a 3-level mask, during the process of marking target and background regions on a training scene.

* *

void edj02 (dstb, dsta, src, skr, opr, minlen, maxwid, minavgz, mingrad) file emlines.c, MASTER
void edj02 (dstb, dsta, src, opr, minlen, maxwid, minavgz, mingrad) file eslines.c, SLAVE

I32 dstb	number of the image to receive the edge intensity values
I32 dsta	number of the image to receive the edge angle values
I32 src	number of the source image, in which edges are being sought

I32 skr	number of an image to be used for scratch space
I32 opr	number of the kernel to be used for finding lines
F32 minlen	the minimum acceptable length of a line
F32 maxwid	the maximum acceptable width of a line
F32 minavgz	the minimum acceptable edge intensity
F32 mingrad	the minimum acceptable edge gradient

The intent of edj02 is to find edges, regions in an image such that the intensity is higher on one side of a more-or-less linear boundary than on the other side. This edj02 uses xgrad followed by lin02 and ygrad followed by lin02 to find sharp gradients in the image src. Image dstb is set to the magnitude of the gradient, and image dsta is set to the tangent of the angle between the gradient direction and the x-axis. The opr, minlen, maxwid, and minavgz parameters are those used in lin02. mingrad is a gradient threshold; gradient amplitudes less than mingrad are set to zero and ignored. skr is the number of an image than can be used for scratch space. Either skr must be 0, in which case edj02 defines an image and uses it and destroys it; or, image skr must be previously defined with enough overlap rows to accomodate kernel opr, in which case the contents of image skr will be undefined after this edj02 operation. Image src should have one overlap row. All four images should be different, all of the same size. Image dstb pixels in the excluded edge region defined by the domain of kernel opr are set to zero. edj02 implements the EDJ02 user command.

* *

void eint2 (k, xc, mom, nmom, e0, e1, e2) file fsopt.c, SLAVE, F only

I32 k	a code indicating which tail to integrate over: 1, from xc to positive infinity; 2, from negative infinity to xc
F32 xc	the finite limit of integration
F32 * mom	the values of the moments that specify quantitatively the distribution function
I32 nmom	the number of moments used to describe the distribution
F32 * e0	the address of a variable to receive the value of the integral
F32 * e1	the address of an array to receive the values of the first derivatives of the integral with respect to the distribution moments
F32 * e2	the address of an array to receive the values of the second derivatives of the integral with respect to the moments

eint2 integrates the distribution function that describes the pixel values in a feature image. The integral is done from pixel value xc to positive infinity if k is 1, or from negative infinity to pixel value xc if k is 2. eint2 also calculates the first and second derivatives of the integral with respect to the moments that quantitatively describe the distribution. eint2 is used by cw1b in the training process.

* *

void error9 (void) file esinfo.c, SLAVE

error9 sends a message to MASTER, indicating that this node received a bad command and giving some values present in that bad command. MASTER responds by using abort9 to display the message on the operator's console and stop program execution.

* *

void etime (str) file emmain.c, fmmain.c, & gmmain.c, MASTER

char * str	the message to be printed

etime prints the specified message, and also prints the current host computer clock time and the number of seconds elapsed since the last previous call to etime. etime implements the ECHOTIME user command.

* *

void exsub (void) file emntrp.c, MASTER, E and F only

exsub simply takes values from the programmer's stack and assigns them to the subroutine arguments. exsub is called by the function jump in response to the GOSUB user command, to execute a subroutine in the user command file.

* *

void extract (dst, src, row, col) file emman.c, MASTER & esman.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 row	the source image row from which the top destination image row will be copied
I32 col	the source image column from which the left destination image column will be copied

extract copies all of image dst from the region of image src with top left corner at row row, column col. If part of the requested region lies outside of image src, that part of image dst is left unchanged. This operation does set the overlap rows in dst. extract implements the user command EXTRACT.

* *

void f2c (dst, src, cnst, kod) file esmath.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 cnst	value of the constant to be used
I32 kod	operation code

f2c does single-pixel arithmetic operations that require one image source and perhaps one constant (scalar) source value. Some operations, square root, for example, set each destination image pixel equal to a value calculated from the corresponding source image pixel. Other operations use a constant value, which is the same for all pixels, in addition to the source image pixel value to determine the destination image pixel value, such as the operation of adding a constant to each pixel value, or the operation of clipping pixel values that are higher than the specified constant value. dst and src do not need any overlap rows, they must be the same size, and they can be the same image. The specific operation to be done is specified by the value of kod:

kod	operation
136	copy
125	square root
126	absolute value
127	clip low values
128	clip high values
131	add constant
132	subtract constant
133	multiply by constant
134	divide by constant

135	set image to a constant
151	tangent
152	arctangent
154	natural logarithm
155	exponential

f2c implements the user commands COPY, SQRT, ABS, MAXCON, MINCON, ADDCON, SUBCON, MULCON, DIVCON, SETCON, TAN, ATAN, LOG, and EXP.

* *

void f2cc (dst, src, a, b, kod) file esmath.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 a	value of the first constant to be used
F32 b	value of the second constant to be used
I32 kod	operation code

f2cc does single-pixel arithmetic operations that require one image source and two constant (scalar) source values. The only two operations now installed are (kod=137) set the destination pixel equal to the corresponding source pixel if the values of that pixel is greater than a, or to the value b otherwise; or (kod=138) set the destination pixel equal to the corresponding source pixel if that pixel value is not greater than a, or to b otherwise. dst and src do not need any overlap rows, they must be the same size, and they can be the same image. The specific operation to be done is specified by the value of kode:

kod	operation
137	replace low
138	replace high

f2cc implements the user commands REPLO and REPHI.

* *

void f2udv (kode) file emmath.c, MASTER

I32 kode	function code

f2udv does a mathematical operation using the second user command parameter as the source and the first user command parameter, which must be a user-accessible variable, as the destination. The operation is specified by the value of kode:

kode	operation
20	copy, numeric value or string
25	square root
26	absolute value
51	tangent [radians]
52	arctangent [radians]
54	natural logarithm
55	exponential

f2udv implements the user commands SEQ, SSQRT, SABS, STAN, SATAN, SLOG, and SEXP.

* *

void f3 (dst, sr1, sr2, kod) file esmath.c, SLAVE

I32 dst	number of the destination image
I32 sr1	number of the first source image
I32 sr2	number of the second source image
I32 kod	operation code

f3 does single-pixel arithmetic operations that require two images as sources. An example is the subtraction of two images, in which each pixel of the destination image is set equal to the corresponding pixel of the first source image minus the corresponding pixel of the second source image. The images do not need any overlap rows, they must be the same size, and they can be the same image. The specific operation to be done is specified by the value of kod:

kod	operation
121	add
122	subtract
123	multiply
124	divide
153	arctangent 2
129	maximum
130	minimum

f3 implements the user commands ADD, SUB, MUL, DIV, ATAN2, MAX, and MIN.

* *

void f3udv (kode) file emmath.c, MASTER

I32 kode	function code

f3udv does a mathematical operation using the second and third user command parameters as sources and the first user command parameter, which must be a user-accessible variable, as the destination. The operation is specified by the value of kode:

kode	operation
21	add, or concatenate strings
22	subtract
23	multiply
24	divide
27	maximum
28	minimum
53	arctangent [radians]

f3udv implements the user commands SADD, SSUB, SMUL, SDIV, SMAX, SMIN, and SATAN2.

* *

void ffeats (void) files emntrp.c & gmntrp.c, MASTER

ffeats interprets the user commands. ffeats uses function read2 or function read3 to read the user command and parse it, and ffeats then implements the command, usually by calling a function with the arguments implied by the user command.

* *

I32 findHFAobject (initial_offset, n, name, type, device, filename, fp)

I32 initial_offset	file offset for beginning of search
I32 n	occurrence number
char * name	sought HFA object name
char * type	sought HFA object type
I32 device	number of the file device (SCSIDISK, SCSIHOST, etc.)
char * filename	name of the HFA file
FILE * fp	fill pointer, if the file is on the host (if device=SCSIHOST)

findHFAobject scans an HFA file to find the n-th occurrence of an HFA object with object name and object type that match the function arguments name and type. If either argument name or type is " * ", any HFA object name or type is construed as matching. findHFAobject returns the offset (byte number) of the object in the HFA file, or 0 if the object is not found.

* *

I32 findudv (name) file emmath.c, MASTER

char * name	the name of a user-accessible variable

findudv finds a user-accessible variable name in the list, and returns the index of that variable in the list. findudv returns -1 if the name is not in the list. If the name includes brackets [...], the brackets are not considered part of the name searched for in the list.

* *

I32 fixnam (fnam) file emio.c, MASTER

char * fnam	a character string containing a file name

fixnam deletes any leading spaces from the string pointed to by fnam, deletes the final '\n' if it exists, and adds a final '\0' if it was not already present. fixnam returns the number of characters (excluding the '\0') in the string.

* *

void FreeInn (void) file comm.inc, MASTER & SLAVE

FreeInn changes the flags associated with the input data buffer bufinni to indicate that the main thread is finished with bufinni, so bufinni is available to receive the next incoming data. Data is put into bufinni by function rcv running in a separate thread. rcv cannot receive more data from other nodes until the bufinni flags are cleared by FreeInn.

* *

void FromNodeDN (notused, linkin, internal) file comm.inc, SLAVE, ATR3 only

Process * notused	the process pointer returned by ProcAlloc
Channel * linkin	a channel pointer to the link used for input from the lower node
Channel * internal	a channel pointer to an internal channel

FromNodeDN is a (hopefully) temporary patch, to be used until certain improvements are made in the system software. FromNodeDN is a simple relay, receiving messages from the link that is connected to the next lower node and sending those messages via an internal channel to the rcv function to be put

into the buffer bufinni.

* *

void FromNodeUP (notused, linkin, internal) file comm.inc, SLAVE, ATR3 only

Process *notused	the process pointer returned by ProcAlloc
Channel *linkin	a channel pointer to the link used for input from the lower node
Channel *internal	a channel pointer to an internal channel

FromNodeUP is a (hopefully) temporary patch, to be used until certain improvements are made in the system software. FromNodeUP is a simple relay, receiving messages from the link that is connected to the next higher node and sending those messages via an internal channel to the rcv function to be put into the buffer bufinni.

* *

void fsums (void) file fmopt.c, MASTER, F only

fsums does reformatting and rearranging and normalization of the sums (of feature image pixel values and their products) needed for the training process. If a file name was specified in the .cmd file, fsums also writes the sums to a file. fsums is normally executed after the raw sums are accumulated by asums, and before the optimization performed by cw1.

* *

void getblk (data_offset, img, bpp, row0, col0, nrow, ncol, row1, col1, wide, high, device, filename, fp)
file emio.c, MASTER
void getblk (void) file esio.c, SLAVE

I32 data_offset	data block's offset (byte address) in the source file
I32 img	image number
I32 bpp	bytes-per-pixel code
I32 row0	first file image row to be read
I32 col0	first file image column to be read
I32 nrow	number of rows to be read
I32 ncol	number of columns to be read
I32 row1	image row number or the first row in the block
I32 col1	image column number of the first column in the block
I32 wide	number of columns in the block
I32 high	number of rows in the block
I32 device	number of source file device (SCSIHOST, SCSIDISK, etc.)
char *filename	name of source file
FILE *fp	file pointer, if the file is on the host

The MASTER getblk sends a block of image data, as from an HFA file, to the slave nodes, and the SLAVE getblk receives blocks of data, converts it to F32 format, and writes it into the appropriate image memory space. Each call to MASTER getblk sends one block; however, the block may be sent as several parts in several separate messages, and SLAVE getblk interprets each of these messages as a separate block. The arguments row0 and col0 refer to the location in the overall file image which corresponds to the top left pixel of the total memory image being read, and nrow and ncol are the size of the total memory image being read, not just referring to the current block. The arguments row1 and col1 indicate the position of the top left pixel of the current block in the overall file image. MASTER getblk is called by function inn8HFA.

* *

I32 getfilnam (filnam, inn) file emio.c, MASTER

char *filnam	a buffer to receive the new file name
FILE *inn	the file from which the new file name is being read

getfilnam reads a file name from the file pointed to by inn, and invokes the function fixnam. The file name is assumed to occur in a line by itself in the file. getfilnam returns the number of characters in the "fixed" file name, or -1 if the attempt to read the file name was unsuccessful.

* *

void getheader (filename, hdrtyp, head, bpr, nrows, nbands, bpp) file emio.c, MASTER

char *filename	name of the file to be read
I32 *hdrtyp	pointer to file header type code variable
I32 *nhead	pointer to file header length variable
I32 *bpr	pointer to bytes per row variable
I32 *nrows	pointer to number of rows variable
I32 *nbands	pointer to number of bands variable
I32 *bpp	pointer to bytes per pixel variable

getheader attempts to read the header from image file filename, and attempts to set the values of certain image file descriptors using information in that header. The arguments of getheader (except filename) are pointers to variables that are assumed to contain either tentative values for the parameters, or the value -1 to indicate that there is no tentative value. If echo is on, getheader warns of differences between tentative values and values implied by the file header. If there is a difference, the variable value is not changed unless the value was -1, in which case it is set to the value determined from the file header. The function getheader also attempts to replace values of -1 in the rband structure, if the file is of the row-interleaved type. getheader is used by inn8 and by phead; this implies that the functions inn8 and phead may indirectly change the rband values.

* *

void getheaderHFA (filename, br, nr, nb, bp) file emio.c, MASTER

char *filename	name of an image file
I32 *br	pointer to a location to receive a bytes-per-row value
I32 *nr	pointer to a location to receive a number-of-rows value
I32 *nb	pointer to a location to receive a number-of-bands value
I32 *bp	pointer to a location to receive a bytes-per-pixel value

getheaderHFA is called by function getheader, to read an HFA file and attempt to determine the I32 values from information in the file. getheaderHFA uses the hfaobject.n-th occurrence of the Eimg_Layer type object with a name matching hfaobject.name, to determine the values of *br, *nr, and *bp. If hfaobject.name is "*", every HFA file object name is construed as matching; otherwise, the HFA file object name must exactly match hfaobject.name. The value returned for *nb is the total number of name-matched Eimg_Layer type objects in the HFA file.

* *

void getopr (opr,filename) file emio.c, MASTER
void getopr (void) file esio.c, SLAVE

I32 opr	number of the kernel to be read
char * filename	name of the file to be read

getopr reads kernel number opr from file filename into the memory of each slave and the master. The program assumes that the kernel buffer has already been defined (function defopr, user command DEFKERN) and that the buffer and the file are compatible. opr must be greater than 0. The file is assumed to contain ASCII characters, with the several values separated by spaces, commas, or end-of-line characters. getopr implements the user command READKERNEL.

* *

void getpixo (void) file esinfo.c, SLAVE

getpixo is like the SLAVE getpixp, in that it responds to a request from another node for a pixel value. getpixo is different from getpixp in that getpixo will use a pixel value from an overlap row to satisfy the request, whereas getpixp will use only primary rows.

* *

void getpixp (img, row, col, nparm) file eminfo.c, MASTER
void getpixp (void) file esinfo.c, SLAVE

I32 img	image number
I32 row	row number
I32 col	column number
I32 nparm	number of the user command parameter that is to receive the pixel value, or 0

getpix gets the value of the pixel at row row, column col, in image img, and puts the value in the global variable _AVG (user-accessible variable $AVG). If echo is on, the pixel value is printed on the operator's console. If nparm is greater than 0, the pixel value is also assigned to the user-accessible variable which is given as the nparm-th parameter in the user command line. getpix implements the GETPIX user command.

* *

void getri (void) file esio.c, SLAVE

getri receives one row of an image from another node (usually the master), decides whether that row should be resident in that slave, and, if appropriate, converts and writes to memory as F32 values that row of image. getri assumes that the image is already defined.

* *

void gosub (void) file emntrp.c, MASTER, E and F only

gosub copies values from the user command line parameters into the programmer's stack, and then uses the jump function to go to the correct DEFSUB... line in the user command file so the subroutine can be executed. gosub implements the GOSUB user command.

* *

void gradcon (dstb, dsta, srcb, srca) file esmath.c, SLAVE

I32 dstb	number of the image that will receive the gradient magnitude values

I32 dsta	number of the image that will receive the gradient angle values
I32 srcb	number of the image that holds the old gradient magnitude values
I32 srca	number of the image that holds the old gradient angle values

gradcon concentrates (or sharpens) the peaks in a vector field, in the direction of the vector. This is intended to concentrate a gradient vector field, such as is obtained from function gradt or from the combination of gradx, grady, and xy2rt. Image srcb contains the source vector magnitude, image srca contains the tangent of the angle between the vector and the x axis, and images dstb and dsta will contain the corresponding quantities for the concentrated vector field. gradcon assumes that the single rows and columns of pixels at the edges of the source images are all zero, and it sets these edge pixels to zero in dstb. Each source image should have one overlap row. All the images should be of the same size. The destination images need not be distinct from the source images or from each other. If the two destination images dstb and dsta are the same, the destination image will contain the concentrated vector magnitudes. This operation is not perfect, but is quite good. There is sometimes some concentration in the wrong direction, which gets worse with repeated application of gradcon. gradcon implements the GRADCON user command.

* *

void gradcon2 (x, y, absbc, xc, yc, alfac, pb, pa, n) file esmath.c, SLAVE

F32 *x	address for new x-component of gradient
F32 *y	address for new y-component of gradient
F32 absbc	source gradient magnitude
F32 xc	x-component of source gradient
F32 yc	y-component of source gradient
F32 alfac	square of cosine of gradient direction
F32 *pb	address of source gradient magnitude pixel
F32 *pa	address of source gradient tangent(angle) pixel
I32 n	number of pixels offset to neighbor pixel

gradcon2 is used by gradcon.

* *

void gradt (dstb, dsta, src) file esmath.c, SLAVE

I32 dstb	number of the image to receive the gradient magnitude values
I32 dsta	number of the image to receive the gradient angle values
I32 src	number of the source image

gradt sets image dstb equal to the magnitude of the gradient, and image dsta equal to the tangent of the angle between the gradient direction and the x (horizontal) axis, for the gradient of the intensity in image src. This operation uses one overlap row for src. The destination images need not be different. If the two destination images dstb and dsta are the same, the destination image will contain the gradient magnitudes and the gradient direction values will not be written to any image. The top and bottom rows, and the left and right columns, of dstb are set to zero. Both dstb and dsta should be different from src, and all three images should be of the same size. gradt implements the GRADT user command.

* *

void gradx (dst, src) file esmath.c, SLAVE

I32 dst	number of the image to receive the x-derivative values
I32 src	number of the source image

gradx sets image dst equal to the x-gradient (derivative with respect to x, the coordinate that increases from left to right) of the intensity in image src. In each row, the gradient value at column i is calculated as gradient = (intensity[i+1] - intensity[i-1]) / 2. The first and last columns (left and right edges) of dst are set to 0. dst and src should be different images, of the same size. gradx implements the XGRAD user command.

* *

void grady (dst, src) file esmath.c, SLAVE

I32 dst	number of the image to receive the y-derivative values
I32 src	number of the source image

grady sets image dst equal to the y-gradient (derivative with respect to y, the coordinate that increases from top to bottom of an image) of the intensity in image src. The gradient values are calculated as gradient = (intensity[j+1] - intensity[j-1]) / 2. The top and bottom rows of dst are set to 0. dst and src should be different images, of the same size. This operation requires one overlap row for image src. grady implements the YGRAD user command.

* *

I32 hdist (void) file fsopt.c, SLAVE, F only

hdist simply returns the number of moments (excluding the zeroth moment) used to specify the quantitative details of the distributions of pixel values in the feature images. The qualitative form of the distribution function is assumed. In the current version, this value is 2, with Gaussian distributions assumed. hdist is used by cw1b in the training process.

* *

void hdist1 (x, mom, h0, h1) file fsopt.c, SLAVE, F only

F32 x	pixel value at which the distribution function and its derivatives are to be evaluated
F32 * mom	values of the moments that specify quantitatively the distribution function
F32 * h0	address of a variable to receive the value of the distribution function at pixel value x
F32 * h1	address of the array to receive the values of the derivatives with respect to the moment values of the distribution function at pixel value x

hdist1 calculates the value of the feature image pixel value distribution function for the pixel value x, and its derivatives with respect to each of the moments whose values are given in the array mom. hdist1 is used by cw1b in the training process.

* *

void hdist2 (x, mom, h0, h1, h2) file fsopt.c, SLAVE, F only

F32 x	pixel value at which the distribution function and its derivatives are to be evaluated
F32 * mom	values of the moments that specify quantitatively the distribution function
F32 * h0	address of a variable to receive the value of the distribution function at pixel value x
F32 * h1	address of the array to receive the values of the first derivatives with respect to the moment values of the distribution function at pixel value x

F32 *h2	address of the array to receive the values of the second derivatives with respect to the moment values of the distribution function at pixel value x

hdist2 calculates the value of the feature image pixel value distribution function for the pixel value x, and its first and second derivatives with respect to each of the moments whose values are given in the array mom. hdist2 is used by cw1b in the training process.

* *

void hdistx (x, mom, h0, d0) file fsopt.c, SLAVE, F only

F32 x	pixel value at which the distribution function and its derivative are to be evaluated
F32 *mom	values of the moments that specify quantitatively the distribution function
F32 *h0	address of a variable to receive the value of the distribution function at pixel value x
F32 *d0	address of a variable to receive the value of the derivative with respect to x of the distribution function at pixel value x

hdistx calculates the value of the feature image pixel value distribution function for the pixel value x, and its derivative with respect to x. hdistx is used by cw1b in the training process.

* *

void headbmpw (img) file emio.c, MASTER

I32 img	number of the image in memory for which the header is being constructed

headbmpw constructs a windows bitmap type file header for the specified image, in preparation for writing an image from memory to a file. This function is not fully developed, and the header may be incomplete.

* *

void headdt (img, bpp) file emio.c, MASTER

I32 img	number of the image in memory for which the header is being constructed
I32 bpp	number of bytes per pixel

headdt constructs a Data Translation type file header for the specified image, in preparation for writing an image from memory to a file.

* *

void headp11 (img) file emio.c, MASTER

I32 img	number of the image in memory for which the header is being constructed

headp11 constructs a Perceptron type file header for the specified image, in preparation for writing an image from memory to a file. This function is not fully developed, and the header may be incomplete.

* *

void hexdump (buf, ilo, ihi) file eminfo.c, MASTER

char *buf	pointer to data that is to be printed
I32 ilo	number of the first byte to be printed
I32 ihi	1 + number of the last byte to be printed

hexdump prints on the operator's console the contents of character buffer buf, in hexadecimal and in ASCII.

