Pet2PegTool and 2Dicom:

Two Programs for Medical Image File Format Exchange

A Senior Research Project

Presented for the

Bachelor of Arts Degree

in College Scholars

The University of Tennessee, Knoxville

Marc Cruser

November 21, 1994

Establishment of a standard format for analysis of digital images at the University of Tennessee Medical Center Image Processing Lab.

The Image Processing Lab (IPL) at UT Medical Center analyzes digital images from four modalities: PET (Positron Emission Tomography), MRI (Magnetic Resonance Imaging), CT (Computed Tomography), and SPECT (Single Photon Emission Computed Tomography), each with a unique file format. The capability exists to convert MRI and PET images to flat file format and to convert those flat images to PET format. CT conversion software is still in development, and the bulk of the image analysis and rendering software in the lab works with PET and MRI files. Because of recent developments in medical image standardization, this situation is likely to change, hence the need to work with images of standard format. The IPL was recently provided with software to convert PET images into SPECT format so they can be viewed on Nuclear Medicine Pegasys workstations and subjected to additional analysis performed by proprietary hardware and software on those machines. This conversion software requires both testing and modifying in order to work with the images and file systems in the IPL. The first step in this project will consist of testing and modifying this program and providing a graphical interface that will work within the SunView windowing system used by the Pegasys workstations.

The second step is an evaluation of file formats for the development of a standard image format for interconverting the different modalities. Standard formats, such as Interfile, have been used for this purpose, and a new standard for medical imaging, DICOM, was introduced last year. DICOM has the additional advantage that it was designed for ready transmission on networks and may lend itself to telemedicine applications. Following the format evaluation, a standard imaging format will be recommended and details of the third step, in which software will be written to convert images to and from the recommended standard, will be refined.

February 1994

Purpose of Program

Pet2PegTool is a utility for use with medical imaging systems which work with both CTI and ADAC medical images. The program converts image files created on a CTI imaging device to a format readable on ADAC Pegasys workstations. Benefits include consolidation of sets of related images of different modalities and application of ADAC diagnosis software to CTI PET images.

Program Requirements

Pet2PegTool runs on ADAC Pegasys workstations running SunView. The program uses files included with the ADAC Programmer's Toolkit.

Installation

To extract the files from a 3.5" disk into the current directory, type `tar xf /dev/rfd0`. To extract the files from a tar file, type `tar xf pet2pegtool.tar`. This will create a new directory, *pet2peg*, that contains the program files. In this directory you will need to edit *Makefile* so it knows where to find the toolkit files on your system. Type `make` to create the executable file if it does not already exist. The cc compiler must be on your path in order to compile the program. Type `pet2pegtool` to run the program.

Running the Program

Opening Files

When Pet2PegTool begins, a file selection window appears on the screen. Type in the preferred directory and file filter, then press the "List" button to view the list of files. To select a file to convert, *triple-click* on one of the files listed in this box. If you wish to convert this file, click on "Prepare Conversion." You may also end the program by pressing the "Quit" button in this window.

Conversion Options Panel

If the selected file is in a recognizable CTI format, an option panel will appear over the file selection window. At the top of the panel is a message bar for communicating the current status of the conversion. Below this are the names of the source and output files. The default output filename is that of the source file with the extension ".adac." (If the file is immediately sent to the ADAC database, the output filename is irrelevant since it is removed during this process.) Specify whether the patient was scanned head-first or feet-first by clicking the value beside the "Orientation" label. If the source file contains static series, you may select a range of planes to convert. If it contains dynamic or gated images, options are available for converting a range of planes

from a selected frame and for a range of frames from a specified plane. Check the box beside the preferred option if this choice is available. Next, you may select to reslice the image, an option recommended for transverse images. If the converted images are to be immediately added to the ADAC database, make sure this option has been selected. Below these options are a number of header fields which may be modified if needed. When the conversion options have been set, click on the "Convert" button to create the ADAC image file.

Known Bugs

This program will not work effectively with whole body projections. While these images can be processed, the result is typically not useful and somehow manages to include pieces of other images with its data set. Some images appear to be off-center after conversion. A "Notifier Error" message results when the option panel appears but does not seem to affect functionality. On some machines, the plane and frame selection fields appear to be askew.