* *

void hist02 (src, xcl, nbin, vmin, vmax, hst) file eminfo.c, MASTER
void hist02 (src, xcl, nbin, vmin, vmax) file esinfo.c, SLAVE

I32 src	number of the image for which the histogram is being generated
I32 xcl	number of the kernel that defines the excluded edge region
I32 nbin	number of bins in the histogram, not including the high and low out-of-range bins
F32 vmin	low pixel value limit
F32 vmax	high pixel value limit
F32 *hst	an array to hold the histogram

hist02 creates a histogram of intensity values for image src. If echo is on, hist02 calls phst02 to display the histogram on the operator's console. Kernel xcl defines an excluded edge region; xcl may be 0. The histogram will have nbin bins representing the pixel values from vmin to vmax, plus two more bins, for valus less than vmin (in bin 0) and for values greater than or equal to vmax (in bin nbin+1). This operation sets the user-accessible variables $N = number of pixels in the histogram range, $AVG = average value of those pixels, and $SIG = standard deviation of those pixel values. These statistics are calculated from the histogram, not directly from the image. hist02 implements the HIST2 user command.

* *

I32 I32c4 (c)

char *c	pointer to a sequence of 4 characters to be read

I32c4 constructs an I32 value by using the first character at location c as the least significant byte of the I32 value, the next character as the second least significant byte, etc. This I32 value is returned by the function.

* *

I32 I32c2 (c)

char *c	pointer to a sequence of 2 characters to be read

I32c2 constructs an I32 value by using the first character at location c as the least significant byte and the next character as the second least significant byte, and setting the more significant 2 bytes to zero. This I32 value is returned by the function.

* *

(macro) I32 IEQ(x) file efg.h

The function sendshrt uses values supplied as 32-bit integer (I32) arguments, and puts these values into the I32 array bufouti. IEQ is used to disguise a 32-bit floating point (F32) variable as an I32 argument, allowing the bit pattern of the F32 variable to be put into the I32 array bufouti so that the F32 bit pattern can be sent to another node.

* *

void incfex (kf) file fmopt.c, MASTER, F only

I32 * kf	kf[i]=1 if feature image i is to be included, 0 otherwise

incfex increments the indexes that are used to keep track of which combinations of feature images have been used and which combination should be used next, in the sequence of optimizing the coefficients for successive subsets of feature images in the training process. incfex is used by cw1.

* *

void incfil (name, nleft, nrite) file emio.c, MASTER

char * name	a file name
I32 nleft	number of the left-most character to be incremented
I32 nrite	number of the right-most character to be incremented

incfil increments the given file name. Only those characters in the base part (not the path or the extension) are incremented. Only the characters from the nleft'th through the nright'th before the extension (or before the end of the file name) are incremented.

"Incrementing a file name" means that we treat the base file name, exclusive of the path and extension, as if it were a kind of string of digits representing a number, and we increase it by one. For example, with the broadest range specified by the values for nleft or nright, C:FILE15.IMG increments to C:FILE16.IMG; NAME29 to NAME30; FILE9 to FILF0; TESTA to TESTB; F7Z to F8A; XZZ to YAA; TT99 to TU00; etc. The path and extension are never changed. Alphabetic characters always increment to other alphabetic characters, and numerals to other numerals. For nleft=2 and nright=2, TT99 increments to TT09; for nleft=3 and nright=2, TT99 increments to TU09. incfil implements the INCFIL user command.

* *

void indent (level) file eminfo.c, MASTER

I32 level	level of indentation; number of space blocks to indent

indent is used when printing, to accomplish a function like tab stops. indent inserts 3 * level spaces into the current print line.

* *

void inn8 (filename, np, p1, p2, p3, p4, p5, p6, p7) file emio.c, MASTER

char * filename	image file name
I32 nparm	how many of the parameters p1-p7 are meaningfully defined
I32 p1	the number of the memory image to be read, or -1 to indicate reading a multi-band file
I32 p2	image file type code
I32 p3	number of columns in the image file; or, for a multi-band file, number of bytes per composite row in the file
I32 p4	number of rows to read

I32 p5	number of file image rows to skip at the start
I32 p6	number of file image columns to skip in each row
I32 p7	bytes per pixel

inn8 is the primary function for reading an image from a file into memory.
The value of the parameter p2 indicates the type of header on the image file, as follows:

p2	meaning
0	no header
1	Data Translation 512-byte header
2	Perceptron 11-byte header
3	Microsoft OS2 bit map file header
4	Microsoft Windows bit map file, 8 bits per pixel
5	Daedalus 2048-byte header
6	Casi 1024-byte header
7	AMPS synthetic aperture radar
8	Erdas .lan 8-bit and 16-bit, and .gis

If the value -1 is given for p2, inn8 will attempt to determine the header type from the file itself.

p3 is the number of columns in the file image or, for a row-interleaved multi-image file, the number of bytes per composite row of the file. p4 is the number of rows of pixels to be read. If the memory image is already defined, inn8 will not read more rows than can fit into the defined image. When reading multiple images from a row-interleaved file, inn8 will not read more rows than can fit into the smallest previously defined image. If the file image does not have enough rows to fill the memory image, the unfilled memory image rows are left with the same values they had before.

The memory image is taken from that region of the file image with top left corner at row p5 and column p6. Usually, p5 and p6 will be 0, indicating that the top left corner of the memory image is at the top left corner of the file image.

p7 is the number of bytes per pixel. Acceptable values are 1; 2, implying the less significant byte of each pair is first in the file; and -2, implying two bytes per pixel with the more significant byte first. (In all cases, the values are assumed to be integer, not floating point.)

Unlike most operations, inn8 DOES set the values of the pixels in the overlap rows of the memory image.

If the value -1 is given for any of the parameters p2-p7, that parameter is assumed to be unknown by the calling function and inn8 attempts to determine the correct value or a reasonable value from the image file, from the other parameters, from the descriptors of already-defined images, etc. Also, inn8 will attempt to replace -1 values in the rband structures, both directly and by calling the function getheader, if the file is of the row-interleaved type. If the destination image is not already defined, inn8 will define it (with 0 overlap rows) if enough information is available.

If the header is type 1, the program will attempt to read the values of the scale factors A and B from the header in the image file, and set the image table A and B values and the pre-defined user-accessible variables _A and _B (user names $A and $B) values accordingly. (These A and B values are not part of the standard Data Translation header, but this software package -- function headdt -- writes them as an optional comment in the header.) Otherwise, these will be set to 0.0 and 1.0.

If the value -1 instead of an image number is given for p1, inn8 assumes that the file contains several images with interleaved rows. That is, the file contains the first row of the first image, the first row of the second image, ..., the first row of the Nth image, the second row of the first image, the second row of the second image, ..., perhaps with additional bytes interspersed between the separate image rows and perhaps with a file header. This command reads several images from the file in one operation. The information about each image or band is assumed to be already specified in the rband structure, set by function setrband (user command BANDR).

inn8 is used by the user commands READIMAGE, READSCENE, INS, and INM.

* *

void inn8HFA (img, bpp, row0, col0, nrow, ncol, ncols, device, filename, fp)

I32 img	destination image number
I32 bpp	bytes-per-pixel code
I32 row0	file image row at which reading starts
I32 col0	file image column at which reading starts
I32 ncols	number of columns in the file image
I32 device	file device number (SCSIHOST, SCSIDISK, etc.)
char *filename	name of the image file
FILE *fp	file pointer, if the file is on the host (if device=SCSIHOST)

inn8HFA is called by function inn8 when reading an image from an HFA file.

* *

void insert (dst, row, col, src) file emman.c, MASTER & esman.c, SLAVE

I32 dst	number of the destination image
I32 row	destination image row to which the top row of the source image will be copied
I32 col	destination image column to which the left column of the source image will be copied
I32 src	number of the source image

This command copies image src into image dst, with the top left pixel of image src going into dst pixel (row, col). This operation copies only that part of src that will fit into the specified area of dst. src and dst should be different images. This operation does set the overlap rows in dst. insert implements the INSERT user command.

* *

(macro) I32 iroundf(x) file efg.h

F32 x	quantity to be rounded to an integer

iroundf rounds the F32 (32-bit floating point) argument x to the nearest integer value and returns it as an I32 (32-bit integer) value.

* *

void jump (void) file emntrp.c, MASTER, E and F only

The jump function moves to the specified line in the user command (.fc) file. If the jump function is called in response to a user command in which the destination line number is already known, the jump operation is quite simple. If the destination line number is not yet known explicitly but is specified in terms of a not-yet-defined symbol (the argument of a LABEL or SUBDEF user command), then the jump function scans forward (never backward) through the user command file until the symbol is found in the appropriate context. When the symbol is found, it is defined by function linum. If the symbol is the argument of a SUBDEF user command, the subroutine is executed by the function exsub. As the jump function scans through the user command file, it uses the linum function to define every not-yet-defined symbol that appears as a parameter of a LABEL or SUBDEF command. The jump function is used by the user commands JUMP, BRANCH, and GOSUB.

* *

I32 kcurs (jcurs, icurs, minic, maxic, minjc, maxjc) file gmmask.c, MASTER, G only

I32 *jcurs	pointer to cursor row number
I32 *icurs	pointer to cursor column number
I32 minic	minimum allowed cursor column number
I32 maxic	maximum allowed cursor column number
I32 minjc	minimum allowed cursor row number
I32 maxjc	maximum allowed cursor row number

kcurs is used during the construction of a 3-level mask, while marking target and background regions in a training image. kcurs moves the cursor on the screen in response to cursor key hits, and kcurs returns the numerical value corresponding to the key when any non-cursor key is hit.

* *

I32 keyhit (void) file gmmask.c, MASTER, G only, ATR3 only

keyhit is used during the 3-level mask construction process. keyhit removes from a buffer a keystroke that was placed there by function keys, and returns the numerical code for that keystroke. The combination of the two functions keyhit and keys simply gets keystrokes from the keyboard and makes them available to the calling function.

* *

void keys (pp) file gmmask.c, MASTER, G only, ATR3 only

pthread_addr_t pp	an unused but essential argument

keys runs in its own thread and monitors the keyboard and puts keystrokes into a buffer during the 3-level mask construction process. Keystrokes are removed from the buffer by the keyhit function.

* *

I32 ldblk (block) file fmopt.c, MASTER, F only

I32 block	the block sequence number, or 0

ldblk loads each slave node with one row of each feature image (a different row for each slave) and with the appropriate products of feature image rows. The variable block indicates which block of rows is to be loaded. A block of rows comprises one row for each of the slave nodes present in the system. The blocks are numbered 1,2,... ldblk returns the number of the highest node loaded. If the value 0 is given for block, ldblk does not load any image data, but simply returns the number of blocks required to include the full feature image. ldblk is used by sums2.

* *

void lin01 (dstb, dsta, src, opr, minlen, maxwid) file eslines.c, SLAVE

I32 dstb	number of the image to receive the line intensity values
I32 dsta	number of the image to receive the line angle values
I32 src	number of the source image, in which lines are being sought
I32 opr	number of the kernel that defines the local region and its pixel weights
F32 minlen	the minimum acceptable length of a line
F32 maxwid	the maximum acceptable width of a line

lin01 is intended to find lines in image src. lin01 sets each pixel in image dstb equal to the brightness of the line (if any) passing through the corresponding pixel in image src, and sets the corresponding pixel in image dsta equal to the tangent of the angle between the line and the x (horizontal) axis. lin01 will not find lines with negative brightness. dstb, dsta, and src should all be the same size, and dstb and dsta should be different from src. If dstb and dsta are the same image, the image will be set equal to the brightness values and the angle information will not be stored in any image. Kernel opr contains weights for the local region which is analyzed for the presence of a line. minlen is the minimum acceptable line length parameter, and maxwid is the maximum acceptable line width parameter. length and width are measured in pixels, but they are floating point values and fractional parts are meaningful. dstb pixels in the excluded edge region defined by the domain of opr are set to zero. Image src should have enough overlap rows to accommodate kernel opr.

The lin01 algorithm is designed to find bright lines on a zero-intensity background, with no negative pixel values. This algorithm treats the src * opr intensity versus position data as a bivariate probability density function, finds the principal axes, and compares the standard deviations in the principal directions with minlen/sqrt(12) and maxwid/sqrt(12) to determine whether the distribution is "long" and "narrow" enough to be construed as a line. (For a line of uniform intensity, the standard deviations of the distribution are equal to length/sqrt(12) and width/sqrt(12).) In other words, this algorithm looks at the peak in the scr * opr values, regarded as a function of the two position coordinates x and y, and checks to see whether this peak is long and narrow enough to be considered a line. Lines that do not pass through the "central" pixel of the local region defined by opr are rejected.

Note that positive x is to the right, positive y is downward, and positive angles are clockwise from the positive x axis.

lin01 implements the LIN01 user command.

* *

void lin02 (dstb, dsta, src, opr, minlen, maxwid, minavgz) file eslines.c, SLAVE

I32 dstb	number of the image to receive the line intensity values
I32 dsta	number of the image to receive the line angle values
I32 src	number of the source image, in which lines are being sought
I32 opr	number of the kernel that defines the local region and its pixel weights
F32 minlen	the minimum acceptable length of a line
F3 maxwid	the maximum acceptable width of a line
F32 minavgz	the minimum acceptable line intensity

This is a line-finding operation like lin01, except that lin02 rejects lines if the absolute value of the average intensity in the local region is less than minavgz. lin02 can find lines with negative brightness. lin02 implements the LIN02 user command.

* *

I32 lineq (mat, vec, neq, sol) file emmath.c, MASTER & file esmath.c, SLAVE

F64 * mat[]	an array of pointers, each pointing to an array of F64 values that is one row of the coefficient matrix
F64 * vec	a pointer to an array containing the right side vector
I32 neq	the number of equations, the number of unknowns
F64 * sol	an array to receive the solution values

lineq solves a set of neq linear equations in neq unknowns. The equations are specified by
sum over col=0 to neq-1 of {mat[row][col] * sol[col]} = vec[row],
for each row=0,1,...,neq-1.

lineq maintains crude uncertainty estimates, which serve to detect ill-conditioned sets of equations and excessive roundoff error. The uncertainty of sol[i] is returned in vec[i]. lineq changes the values in mat.

* *

I32 linum (void) file emntrp.c, MASTER, E and F only

linum creates the user-defined variables associated with labels (the LABEL user command) and subroutines (the SUBDEF user command). linum is called in direct response to the LABEL user command, as might be expected. Note that linum may also be called when the jump function is executing a JUMP, BRANCH, or GOSUB user command.

* *

int main (argc, argv) files emmain.c, fmmain.c, & gmmain.c, MASTER
int main (void) file esntrp.c, SLAVE, ATR1 & ATR2

int argc	the command line token count
char * *argv	an array of pointers to the command line tokens

The MASTER and SLAVE main functions do the usual chores of initiallizing variables and controlling the overall flow of the program. The SLAVE main function also receives command codes from other nodes and starts the appropriate SLAVE function. (The ATR3 SLAVE "main" function is named "slave".)

* *

void mark (void) file gmmask.c, MASTER, G only

mark allows the user to interactively create or modify a 3-level mask to accompany a scene image to be used in the training process. This mask creation process is described in the user's manual.

* *

void med1x (dst, src, n, opr) file esfilt.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 n	number of pixels in the median filter window
I32 opr	number of the kernel that defines the excluded edge region

med1x sets image dst equal to image src median filtered, with a window of n pixels in the x (horizontal) direction by 1 pixel in the y (vertical) direction. The domain of the kernel opr defines an excluded edge region, in which dst pixel values are left unchanged and src pixels are not used. dst and src should be different images of the same size. n should be an odd integer. med1x uses the functions alist, slist, and dlist. med1x implements the MED1X user command.

* *

void med1y (dst, src, n, opr) file esfilt.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 n	number of pixels in the median filter window
I32 opr	number of the kernel that defines the excluded edge region

med1y sets image dst equal to image src median filtered, with a window of n pixels in the y (vertical) direction by 1 pixel in the x (horizontal) direction. The domain of the kernel opr defines an excluded edge

region, in which dst pixel values are left unchanged and src pixels are not used. dst and src should be different images of the same size. n should be an odd integer. Image src should have at least (n-1)/2 overlap rows. med1y uses the functions alist, slist, and dlist. med1y implements the MED1Y user command.

* *

void median (dst, src, opr, nlo, nhi) file esfilt.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel that defines the median filter window
I32 nlo	number of excluded values at the low-value end of the list of values in the filter window
I32 nhi	number of excluded values at the high-value end of the list of values in the filter window

median does an order sort filter, in which the output (filtered) value is neither the largest nor the smallest of the values in the local region of the source image. This is a non-linear filter that removes local minima with domains of nlo or fewer pixels, and local maxima with domains of nhi or fewer pixels. For each pixel in image dst, median finds the corresponding pixel in image src and aligns thereon the origin pixel of kernel opr. For each non-zero element of the kernel, the corresponding src pixel is put into a list of pixel values. The list is sorted according to value. The dst pixel value is set equal to the corresponding src pixel value, unless this value is smaller than the (nlo+1)th smallest value or larger than the (nhi+1)th largest value in the list, in which case the limiting list value is used for the dst pixel value. Thus, if nlo is 1 and nhi is 2, for example, the dst pixel value cannot be the smallest or the largest or the second largest value in the kernel domain of src pixels. The kernel values are not used except to specify which source pixels are in the local neighborhood, which is that part of the kernel domain for which the kernel values are not zero. If the kernel has an odd number of non-zero elements, and nlo and nhi are both half of that number (integer division by 2), this median operation is the standard median filter in two dimensions. Pixels in the excluded edge region of dst are set to zero. dst and src should be different images, and they may be different sizes. src should have enough overlap rows to accommodate opr.

Note that if nlo is 0 and nhi is one less than the number of non-zero elements in the kernel, this median operation gives the minimum of the pixel values in the local region; this and the similar nhi=0, nlo=(number of non-zero kernel elements)-1 are convenient ways to get a local minimum or maximum. The code implements these two special cases more efficiently than the general case of median.

median implements the MEDIAN user command.

* *

void modmsk (new, old, res, opr, region, q) file esmath.c, SLAVE

I32 new	number of the destination image, which will receive the modified mask
I32 old	number of the source image, which holds the old mask
I32 res	number of the old result image, which is used to make decisions about how to modify the old mask
I32 opr	number of a kernel that defines the excluded edge region
I32 region	code for region to be modified
F32 q	target/background pixel value criterion

modmsk creates a new 3-level mask (used in the training process) in image new, by modifying an old mask from image old, using a result image from image res from a previous program E calculation. If region is 2, then any pixel with value 2 in old is changed to 0 in new if the res pixel value is less than q. If region is 1, then any pixel with value 1 in old is changed to 0 in new if the res pixel value is greater than q. If region is 3, both operations are done. Other pixels are simply copied from old to new. Thus, the

designated target (2) and background (1) regions in the mask are shrunk so that they do not extend beyond the target and background regions indicated in the result image res. This hopefully makes the mask more efficient without damaging its intended target and background designations. Kernel xcl defines an excluded edge region in which new pixels are left unchanged. All three images should be the same size, and new may be the same image as either old or res. modmsk implements the MODMSK user command.

* *

void mom1 (src, xcl, vmin, vmax, kode) file eminfo.c, MASTER
void mom1 (src, xcl, vmin, vmax) file esinfo.c, SLAVE

I32 src	number of the image
I32 xcl	number of the kernel that defines the excluded edge region
F32 vmin	minimum acceptable pixel value
F32 vmax	maximum acceptable pixel value
I32 kode	operation code, 210 or 211

mom1 analyzes image src, excluding the edge pixels specified by kernel xcl, and including only those pixels with values greater than or equal to vmin and less than vmax. If vmin is greater than vmax, all pixel values are included. mom1 treats these pixels as a distribution (a probability density function) in two dimensions and calculates first and second moments and other data. If kode is 210 instead of 211, the included pixel values are all assumed to be 1 when calculating the moments. mom1 sets some user-accessible variables to the values of these moments (x=i=column number, increasing to the right; y=j=row number, increasing downward):

_MINI ($MINI)	minimum column number with in-range pixel
_MAXI($MAXI)	maximum column number with in-range pixel
_MINJ ($MINJ)	minimum row number with in-range pixel
_MAXJ($MAXJ)	maximum row number with in-range pixel
_NUM($N)	number of in-range pixels
_AVGX($AVGX)	average (distribution mean) x
_SIGX($SIGX)	standard deviation of x
_AVGY($AVGY)	average y
_SIGY($SIGY)	standard deviation of y
_COXY($COXY)	covariance of x and y

mom1 implements the user commands PDFXYZ (for kode=211) and PDFXY1 (for kode=210).

* *

void momuv (src, opr, c1, cu, cv, cuu, cvv, tn1) file esconv.c, SLAVE

I32 src	number of the source image
I32 opr	number of the kernel that contains the weights for the pixels in the local region
I32 c1	number of the image to receive the 1 moments
I32 cu	number of the image to receive the u moments
I32 cv	number of the image to receive the v moments
I32 cuu	number of the image to receive the uu moments
I32 cvv	number of the image to receive the vv moments
I32 tn1	number of the image to receive the values of the tangent of the angle between the major principal axis u and the x axis

momuv treats the intensity values in the local region as if they were an un-normalized probability

density function (PDF), and calculates the second and lower moments of the PDF about the origin pixel of the local region (not about the mean). momuv does a coordinate rotation to maximize the second moment in the u direction. If the PDF indicates no preferred direction, the u axis is along the x axis (horizontal, positive to the right). The angle between the u axis and the x axis is always between -90 and +90 degrees. Image src is the source image, and kernel opr defines the local region and the weights for the pixels in the local region. Images c1, cu, cv, cuu, cvv and tn1 are destination images for the weighted average intensity, the moments u, v, uu, and vv (the uv moment is always 0), and the tangent of the angle from the x axis to the u axis. If the value 0 is used for any of the destination image numbers, the corresponding quantity is not written to any image. The x coordinate is positive to the right, and the y coordinate is positive downward, and the origin is at the origin pixel of the local region. src must have enough overlap rows to accommodate opr. The images must all be the same size. src should normally be different from all the destination images. The excluded edge pixels in each dst are set to zero. momuv implements the MOMUV user command.

* *

I32 newimg (void) file emman.c, MASTER

newimg returns the number of a not-yet-defined image. If all images are already defined, newimg returns -1. It also sets the user-accessible variable $N (internal variable _NUM) equal to the new image number. If the global variable np is greater than zero, the variable specified as the first parameter in the user command line will be set to this new image number. newimg implements the NEWIMG user command.

* *

void nlin01 (dst, srcb, srca, opr) file eslines.c, SLAVE

I32 dst	number of the destination image
I32 srcb	number of the image containing the line brightness values
I32 srca	number of the image containing the line tangent(angle) values
I32 opr	number of the kernel that specifies how far to extend each line

nlin01 attempts to count how many lines or line extensions pass through each pixel. This operation is similar to xlin01. But, whereas xlin01 yeilds the sum of the intensities of all the line segments whose extensions would pass through the central pixel, nlin01 is an attempt to count the number of line segments whose extensions would pass through the central pixel, independent of the line segment intensities. nlin01 does not work very well. The weights in the kernel opr should sum to 1.0 along any one ray from the origin pixel. nlin01 implements the NLIN01 user command.

* *

void ntrp00 (dst, src, opr) file esman.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel that contains the local region weights for interpolation

ntrp00 does an interpolation, replacing pixels that have value 0.0 in the source image src with new values in the destination image dst. The new value is the weighted average of all the non-zero-value pixels in the local region of image src, with the weights contained in the kernel opr. src pixels with non-zero values are simply copied to the corresponding pixel in dst. src should include enough overlap rows to accommodate kernel opr. dst and src should be the same size. dst should usually be different from src. ntrp00 implements the NTRP00 user command.

* *

void ntrp01 (dst, src, opr) file esman.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel that contains the local region weights for interpolation

ntrp01 does an interpolation, replacing pixels that have value 0.0 in the source image src with new values in the destination image dst. The new value is determined by a weighted least squares fit of a linear function of position to all the non-zero-value pixels in the local region, with the weights contained in the kernel opr. src pixels with non-zero values are simply copied to the corresponding pixel in dst. src should include enough overlap rows to accommodate kernel opr. dst and src should be the same size. dst should usually be different from src. ntrp01 implements the NTRP01 user command.

* *

I32 nvrt (mat, neq, inv) file emmath.c, MASTER

F64 * mat[]	an array of pointers, each pointing to an array of F64 values that is one row of the matrix
I32 neq	the number of rows and columns in the matrix
F64 * inv[]	an array of pointers, each pointing to an array of F64 values that will receive one row of the inverse matrix

nvrt inverts a square matrix. nvrt maintains crude estimates of the uncertainties due to roundoff error, to detect cases in which the matrix cannot be inverted reasonably. This function changes the values of * mat.

* *

void out5 (img, kode, filename, bpp) file emio.c, MASTER

I32 img	number of the image to be written
I32 kode	file type code
char * filename	name of the destination file
I32 bpp	bytes per pixel in file

out5 writes memory image img to file filename. The file is written with the header type specified by the value of kode, as listed with the inn8 function description. The current version of out5 accepts only two values of kode: 0 (no header) or 1 (Data Translation header). If a Data Translation header is specified, bytes 127 through 152 of the header (the first byte of the header is called number 1) will be the values of scale coefficients A and B, written in ASCII characters. For the common case of one byte per pixel in the file (bpp=1), the memory image pixel values should be between 0.0 and 255.0 (see function scale, user command SCALE). out5 implements the user commands WRITEIMAGE, WRITERESULT, WRITEFEAT, OUTM, and OUTS.

* *

void pacc1 (src, sum1, sumx, sumy, sumxx, sumyy, sumxy) file empeak1.c, MASTER & espeak1.c, SLAVE

I32 src	number of the source image containing the peaks that we are

manipulating.

I32 sum1	number of the image containing peak height information, generated by function plnk1 or function pmrg1.
I32 sumx	number of the image containing the x-coordinates of the accumulators, generated by function plnk1 or function pmrg1.
I32 sumy	number of the image containing the y-coordinates of the accumulators, generated by function plnk1 or function pmrg1.
I32 sumxx	number of the image to receive the xx moments of the peaks
I32 sumyy	number of the image to receive the yy moments of the peaks
I32 sumxy	number of the image to receive the xy moments of the peaks

pacc1 is used with the functions listed in the function peak1 description. pacc1 calculates sums for each src image peak previously defined by peak1 and plnk1, which sums can be used to calculate the moments of each peak as if the peak were an un-normalized distribution function. On input, sum1 should be the same as pv output by plnk1 or pmrg1, sumx should be px, and sumy should be py. This function treats each peak in src as a probability density function (unnormalized), calculates sums for each peak, and assigns the sum values to the destination image pixels corresponding to the accumulator pixel in peak (hence the term "accumulator"). sum1 is set equal to the sum of the src pixel values in the peak. sumx and sumy are set equal to the sums of the distances (in pixels) from the accumulator pixel to the other pixels in the peak, multiplied by the src pixel value. sumxx, sumyy, and sumxy are set equal to the sums of the products of the distances, multiplied by src. The sums appear in the accumulator pixels only; the donor pixels are set to zero (except for src, which is unchanged). These images should all be the same size, and they should all be distinct. No overlap rows are needed for this operation.

* *

I32 pacc1a (jlo, jhi, ncol, p1, px, py, pxx, pyy, pxy) file espeak1.c, SLAVE

I32 jlo	number of lowest row resident in this node
I32 jhi	number of highest row resident in this node
I32 ncol	number of columns in images
F32 * p1	pointer to sum1 image data
F32 * px	pointer to sumx image data
F32 * py	pointer to sumy image data
F32 * pxx	pointer to sumxx image data
F32 * pyy	pointer to sumyy image data
F32 * pxy	pointer to sumxy image data

pacc1a is used by pacc1. pacc1a receives a donor contribution from another slave node and either adds it to the accumulator pixels in this node or relays it on to another node.

* *

I32 passtest (void) file emntrp.c, MASTER

passtest uses parameters 1, 2, and 3 of the user command line. These parameters are assumed to be the first three tokens following an "IF" command, with the second parameter being an arithmetic relational operator and the first and third parameters being numerical quantities. passtest returns 1 if the arithmetic relationship in the IF command is true, 0 otherwise. If the command is true, passtest deletes the first four tokens (the IF command and the following three parameters) from the command line buffer and returns the remainder of the old command line to be interpreted as a new command. passtest implements the IF user command.

* *

void peak1 (src, pv, px, py) file espeak1.c, SLAVE

I32 src	number of the source image, in which we seek peaks
I32 pv	number of the image to receive peak height values
I32 px	number of the image to receive accumulator x-coordinates
I32 py	number of the image to receive accumulator y-coordinates

Several functions are intended to be used in sequence,
with the images output by one being used as input for the next.
The normal order of the functions is as in the following list.
before peak1 :
src = source image, at least one overlap row.
src is never changed by any of the following functions.
src is assumed to have only non-negative pixel values.
any pixel with value 0 in src is ignored by the following
functions.
images pv, px, py, qx, qy, and qv must be defined and must be
compatible with src and must have at least one overlap row,
but their contents do not matter.
an accumulator pixel is one at the top of a peak in src .
a donor pixel is part of a peak, but not the highest point in the peak;
donor pixels in src have values greater than 0 and not greater than
the accumulator pixel value for that peak.
peak1 (src,pv,px,py)
pv = src pixel value for accumulator pixels, 0 for donor and non-peak pixels.
px = 0.5 + accumulator i for accumulators, 0 otherwise.
py = 0.5 + accumulator j for accumulators, 0 otherwise.
plnk1 (src,pv,px,py)
pv = this peak's accumulator value, for peak pixels; 0 for non-peak pixels.
px = this peak's accumulator's i + 0.5 for peak pixels; 0 otherwise.
py = this peak's accumulator's j + 0.5 for peak pixels; 0 otherwise.
that is, every pixel in each peak has the same information
about the peak's accumulator pixel, its value and location.
pmrg1 (src,pv,px,py,qx,qy,qv,step,maxsag) [optional]
The meanings of pv, px, and py are unchanged by this function,
although the values may be changed to indicate the merging of
several peaks into one peak.
qx, qy, and qv are used for scratch by pmrg1 , and their values
after pmrg1 may be anything.
pacc1 (src,pv ,px ,py ,sumxx,sumyy,sumxy)
pacc1 (src,sum1,sumx,sumy,sumxx,sumyy,sumxy)
the pv input image becomes the sum1 image on output.
the px input becomes the sumx output.
the py input becomes the sumy output.
it does not matter what sumxx, sumyy, and sumxy are on input.
on exit from pacc1: for each accumulator pixel:
sum1 = sum of src pixel values v in that accumulator"s peak.
sumx = sum of v * x in that peak, x = donor i - accumulator i.
sumy = sum of v * y in that peak, y = donor j - accumulator j.
sumxx = sum of v * x * x in that peak.
sumyy = sum of v * y * y in that peak.
sumxy = sum of v * x * y in that peak.
for each donor pixel: sum1, sumx, and sumy = 0.
now use either pmomxy OR pmomuv, not one after the other.
pmomxy (sum1,sumx,sumy,sumxx,sumyy,sumxy)

pmomxy (mom0,avgx,avgy,varxx,varyy,varxy)
the sum1 input becomes the mom0 output.
the sumx input becomes the avgx output.
the sumy input becomes the avgy output.
the sumxx input becomes the varxx output.
the sumyy input becomes the varyy output.
the sumxy input becomes the varxy output.
on exit from pmomxy: for each accumulator pixel:
mom0 = sum of src values in peak, same as sum1 input.
avgx = x-distance from accumulator to peak centroid.
avgy = y-distance from accumulator to peak centroid.
varxx = variance of peak about centroid, in x direction.
varyy = variance of peak about centroid, in y direction.
varxy = covariance of peak about centroid.

peak1 finds peaks in image src. If a pixel in image src has value 0, it is not part of a peak, and the corresponding pixel in image pv is set to 0. Each pixel with value greater than 0 in image src is compared with its 8 nearest neighbors. If the src pixel value is less than any of its neighbors, or if it is equal to a neighbor with a lower address (lower row number or same row and lower column number), the src pixel is declared a donor and the corresponding pixel in pv is set to zero. Otherwise, if the src pixel value is greater than its neighbor pixel values, the src pixel is considered an accumulator, and its value (the peak height) is assigned to the corresponding pixel in pv. This function is intended to work with src images that have only non-negative pixel values. Images px and py are set to zero for non-accumulator pixels, and they are set to 0.5+i and 0.5+j for accumulator pixels, where i and j are the accumulator pixel coordinates. The three images px, py, and pv are set in that order, so that, for example, if pv and px are the same image, the image will be left with the peak height values. All the images should be distinct, and all should be the same size. src should have at least 1 overlap row. peak1 implements the PEAK1 user command.

* *

void phead (filename, type, length) file eminfo.c, MASTER

char * filename	name of image file
I32 type	image file type code
I32 length	number of bytes in image file header

phead reads and prints the header from the image file filename. type is an integer code specifying the type of header (see function inn8). If the value -1 is given for type, the program will attempt to determine the header type by itself. If the value 0 is given for type, the program simply prints the first length bytes of the file in hexadecimal and ASCII. The parameter length is not used unless type is 0. phead uses the function getheader, and may therefore replace values of -1 in the rband structures if the file is of the row-interleaved type. phead implements the PHEAD user command.

* *

void pheadHFA (filename) file eminfo.c, MASTER

char * filename	name of the image file

pheadHFA is called by function phead, to read header information from an HFA file.

* *

void phst02 (hst, nbin) file eminfo.c, MASTER

F32 * hst	the array holding the histogram
I32 nbin	the number of bins in the histogram

phst02 draws (crudely) on the operator's console the histogram data created by hist02.

* *

void plnk1 (src, pv, px, py) file empeak1.c, MASTER & espeak1.c, SLAVE

I32 src	number of the source image, in which occur the peaks that we are characterizing
I32 pv	number of the image containing the values of the accumulator pixels, set by function peak1
I32 px	number of the image containing the x-coordinates of the accumulator pixels, set by function peak1
I32 py	number of the image containing the y-coordinates of the accumulator pixels, set by function peak1

plnk1 associates each donor pixel in image src, with a nearby accumulator pixel. The peak1 function should be used to set the values in pv, px, and py, before this plnk1 function is used; see the peak1 description.

For a peak in image src, we will speak of an accumulator pixel and (usually) several donor pixels. The accumulator for a peak is the pixel with the greatest intensity in src in that peak; a donor is any pixel in that peak other than the accumulator (and with a src value greater than 0). The result of this plnk1 function is that each donor pixel in a peak is associated with the accumulator for that peak, by being assigned the accumulator pixel's values in images pv, px, and py. That is, for each donor pixel, the values of px - 1/2 and py - 1/2 are the coordinates of that donor's accumulator pixel, and the value of pv is the value of that donor's accumulator pixel in src. Each donor pixel is associated with the same peak as its nearest (of 8) neighbor pixel with the largest value in src. (This neighbor pixel value is larger than the donor's own value, or the "donor" is actually an accumulator.) If there is a tie for highest value nearest neighbor, the lower address neighbor is favored. This plnk1 function is intended for src images with non-negative pixel values. Pixels with value zero in image src are not assigned to any peak. plnk1 implements the PLNK1 user command.

* *

void pmom (kode, src, mom0, momu, momv, momuu, momvv, tang, maxsag) file empeak1.c, MASTER

I32 kode	if kode = 224, calculate the moments along the x and y axes; if kode = 225, calculate the moments along the principal axes of each peak separately, with u and v representing the major and minor axes.
I32 src	number of the source image
I32 mom0	number of the image to receive the values of the sum of the pixels in each peak
I32 momu	number of the image to receive the values of the x or u distance from the peak maximum to the peak centroid
I32 momv	number of the image to receive the values of the y or v distance from the peak maximum to the peak centroid
I32 momuu	number of the image to receive the values of the xx or uu moment about the centroid of the peak
I32 momvv	number of the image to receive the values of the yy or vv moment about the centroid of the peak
I32 tang	number of the image to receive the values of the tangent of the angle between the x-axis and the u-axis, for kode = 225, or the values of the xy moment about the peak centroid, for kode = 224.

F32 maxsag	the maximum allowable depth of a valley between two peaks that are to be merged into one single peak

pmom finds intensity peaks in image src, treats each peak as an un-normalized probability density function, and calculates the moments of that distribution. pmom uses the sequence of functions listed in the function peak1 description. The values for the various moments for each peak are put into the pixels corresponding to the highest value in image src for that peak; the other pixels in the several destination images are set to zero. pmom implements the PMOMXY and PMOMUV user commands.

* *

void pmomuv (mom0, avgu, avgv, varuu, varvv, tang) file espeak1.c, SLAVE

I32 mom0	number of the image containing sums of peak pixel values
I32 avgu	number of the image containing sums of peak pixel values, weighted with x distance from the accumulator pixel; to receive u distance from accumulator to centroid
I32 avgv	number of the image containing sums of peak pixel values, weighted with y distance from the accumulator pixel; to receive v distance from accumulator to centroid
I32 varuu	number of the image containing sums of peak pixel values, weighted with the square of the x distance from the accumulator pixel; to receive u variance about the centroid
I32 varvv	number of the image containing sums of peak pixel values, weighted with the square of the y distance from the accumulator pixel; to receive v variance about the centroid
I32 tang	number of the image containing sums of peak pixel values, weighted with the product of the x and y distances from the accumulator pixel; to receive the tangent of the angle between the x-axis and the u-axis

pmomuv is like pmomxy, except that pmomuv converts the results for each peak to a u-v coordinate system for that peak. The u coordinate is chosen to be in the direction of the largest second moment for the peak or, if there is no preferred direction, the u axis is along the x axis. The u-v covariance (the cross second moment about the mean) is always zero; instead of this value, the last destination image is set equal to the tangent of the angle between the u axis and the x axis. pmomuv implements the PMOMUV1 user command.

* *

void pmomxy (mom0, avgx, avgy, varxx, varyy, varxy) file espeak1.c, SLAVE

I32 mom0	number of the image containing sums of peak pixel values
I32 avgx	number of the image containing sums of peak pixel values, weighted with x distance from the accumulator pixel; to receive x distance from peak to centroid
I32 avgy	number of the image containing sums of peak pixel values, weighted with y distance from the accumulator pixel; to receive y distance from peak to centroid
I32 varxx	number of the image containing sums of peak pixel values, weighted with the square of the x distance from the accumulator pixel; to receive x variance about the centroid
I32 varyy	number of the image containing sums of peak pixel values, weighted with the square of the y distance from the accumulator pixel; to receive y variance about the centroid
I32 varxy	number of the image containing sums of peak pixel values, weighted with

the product of the x and y distances from the accumulator pixel; to receive the xy covariance about the centroid

pmomxy is part of the series of functions listed in the function peak1 description. pmomxy converts the sums from pacc1 into moments. On input, mom0 should be the same as sum1 output by pacc1, avgx should be sumx, avgy should be sumy, varxx should be sumxx, varyy should be sumyy, and varxy should be sumxy. This command treats each peak in peak1's src as a probability density function (unnormalized), calculates the moments for each peak, and assigns the moment values to the destination image pixels corresponding to the accumulator pixel in peak1's src image. mom0 is left unchanged, equal to sum1, the sum of the src pixel values in the peak. avgx and avgy are set equal to the distance in pixels from the accumulator pixel to the centroid of the peak. varxx, varyy, and varxy are set equal to the second moments (the variances and the covariance) about the centroid. These images should all be the same size, and they should all be distinct. No overlap rows are needed for this operation. pmomxy implements the PMOMXY1 user command.

* *

void pmrg1 (src, pv, px, py, qx, qy, qv, maxsag) file empeak1.c, MASTER
void pmrg1 (src, pv, px, py, qx, qy, qv, step, maxsag) file espeak1.c, SLAVE

I32 src	number of the source image containing the peaks that we are manipulating.
I32 pv	number of the image containing peak height information (accumulator pixel value in src), generated by function plnk1.
I32 px	number of the image containing the x-coordinates of the accumulator pixels, generated by function plnk1.
I32 py	number of the image containing the y-coordinates of the accumulator pixels, generated by function plnk1.
I32 qx	number of a scratch image
I32 qy	number of a scratch image
I32 qv	number of a scratch image
I32 step	code specifying which step of the overall process to do next
F32 maxsag	the maximum allowable depth of a valley between two peaks that are to be merged into one single peak

pmrg1 is used with several other functions, as listed in the function peak1 description. pmrg1 merges peaks that are connected by a path in src such that the lowest pixel value along that path is not lower than the higher peak value minus the value maxsag and the lowest pixel is also greater than 0. All the pixels in two or more merged peaks are assigned to the same accumulator. The images src, pv, px, and py should be set by a prior call to the function plnk1, and these four images will have the same meanings (although perhaps different values) after pmrg1 as after plnk1. The images qx, qy, and qv are used for scratch by pmrg1. All seven images should have at least one overlap row, all should be the same size, and all should be distinct. pmrg1 implements the PMRG1 user command.

* *

void pmrg1b (jlo, jhi, jmin, jmax, ncol, ip, jp, vq, xq, yq, ppv, ppx, ppy) file espeak1.c, SLAVE

I32 jlo	lowest primary row in this node
I32 jhi	highest primary row in this node
I32 jmin	lowest overlap row in this node
I32 jmax	highest overlap row in this node
I32 ncol	number of columns in image
I32 ip	old accumulator column number
I32 jp	old accumulator row number

F32 vq	new accumulator peak magnitude
F32 xq	new accumulator x coordinate
F32 yq	new accumulator y coordinate
F32 * ppv	address of peak magnitude image
F32 * ppx	address of accumulator x coordinate image
F32 * ppy	address of accumulator y coordinate image

pmrg1b is used by SLAVE pmrg1. pmrg1b does the actual remapping of donor pixels to new accumulator pixels, and in some cases sends messages to other nodes instructing them to do remapping.

* *

I32 pmrg1c (jlo, jhi, jmin, jmax, ncol, ppv, ppx, ppy, pqv, repeat) file espeak1.c, SLAVE

I32 jlo	lowest primary row in this node
I32 jhi	highest primary row in this node
I32 jmin	lowest overlap row in this node
I32 jmax	highest overlap row in this node
I32 ncol	number of columns in image
F32 * ppv	address of accumulator values (peak heights)
F32 * ppx	address of accumulator x coordinates
F32 * ppy	address of accumulator y coordinates
F32 * pqv	address of temporary peak heights
I32 * repeat	address of repeat flag

pmrg1c is used by SLAVE pmrg1. pmrg1c accepts remapping commands from other slave nodes while coordinating the required remapping of donor pixels in this node.

* *

void pmrg1d (src, pv, px, py, qx, qy, qv, kode, maxsag) file empeak1.c, MASTER

I32 src	number of the source image containing the peaks that we are manipulating.
I32 pv	number of the image containing peak height information, generated by function plnk1.
I32 px	number of the image containing the x-coordinates of the accumulator pixels, generated by function plnk1.
I32 py	number of the image containing the y-coordinates of the accumulator pixels, generated by function plnk1.
I32 qx	number of a scratch image
I32 qy	number of a scratch image
I32 qv	number of a scratch image
I32 kode	code specifying which detailed step should be done
F32 maxsag	the maximum allowable depth of a valley between two peaks that are to be merged into one single peak

pmrg1d is used by MASTER pmrg1. pmrg1d sends a message to the slaves, instructing them to do the detail step specified by the value of kode.

* *

I32 pmrg1e (kode) file empeak1.c, MASTER

I32 kode	expected value of code returned by slaves

pmrg1e is used by MASTER pmrg1. pmrg1e waits for a status report from the slave nodes, and checks to see that the code sent by the slave agrees with kode. pmrg1e returns the value of a status code.

* *

I32 pop (void) file emntrp.c, MASTER

pop removes an I32 value from the programmer's stack and returns that value. pop is used by functions exsub and ret.

* *

F32 * powell (func, fparm, iparm, linmin, linatol, linrtol, linitmax, atol, rtol, itmax) file emmath.c, MASTER

F32 * func(...)	represents the multi-variate function to be minimized
F32 * fparm	an array of F32 parameters for func, linmin, and powell
I32 * iparm	an array of I32 parameters for func, linmin, and powell
F32 * linmin(...)	minimizes a multivariate function (like func) along one line
F32 linatol	linmin argument, usually a fraction-of-range convergence criterion
F32 linrtol	linmin argument, usually a fraction-of-value convergence criterion
I32 linitmax	linmin argument, usually the maximum allowed number of iterations
F32 atol	absolute convergence criterion for powell
F32 rtol	relative convergence criterion for powell
I32 itmax	maximum allowed number of iterations for powell

This function powell is adapted from William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, "Numerical Recipes in C, Second Edition", Cambridge University Press, 1992.

powell minimizes the multivariate function represented by func (see reg3c, for example), which must be of a particular form:

* The second and third arguments of func are fparm and iparm, the same as the second and third arguments of powell. The first argument of func is an F32 value, call it x.
* the function is to be minimized with respect to n variables, fparm[1], fparm[2], ... fparm[n], where n = iparm[0].
* func evaluates the function at the point fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n], and sets fparm[0] equal to the function value.
* func returns a pointer to an F32 array. Array element 0 is set to the function value; element 1 is set to the x value used in the function evaluation (which may be different from the x value supplied as an argument, depending on how func is programmed); element 2 is a status code: 0 for normal return, 1 for unspecified error, 2 for argument out of acceptable range. Other array elements are available for special uses as the programmer wishes.

On entry to powell, fparm[1], fparm[2], ... fparm[n] must be a valid point in the function domain. An earlier version of powell, commented VERSION 2 in the source code text, required further that fparm[0] must be the function value at that point, but this requirement for fparm[0] does not apply to the current VERSION 3.