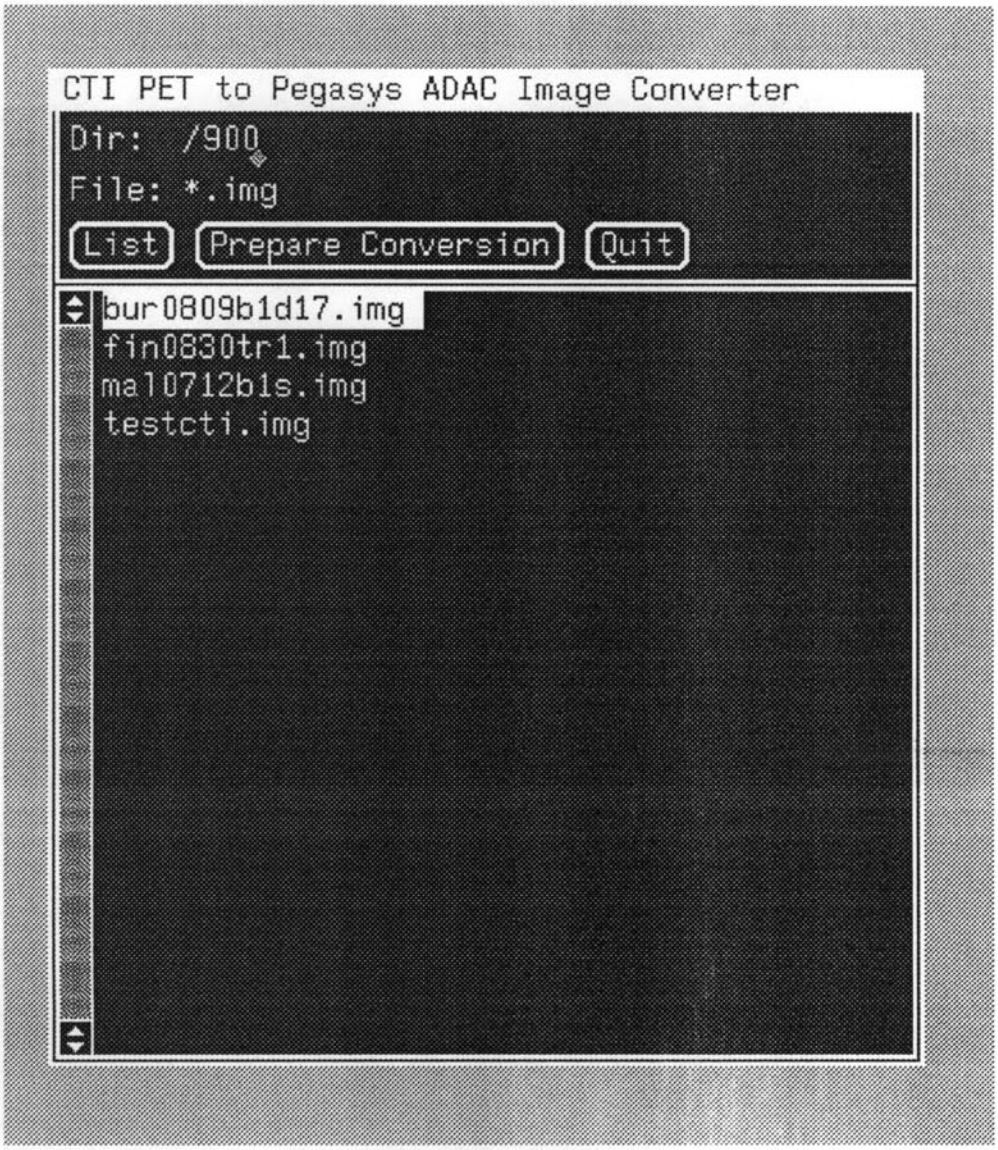
Program History

The code for the original pet2peg program was written and modified by Ed Glass from 1989 to 1992. The SunView interface, enhanced dynamic image capabilities, and other functions (including some intended for whole body projections) were added by Marc Cruser as undergraduate research in November 1994.