The array elements fparm[n+1], ..., fparm[n+n] are used by powell. The contents of these array elements will be destroyed by powell, and will not be passed through powell to func or to linmin. Any other elements of fparm and iparm, not previously mentioned in this discussion, can be used as the programmer wishes, perhaps for values for func or for linmin.

linmin is a function (see brent3a, for example) which minimizes a multivariate function, represented by func, along a line through the multiple variable space. linmin must have this form:

* The first three parameters of linmin are the same as the first three parameters of powell, and the next three parameters of linmin are the powell arguments linatol, linrtol, and linitmax. linmin does not necessarily need to use all of its six parameters.
* linmin returns a pointer to an F32 array. Array element 0 is the minimized function value;

element 1 is the value of x at the minimum; element 2 is a status code: 0 for normal completion, 1 if no acceptable minimum was found, 2 if the initial values given to linmin represented a converged condition (which is not normally an error); element 3 is the number of iterations done. powell actually uses only elements 0 and 2 of this array. If linmin returns a status code of 0 or 2, fparm[1], ... , fparm[n] is the position of a minimum along the line, and fparm[0] is the function value at that minimum; otherwise, fparm[0] ... fparm[n] are unchanged from their values on entry to linmin.

* Generally, linmin (or its subsidiary functions) must minimize func(x,fparm,iparm) with respect to x, which usually suggests that linmin must select and try different values of x. linmin may or may not use or set fparm and iparm values, apart from the required setting of fparm[0] ... fparm[n] as mentioned above. Normally, linmin should not change the values of fparm[1+n] ... fparm[n+n].

On return from powell, fparm[1], fparm[2], ... fparm[n] is the location of the function minimum, and fparm[0] is the function value at the minimum. powell returns a pointer to an F32 array. Array element 0 is the minimized function value; element 1 is not used; element 2 is a status code: 0 for normal completion, 1 if no acceptable minimum was found; and element 3 is the number of iterations done.

* *

void print (msg, numchar, format) file comm.inc, SLAVE

char * msg	pointer to the data to be sent
I32 numchar	number of bytes to be sent
char format	print format code

print is used by a slave node that does not have direct access to the host devices, to print a message on the operator's console. The message is not made available to the MASTER main thread. Each byte is printed as an ASCII character, in octal notation, or in hexadecimal notation, depending on whether the format argument is 'c', 'o', or 'x'.

* *

void printpar (i) file emmath.c, MASTER

I32 I	the number of the command line parameter to be printed

printpar prints the value of command line parameter number i (the command itself is parameter 0), on the operator's console.

* *

void prints (void) file emmath.c, MASTER

If the global variable np (the number of parameters in the current command line) is 0 (that is, no parameters were specified with the command), this function prints the values of all the user-accessible variables, on the operator's console. If np is greater than 0, this function prints the values of the command line parameters, not including the command itself.

* *

void printvar (i) file emmath.c, MASTER

I32 i	the index of a user-accessible variable

printvar prints the value of user-accessible variable number i, on the operator's console.

* *

I32 printvar2 (typ) file emmath.c, MASTER

I32 typ	a user-accessible variable type code

printvar2 is used with printvar and printpar, to print information about the type of the variable or the parameter on the operator's console.

* *

void push (i) file emntrp.c, MASTER

I32 i	the value to be pushed onto the programmer's stack

push puts an I32 value on the programmer's stack. push is used by function gosub.

* *

void putbi (img, dstlo, dsthi, bpp) file esio.c, SLAVE

I32 img	number of the image to be sent
I32 dstlo	destination node low limit
I32 dsthi	destination node high limit
I32 bpp	bytes per pixel

putbi sends the resident part of image img to all nodes with numbers between dstlo and dsthi inclusive. The image is converted from F32 values to integer values corresponding to the bytes-per-pixel code (see function inn8) bpp before being sent. putbi is used by such operations as writing an image to a file and displaying an image on the high-resolution RGB screen. putbi uses function putri to send individual rows of the image.

* *

void putbires (img, dstlo, dsthi, res, q) file esio.c, SLAVE

I32 img	number of the image to be sent
I32 dstlo	destination node low limit
I32 dsthi	destination node high limit
I32 res	number of the image that holds the pixel color criterion values
F32 q	red/green pixel color criterion

putbires sends the resident part of image img to nodes dstlo to dsthi inclusive. The image is sent with a special 8-bit-per-pixel code: the low 7 bits indicate the pixel's value (intensity), and the highest bit is 0 or 1 according to whether the corresponding pixel in image res has value less than or greater than q. putbires uses function putrires to send individual rows of the image. putbires is used with the DISPRES user command.

* *

void putri (img, row, dstlo, dsthi, bpp) file esio.c, SLAVE

I32 img	number of the image to be sent
I32 row	number of the row to be sent
I32 dstlo	destination node low limit

I32 dsthi	destination node high limit
I32 bpp	bytes per pixel

putri is used by function putbi, to send one row of the image. That is, putbi calls putri once for each resident row of image img.

* *

void putrires (img, row, dstlo, dsthi, res, q) file esio.c, SLAVE

I32 img	number of the image to be sent
I32 row	number of the row to be sent
I32 dstlo	destination node low limit
I32 dsthi	destination node high limit
I32 res	number of the image that holds the pixel color criterion values
F32 q	red/green pixel color criterion

putrires is used by function putbires, to send one row of the image. That is, putbires calls putrires once for each resident row of image img.

* *

void qt9d (img, row0, col0, q, caption) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char * caption	image caption

qt9d displays image img on the high-resolution RGB monitor. The top left corner of the image will appear at row0,col0 in the screen display. Pixel values in image img should be between 0.0 and 255.0 (see function scale, user command SCALE). Red will be used for pixels with value less than q, green for pixels with value greater than q. The character string caption would be printed below the image on the screen, except for defects in the software supplied with the QT9D interface hardware. If echo is on, the image will remain on the screen, and the program will stop execution, until the operator hits a key on the keyboard. qt9d implements the DISP user command in ATR2.

* *

void qt9dinit (void) file emio.c, MASTER

qt9dinit initiallizes the high-resolution RGB display software. qt9dinit is invoked one time, when the program is started. qt9d is used only in ATR2.

* *

void qt9dres (img, row0, col0, q, caption, res) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char * caption	image caption

I32 res	number of the image with the red/green criterion pixel values

qt9dres is like qt9d, except that the pixels are colored red or green according to whether the corresponding pixel in the image res (normally, but not necessarily, a result image) has a value less than q. This allows an input image to be displayed with color coding based on a result image. qt9dres implements the DISPRES user command in ATR2.

* *

void quadfit (opr) file emconv.c, MASTER

I32 opr	number of a kernel

quadfit checks the values in the kernel opr, to see that the equations will be solvable when kernel opr is used as the weights for fitting a quadratic function to the intensities in a local region of an image. quadfit is used with user commands QUADXY and QUADUV.

* *

void quaduv (src, opr, c1, cu, cv, cuu, cvv, tn1) file esconv.c, SLAVE

I32 src	number of the source image
I32 opr	number of the kernel that holds the weights for the pixels in the local area
I32 c1	number of the image to receive a1 values
I32 cu	number of the image to receive au values
I32 cv	number of the image to receive av values
I32 cuu	number of the image to receive auu values
I32 cvv	number of the image to receive avv values
I32 tn1	number of the image to receive tangent(angle) values

Like quadxy, quaduv fits the local region with a quadratic function. quaduv then does a coordinate rotation from the x-y to the u-v coordinates, with the u-axis chosen in the direction that maximizes the second derivative with respect to u. Images c1, cu, cv, cuu, and cvv are destination images for the coefficients of the 1, u, v, uu, and vv terms in the fitted polynomial. (The uv term is always zero.) Image tn1 is the destination for the tangent of the angle between the x axis and the u axis. quaduv implements the QUADUV user command.

* *

void quadxy (src, opr, c1, cx, cy, cxx, cyy, cxy) file esconv.c, SLAVE

I32 src	number of the source image
I32 opr	number of the kernel that holds the weights for the pixels in the local area
I32 c1	number of the image to receive a1 values
I32 cx	number of the image to receive ax values
I32 cy	number of the image to receive ay values
I32 cxx	number of the image to receive axx values
I32 cyy	number of the image to receive ayy values
I32 cxy	number of the image to receive axy values

quadxy does a weighted least squares fit of a quadratic function of position,

intensity = a1 + ax * x + ay * y + axx * x * x + ayy * y * y + axy * x * y,

to the pixels in a local region of the source image, and writes the 6 polynomial coefficients to the 6

destination images. If the image number supplied for any destination image is 0, the corresponding quantity is not written to any image. The x coordinate is positive to the right, and the y coordinate is positive downward, and the origin is at the origin pixel of the local region. Kernel opr defines the local region and contains the weights. src must have enough overlap rows to accommodate opr. The images must all be the same size. src should normally be different from all the destination images. The excluded edge pixels in each destination image are set to zero. quadxy implements the QUADXY user command.

* *

void rcv (pp) file comm.inc, MASTER, ATR3

pthread_addr_t pp an unused but essential argument

void rcv (pp) file comm.inc, MASTER except ATR3 & SLAVE

Process * pp process pointer from ProcAlloc

rcv receives data from another node, either directly from a hardware link or indirectly via functions FromNodeUP or FromNodeDN. rcv puts the data into buffer bufinni and notifies either snd to relay the data to the next node, or the main thread to use the data, or both. rcv runs in a thread by itself.

* *

I32 read2 (sSL) file emntrp.c, MASTER, E & F only

char * sSL pointer to a character string that will hold a copy of the command line.

read2 reads and parses a user command line, and puts a copy of the command line into the string sSL. read2 returns the number of parameters in the command line, not including the command itself. read2 is used by ffeats.

* *

I32 read3 (sSL) file gmntrp.c, MASTER, G only

char * sSL pointer to a character string that will hold a copy of the command line.

read3 reads and parses a user command line, and puts a copy of the command line into the string sSL. read3 returns the number of parameters in the command line, not including the command itself. read3 is used by ffeats.

* *

I32 readimage (filnam) file gmntrp.c, MASTER, G only

char * filename the name of the file from which the image is to be read

readimage does some preliminary operations before reading an image from a file. readimage returns 0 for normal completion, or -1 if an error occurs. readimage is used by the READIMAGE, INS, and INM user commands.

* *

void ready (void) file emmain.c, fmmain.c, & gmmain.c, MASTER

ready queries all the nodes except MASTER, and waits for a reply from each. If echo is on, ready displays a message for each node's reply. Receiving a reply from a node implies that the node is finished with any previous activities. ready implements the READY user command.

* *

void redraw (nrow, ncol, nv) file gmmask.c, MASTER, G only

I32 nrow	number of rows in the image
I32 ncol	number of columns in the image
I32 nv	number of verrices in the user-drawn polygon

redraw is used to refresh the display during the construction of a 3-level mask, during the process of marking a training scene with target and background regions. redraw is invoked when the operator hits the ";" key.

* *

void reg2 (hhh, ggg, fff, fs1, gs1, gs2, gs3, gs4, bux, buy, bu1, bvx, bvy, bv1, dux, duy, du1, dvx, dvy, dv1)

Function reg2 does not exist as a separate function in the current version of the software. reg2 can be easily created by modifying reg3, following the comments in the text of the reg3 function. There are two significant differences: in reg2, brent2a and reg2c should be used instead of brent3a and reg3c, and reg2 must set fparm[0] before calling powell.

* *

F32 * reg2b (b, sig, hhh, ggg, fff, fs1, gs1, gs2, gs3, gs4) file emman.c, MASTER

F32 b[6]	mapping coefficients, the 6 function variables
F32 sig[6]	smoothing widths
I32 hhh	number of the destination image, which will be the smoothed source image mapped to match template image; compatible with image fff
I32 ggg	number of the source image
I32 fff	number of the template image, the image to be matched
I32 fs1	number of a scratch image, compatible with image fff
I32 gs1	number of a scratch image, compatible with image ggg
I32 gs2	number of a scratch image, compatible with image ggg
I32 gs3	number of a scratch image, compatible with image ggg
I32 gs4	number of a scratch image, compatible with image ggg

reg2b is not used in the current version of the software; it is replaced by reg3b, which calculates the number of degrees of freedom in addition to the correlation.

reg2b is used with function reg2. reg2b calculates the cross-correlation of image fff with a smoothed, mapped image ggg.

The goal is to calculate the correlation at many points in the 6-dimensional space spanned by the mapping coefficients b[i], to smooth the correlation in each of the 6 b[i] directions with a smoothing function that has standard deviation sig[i], and to return the value of the smoothed correlation function at the point specified by the b argument. reg2b does an indirect calculation to produce the same result. reg2 uses the sig values to construct image ddd which contains local smoothing width values, uses image ddd to smooth image ggg into image sss, maps image sss into image hhh using the b values, and calculates and returns the correlation of the mapped-to region of image hhh with the corresponding region of image fff. The returned correlation values should always be between -1.0 and 1.0, with 1.0 representing perfect correlation, 0.0 representing no correlation, and -1.0 representing a perfect negative correlation (eg, one image is the negative of the other). reg2b returns the value -2.5 (not a possible correlation value) if the

two images -- fff and mapped ggg -- have no overlap, or -3.5 if all the pixels in the overlap region of either image have the same value. These two error conditions arise because (1) the correlation obviously cannot be calculated if the images do not overlap, and (2) the correlation involves a division by the standard deviation of the pixel values, which division cannot be done if the pixels all have the same value so the standard deviation is 0. Specifically, reg2b returns a pointer to an F32 array, in which element 0 is the correlation value (or the error codes -2.5 or -3.5), element 1 is 0.0, and element 2 is the number of mapped pixels in the destination image hhh.

The optimization problem which is addressed by functions powell and brent1a in reg2 is to maximize this function reg2b, the correlation, by adjusting the values of the 6 b[i] coefficients.

* *

F32 * reg2c (x, fparm, iparm) file emman.c, MASTER

F32 x	the single variable
F32 * fparm	an array of F32 parameters
I32 * iparm	an array of I32 parameters

reg2c is not used in the current version of the software; it is replaced by reg3c.

reg2c is used with reg2. reg2c is an example of the function func used as an argument of function powell; reg2c represents the 6-variable function reg2b in the 1-variable form required for use with powell. reg2c uses the fparm and iparm values to generate appropriate argument values for function reg2b, and reg2c puts the function value returned by reg2b and other return values into the required format. reg2c also converts from the correlation value calculated by reg2b to the 1-correlation value required for the minimization procedures.

x is the single variable, an argument that measures position along a line in the 6-dimensional parameter space of function reg2b, the line along which a single-variable minimization is done. Specifically, reg2c evaluates reg2b at the point fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n] (with n=iparm[0]=6).

fparm contains values that supply the information necessary to construct a (constrained) 6-dimensional position vector from the single parameter x. These include fparm[1]-fparm[6], which specify the x=0 point, and fparm[7]-fparm[12], which specify the direction of the line through the 6-dimensional space. fparm and iparm also contain values for the other arguments required by reg2b, and fparm contains some scratch space used by reg2c to hold argument values for reg2b. Note that the arrays fparm and iparm are defined in the function reg2 which calls reg2c.

reg2c evaluates the function reg2b at the point fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n], and sets fparm[0] equal to the required function value 1-reg2b (1-correlation). reg2c returns a pointer to an F32 array, with array element 0 set to the function value (1-correlation, 1-reg2b); element 1 set to the x value used in the function evaluation (which in this case is the same as the x value supplied as an argument); element 2 set to 0 for normal completion, -1 (unspecified error) if reg2b returned -3.5, or -2 (variable out of range error) if reg2b returned -2.5; and element 3 set to 0.

* *

void reg3 (hhh, ggg, fff, fs1, gs1, gs2, gs3, gs4, bux, buy, bu1, bvx, bvy, bv1, dux, duy, du1, dvx, dvy, dv1)
file emman.c, MASTER

I32 hhh	number of the destination image
I32 ggg	number of the source image
I32 fff	number of the image to be matched
I32 fs1	number of a scratch image
I32 gs1	number of a scratch image
I32 gs2	number of a scratch image
I32 gs3	number of a scratch image
I32 gs4	number of a scratch image

F32 b... first guess values for mapping coefficients
F32 d... allowed deviations for mapping coefficients

reg3 finds and executes a mapping of image ggg into image hhh, such that the correlation of the mapped part of image hhh and image fff is maximized.

The row and column coordinates of corresponding pixels in the images hhh and ggg are related by

gggcol = bux * hhhcol + buy * hhhrow + bu1
gggrow = bvx * hhhcol + bvy * hhhrow + bv1

The b values given as arguments for this function are first guesses at the optimal values. The b values are adjusted by reg3 to achieve optimal mapping; these 6 b values are the 6 variables of the multi-variate function that is minimized by powell.

The d arguments are the allowed deviations from the initial guess values of the b coefficients; initial b - d <= final b <= initial b + d. If a negative value is given for any d, this function reg3 assigns a reasonable value to that d.

Images ggg and fff are not changed by this operation. Images fff and hhh must be the same size and must be compatible for arithmetic operations (they must be distributed the same way among the nodes); image ggg may be different in size. fs1, gs1, gs2, gs3, and gs4 are images that can be used as scratch space by this operation. If 0 is given as the number for any of these scratch images, reg3 will temporarily define suitable scratch images (if sufficient memory is available). If a valid image number is given for any of these scratch images, then that image must be already defined. fs1 must be compatible with fff, and gs1, gs2, gs3, and gs4 must be compatible with ggg. All of the images must be distinct; none of the images needs any overlap rows.

reg3 attempts to maximize the correlation between the mapped regions of image hhh and image fff. If no reliable correlation is found, the 6 "optimized" b values will be the same as the values input as function arguments. The final optimized values of bux, buy, bu1, bvx, bvy, and bv1 are returned in the user-accessible variables $AVGX, $AVGY, $AVG, $SIGX, $SIGY, and $SIG. The value of [1-correlation] is returned in $COXY. Possible correlation values range from -1 to 1, so $COXY=[1-correlation] values range from 0 (perfect correlation) to 2.

This function reg3 uses function powell, with brent3a as linmin and reg3c as func, to maximize the correlation (minimize 1-correlation) by adjusting the 6 b coefficients.

The following table indicates whether reg3 and its subsidiary functions use or set (change) the values of fparm[] and iparm[]. The extreme right column indicates the values returned by each function.

reg3										
define	fparm [0]	fparm [1-6]	fparm [7-12]	fparm [13-18]	fparm [19-24]	fparm [25-30]	fparm [31-36]	iparm [0]	iparm [1-8]	
alias		b	u	minb	maxb		sig			
		set		set	set		set	set	set	
powell										{1-corr,0,code,powell iter}
alias		p	u					n		
			set					use		
brent3a										{1-corr,x,code,brent3 iter}
alias		b0	u	minb	maxb			n		
		use	use	use	use			use		
	set	set (fparm[0-6] set only if brent3 returns a valid minimum)								
brent3										{1-corr,x,code,brent3 iter}
	(no direct reference)									
reg3c										{1-corr,x,code,DOF}
alias		b0	u			b		n		
	set	use	use			set		use		
reg3b										{corr,DOF,overlap}
argument						b	sig		* * *	
						use	use		use	

reg3 implements the REGISTER user command.

* *

F32 * reg3b (b, sig, hhh, ggg, fff, fs1, gs1, gs2, gs3, gss4) file emman.c, MASTER

F32 b[6]	mapping coefficients
F32 sig[6]	smoothing widths
I32 hhh	number of the destination image, which will be the smoothed source image mapped to match the template image; compatible with image fff
I32 ggg	number of the source image
I32 fff	number of the template image, the image to be matched by the mapped source image
I32 fs1	number of a scratch image, compatible with image fff
I32 gs1	number of a scratch image, compatible with image ggg
I32 gs2	number of a scratch image, compatible with image ggg
I32 gs3	number of a scratch image, compatible with image ggg
I32 gs4	number of a scratch image, compatible with image ggg

reg3b is used with function reg3. reg3b calculates the cross-correlation of a smoothed image fff with a smoothed, mapped image ggg. reg3b also calculates an estimate of the number of degrees of freedom associated with the cross-correlation calculation.

The goal is to calculate the correlation at many points in the 6-dimensional space spanned by the mapping coefficients b[i], to smooth the correlation in each of the 6 b[i] directions with a smoothing function that has standard deviation sig[i], and to return the value of the smoothed correlation function at the point specified by the b argument. reg3b does an indirect calculation to produce the same result. reg3b calculates the correlation of the appropriately smoothed copy of image fff with the smoothed and mapped image ggg. The returned correlation values should always be between -1.0 and 1.0, with 1.0 representing perfect correlation, 0.0 representing no correlation, and -1.0 representing a perfect negative correlation (eg, one image is the negative of the other). reg3b returns the value -2.5 (which is not a possible correlation value) if the two images -- fff and mapped ggg -- have no overlap, or -3.5 if all the pixels in the overlap region of either image have the same value. These two error conditions arise because (1) the correlation obviously cannot be calculated if the images do not overlap, and (2) the correlation involves a division by the standard deviation of the pixel values, which division cannot be done if the pixels all have the same value so the standard deviation is 0. Specifically, reg3b returns a pointer to an F32 array, in which element 0 is the correlation value (or the error codes -2.5 or -3.5), element 1 is the estimated number of degrees of freedom, and element 2 is the number of mapped pixels in the destination image hhh.

The optimization problem which is addressed by functions powell and brent3a in reg3 is to maximize this function reg3b, the correlation, by adjusting the values of the 6 b[i] coefficients.

* *

F32 * reg3c (x, fparm, iparm) file emman.c, MASTER

F32 x	the single variable
F32 * fparm	an array of F32 parameters
I32 * iparm	an array of I32 parameters

reg3c is used with reg3. reg3c is an example of the function func used as an argument of function powell; reg3c represents the 6-variable function reg3b in the 1-variable form required for use with powell. reg3c uses the fparm and iparm values to generate appropriate argument values for function reg3b, and

reg3c puts the function value returned by reg3b and other return values into the required format. reg3c also converts from the correlation value calculated by reg3b to the 1-correlation value required for the minimization procedures.

x is the single variable, an argument that measures position along a line in the 6-dimensional parameter space of function reg3b, the line along which a single-variable minimization is done. Specifically, reg3c evaluates reg3b at the point fparm[1]+x * fparm[1+n], fparm[2]+x * fparm[2+n], ... fparm[n]+x * fparm[n+n] (with n=iparm[0]=6).

fparm contains values that supply the information necessary to construct a (constrained) 6-dimensional position vector from the single parameter x. These include fparm[1]-fparm[6], which specify the x=0 point, and fparm[7]-fparm[12], which specify the direction of the line through the 6-dimensional space. fparm and iparm also contain values for the other arguments required by reg3b, and fparm contains some scratch space used by reg3c to hold argument values for reg3b. Note that the arrays fparm and iparm are defined in the function reg3 which indirectly and directly calls reg3c.

reg3c sets fparm[1+4 * n] ... fparm[6+4 * n] (fparm[25] ... fparm[30]) equal to the value of the 6-component vector position at which reg3b is to be evaluated, fparm[1]+x * fparm[1+n] ... fparm[6]+x * fparm[6+n] (fparm[1]+x * fparm[7] ... fparm[6]+x * fparm[12]). reg3c then evaluates reg3b at this point, and sets fparm[0] equal to the required 1-correlation value. reg3c returns a pointer to an F32 array, with array element 0 set to the function value (1-correlation); element 1 set to the x value used in the function evaluation (which in this case is the same as the x value supplied as an argument); element 2 set to 0 for normal completion, 1 (unspecified error) if reg3b returned -3.5, or 2 (variable out of range error) if reg3b returned -2.5; and element 3 set to number of degrees of freedom in the reg3b correlation calculation.

* *

void remap1 (dst, src, bux, buy, bu1, bvx, bvy, bv1) file emman.c, MASTER & esman.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 bux	mapping coefficient
F32 buy	mapping coefficient
F32 bu1	mapping coefficient
F32 bvx	mapping coefficient
F32 bvy	mapping coefficient
F32 bv1	mapping coefficient

remap1 re-maps image src to produce image dst. The row and column values in the new image dst are related to the row and column values in the old image src by an affine transformation:

(src col) = bux * (dst col) + buy * (dst row) + bu1,
(src row) = bvx * (dst col) + bvy * (dst row) + bv1.

This type of transformation allows for stretching or compression of the image in the x direction and in the y direction; rotation; top-bottom and left-right reflection; and shear distortion.

If the src pixel corresponding to any dst pixel does not exist, the dst pixel is left with the value it had before this operation. This remap1 function uses remap1a, and implements the REMAP user command.

* *

void remap1a (updn, more, dst, src, bux, buy, bu1, bvx, bvy, bv1) file esman.c, SLAVE

I32 updn	code for addressing higher, lower, or this node
I32 * more	address of condition flag
I32 dst	number of the destination image
I32 src	number of the source image

F32 bux (see remap1)
F32 buy
F32 bu1
F32 bvx
F32 bvy
F32 bv1

remap1a is used by remap1. remap1a determines which pixel values from this node will be needed by other nodes, and sends the appropriate pixel values; remap1 coordinates the overall process, making sure that inter-node messages do not collide and lock the system.

* *

I32 resam1 (dst, src, opr, ncold, nrowd, novld) file emman.c, MASTER
void resam1 (dst, src, opr, mcold, mrowd, movld) file esman.c, SLAVE

I32 dst	number of the destination image, which will be a re-sampled version of the source image
I32 src	number of the source image, which will be re-sampled to create the destination image
I32 opr	number of the kernel that defines an excluded edge region of the source image
I32 ncold/mcold	number of columns in the destination image
I32 nrowd/mrowd	number of rows in the destination image
I32 novld/movld	number of overlap rows in the destination image

resam1 resamples image src and put the result into image dst. The kernel xcl defines an excluded edge region of image src; xcl may be 0. There is no excluded edge in dst. If image dst is not yet defined, it will be defined with ncol columns, nrow rows, and novl overlap rows, and its scale factors A and B will be set to 0.0 and 1.0. Note that images defined by this resam1 function may be distributed among the slaves differently than images of the same size defined by other processes. If image dst was previously defined, it is assumed to have been defined with the correct size and with the correct parts allocated to each slave, and the values supplied for ncol, nrow, and novl are ignored. The values of the overlap row pixels in dst are NOT set by this resam1 function. The resampling is done by the simplest nearest-pixel method, with no interpolation or smoothing. If image src has fewer rows than image dst and image src has overlap rows, the overlap rows will probably be used in the resam1 procedure, so their values (if not already current) should be updated with the share function (OVERLAP user command) before this resam1 function is used. src overlap rows may also be used if the src and dst images are not distributed exactly correctly among the slave nodes, as might happen if one of the images is defined by a process other than the resam1 function. resam1 implements the RESAMPLE user command.

* *

void resam2 (void) file esman.c, SLAVE

resam2 is an experimental program, not recommended for general use and not guaranteed to work. It accomplishes (maybe) the same thing as resam1, but using different and probably much slower methods.

* *

void ret (void) file emntrp.c, MASTER

ret returns from a subroutine in the user command (.fc) file. ret recovers from the programmer's stack the line number of the GOSUB user command that invoked the subroutine, and uses the jump

function to go to that line in the command file. ret implements the RETURN user command.

* *

void rijcon (dst, src, opr) file emconv.c, MASTER & esconv.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel

The intent of rijcon is to concentrate ridges, but this operation is not accomplished perfectly. A ridge is an intensity maximum that extends much further in one direction then in the perpendicular direction. We would like to make the ridge sharper, narrower in the narrow direction, but not change it significantly in the long direction. A weighted least squares fit is used to fit a quadratic polynomial to the local region in the source image src, with kernel opr containing the weights and specifying the local region. The kernel should be invariant to 90-degree rotations about the origin pixel. If the fitted polynomial describes a ridge, the ridge is smoothed along its length and concentrated in the direction perpendicular to its length, in the destination image dst. dst should be different from src, and they should be the same size. src should include enough overlap rows to accommodate opr. The excluded edge pixels in dst are set to zero. rijcon implements the RIJCON user command.

* *

void rmul (pimg, ncol, dst, src1, src2) file fsopt.c, SLAVE, F only

F32 * pimg	pointer to the image data memory area
I32 ncol	number of columns in the images
I32 dst	number of the destination image
I32 src1	number of the first source image
I32 src2	number of the second source image

rmul multiplies one image row by another image row. rmul is used by sums2 to form rows of product pixels during the accumulation of sums for the training process. In this context, only one row of each image is stored in the memory in any one node, and all the image data is in the same memory block which is pointed to by pimg. Thus, the address of the one row of image src1, for example, is pimg+src1 * ncol.

* *

void rmulc (pimg, ncol, row, c) file fsopt.c, SLAVE, F only

F32 * pimg	pointer to the image data memory area
I32 ncol	number of columns in the images
I32 row	number of the destination image
F32 c	value of the constant

rmulc multiplies one image row by a constant. rmulc is used by sums2 to apply the appropriate scale factor (the B coefficient) to the feature image data during the accumulation of sums for the training process. In this context, only one row of each image is stored in the memory in any one node, and all the image data is in the same memory block which is pointed to by pimg. Thus, the number of the row in the memory block is the same as the image number.

* *

void RollCallM (cpi, cpo, nd, ni, nv) file comm.inc, MASTER

Channel *cpi[]	an array of channel pointers, to be used for input
Channel *cpo[]	an array of channel pointers, to be used for output
I32 *nd	pointer to number-of-slaves variable
I32 *ni	pointer to SCSI-interface-node-number variable
I32 *nv	pointer to video-interface-node-number variable

RollCallM is used to establish initial communication with the slave nodes, to determine which links will be used, and to count the daisy nodes. cpi and cpo are arrays of channel pointers, with each array having more elements than the number of inter-node links. RollCallM sets values for some of these channel pointers. nd should point to the global variable ndaisy, the number of daisy nodes in the system; RollCallM sets this variable. RollCallM also sets the variables pointed to by ni and nv, which are meaningful only in ATR1 and are set to zero in ATR2 and ATR3.

* *

void RollCallS (cpi, cpo) file comm.inc, SLAVE

Channel *cpi[]	an array of channel pointers, to be used for input
Channel *cpo[]	an array of channel pointers, to be used for output

RollCallS is used to establish initial communication with the other nodes, to determine which links will be used, and to count the number of daisy nodes. cpi and cpo are arrays of channel pointers, with each array having more elements than the number of inter-node links. RollCallS sets values for some of these channel pointers.

* *

void rsums (void) file fmopt.c, MASTER, F only

rsums reads from a file the sums previously written by fsums, which sums are used in the training process. This allows trying variations of training processes that use the same feature images, without re-forming all the feature images and re-calculating the sums. That is, one call to rsums replaces a sequence of calls to zsums, asums, and fsums.

* *

void sbend (dst, src, tcol0, taddmax, tmax) file emman.c, MASTER
void sbend (void) file esman.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 tcol0	number of the pixel column that is on the camera centerline
F32 taddmax	number of pixel columns between the camera centerline and the edge of the field of view
F32 tmax	angle [radians] from camera centerline to edge of field of view

This sbend function does the s-bend geometric correction. For image src, the pixels within each row (or the columns in the image) are assumed to represent samples that are uniformly spaced in terms of the angle to the side of the camera center line. Image dst will have pixels that represent samples that are uniformly spaced in terms of the distance from the camera center line on a flat scene plane or a flat film plane. src and dst must be the same size, and they may be the same image. tcol0 is the number of the image column corresponding to the camera center line, where angle = 0. tcol0 is the same for src and dst, and the src tcol0 pixels are mapped directly to dst tcol0. tmax is usually the maximum angle included in the camera's field of view, corresponding to pixels at the edge (largest column number) of the images, and column tcol0 + taddmax is the pixel column number corresponding to this maximum angle. However,

you are not required to use tmax and taddmax values with this simple physical interpretation. Formally, the src pixels in columns tcol0 + taddmax and tcol0 - taddmax are mapped directly to the same columns in dst. Pixels in columns other than these three special columns are shifted laterally (to different columns) in the SBEND operation. The columns numbered tcol0, tcol0 + taddmax, and tcol0 - taddmax do not have to actually exist in the images src and dst. sbend uses sbenda, and it implements the SBEND user command.

* *

void sbenda (cold, xtcol0, f1, f2, ncol, nrow, ps1, pd1) file esman.c, SLAVE

I32 cold	destination column number
F32 xtcol0	number of column at which angle=0, distance=0
F32 f1	columns per angle
F32 f2	normalized distance per column
I32 ncol	number of columns in image
I32 nrow	number of active rows in this node
F32 * ps1	address of the source image
F32 * pd1	address of the destination image

sbenda is used by sbend. sbend divides the image into as many as four groups of columns, sets parameters for each group, and calls sbenda to do the actual mapping for each group.

* *

void scale (dst, src, opr) file emio.c, MASTER
void scale (dst, src, opr, a, b) file esio.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel that defines the excluded edge region
F32 a	value of A in (dst image) = A + B * (src image)
F32 b	value of B in (dst image) = A + B * (src image)

scale sets image dst equal to image src multiplied by a scale factor chosen to make the largest value in image dst equal to 254. The edge pixels, defined by kernel xcl, are excluded; their values in src are not considered in determining the scale factor, and their values in dst are left unchanged. The image dst coefficient A is set to 0.0 and B is set to the scale factor used in this scaling operation, without regard to the A and B coefficient values for image src, and the user-accessible variables $A and $B are set equal to A and B. dst and src should be of the same size, and they may be the same image. Usually, scale should be used immediately before displaying or writing an image with an 8-bit integer format. scale implements the SCALE user command.

* *

void SCSIDISKclose (void) file emio.c, MASTER

SCSIDISKclose closes the currently open file on the ATR1 SCSI disk. This function is also present in the ATR2 and ATR3 programs, but it should never be used except in ATR1.

* *

void SCSIDISKopen (filename, nskip) file emio.c, MASTER

char * filename	the name of the ATR1 SCSI disk file

I32 nskip	the number of bytes to be skipped at the beginning of the file

SCSIDISKopen opens a file on the ATR1 SCSI disk, and immediately skips nskip bytes in that file. This function is also present in the ATR2 and ATR3 programs, but it should never be used except in ATR1.

* *

void SCSIDISKread (buf, nbytes) file emio.c, MASTER

char * buf	the buffer to receive the data from the file
I32 nbytes	the number of bytes to be read

SCSIDISKread reads nbytes of data from the currently open file on the ATR1 SCSI disk. This function is also present in the ATR2 and ATR3 programs, but it should never be used except in ATR1.

* *

void SCSIDISKskip (bytes) file emio.c, MASTER

I32 bytes	the number of bytes to skip

SCSIDISKskip skips the specified number of bytes in the currently open file on the ATR1 SCSI disk. This function is also present in the ATR2 and ATR3 programs, but it should never be used except in ATR1.

* *

void seglab (dst, src) files emmath.c, MASTER & esmath.c, SLAVE

I32 dst	the number of the destination image
I32 src	the number of the source image

seglab does a certain segment labeling operation. Image src is assumed to contain segments, patches of pixels with values greater than 0 separated by regions of background pixels with values less than or equal to 0. For each segment pixel in image src, the corresponding pixel in image dst is set to a value that is unique to that segment. The value is 1+r+c/nc, where nc is the number of columns in the images, r is the number of the lowest row occupied by the segment, and c is the number of the lowest column occupied by the segment within row r. The image dst pixels that do not correspond to an image src segment pixel are left unchanged.

Normally, one would start by setting all image dst pixels to zero. Then seglab could be used once or repeatedly with the same dst image but with different src images, to accumulate information about segments in different source images. If the image dst pixel corresponding to an image src segment pixel is already non-zero, this is an indication of overlapping segments, and the results are not specified. This seglab function is not reliable for images with more than about 8 million pixels, because of the limited precision of the 32-bit floating point variables that hold the 1+r+c/nc values. The two images must be distinct, they must be the same size, and each must have at least one overlap row. seglab implements the SEGLAB user command.

* *

void sendlong (nodelo, nodehi, kode, k1, k2, k3) file comm.inc, MASTER & SLAVE

I32 nodelo	the number of the lowest node to receive this message
I32 nodehi	the number of the highest node to receive this message

I32 kode	user data, usually a command code
I32 k1	user data, usually a command parameter
I32 k2	user data, usually a command parameter
I32 k3	user data, usually a command parameter

sendlong is used to send a long (more than 4 I32 values) message to one or more other nodes. The message will be sent to all nodes with numbers from nodelo through nodehi inclusive. The four I32 values given for kode, k1, k2, and k3 will be sent as part of the message. Before sendlong is called, bufouti[9] should be set to the number of additional bytes to be sent, and those bytes should be already written into bufouti[10]... (or, equivalently, bufoutc[0]... or bufoutf[10]...). WaitOutFree should be called before writing into bufouti. sendlong also sends the values of nodelo and nodehi (sendlong puts these values into bufouti[3] and bufouti[4]) and the number of the sending node (bufouti[2]) with the message; however, the values of nodelo and nodehi in the message are changed as the message moves from one node to another.

* *

void sendshrt (nodelo, nodehi, kode, k1, k2, k3) file comm.inc, MASTER & SLAVE

I32 nodelo	the number of the lowest node to receive this message
I32 nodehi	the number of the highest node to receive this message
I32 kode	user data, usually a command code
I32 k1	user data, usually a command parameter
I32 k2	user data, usually a command parameter
I32 k3	user data, usually a command parameter

sendshrt is used to send a short (4 I32 values) message to one or more other nodes. The message will be sent to all nodes with numbers from nodelo through nodehi inclusive. The four I32 values given for kode, k1, k2, and k3 will be sent as part of the message. sendshrt also sends the values of nodelo and nodehi (sendshrt puts these values into bufouti[3] and bufouti[4]) and the number of the sending node (bufouti[2]) with the message; however, the values of nodelo and nodehi in the message are changed as the message moves from one node to another. sendshrt uses bufouti, but sendshrt includes a call to WaitOutFree, so WaitOutFree does not need to be called separately before sendshrt.

* *

void setab (img, a, b) file emman.c, MASTER

I32 img	image number
F32 a	coefficient A
F32 b	coefficient B in (this image) = A + B * (other image)

If the global variable np is at least 3, setab sets the A and B coefficients for image img to floating point values a and b. ([This image] = A + B * [original image].) These coefficients were initially set to A=0.0, B=1.0 when the image was defined. This operation does NOT rescale the image; it merely sets the values of the stored coefficients. If np is greater than 0, setab sets the user-accessible variables $A and $B (internal variables _A and _B) equal to the A and B values for image img. If echo is on, setab also prints the A and B values for image img on the screen. setab implements the SET_AB user command.

* *

void setext (filename, ext) file emio.c, MASTER

char * filename	pointer to file name character string
char * ext	pointer to extension string

setext is intended to set the extension part of a DOS-type file name. setext assumes that the string filename is a file name, and that the string ext is a 3-character string. The file name is modified to make the new file name extension (the part after the '.') equal to the string ext. Any previous extension is discarded. The string ext should not include the '.'. If the '.' was not already in the string filename, it is appended at the end of the original filename string before ext is added.

* *

void setfex (kf) file fmopt.c, MASTER, F only

I32 * kf	kf[i]=1 if feature image i is to be included, 0 otherwise

setfex initiallizes a set of indexes that are used to keep track of which combinations of feature images have been used and which combination should be used next, in the sequence of optimizing the coefficients for successive subsets of feature images in the training process. setfex is used by cw1.

* *

void setHFA (nparm, n, name, type) file emio.c, MASTER

I32 nparm	number of parameters given in the user command line
I32 n	specifies which occurrence of the HFA object to use
char * name	name of the HFA object
char * type	type of the HFA object

If nparm is at least 1, setHFA sets the value of hfaobject.n to n; if nparm is at least 2, setHFA sets hfaobject.name to name; if nparm is at least 3, setHFA sets hfaobject.type to type; and, if the echo is on, setHFA prints the current values of all 3 members of the structure hfaobject. This function is used in preparation for reading an image from an HFA file, for example, in which context the image being read is the hfaobject.n-th occurrence of an image from an HFA Eimg_Layer type object with name equal to hfaobject.name. setHFA implements the SETHFA user command.

* *

void setpix (img, row, col, value) file esman.c, SLAVE

I32 img	image number
I32 row	row number
I32 col	column number
F32 value	value to be assigned to pixel

setpix sets the value of the pixel in image img, row row, column col, to the floating point value value. This operation sets pixels in both the primary and the overlap rows. Note that images are stored in memory as type F32, 4-byte floating point data. setpix implements the SETPIX user command.

* *

void setrband (nparm, band, img, byt0, ncol, bpp) file emio.c, MASTER

I32 nparm	number of parameters in the command line
I32 band	band number
I32 img	number of memory image to receive this band
I32 byt0	number of bytes from beginning of file image composite row to beginning of memory image row for this band
I32 ncol	number of pixels in one memory image row

I32 bpp number of bytes per pixel

setrband sets and displays values in the structure rband[band], which is used by function inn8 for reading multi-image, row interleaved files. The bands are numbered 1, 2, ... rband[0] is used as temporary storage space for certain operations, and rband[0].img is the number of bands in the file.

If nparm is 0, setrband lists the rband entries for which the values are not the default values.

If nparm is 1, setrband sets the number of bands (rband[0].img) equal to the value of the parameter band.

If nparm is greater than 1, setrband sets parameters for band (or sub-image) number band (band = 1, 2, ... nbands), as follows:

If nparm > 1, set the memory image number for the specified band equal to img. If the value 0 is given as the image number, the specified band is not used for any memory image.

If nparm > 2, use the byt0 value as the number of bytes skip before image img in each composite row of the image file.

If nparm > 3, use the ncol value as the number of pixels in each row for the specified band.

If nparm > 4, use the bpp value as the bytes-per-pixel value for the specified band. (The current version of function inn8 assumes that this is the same for all images in any one row-interleaved file.)

If the value -1 is given for byt0 or ncol or bpp, for any band, the function inn8 (user command READIMAGE -1) will attempt to determine the correct values for these parameters from the file header. setrband implements the BANDR user command.

* *

void share (img) file emman.c, MASTER & esman.c, SLAVE

I32 img number of the image

share causes the slave nodes to exchange pixel values for image img, so that the values of all overlap rows are updated from the current values in primary rows. That is, each slave obtains (from other slaves) the correct, current values of the pixels in its own overlap rows. This should be done before operations like convolution with a kernel or median filtering in the y direction, which use the overlap rows. The overlap rows are automatically assigned the correct values when an image is read from a file, but the overlap rows are NOT assigned the correct values by most other operations. share uses shrinn and shrout. share implements the OVERLAP user command.

* *

I32 shrinn (void) file esman.c, SLAVE

shrinn is used by share and other functions. shrinn is called when the input data buffer bufinni is known to contain image data. shrinn copies the data from the buffer, in F32 format, to the appropriate pixels in the image array, and releases the buffer.

* *

void shrout (img, row, node) file esman.c, SLAVE

I32 img	image number
I32 row	row number
I32 node	destination node number

shrout is used by share and other functions. shrout sends one row of an image to another node, in F32 format.

* *

int slave (void) file esntrp.c, SLAVE, ATR3 only

slave is the "main" program for the slave nodes in ATR3. Its primary function is to interpret commands or messages from other nodes.

* *

F32 slist (pv, pi, listlen) file esfilt.c, SLAVE

F32 * pv	the values in the list
I32 * pi	the indices associated with the values in the list
I32 listlen	the number of values in the list

slist is used with the one-dimensional median filter functions med1x and med1y. slist sorts the list and returns the value in the middle of the list, the median value.

* *

void smthx (dst, src, sig, n, opr) file emfilt.c, MASTER
void smthx (dst, src, a, opr) file esfilt.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 sig	width of the overall smoothing function
I32 n	number of smoothing passes
I32 opr	number of the kernel that defines an excluded edge region
F32 a	a parameter related to the smoothing width

smthx sets image dst equal to image src smoothed in the x (horizontal) direction. One call to SLAVE smthx does 2 passes of an exponential smoothing filter, one pass with increasing x and one pass with decreasing x. This double pass is counted as one pass by MASTER smthx. The value of the parameter a is given by function asig. The combination of the functions MASTER smthx and asig calculate the value of the SLAVE smthx parameter a so that the standard deviation of the total effective smoothing function represented by n double passes (n calls to SLAVE smthx) is equal to sig. The domain of the kernel opr specifies an excluded edge region, in which the dst pixel values are not changed. (The kernel opr is not used for anything except specifying this excluded edge region.) dst should be the same size as src, and they may be the same image. smthx implements the SMTHX user command.

* *

void smthxs (dst, src, sig, opr) file emfilt.c, MASTER & esfilt.c, SLAVE

I32 dst	number of the destination (smoothed) image
I32 src	number of the source (to be smoothed) image
I32 sig	number of the image that contains the local smoothing width values
I32 opr	number of the kernel that defines the excluded edge region

smthxs does a kind of exponential smoothing of an image, in the x (horizontal) direction. In smthxs, the width of the exponential smoothing function is not constant, but can vary from one pixel to the next. The value of this width -- the standard deviation of the smoothing function -- for each pixel is the value of the corresponding pixel in the image sig. The smoothing is actually done in two passes, one with increasing x and one with decreasing x, with the total effect of the two passes representing a smoothing

function with the standard deviation value specified in sig. Smoothing in this manner, with an ever-changing width of the smoothing function, is not a standard operation, and the results of this smoothing are not simply predictable in an intuitive sense. One implication is that this smoothing operation is not strictly linear, so such quantities as the area under a peak may not be preserved. The domain of the kernel opr specifies an excluded edge region, in which the dst pixel values are not changed. (The kernel opr is not used for anything except specifying this excluded edge region.) dst, src, and sig should all be the same size. dst and src may be the same image, but sig should be different from both dst and src. smthxs implements the SMTHXS user command.

* *

void smthy (dst, src, sig, n, opr) file emfilt.c, MASTER
void smthy (dst, src, a, opr) file esfilt.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
F32 sig	width of the overall smoothing function
I32 n	number of smoothing passes
I32 opr	number of the kernel that defines an excluded edge region
F32 a	a parameter related to the smoothing width

smthy is like smthx except that smthy smooths in the y (vertical) direction instead of the x (horizontal) direction. This y direction smoothing requires exchanging data between adjacent slaves, so the SLAVE smthy function is more complicated than its x direction counterpart. smthy implements the SMTHY user command.

* *

void smthys (dst, src, sig, opr) file emfilt.c, MASTER & esfilt.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 sig	number of the image that contains the local smoothing width values
I32 opr	number of the kernel that defines the excluded edge region

smthys is like smthxs except that smthys smooths in the y (vertical) direction instead of the x (horizontal) direction. This y direction smoothing requires exchanging data between adjacent slaves, so the SLAVE smthys function is more complicated than its x direction counterpart. smthys implements the SMTHYS user command.

* *

void snd (pp) file comm.inc, MASTER, ATR3 only

pthread_addr_t pp	an unused but essential argument

void snd (pp) file comm.inc, MASTER except ATR3 & SLAVE

Process * pp	the process pointer from ProcAlloc

snd transmits data via a hardware link to another node. The data must be in either of the two buffers bufinni or bufouti. snd is invoked by the functions sendlong and sendshrt from the main thread (using bufouti), and by rcv from its thread (using bufinni). snd uses snd1, except in ATR3 MASTER. snd and snd1 run in a separate thread.

* *

void snd1 (buf) file comm.inc, MASTER except ATR3 & SLAVE

I32 * buf	a pointer to the data to be sent

snd1 is used by snd to send the specified buffer's data. The buffer pointer buf should be either bufinni or bufouti, described in the listing of global structures.

* *

void ssq (dst, src, opr) file esconv.c, SLAVE

I32 dst	number of the destination image
I32 src	number of the source image
I32 opr	number of the kernel that has the weights for the local region pixels

ssq sets image dst equal to the local sum of squares of image src, with weighting coefficients from kernel opr. That is, ssq sets each image dst pixel value equal to the sum over the domain of kernel opr of {src * src * opr}. The dst pixels in the excluded edge region, defined by the domain of the kernel opr, are set to zero. src should have enough overlap rows to accommodate kernel opr. opr must be greater than 0. dst and src should be different images. The image dst does not need to be the same size as the image src, but the two images do need to be distributed among the slaves in a manner compatible with the resam1 (user command RESAMPLE) algorithms. ssq implements the SSQ user command.

* *

void stats (src, xcl) file eminfo.c, MASTER & esinfo.c, SLAVE

I32 src	number of the source image
I32 xcl	number of the kernel that defines the excluded edge region

stats calculates pixel value statistics for image img, excluding the edge region implied by the domain of kernel xcl. stats sets some user-accessible variables:

_NUM($N)	number of pixels
_AVG($AVG)	average of pixel values
_SIG($SIG)	standard deviation of pixel values
_MAX($MAX)	maximum pixel value
_MIN($MIN)	minimum pixel value
_NROW($NROW)	number of rows
_NCOL($NCOL)	number of columns
_A($A)	value of A coefficient in table
_B($B)	value of B coefficient in table

stats implements the STATS user command.

* *

I32 streq (str1, str2, nchar) file emmain.c, fmmain.c, & gmmain.c, MASTER

char * str1	first string
char * str2	second string
I32 nchar	number of characters to compare

streq compares the first nchar characters of the two strings, without regard to case, and returns the number of equal characters before the first unequal character pair.

* *

F32 sumpix (src, xcl) file eminfo.c, MASTER
void sumpix (src, xcl) file esinfo.c, SLAVE

I32 src	number of the source image
I32 xcl	number of the kernel that defines the excluded edge region

sumpix calculates the sum of all the pixels in the image img, excluding the edge region defined by the domain of kernel xcl. This operation sets the user-accessible variables $N(_NUM) = number of pixels, $AVG(_AVG) = average value of pixels. MASTER sumpix returns the sum value. sumpix implements the SUMPIX user command.

* *

void sums2 (void) file fmopt.c (& emdummy.c), MASTER & fsopt.c (& esdummy.c), SLAVE, F only

sums2 is used by asums to accumulate the sums used in the training process. sums2 uses ldblk.

* *

void ttg3 (img, row0, col0, q, caption) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char *caption	image caption

ttg3 displays image img on the high-resolution RGB monitor. The top left corner of the image will appear at row0,col0 in the screen display. Pixel values in image img should be between 0.0 and 255.0 (see function scale, user command SCALE). Red will be used for pixels with value less than q, green for pixels with value greater than q. The character string caption will be printed below the image on the screen. If echo is on, the image will remain on the screen, and the program will stop execution, until the operator hits a key on the keyboard. ttg3 implements the DISP user command in ATR1.

* *

void ttg3res (img, row0, col0, q, caption, res) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char *caption	image caption
I32 res	number of the image with the red/green criterion pixel values

ttg3res is like ttg3, except that the pixels are colored red or green according to whether the corresponding pixel in the image res (normally, but not necessarily, a result image) has a value less than q. This allows an input image to be displayed with color coding based on a result image. ttg3res implements the DISPRES user command in ATR1.

* *

void vga5 (img, row0, col0, q, text) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char * text	image caption

vga5 is used to display an image on the operator's console, as opposed to the usual RGB monitor. The top left corner of the image will appear at row0,col0 in the screen display. Pixel values in image img should be between 0.0 and 255.0 (see function scale, user command SCALE). Red will be used for pixels with value less than q, green for pixels with value greater than q. The character string caption will be printed below the image on the screen. If echo is on, the image will remain on the screen, and the program will stop execution, until the operator hits a key on the keyboard. vga5 is slow and primitive and buggy, and it is not recommended for general use. vga5 requires that the device handler VGA be installed on the host, which must be an IBM DOS type of computer (ATR1 or ATR2). vga5 implements the VGA user command.

* *

void wait (buf, j) file comm.inc, SCSI & VIDEO

I32 * buf	an internode communication buffer, bufinni or bufouti
I32 j	the waited-for value of buf[0]

This function is used only by the ATR1 programs SCSI and VIDEO. It causes the calling thread to stop execution until buf[0] is equal to j. This is used to make the main thread wait for the desired condition of the flags for the bufinni and bufouti communication buffers.

* *

void WaitInnData (void) file comm.inc, MASTER & SLAVE

WaitInnData blocks the calling thread (normally the main thread) until data is available in buffer bufinni. Data is put into bufinni by the function rcv running is a separate thread.

* *

void WaitOutFree (void) file comm.inc, MASTER & SLAVE

WaitOutFree blocks the calling thread (normally the main thread) until the data buffer bufouti is free for a new use by the main thread. This function should be used before writing to bufouti, to avoid overwriting data in bufouti that is being sent to another node by the function snd which runs in a separate thread.

* *

void warn (mess) file emmain.c, fmmain.c, gmmain.c, MASTER

char * mess	string to be printed

If echo is on, warn prints the specified string as part of a warning. For programs E and F, warn also prints the line number for the .fc file containing the current command.

* *

void warn1 (dst, src) file emntrp.c, gmntrp.c, MASTER

I32 dst	number of the first image
I32 src	number of the second image

warn1 prints a warning message if dst and src both refer to the same image.

* *

void warn2 (img, opr) file emntrp.c, gmntrp.c, MASTER

I32 img	number of the image
I32 opr	number of the kernel

warn2 prints a warning message if image img does not have enough overlap rows to accommodate kernel opr.

* *

void warn3 (img, novl) file emntrp.c, gmntrp.c, MASTER

I32 img	number of the image
I32 novl	number of required overlap rows

warn3 prints a warning message if image img has fewer than novl overlap rows.

* *

void wtgtz (opr) file emconv.c, MASTER

I32 opr	number of the kernel

wtgtz checks to see that the sum of the values in the kernel opr is greater than zero, and stops program execution if it is not. wtgtz is used with user command MOMUV.

* *

void ximg (dst) file esmath.c, SLAVE

I32 dst	number of the destination image

ximg sets the value of each pixel in image dst equal to the pixel's x (column) coordinate. Column numbers start with 0 for the left-most column, and increase toward the right. ximg implements the XIMG user command.

* *

void xlin01 (dst, srcb, srca, opr) file eslines.c, SLAVE

I32 dst	number of the destination image
I32 srcb	number of the image with brightness values
I32 srca	number of the image with tangent(angle) values
I32 opr	number of the kernel that specifies how much to extend the lines

xlin01 extends lines found by lin01. The "central" pixel in the local region of image dst is set equal to the weighted sum of the brightnesses of the line segments that are in the local region defined by the kernel opr and are oriented in the direction so that their extensions would pass through the central pixel. The weights are the values of the pixels in opr. srcb and srca are the images containing the brightness and orientation data from lin01. dst, srcb, and srca should all be different images, all of the same size. This operation uses overlap rows for both images srcb and srca. dst pixels in the excluded edge region defined by the domain of opr are set to zero. xlin01 implements the XLIN01 user command.

* *

void xwin (img, row0, col0, q, caption) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char * caption	image caption

xwin displays image img on the high-resolution RGB monitor. The top left corner of the image will appear at row0,col0 in the screen display. Pixel values in image img should be between 0.0 and 255.0 (see function scale, user command SCALE). Red will be used for pixels with value less than q, green for pixels with value greater than q. The character string caption will be printed below the image on the screen. If echo is on, the image will remain on the screen, and the program will stop execution, until the operator hits a key on the keyboard. xwin implements the DISP user command in ATR3.

* *

void xwininit (argc, argv) file emio.c, MASTER

int argc
char * *argv

xwininit initiallizes the high-resolution RGB display software. Its arguments argc and argv are the token count and token pointer arguments of the function main, referring to the arguments on the operating system command line that invoked this software (no arguments should be specified in this command line). The ATR3 system uses the environment variable DISPLAY, and assumes that the window manager program mwm is running in the background. xwininit is invoked one time, when the program is started. xwininit is used only on ATR3.

* *

void xwinres (img, row0, col0, q, caption, res) file emio.c, MASTER

I32 img	number of the image to be displayed
I32 row0	screen row for top of image
I32 col0	screen column for left edge of image
F32 q	red/green criterion
char * caption	image caption
I32 res	number of the image with the red/green criterion pixel values

xwinres is like xwin, except that the pixels are colored red or green according to whether the corresponding pixel in the image res (normally, but not necessarily, a result image) has a value less than q. This allows an input image to be displayed with color coding based on a result image. xwinres implements the DISPRES user command in ATR3.

* *

void xy2rt (dstr, dstt, srcx, srcy, xcl) file esmath.c, SLAVE

I32 dstr	number of the image to receive radius values
I32 dstt	number of the image to receive tangent(angle) values
I32 srcx	number of the image with x-coordinate values
I32 srcy	number of the image with y-coordinate values
I32 xcl	number of the kernel that defines the excluded edge region

xy2rt converts a vector field from x-y representation to r-tangent(angle) representation. Images srcx and srcy contain the x and y component values. The vector magnitude will be put into image dstr, and the tangent of the angle between the vector and the x axis will be put into image dstt. xcl specifies an excluded edge region, in which the dstr and dstt pixel values will not be changed. None of the images need be different from any of the others, although it would normally not be sensible for srcx and srcy to be the same image. If the two destination images dstr and dstt are the same, the destination image will contain the vector magnitudes and the vector direction values will not be written to any image. All these images should be the same size. xy2rt implements the XY2RT user command.

* *

void yimg (dst) file esmath.c, SLAVE

I32 dst	number of the destination image

Set the value of each pixel in image dst equal to the pixel's y (row) coordinate. Row numbers start with 0 for the top row, and increase toward the bottom of the image. yimg implements the YIMG user command.

* *

void zerbuf (img) file esmath.c, SLAVE

I32 img	number of the destination image

zerbuf sets all the pixels in image img, including the overlap rows, to zero. zerbuf implements the ZEROIMAGE user command.

* *

void zsums (void) file fmopt.c, MASTER, F only

zsums initiallizes (sets to zero) certain sums of feature image pixel values and their products, which are used in the training process. zsums is called before the first call to asums for the first of a sequence of training scenes.

* *

void ztbrows (img, jlo, jhi) file esmath.c, SLAVE

I32 img	number of the destination image
I32 jlo	upper limit of low-number rows to be zeroed
I32 jhi	lower limit of high-number rows to be zeroed

ztbrows sets to zero all the pixels in the indicated top and bottom rows of image img. The zeroed

rows are all rows with row number less than jlo and those with row number greater than jhi.

* *

COMMAND CODES

Numeric values are used as command codes throughout the software. The following table summarizes the interpretations that the different types of node apply to different command code values. For example, command code 2 is interpreted in the MASTER program (in function feats) as the READY user command, and the SLAVE (S), SCSI (I), and VIDEO (V) programs interpret command code 2 as a command to simply reply to MASTER. (The SCSI and VIDEO nodes exist only in ATR1.)

CODE	MASTER response	other node response	
0	REM		
1	ECHO		
2	READY	S,I,V: reply when ready	
3	ECHOTIME		
4	ECHO_ON		
5	ECHO_OFF		
6	PAUSE		
9	STOP,END	S: stop feature calculation subroutine	
11	PRINTS		
12	SDEFI32		
13	SDEFF32		
20	SEQ		
21	SADD		
22	SSUB		
23	SMUL		
24	SDIV		
25	SSQRT		
26	SABS		
27	SMAX		
28	SMIN		
51	STAN		
52	SATAN		
53	SATAN2		
54	SLOG		
55	SEXP		
101	NEWIMG		
102	READIMAGE	S: receive image row	I: SCSI input
103	READKERNEL	S: receive kernel	
104	CONVOLVE	S: convolve	
105	WRITEIMAGE	S: send image	I: SCSI output
106	OVERLAP	S: share overlap values	
107	COPYEDGES	S: copy edges	
108	ZEROIMAGE	S: zero image	
109	DISPRES	S: send image, color coded	V: display, color coded
110	SCALE	S: scale image	
111	SSQ	S: local sum of squares	
112	STATS	S: calculate image statistics	
113	ORDFLT	S: order filter	
114	DEFIMG	S: define image	
115	DEFKERN	S: define kernel	
116	UNDERSAMPLE	S: DELETED	

117	COPDEF	S: copy image definition
118		S: (F only) calculate feature weights
119	WRITERESULT	
120	RESAMPLE	S: scale resample
121	ADD	S: add
122	SUB	S: subtract
123	MUL	S: multiply
124	DIV	S: divide
125	SQRT	S: square root
126	ABS	S: absolute value
127	MAXCON	S: maximum, constant
128	MINCON	S: minimum, constant
129	MAX	S: maximum
130	MIN	S: minimum
131	ADDCON	S: add constant
132	SUBCON	S: subtract constant
133	MULCON	S: multiply by constant
134	DIVCON	S: divide by constant
135	SETCON	S: set to a constant
136	COPY	S: copy image
137	REPLO	S: replace low values
138	REPHI	S: replace high values
141	SETPIX	S: set a pixel
142		S: get a pixel, overlap, getpixo
143	GETPIX	S: get a pixel, primary, getpixp
144	SMTHX	S: exponential smooth x
145	SMTHY	S: exponential smooth y
146	HIST2	S: histogram
147		S: (F only) accumulate sums
148		S: linear resample, resam2
149		S: getblk
150	SETHFA	
151	TAN	S: tangent
152	ATAN	S: arctangent
153	ATAN2	S: arctangent2
154	LOG	S: natural logarithm
155	EXP	S: exponential
161		I: SCSI commands
162	TTG3 diagnose	V: diagnose
171	READSCENE	
172	WRITEFEAT	
173	FEAT *	
174	CLEAR	S: clear tables
176	MED1X	S: median filter x
177	MED1Y	S: median filter y
178	VGA	
179	SET_AB	
181		S: send one row
182	FEATFIL	
183	MODMSK	S: modify mask
184	LIN01	S: find lines 1
185	XLIN01	S: extend lines 1
186	NLIN01	S: count lines 1
188	XGRAD	S: gradient x
189	YGRAD	S: gradient y

190	LIN02	S: find lines 2
191	EDJ02	S: find edges 2
192	DISP	V: display
193	GRADT	S: gradient
194	XY2RT	S: rectangular to polar
195	GRADCON	S: concentrate gradient
196	RIJCON	S: concentrate rigdes
197	CONTOUR	S: make contours
198	SEGLAB	S: label segments
199	COMPOZ	S: insert image
200	NTRP00	S: interpolate, constant
201	NTRP01	S: interpolate, linear
202	QUADXY	S: quadratic fit
203	QUADUV	S: quad fit, rotated
204	EXTRACT	S: extract image
205	MOMUV	S: PDF moments
206	LABEL	
207	JUMP	
208	IF	
209	BRANCH	
210	PDFXY1	S: image PDF
211	PDFXYZ	
221	PEAK1	S: find peaks
222	PLNK1	S: link peak pixels
223	PACC1	S: accumulate peak sums
224	PMOMXY1	S: sums to moments
225	PMOMUV1	S: sums to moments
226	PMOMXY	
227	PMOMUV	
228	PMRG1	S: merge peaks
231	BANDR	
232	PHEAD	
233	INCFIL	
234	REMAP	S: remap1
235	SUMPIX	S: sumpix
236	REGISTER	S: reg1a (DELETED)
237	XIMG	S: ximg
238	YIMG	S: yimg
239	SMTHXS	S: smthxs
240	SMTHYS	S: smthyx
241	SBEND	S: s-bend correction
242	GOSUB	
243	SUBDEF	
244	RETURN	
256		V: erase
257		V: cursor 1 on
258		V: cursor 1 off
259		V: position cursor 1
260		V: start line
261		V: draw line
262		V: write pixels
263		V: write text
271	INS (G only)	
272	INM (G only)	
273	MARK (G only)	

274 OUTS (G only)
275 OUTM (G only)

ATR1 SCSI SOFTWARE

These sections describe software that is used exclusively on the ATR1 SCSI interface node. (ATR2 and ATR3 do not have a SCSI node or a TSCSI program.) The software described here is for version 7 of program TSCSI and for version 14 of programs E, F, and G. Some software that is used on both the SCSI node and either the MASTER or the SLAVE node is described in earlier parts of this programmer's manual. Many of the functions included in the SCSI node software are accessible from the MASTER program of TSCSI but not from the programs E, F, or G. However, the presence of these inaccesssible functions does not cause any problems for E, F, or G, and in the interest of simplicity of software maintenance the same SCSI node software is used for E, F, and G as is used for TSCSI.

The SCSI node source code comprises the files scsi7.c, disk7.c, tape7.c, scsilib.c, scsilib.h, efg.h, and comm.inc. scsilib.c is primarily low level SCSI functions supplied by Alta Technology Corp., and those functions are not described here.

The TSCSI program is for manipulating the devices that are attached to the ATR1 SCSI interface node. The interesting software is that which runs on the SCSI node; the MASTER node software merely sends commands to the SCSI node, and the SLAVE and VIDEO software serve no real function except to make the overall TSCSI program complete enough to function.

GLOBAL CONSTANTS AND VARIABLES

This section describes some global variables and constants that are important in the SCSI node software and are not described in previous sections that discuss MASTER and SLAVE node software.

char Gbuf[1600]	a scratch buffer for general use
I32 Gbufsiz	size of the scratch bufffer Gbuf
#define MDEV 5	maximum allowed number of devices
#define SCSIHOST 0 #define SCSIDISK 1 #define SCSITAPE 2 #define BUFFER 3	device numbers
char * devnam[MDEV]={"SCSIHOST:","SCSIDISK:","SCSITAPE:","BUFFER:"," "}	an array of names of the hardware devices and pseudo-devices, ending with a blank name
DEV device[MDEV]	an array of structures describing the hardware devices and pseudo-devices
I32 iddev[MDEV]	iddev[dev] is the SCSI ID for device dev. If iddev[dev]==-1, there is no SCSI ID for device dev, probably because device dev does not exist. If iddev[dev]==8, the device exists but does not actually use the SCSI bus.
I32 devid[8]={SCSIHOST,-1,-1,-1,-1,-1,-1,-1}	devid[id] is the device number for the device with SCSI ID id, or -1 if no device has SCSI ID id. SCSI ID 0 always goes with device SCSIHOST.

STRUCTURES

This section describes some structures that are important in the SCSI node software and are not described in previous sections that discuss MASTER and SLAVE node software.

* *

STRUCTURE TYPE: DEV

STRUCTURE INSTANCES: device[MDEV]

STRUCTURE MEMBERS:

I32 bufsiz	buffer size, in bytes
I32 (* openr)(FILE7 * fil)	pointer to device-specific openr function
I32 (* openw)(FILE7 * fil, I32 tim)	pointer to device-specific openw function
I32 (* fillbuf)(FILE7 * fil)	pointer to device-specific fillbuf function
I32 (* skipb)(FILE7 * fil, I32 skipbytes)	pointer to device-specific skipb function
I32 (* emptybuf)(FILE7 * fil)	pointer to device-specific emptybuf function
I32 (* close)(FILE7 * fil)	pointer to device-specific close function
I32 flags	device flags

The array device[] includes a structure of this type for each hardware device (such as SCSIDISK, SCSITAPE, and SCSIHOST) and each pseudo-device (such as BUFFER).

The device SCSIDISK is a hard disk, DOS formatted.

The device SCSITAPE is an Exabyte 8 mm tape drive, or perhaps a compatible device.

The device SCSIHOST is the link from the SCSI node to the MASTER node. No more than one input (read) file and one output (write) file can be opened at any one time for the SCSIHOST device. Since this device uses the bi-directional link to the MASTER node and the communication buffers bufinni and bufouti, etc., other uses of these buffers and this link while a SCSIHOST file is open may cause problems. For SCSIHOST files, the .ptr element of the FILE7 structure points to an array of 9 I32 values. The scsi7_fillbuf or host7_fillbuf command waits until there is data in the communication input buffer bufinni, and it sets the .bufsiz element of the FILE7 structure equal to bufinni[9] and returns with the FILE7 pointers pointing to bufinnc. Data received in the lower elements of bufinni[] is copied into the I32[9] array pointed to by the .ptr element of the FILE7 structure. The scsi7_emptybuf or host7_emptybuf command causes the data buffer to be sent out over the link, with the I32[9] array at FILE7.ptr used as the values for the first 9 elements of bufouti[] in the sendlong function call. The number of bytes sent in bufoutc[] (that is, beyond the first 9 I32 elements of bufouti[]) is the value of the .nbytes element of the FILE7 structure. Some of the programs that use the SCSIHOST device use the convention that a message with zero length (bufinni[9] or bufouti[9] is 0) indicates an end-of-file condition.

The device BUFFER is simply a FILE7 structure and a buffer (a block of memory) that the FILE7 structure points to, with no physical device that the buffer can be copied to or from. You can create multiple instances of pseudo-files for this BUFFER device, and they are all maintained independently. These pseudo-files can be written to and read from by functions like copfil, as long as not more than one buffer full of data is written or read without resetting the FILE7 pointers. A BUFFER pseudo-file is created by calling scsi7_open with the mode parameter either NULL or the name of an existing character buffer that is to be used for this pseudo-file; mode should not be "W" or "R" or "w" or "r". The scsi7_emptybuf function, or the buff7_emptybuf function, should be called before reading from the buffer, and the scsi7_fillbuf function, or the buff7_fillbuf function, should be called before writing to the buffer. The buffer contents can also be accessed directly, by using the .buf element of the FILE7 structure as a pointer to the buffer, but of course such access does not automatically maintain the pointers in the FILE7 structure. The scsi7_close function, or the buff7_close function, destroys the pseudo-file and, if the buffer was allocated by the scsi7_open (buff7_openw) function, de-allocates the buffer.

* *

STRUCTURE TYPE: FILE7

STRUCTURE MEMBERS:

I32 dev	hardware device number for this file
char * nam	pointer to file name, without device name
char * buf	pointer to data buffer, or NULL
I32 bufsiz	size of data buffer, in bytes
I32 pbytes	number of bytes before this buffer
I32 nbytes	position in buffer; number (relative to start of buffer) of next byte to be read from buffer or written to buffer
I32 filsiz	size of file, or -1
I32 flags	1 --> read; 2 --> write; 4 --> borrowed buffer
void * ptr	pointer to other (device-specific) data for this file

A structure of this type is created (memory is allocated) for each file and pseudo-file that is opened, and this structure is destroyed when the file is closed.

* *

STRUCTURE TYPE: HFP

STRUCTURE MEMBERS:

I32 buf9[9]	values to be put into bufouti[0-8] when sending data, or copied from bufinni[0-8] when receiving data

This structure is used as the SCSIHOST-specific data pointed to by the .ptr member of the FILEL7 structure.

* *

STRUCTURE TYPE: DFP

STRUCTURE MEMBERS:

I32 loc	location (sector number) of file
I32 dir	location of this file's home directory
I32 off	offset of this file's entry from the beginning of the directorry
I32 sector	number of last sector transferred between disk and memory, or -1
I32 nbytes	number of bytes written or read, including those written to a memory buffer but not yet written to the disk, not including those read from the disk but not yet read from the memory buffer

This structure is used as the SCSIDISK-specific data pointed to by the .ptr member of the FILE7 structure.

* *

STRUCTURE TYPE: DIRE

STRUCTURE MEMBERS:

char nam[DIREnamlen]	file name, 8-char base + 3-char ext
char typ	file type code: bit 0 --> read only bit 1 --> hidden bit 2 --> system bit 3 --> volume name bit 4 --> subdirectory bit 5 --> archive (normal file)
char res[10]	reserved
U16 tim	file creation time
U16 dat	file creation date
U16 loc	file location, cluster number
I32 siz	file size, bytes

This structure DIRE is the standard DOS disk file directory entry structure.

* *

FUNCTIONS

This section describes all the functions in the SCSI node software, except those in the Alta Technology Corp. library.

* *

I32 buff7_close (fil)

FILE7 * fil	pointer to the BUFFER file

buff7_close destroys the pseudo-file. buff7_close always returns 0. buff7_close is the BUFFER-specific close function and should normally be called only by scsi7_close. See the discussion of the BUFFER device in the DEV structure description.

* *

I32 buff7_emptybuf (fil)

FILE7 * fil	pointer to the BUFFER file

buff7_emptybuf resets the pointers to indicate an empty buffer and allow writing to a BUFFER pseudo-file: fil->nbytes=0; fil->filsiz=-1; fil->pbytes=0. buff7_emptybuf always returns 0. buff7_emptybuf is the BUFFER-specific emptybuf function and should normally be called only by scsi7_emptybuf. See the discussion of the BUFFER device in the DEV structure description.

* *

I32 buff7_fillbuf (fil)

FILE7 * fil	pointer to the BUFFER file

buff7_fillbuf resets the pointers to allow reading data from a BUFFER pseudo-file: fil->pbytes=0; fil->filsiz=fil->nbytes; fil->nbytes=0. buff7_fillbuf always returns 0. buff7_fillbuf is the BUFFER-specific fillbuf function and should normally be called only by scsi7_fillbuf. See the discussion of the BUFFER device in the DEV structure description.

* *

I32 buff7_openw (fil, size)

FILE7 * fil	pointer to the BUFFER file
I32 size	size of buffer to be created

buff7_openw opens a pseudo-device file, for device BUFFER. The device is open for both read and write operations; there is no buff7_openr function. If fil->ptr is NULL, buff7_openw allocates a buffer with size bytes; if fill->buf is not NULL, the buffer pointed to by fil->ptr is used as the file buffer. On return from buff7_openw, fil->buf points to the buffer; fil->ptr is NULL; fil->nbytes is 0; fil->pbytes is 0; fil->bufsiz is equal to the input parameter size; fil->filsiz is -1; and fil->flags is 0 if the buffer was created by buff7_openw, or 4 if the buffer is borrowed (if fil->ptr was not NULL on entry to buff7_openw). buff7_openw always returns 0. buff7_openw is the BUFFER-specific open function and should normally be called only by scsi7_open. See the discussion of the BUFFER device in the DEV structure description.

* *

I32 buff7_skipb (fil, skipbytes)

FILE7 * fil	pointer to the BUFFER file
I32 skipbytes	number of bytes to be skipped

buff7_skipb skips skipbytes in reading from a BUFFER pseudo-file. buff7_skipb always returns 0. buff7_skipb is the BUFFER-specific skipb function and should normally be called only by scsi7_skipb. See the discussion of the BUFFER device in the DEV structure description.

* *

void c2I32 (c, i)

char * c	a pointer to a character array
I32 i	the value to be put into the character array

c2I32 puts the two least significant bytes of i into the first two bytes of character buffer c, with the less significant byte first.

* *

void c4I32 (c, i)

char * c	a pointer to a character array
I32 i	the value to be put into the character array

c4I32 puts the four bytes of i into the first four bytes of character buffer c, with the less significant bytes first.

* *

I32 CS (S)

I32 S	a sector number

CS returns the number of the cluster that includes sector number S, or 0 if S is less than the first sector of cluster 2.

* *

void disk2_fixnam (oldnam, newnam)

char * oldname	an ATR1 SCSI node device&file name, input
char * newname	a modified form of oldname, returned

disk2_fixnam starts with the ATR1 SCSI device&file name oldname, removes any ATR1 SCSI node device name (such as SCSIDISK:, SCSITAPE:, SCSHOST:) and converts the remainder of the file name in oldname to upper case characters, and copies this modified file name into the string newname. The character pointers newname and oldname can be the same; that is, the file name can be modified in place without being copied to a new string.

* *

l32 disk2_namfit (entry, wildnam)

char *entry	pointer to a DOS file name, with no wild characters
char *wildnam	pointer to a DOS file name template, with wild characters

disk2_namfit compares a DOS file name entry, with no wild characters, with another DOS file name wildnam, which may include wild characters. disk2_namfit returns 0 if the names do not match, or the number of matching characters (not including the terminating '\0', '\\', or ' ') in wildnam if the names do match. Any occurrence of "?" in wildnam is construed as matching any one character except delimiters in entry. Any occurrence of "*" in wildnam is construed as matching any substring not including delimiters in entry. A substring is any sequence of characters that does not include a delimiter character. The only delimiter in this context is the implied '.' (period) that does not appear explicitly but is assumed between the eighth and ninth characters of entry. The substring that matches the '*' can have 0 length. If a "*" in wildnam is immediately followed by a non-delimiter character, the entry substring that is replaced by the "*" is construed as ending before the next occurrence of that non-delimiter character in entry. Thus, when "*" is encountered in wildnam, characters in entry are skipped until either a delimiter character or the non-delimiter character following the "*" is encountered.

* *

void disk2_parse1 (oldnam, newnam)

char *oldnam	DOS path&file name, input; part of path, returned
char *newnam	part of DOS file name, returned

disk2_parse1 separates the last part of the path&file name input as oldnam, copies the last part into the string newnam, and deletes that last part from the string oldnam. The separation is done after the last '\\' that precedes an alphanumeric character or a wild character ('*' or '?') in the input oldnam. If no alphanumeric character is found, the entire input oldnam is returned in oldnam and newnam is returned as an empty string. If the input oldnam contains an alphanumeric character but no preceding '\\', oldnam is returned as an empty string and the entire input oldnam is returned in newnam. Examples:

input oldnam	**output oldnam**	**output newnam**
\dir\subdir\file.ext	==> \dir\subdir\	+ file.ext
\dir\subdir\	==> \dir\	+ subdir\
\dir\	==> \	+ dir\
dir\	==>	+ dir\
file.ext	==>	+ file.ext
\	==> \	+

* *

void disk2_writD (ptrD, nam, tim, loc, siz)

DIRE *ptrD	pointer to a DOS disk directory entry structure to be written to
char *nam	DOS file name, with no path or device
l32 tim	time&date value from the host system clock
l32 loc	number of the first cluster in the file
l32 siz	number of bytes in the file

disk2_writD constructs a DOS disk directory entry in the structure pointed to by ptrD. In this entry, nam will be the file name (8 character base + 3 character extension), tim will be the file creation time, loc will be the location (cluster number) of the start of the file, and siz will be the number of bytes in the file. It is the programmer's responsibility to ensure that these values really do represent the actual file. If the last character in the string nam is a '\\', the entry is marked as a subdirectory; otherwise, it is marked as

a simple file.

* *

void disk6_delete (filnam)

char * filnam	name of SCSIDISK disk file to be deleted

disk6_delete deletes SCSIDISK disk files. The string filnam may include the wild characters '*' and '?' in the base or extension parts of the file name, but not in the path part.

* *

void disk6_dir (filnam)

char * filnam	path or directory name

disk6_dir prints the directory for the SCSIDISK disk subdirectory specified in the string filnam. filnam can include a file name as well as a path or subdirectory name, and the file name (but not the path) can include the wild characters '*' and '?'.

* *

void disk6_Enew (filnam, tim, ptrnewfil, ptrnewdir)

char * filnam	name of the new file
I32 tim	time&date value for the new file creation time
DFP * ptrnewfil	pointer to a disk file pointer structure for the new file
DIRE * ptrnewdir	pointer to the structure in which the new entry is to be written

disk6_Enew creates a new file on the SCSIDISK disk. It allocates the first cluster for the file, and it writes a directory entry describing the new file. The directory entry is written to a memory location; copying this new directory entry onto the disk must be done separately. disk6_Enew also sets the values in * ptrnewfil appropriate to the new file.

* *

void disk6_find (filnam, filpointr, direntry)

char * filnam	path&file name of the file sought
DFP * filpointr	pointer to a disk file pointer structure that will receive the file pointers
DIRE * direntry	pointer to the structure that will receive the directory entry

disk6_find finds the file specified in the string filnam, and sets the values in the structures filpointr and direntry to describe the file. If the file is not found, disk6_find sets filpointr->loc to 0 and direntry->nam[0] to '\0'. If the last char of filnam is ' ' (a space), the found entry must be a simple file, not a subdirectory. If the last char of filnam is '\\', the found entry must be a subdirectory, not a simple data file.

* *

I32 disk6_linkC (oldC)

I32 oldC	the number of the present cluster in the file

disk6_linkC returns the number of the next cluster in the SCSIDISK disk file or subdirectory that includes cluster number oldC. If oldC is the last cluster in its chain, disk6_linkC finds an unused cluster and adds it to the chain and returns its number. If oldC is 0, disk6_linkC starts a new chain.

* *

I32 disk6_loadDS (DS0, D, bufDS)

I32 DS0	the sector number where the subdirectory starts
I32 D	the number of the entry in the subdirectory
char * bufDS	the destination buffer, to receive one sector of the directory

disk6_loadDS copies one sector of a directory from the SCSIDISK disk into the memory buffer at bufDS. The sector is the one that contains the D-th entry of the directory that starts at sector DS0. (The numbering of entries, the D values, starts at 0 for the first entry in each directory.) disk6_loadDS returns the number of the sector that is copied into memory, or 0 if an error occurs.

* *

I32 disk6_loadFS (relFS, bufFS)

I32 relFS	the relative sector number, relative to the beginning of the FAT
char * bufFS	the destination buffer, to receive one sector of the FAT

disk6_loadFS copies one sector of the first File Allocation Table (FAT) from the SCSIDISK disk into the memory buffer at bufFS. The sector is the one numbered relFS relative to the first FAT sector; relFS=0 refers to the first FAT sector. disk6_loadFS returns the absolute sector number of the loaded sector.

* *

I32 disk6_loadnextS (oldS, buf)

I32 oldS	current sector number
char * buf	memory buffer to receive the sector

disk6_loadnextS copies one sector from the SCSIDISK disk into the memory buffer at buf. The copied sector is the one which is next in the chain for the file that includes sector number oldS. disk6_loadnextS returns the number of the copied sector. If sector oldS is the last one in the chain, disk6_loadnextS returns the value 0 and does not load anything into the buffer buf.

* *

void disk6_mkdir (filnam, tim)

char * filnam	path&file name for the new directory
I32 tim	time&date value for the new directory creation time

disk6_mkdir creates a new subdirectory on the SCSIDISK disk.

* *

void disk6_mkfil (filnam, tim, ptrfil)

char * filnam	path&file name for the new file

I32 tim	time&date value for the new file creation time
DFP * ptrfil	pointer to the disk file pointer structure that will receive information about the new file

disk6_mkfil creates a new file (not a directory) on the SCSIDISK disk. If the subdirectories implied in the path specified in filnam do not already exist, they are created as required. The values in the DFP structure pointed to by ptrfil are set to describe the new file.

* *

I32 disk6_newC (void)

disk6_newC finds a not-yet-used cluster on the SCSIDISK disk, sets its FAT entry to -1 (to indicate that this cluster is the last in its chain), and returns the number of the cluster. It is the programmer's responsibility to use this returned cluster number to link this cluster to the previous cluster or to a directory entry.

* *

I32 disk6_nextC (thisC)

I32 thisC	number of the current cluster

disk6_nextC finds the SCSIDISK disk cluster that is next in the chain for the file that includes cluster number thisC, and returns that next cluster number. If thisC is the last cluster in the chain, disk6_nextC returns 0.

* *

char * disk6_nextD (ptrD, ptrS, buf)

I32 * ptrD	pointer to directory entry number
I32 * ptrS	pointer to sector number
char * buf	pointer to memory buffer

If * ptrD is -1, disk6_nextD copies SCSIDISK disk sector number * ptrS into the memory buffer at buf; sets * ptrD to 0, indicating the first entry of the directory; and returns buf, which points to that first entry which is at the beginning of the buffer buf. If * ptrD is greater than -1, disk6_nextD assumes that buf already holds a copy of sector number * ptrS, that this sector is part of a directory, and that this sector includes entry number * ptrD of that directory. disk6_nextD returns a pointer to the next directory entry in buf and increments the value of * ptrD. If necessary to get to the next directory entry, disk6_nextD reads a new sector from the disk into buffer buf and sets * ptrS to the number of the sector in buf. Thus, this function allows the programmer to step through all the entries in a disk directory. This function will go past the last valid entry in the directory, so the returned pointer might point to an unused entry. If the end of the directory file is passed, disk6_nextD returns char * NULL, leaves * ptrS unchanged, and sets * ptrD to -2.

* *

void disk6_rename (names)

char * names	a string containing two file names separated by a '\0'

disk6_rename changes the names of files on the SCSIDISK disk. The first path&file name in the string names, the target name, is the old file name that is to be changed, and the second file name in the

string names is the replacement file name. The target name can contain the wild characters '*' and '?' in the file name but not in the path name. If the target name contains wild characters, the replacement name should usually contain corresponding wild characters; otherwise, if the target name with wild characters matches more than one existing disk file name and the replacement name does not have corresponding wild characters, all of the matching disk file names will be changed to the same replacement file name. disk6_rename can be used to change subdirectory names as well as simple file names.

* *

void disk6_rmdir (filnam)

char *filnam	name of the subdirectory to be deleted

disk6_rmdir deletes a subdirectory from the SCSIDISK disk. The subdirectory must be empty.

* *

void disk6_saveFS (relFS, bufFS)

I32 relFS	relative FAT sector number, relative to the beginning of the FAT
char *bufFS	pointer to memory buffer

disk6_saveFS copies one sector from memory buffer bufFS to each FAT (File Allocation Table) of the SCSIDISK disk, to relative FAT sector number relFS in each FAT.

* *

I32 disk6_vacant (void)

disk6_vacant returns the number of unused bytes left on the SCSIDISK disk.

* *

void disk7_cd (filnam)

char *filnam	DOS-type disk path

disk7_cd changes the current SCSIDISK directory to that indicated in the string filnam.

* *

I32 disk7_close (fil)

FILE7 *fil	pointer to the SCSIDISK file

disk7_close closes the disk file. If the buffer was not empty, disk7_close calls disk7_emptybuf to write the buffer to the disk. disk7_close completes the disk directory entry describing the file, and returns 0. disk7_close is the SCSIDISK-specific close function, and should normally be called only by scsi7_close.

* *

void disk7_delD (DFP *file)

DFP *file	pointer to the directory to be deleted

disk7_delD deletes the file or subdirectory pointed to by the DFP structure file. Specifically, the first character in the file's entry in its parent directory is set to (hex)E5 to indicate a deleted file entry; and, if the entry had a non-zero file location (cluster number), the FAT is modified to make the cluster chain starting at that location available and the location value in the directory entry is set to 0.

* *

I32 disk7_emptybuf (fil)

FILE7 *fil	pointer to the SCSIDISK file

disk7_emptybuf transfers data from the file's memory buffer to the disk, for a SCSIDISK file open for output (write), and returns 0. disk7_emptybuf is the SCSIDISK-specific emptybuf function, and should normally be called only by scsi7_emptybuf.

* *

I32 disk7_fillbuf (fil)

FILE7 *fil	pointer to the SCSIDISK file

disk7_fillbuf transfers data from the disk to the file buffer, for a SCSIDISK file open for input (read). disk7_fillbuf returns 0 for normal completion, or -1 for an anticipated end-of-file, or -2 for error conditions including an unexpected end-of-file. disk7_fillbuf is the SCSIDISK-specific fillbuf function, and should normally be called only by scsi7_fillbuf.

* *

I32 disk7_openr (fil)

FILE7 *fil	pointer to the SCSIDISK file

disk7_openr opens a SCSIDISK file for input (read). disk7_openr returns 0 for normal completion, or -1 if the requested file is not found. disk7_openr is the SCSIDISK-specific openr function, and should normally be called only by scsi7_open.

* *

I32 disk7_openw (fil, tim)

FILE7 *fil	pointer to the SCSIDISK file
I32 tim	time&date value from the system clock

disk7_openw opens a SCSIDISK file for output (write), and returns 0. disk7_openw is the SCSIDISK-specific openw function, and should normally be called only by scsi7_open.

* *

void disk7_prnpth1 (filnam)

char *filnam	name of a file in the current directory

disk7_prnpth1 prints the total path for the file filnam in the current directory.

* *

I32 disk7_skipb (fil, skipbytes)

FILE7 *fil	pointer to the SCSIDISK file
I32 skipbytes	number of bytes to skip

disk7_skipb skips skipbytes in a SCSIDISK file that is open for input (read). disk7_skipb returns 0 for normal completion, or -1 for an anticipated end-of-file, or -2 for error conditions including an unexpected end-of-file. disk7_skipb is the SCSIDISK-specific skipb function, and should normally be called only by scsi7_skipb.

* *

void disk7_start (void)

disk7_start reads disk sector 0 and the boot record, and sets disk descriptor parameters in the ATR1 SCSI software. This software uses the first partition encountered; other partitions on the disk are ignored. This software assumes that the disk is a DOS disk, and that there is no more than one disk served by the SCSI software (any disk accessed directly by the MASTER node is separate from this SCSI system). This disk7_start function should be executed when the SCSI program is started, before other functions access the disk.

* *

void disk7_zero (tim)

I32 tim	time&date value from the system clock

disk7_zero resets the SCSIDISK disk File Allocation Tables (FATs) and the root directory to indicate an empty disk. This assumes a DOS disk. The boot record and sector 0 are not changed. This function operates only on the first disk partition encountered; other partitions are ignored.

* *

I32 dummy_openr (dummy)

FILE7 *dummy	pointer to a file

dummy_openr is a dummy function that does not do anything. It always returns 0.

* *

I32 host7_close (fil)

FILE7 *fil	pointer to the SCSIHOST file

host7_close closes the SCSIHOST file and returns 0. host7_close is the SCSIHOST-specific close function and should normally be called only by scsi7_close. See the description of the SCSIHOST device in the description of the DEV structure.

* *

I32 host7_emptybuf (fil)

FILE7 *fil	pointer to the SCSIHOST file

host7_emptybuf empties the buffer when the SCSIHOST device is open for output, as a write file operation. host7_emptybuf sends the current buffer of data out over the link. host7_emptybuf always returns 0. host7_emptybuf is the SCSIHOST-specific emptybuf function and should normally be called only by scsi7_emptybuf. See the description of the SCSIHOST device in the description of the DEV structure.

* *

I32 host7_fillbuf (fil)

FILE7 * fil	pointer to the SCSIHOST file

host7_fillbuf fills the buffer when the SCSIHOST device is open for input, as a read file operation. host7_fillbuf waits for a message to arrive on the link and appear in the input communication buffer bufinni. host7_fillbuf returns 0 unless a message with 0 length (bufinni[9]=0) is received on the link, in which case host7_fillbuf assumes an "end of file" condition and returns -2. host7_fillbuf is the SCSIHOST-specific fillbuf function and should normally be called only by scsi7_fillbuf. See the description of the SCSIHOST device in the description of the DEV structure.

* *

I32 host7_openr (fil)

FILE7 * fil	pointer to the SCSIHOST file

host7_openr opens the SCSIHOST device for input, as a read file operation. That is, host7_openr prepares to receive data from MASTER through the hardware link, and to treat this device as a file. host7_openr always returns 0. host7_openr is the SCSIHOST-specific openr function and should normally be called only by scsi7_open. See the description of the SCSIHOST device in the description of the DEV structure.

* *

I32 host7_openw (fil, dummy)

FILE7 * fil	pointer to the SCSIHOST file
I32 dummy	an unused dummy argument

host7_openw opens the SCSIHOST device for output, as a write file operation. That is, host7_openr prepares to send data to MASTER through the hardware link, and to treat this device as a file. host7_openw always returns 0. host7_openw is the SCSIHOST-specific openw function and should normally be called only by scsi7_open. See the description of the SCSIHOST device in the description of the DEV structure.

* *

I32 host7_skipb (fil, skipbytes)

FILE7 * fil	pointer to the SCSIHOST file
I32 skipbytes	number of bytes to skip

host7_skipb skips bytes in the input stream when the SCSIHOST device is open for input, as a read file operation. host7_skipb returns 0 unless a message with 0 length (bufinni[9]=0) is received on the link, in which case host7_skipb assumes an "end of file" condition and returns -2. host7_skipb is the SCSIHOST-specific skipb function and should normally be called only by scsi7_skipb. See the description of the SCSIHOST device in the description of the DEV structure.

* *

I32 I32c2 (c)

char * c	pointer to a sequence of 2 bytes in memory

I32c2 reads two bytes from the memory location * c, and uses the two bytes as the low bytes in the I32 value that is returned by I32c2. The two high bytes in the I32 value are 0. The first byte in memory becomes the lowest byte in the I32 value.

* *

I32 I32c4 (c)

char * c	pointer to a sequence of 4 bytes in memory

I32c4 reads four bytes from the memory location * c, and copies them bit-for-bit into the I32 value that is returned by I32c4. The first bytes in memory become the least significant bytes in the I32 value.

* *

int main (void)

main is the main function for the SCSI node. It receives and interprets command codes from other nodes, primarily MASTER, and generally controls all operations on the SCSI node.

* *

void report (status, msg)

I32 status	status value returned by a SCSI CCS command
char * msg	message to be printed if the status value is not zero

report prints the character string msg on the operator's console if status is not 0. This report function is used with a SCSI Common Command Set function(...) as the argument status, so the message is printed if the SCSI function returns an error code.

* *

void RollCallI (cpi, cpo)

Channel * cpi[]	array of channel pointers for input links
Channel * cpo[]	array of channel pointers for output links

RollCallI sets up inter-node communication links in this SCSI node and informs the MASTER node of the existence of this SCSI node.

* *

I32 SC (I32 C)

I32 C	cluster number

SC returns the number of the first sector in cluster C, or 0 if C is less than 2.

* *

I32 scsi7_close (fil)

FILE7 * fil	pointer to the file to be closed

scsi7_close calls the device-specific function to close the file pointed to by fil. scsi7_close always returns 0.

* *

I32 scsi7_copfil (src, dst, limit)

FILE7 * src	pointer to the source file
FILE7 * dst	pointer to the destination file
I32 limit	maximum number of bytes to be copied

scsi7_copfil copies bytes from file src to file dst until either one of these conditions is met:

1. the number of bytes copied is equal to limit, if the value of limit is greater than -1.
2. the number of bytes copied is equal to the size of either file as indicated in that file's .filsiz structure element, if that value is greater than -1.

The .filsiz condition is checked after the limit condition, so if both conditions are met simultaneously, scsi7_copfil returns as if only the limit criterion had been met.

If the copying stops because of the limit value, scsi7_copfil simply returns. The dst buffers may be left partly full and may not be written to the device.

If the copying stops because of a .filsiz value, scsi7_copfil does one of two things before returning:

1. If the dst buffer is not full, the remaining bytes are set to zero. These zeroed bytes are not included in the byte count.
2. If the dst buffer is full, the data is transferred from the buffer to the device (using scsi7_emptybuf).

scsi7_copfil returns the number of bytes copied to the dst buffer, regardless of whether this data has been transferred from the buffer to the device.

* *

I32 scsi7_copy (srcfilnam, dstfilnam, tim)

char * srcfilnam	name of source file
char * dstfilnam	name of destination file
I32 tim	time&date value

scsi7_copy copies from file srcfilnam to file dstfilnam. srcfilnam and dstfilnam are assumed to begin with device names such as "SCSIHOST:", "SCSIDISK:", or "SCSITAPE:". The value of tim is used if appropriate, as when a disk file is created as the destination. scsi7_copy returns the number of bytes copied, or -1 if an error occurs.

* *

I32 scsi7_dev (whole, part)

char * whole	file name including a device name
char * part	file name without any device name

scsi7_dev returns the number of the device for which the device name matches the first part of whole, or return -1 if there is no match. whole is assumed to begin with a device name such as "SCSIHOST:", "SCSIDISK:", or "SCSITAPE:". part is set equal to whole with the device name removed.

part and whole can both point to the same character array.

* *

I32 scsi7_emptybuf (fil)

FILE7 * fil	pointer to the file open for output

scsi7_emptybuf calls the appropriate device-specific function to transfer data from the buffer to the hardware device during a file write operation. scsi7_emptybuf is used during such operations as copying files.

For the special case in which the device is a BUFFER, scsi7_emptybuf (actually the BUFFER-specific buff7_emptybuf) merely sets the pointers in the structure fil to the beginning of the buffer in such a way that the buffer can be read from. scsi7_emptybuf always returns 0.

* *

I32 scsi7_fillbuf (fil)

FILE7 * fil	pointer to the file open for input

scsi7_fillbuf calls the appropriate device-specific function to transfer data from the hardware device to the data buffer during a file read operation. scsi7_fillbuf is used during such opreations as copying files.

For the special case in which the device is a BUFFER, scsi7_fillbuf (actually the BUFFER-specific buff7_fillbuf) merely sets the pointers in the structure fil to the beginning of the buffer in such a way that the buffer can be written to.

scsi7_fillbuf always returns 0.

* *

I32 scsi7_getb (fil)

FILE7 * fil	pointer to the source file

scsi7_getb returns the I32 equivalent of one byte from file or device fil, or returns -1 if the end of the file is passed. scsi7_getb returns the value -2 only if certain inconsistencies in the FILE7 structure values are detected.

* *

I32 scsi7_getnam (fil, nam)

FILE7 * fil	pointer to the source file
char * nam	destination string for the file name

scsi7_getnam reads a character string from the file fil and writes it into the memory buffer nam. '\r' characters are always skipped. Leading spaces, before the file name, are skipped. A space after a file name character is recognized as a terminating character, and the characters '\0' and '\n' are recognized as terminating characters even if no valid file name character has been read. The terminating character is not copied into the buffer nam; a '\0' character is written into buffer nam at the end of the string. scsi7_getnam returns the number of characters copied, or returns -1 if the end of the file is encountered before a string terminating character.

* *

void scsi7_innrband (src, nrow, nbytes, dstlo, dsthi, nbands, rband)

FILE7 *src	pointer to the source file
I32 nrow	number of image rows to be read
I32 nbytes	number of bytes per composite image row in the source file
I32 dstlo	lowest number destination node
I32 dsthi	highest number destination node
I32 nbands	number of image bands to be read
struct BANDR *rband	description of the assumed band structure in the source file

scsi7_innrband is the SCSI node support for the MASTER function inn8 used to read row-interleaved images. Any image header in the source file should already be skipped over, and the src FILE7 structure should point to the first composite image row desired, before this function is called.

* *

void scsi7_innsmpl (src, nrow, nbytes, dstlo, dsthi, img, bpp)

FILE7 *src	pointer to the source file
I32 nrow	number of image rows to be read
I32 nbytes	number of bytes per image row in the source file
I32 dstlo	lowest number destination node
I32 dsthi	highest number destination node
I32 img	number of the memory image destination
I32 bpp	bytes-per-pixel code

scsi7_innsmpl is the SCSI node support for the MASTER function inn8 used to read simple (not row-interleaved, not HFA) images. Any image header in the source file should already be skipped over, and the src FILE7 structure should point to the first image row desired, before this function is called.

* *

FILE7 *scsi7_open (filnam, mode, tim)

char *filnam	device&file name
char *mode	usually, a read-write code
I32 tim	usually, a time&date value

scsi7_open allocates a FILE7 structure, sets the values of some elements in the structure, and calls the appropriate device-specific function to open a file. filnam is assumed to begin with a device name such as "SCSIHOST:", "SCSIDISK:", or "SCSITAPE:". For normal files, the string mode should be "W" or "w" to open a file for writing, or "R" or "r" to open a file for reading. Otherwise, for a pseudo-file, the parameters in the scsi7_open call may be interpreted differently.

If filnam is "BUFFER:", the "file" is actually just a data buffer that is treated in a limited way as a file. See the discussion of BUFFER in the description of the DEV structure. In this case, if mode is NULL, a buffer is created for the duration of the pseudo-file existence, and the value of tim is used as the number of bytes in the buffer. If mode is not NULL, it is treated as a pointer to a buffer that already exists and will be used for this pseudo-file.

scsi7_open returns a pointer to the FILE7 structure that it allocated, or NULL if the file could not be opened.

* *

void scsi7_poll (void)

scsi7_poll checks all SCSI IDs except ID=0 (which always corresponds to the host device SCSIHOST), tries to match any active SCSI ID with the appropriate known device (SCSITAPE or SCSIDISK), and sets values in tables (arrays iddev[] and devid[]) to match devices and SCSI IDs. Also, for pseudo-devices that do not really use the SCSI bus (such as BUFFER), scsi7_poll sets the SCSI ID to the dummy value 8 (iddev[BUFFER]=8, for example). scsi7_poll prints a listing of devices and SCSI IDs on the operator's console.

* *

I32 scsi7_skipb (fil, skipbytes)

FILE7 * fil	pointer to the file open for input
I32 skipbytes	number of bytes to skip

scsi7_skipb calls the appropriate device-specific function to skip skipbytes bytes when reading an input file. scsi7_fillbuf always returns 0.

* *

I32 scsi7_tart (tarnam)

char * tarnam	name of device&file containing the TAR

scsi7_tart prints on the operator's console a list of the files in the TAR. tarnam should include the device name (SCSIDISK: or SCSITAPE:, for example). If the TAR is on tape, the tape should be positioned at the start of a TAR file header when this scsi7_tart function is called. scsi7_tart returns 0 for normal completion, or -1 if an error occurs.

* *

I32 scsi7_tarx (tarnam, lstnam, tim)

char * tarnam	name of device&file containing the TAR
char * lstnam	name of device&file containing a list of pairs of TAR file names and destination file names
I32 tim	time&date value from host

scsi7_tarx extracts files from a TAR. tarnam is the name of the file or device that contains the TAR. lstnam is the name of the file or device that contains a list of name pairs, each pair comprising a srcnam and a dstnam. srcnam is the file name that is in the TAR header for a TAR file. dstnam is the file that the TAR file will be extracted to; dstnam should include the device name, such as "SCSIDISK:". srcnam and dstnam should be separated by a space, ' '. Successive srcnam-dstnam pairs should be separated by a '\n'. If srcnam is a real file name, the TAR will be scanned in the forward direction (never backward) until the TAR file srcnam is found, and it will then be copied to the device&file dstnam. If the first character of srcnam is '+', the next TAR file will be copied to the destination. If the first character of srcnam is ' * ', dstnam should be a device name followed by an asterisk ("SCSIDISK: * ", for example) and all the remaining TAR files in the TAR will be copied to destination files with the device name as specified in dstnam and with file name the same as the TAR file name (except that '/' will be replaced with '\\'). Regardless of srcnam, if the first character of dstnam (after the device name) is ' * ', the destination file name will be the same as the TAR file name. The list of srcnam-dstnam pairs should be terminated either with a '\0' or with "\n\n". tarnam, lstnam, and dstnam must include a device name prefix. srcnam will generally not include such a prefix. If the TAR is a tape, the tape should be at the start of a TAR file header when this scsi7_tarx function is called. tim is the date&time from the host, and is used if appropriate in creating the destination file. scsi7_tarx returns 0 for normal completion, or -1 if an error occurs.

* *

I32 scsi7_tarx1 (tar, dstfilnam, tim, size)

FILE7 * tar	pointer to the source (input) file that contains the TAR
char * dstfilnam	name of destination device&file
I32 tim	time&date value from host
I32 size	number of bytes to be transferred

scsi7_tarx1 extracts one file from a TAR, into file dstfilnam. The source file pointers in tar should be at the beginning of a TAR file (after the header) on entry to this function. dstfilnam should include a device name, such as "SCSIDISK:". scsi7_tarx1 returns the number of bytes transferred, or -1 if an error occurs.

* *

I32 tape7_close (fil)

FILE7 * fil	pointer to a SCSITAPE file

tape7_close closes a SCSITAPE tape device file. tape7_close always returns 0. tape7_close is the SCSITAPE-specific close function, and it should normally be called only by scsi7_close.

* *

I32 tape7_emptybuf (fil)

FILE7 * fil	pointer to a SCSITAPE file

tape7_emptybuf writes data to the SCSITAPE tape device from the memory buffer, when writing to the SCSITAPE tape as an output file. tape7_emptybuf always returns 0. tape7_emptybuf is the SCSITAPE-specific emptybuf function, and it should normally be called only by scsi7_emptybuf.

* *

I32 tape7_fillbuf (fil)

FILE7 * fil	pointer to a SCSITAPE file

tape7_fillbuf reads data from the SCSITAPE tape device into the memory buffer, when reading the SCSITAPE tape as an input file. tape7_fillbuf returns 0 on normal completion, or -1 for an anticipated end-of-file, or -2 for a tape input error condition (including an unexpected end-of-file condition). tape7_fillbuf is the SCSITAPE-specific fillbuf function, and it should normally be called only by scsi7_fillbuf.

* *

I32 tape7_openr (fil)

FILE7 * fil	pointer to a SCSITAPE file

tape7_openr opens the SCSITAPE tape device as a file for input (read). tape7_openr always returns 0. tape7_openr is the SCSITAPE-specific openr function, and it should normally be called only by scsi7_open.

* *

I32 tape7_openw (fil, dummy)

FILE7 *fil	pointer to a SCSITAPE file
I32 dummy	not used

tape7_openw opens the SCSITAPE tape device as a file for output (write). tape7_openw always returns 0. tape7_openw is the SCSITAPE-specific openw function, and it should normally be called only by scsi7_open.

* *

void tape7_setBL (bytes)

I32 bytes	desired tape block length, in bytes

tape7_setBL sets the SCSITAPE tape block length parameter. It may be necessary to set this parameter correctly before reading a tape.

* *

I32 tape7_skipb (fil, skipbytes)

FILE7 *fil	pointer to a SCSITAPE file
I32 skipbytes	number of bytes to skip

tape7_skipb skips skipbytes when reading the SCSITAPE tape as an input file. tape7_skipb returns 0 on normal completion, or -1 for an anticipated end-of-file, or -2 for a tape input error condition (including an unexpected end-of-file condition). tape7_skipb is the SCSITAPE-specific skipb function, and it should normally be called only by scsi7_skipb.

* *

void tape7_start (void)

tape7_start sets up parameters for other functions that use the SCSITAPE tape device. This function should be executed when the program is started, before other tape access functions are used.

* *

COMMANDS ACCEPTED BY SCSI NODE

This section is an outline of the commands accepted by the SCSI node main program. As an example: the second line in the following list indicates that if bufinni[5] is equal to 102, the command is to input an image from a device attached to the SCSI node. The next line indicates that if bufinni[6] is equal to the value SCSIDISK, the data is to come from a SCSI disk file; a later line indicates that bufinni[6] being equal to SCSITAPE implies that the data should come from the SCSI tape. bufinni[7]=1 is a command to open the file and skip bufinni[8] bytes, with bufinni[10] and following bytes being the name of the file; bufinni[7]=2 is a command to read from the file, with bufinni[8] and following bufinni elements being values of parameters used in the read operation; bufinni[7]=3 is a command to close the file; and bufinni[7]=4 is a command to skip bufinni[8] bytes in the input file. In this list, "dev" is the device number.

Some of these commands are for standard SCSI Command Command Set (CCS) operations, such as *inquiry, ready*, etc., and are used primarily for diagnostic operations. A few of the commands are essentially duplicates, accomplishing the same overall tasks in different ways; the duplicates are retained to allow compatibility with earlier versions of MASTER and SLAVE programs.

```
[5] 2 --> respond, node ready
[5] 102 --> input an image
  [6] SCSIDISK --> from SCSI disk
    [7] 1 --> open file to read
      [8] number of bytes to skip
        [10...] file name, without device name
    [7] 2 --> read
      [8] bytes per pixel
        [10] number of records
          [11] bytes per record
            [12] low node destination
              [13] high node destination
                [14] image number
                  [15] number of bands
                    [16...] band1, ...
    [7] 3 --> close file, end image input operation
    [7] 4 --> skip
      [8] number of bytes to skip
  [6] SCSITAPE --> from SCSI tape
    [7] 1 --> open file to read
      [8] file number (file mark number)
    [7] 2 --> read
      [10] number of records
        [11] bytes per record
          [12] low node destination
            [13] high node destination
              [14] image number
    [7] 3 --> close file, end image input operation
[5] 105 --> write an image
  [6] SCSIDISK --> write to SCSI disk file
    [7] time and date
      [10...] file name, without device name
  [6] SCSITAPE --> write to SCSI tape
[5] 161 --> SCSI command
```

[6] 0x00 --> test unit ready
  [7] dev
[6] 0x01 --> rewind (rezero)
  [7] dev
[6] 0x03 --> request sense
  [7] dev
[6] 0x10 --> write filemark
  [7] dev, SCSITAPE only
    [8] number of filemarks to write
[6] 0x11 --> space
  [7] dev, SCSITAPE only
    [8] number of items to space over
      [10] 0 --> items are blocks, 1 --> items are filemarks,
          3 --> goto end of data.
[6] 0x12 --> inquiry
  [7] dev
[6] 0x1A --> mode sense
  [7] dev
[6] 0x25 --> read capacity
  [7] dev
[6] 0x28 --> read block of SCSI disk
  [7] dev, must be SCSIDISK
    [8] block number
[6] 0x2B --> locate logical block
  [7] dev (implemented only for dev=SCSITAPE)
    [8] block number
[6] 0x34 --> read position
  [7] dev
[6] 0x37 --> read defect data
  [7] dev
[6] -2 --> SCSI bus reset
[6] -3 --> scan disk for non-default blocks (DO NOT USE THIS COMMAND)
[6] -4 --> initiallize device, delete all files
  [7] dev
[6] -5 --> print directory
  [7] dev
    [10...] OPTIONAL file name (for disk only)
[6] -6 --> delete a file
  [7] dev (implemented for dev=SCSIDISK only)
    [10...] file name, without device name
[6] -7 --> set block length
  [7] dev (implemented for dev=SCSITAPE only)
    [8] block length in bytes
[6] -9 --> do nothing
[6] -11 --> copy one tape file to one disk file
  [7] 1 --> open disk file to write
    [8] time and date
      [10...] file name, without device name
  [7] 2 --> copy
    [10] number of records
      [11] bytes per record
        [12] bytes to skip
  [7] 3 --> close files, end copy operation
[6] -12 --> copy one SCSI disk file to host
  [10...] SCSI disk file name, without device name

[6] -13 --> copy one host file to SCSI disk
  [8] time and date
  [10...] SCSI disk file name, without device name
[6] -14 --> rename a SCSI disk file
  [7] dev (implemented for dev=SCSIDISK only)
    [10...] target name, '\0', replacement name, '\0', without device names
[6] -15 --> change current directory
  [7] dev (implemented for dev=SCSIDISK only)
    [10] new directory name, without device name
[6] -18 --> copy TAR to disk
    [8] time and date
      [10...] TAR file name and disk file name, withoug device name
[6] -19 --> general copy
    [8] time and date
      [10...] source file name, '\0', and destination file name, including device names
[6] -20 --> TAR file extract
    [8] time and date
      [10...] TAR device:file name, '\0', and list device:file name, including device names
[6] -21 --> TAR directory
      [10...] TAR file name, including device name

ATR1 VIDEO SOFTWARE

These sections describe software that is used exclusively on the ATR1 video interface node. (ATR2 and ATR3 do not have a video node.) Some software that is used on both the VIDEO node and either the MASTER or the SLAVE node is described in earlier parts of this programmer's manual. The software described here is for version 14 of programs E, F, and G.

Most of the VIDEO software is built on the TTGS library, which is not described here. A few functions in the TTGS library seemed to behave incorrectly, and they were replaced by the GDL_... functions listed here.

FUNCTIONS

This section describes all the functions in the VIDEO node software.

* *

void GDL_attach_cursor (id)

I32 id	cursor number

GDL_attach_cursor does the actual drawing of cursor number id at a previously specified location on the RGB monitor screen.

* *

I32 GDL_create_cursor (cursor, ncol, nrow, ec, ic)

char * cursor[]	array of strings that defines the shape of the cursor being created
I32 ncol	width of cursor
I32 nrow	height of cursor
I32 ec	
I32 ic	

GDL_create_cursor defines a cursor which other functions can draw and move on the RGB monitor screen, and returns the identification number of the new cursor.

* *

void GDL_draw_cursor (id, col, row)

I32 id	cursor number
I32 col	desired column position
I32 row	desired row position

GDL_draw_cursor draws cursor number id at position (row,col) on the RGB monitor screen.

* *

void GDL_unattach_cursor (void)

GDL_unattach_cursor erases whatever cursor may be on the RGB monitor screen.

* *

void movcur (row, col)

I32 row	desired row position
I32 col	desired column position

movcur draws a cursor on the RGB monitor at the indicated row and column position.

* *

int main (void)

main is the main function for the VIDEO node. It receives and interprets command codes from other nodes, primarily MASTER, and generally controls all operations on the VIDEO node.

* *

void RollCallV (cpi, cpo)

Channel * cpi[]	array of channel pointers for input links
Channel * cpo[]	array of channel pointers for output links

RollCallV sets up inter-node communication links in this VIDEO node and informs the MASTER node of the existence of this VIDEO node.

* *

void sub1 (void)

sub1 puts a test pattern on the RGB monitor, for test and diagnostic purposes.

* *

COMMANDS ACCEPTED BY VIDEO NODE

This section is an outline of the commands accepted by the video node main program. The number in brackets [ ] at the beginning of each line is the number of the bufinni[] array element that contains the value, and the following text gives the meaning of the value. A line that is indented is part of the command message that includes the less-indented preceding line.

```
[5] 2  READY, respond to master when ready.
[5] 109  fetch and display an image, result-coded.
  [6]  image number
    [7]  row0
      [8]  col0
        [10]  Q (floating)
          [11]  ncol
            [12]  nrow
              [13]  result image number
[5] 162  special TTG3 diagnostic commands.
  [6] 1  change display parameters and draw test pattern.
    [10]  PixelClock
      [11]  LineFrequency
        [12]  xRes
          [13]  yRes
            [14]  FrameRate
[5] 192  fetch and display an image.
  [6]  image number
    [7]  row0
      [8]  col0
        [10]  Q (floating)
          [11]  ncol
            [12]  nrow
[5] 256  erase
[5] 257  cursor1 on
[5] 258  cursor1 off
[5] 259  position cursor1
  [6]  cursor1 row
    [7]  cursor1 column
[5] 260  position cursor0 (invisible cursor)
  [6]  cursor0 row
    [7]  cursor0 column
[5] 261  draw line using cursor0
  [6]  ending row
    [7]  ending column
      [8]  color
[5] 262  write some pixels on the screen
  [6]  starting row
    [7]  starting column
      [8]  number of pixels
        [10]...  pixel values, one pixel per byte
[5] 263  write text on screen
  [6]  starting row
    [7]  starting column
      [10]...  characters, one per byte
```

REFERENCES

1. G. D. Lassahn, *Automatic TLI Recognition System, General Description*, INEL/EXT-97-00003, February 1997.

2. G. D. Lassahn, *Automatic TLI Recognition System, User's Guide*, INEL/EXT-97-00004, February 1997.