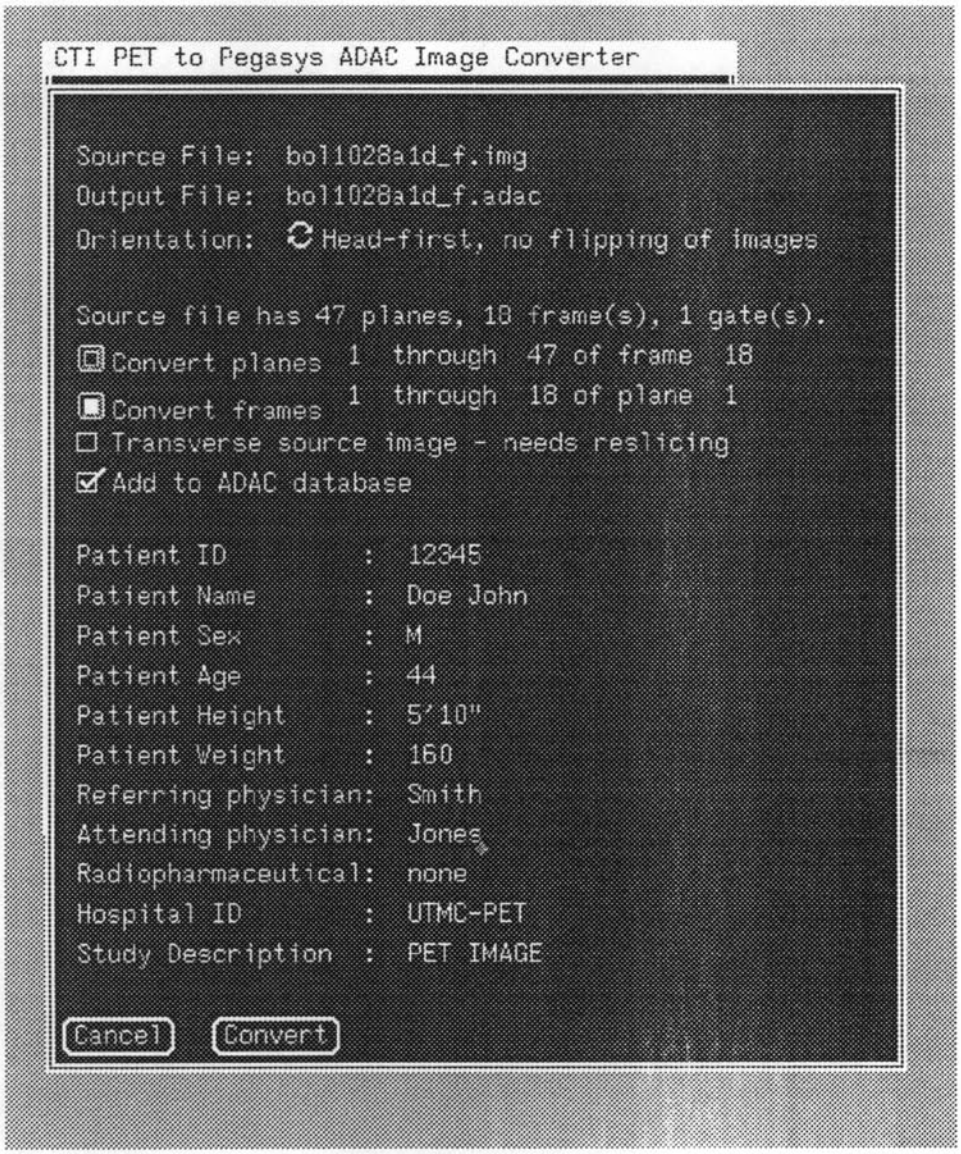
Future Developments

I hope to find out what causes some of the images to appear off-center, and just what causes the whole body projections to become scrambled and mixed with other images. If you have any other suggestions, you may reach me by e-mail at cruser@scanner.hosp.utk.edu.

File Selection Window



Conversion Options Panel



Purpose of Program

2Dicom was written to convert medical image files used within the UTMC IPL to the DICOM imaging standard. The program will be able to receive as input those image files which have been stored in ADAC, CTI, and Genesis formats. This conversion should make the collection of medical image files in the IPL easier to manage and increase compatibility with future imaging equipment which will likely have built-in DICOM compatibility.

Program Requirements

The conversion program runs on Sun Sparc workstations running X-Windows.

Installation

The file currently lies in the directory */home/Se/cruser/project/dicom*. Type `2dicom` to run the program. (In a full implementation of the program, extraction from a tar file and modification of the *Makefile* will also be necessary.)

Running the Program

Opening Files

From the File menu, select "Open ADAC," "Open CTI," or "Open Genesis." This creates a file browser from which you can change directories or select files with a double click of the mouse. If the program can recognize the file as an ADAC, CTI, or Genesis image, the header data from the file will be converted to the DICOM format. The Patient Data Window will appear in the program workspace with the data ready to edit if necessary. If you wish to view or edit the study-specific header information, select "Study Data" from the Edit menu. The two items under the Edit menu will toggle between these two displays, and the contents in each of these windows will be stored in memory when the window is closed.

Saving Files

To save the current set of images as a DICOM file, select "Save As" from the File menu. (In a later version of the program you will be prompted for a filename, and the image data will be transferred to disk. This option is currently not available.)

Additional Features

The header information that define values necessary for the display of the images can be viewed by selecting

"Image Header" from the Window menu. These values cannot be changed. (This is necessary to preserve image integrity.) To print out the header information for the current image, select "Print Header" from the File menu. Under the Help menu you can select options which provide information about the program or this README file. To end the program, select "Quit" from the File menu.

Known Bugs

As mentioned earlier, the program does not yet save the information to a DICOM file, but this will be remedied in early 1995. Some large CTI files (mostly dynamic images) cause the program to crash when they are read. A feature which allows the user to remove the "Image Header" display via the Window menu as well as from the "Done" button makes the program unstable. Resizing the program window can disorient some fields in the "Patient Data" and "Study Data" windows.

Program History

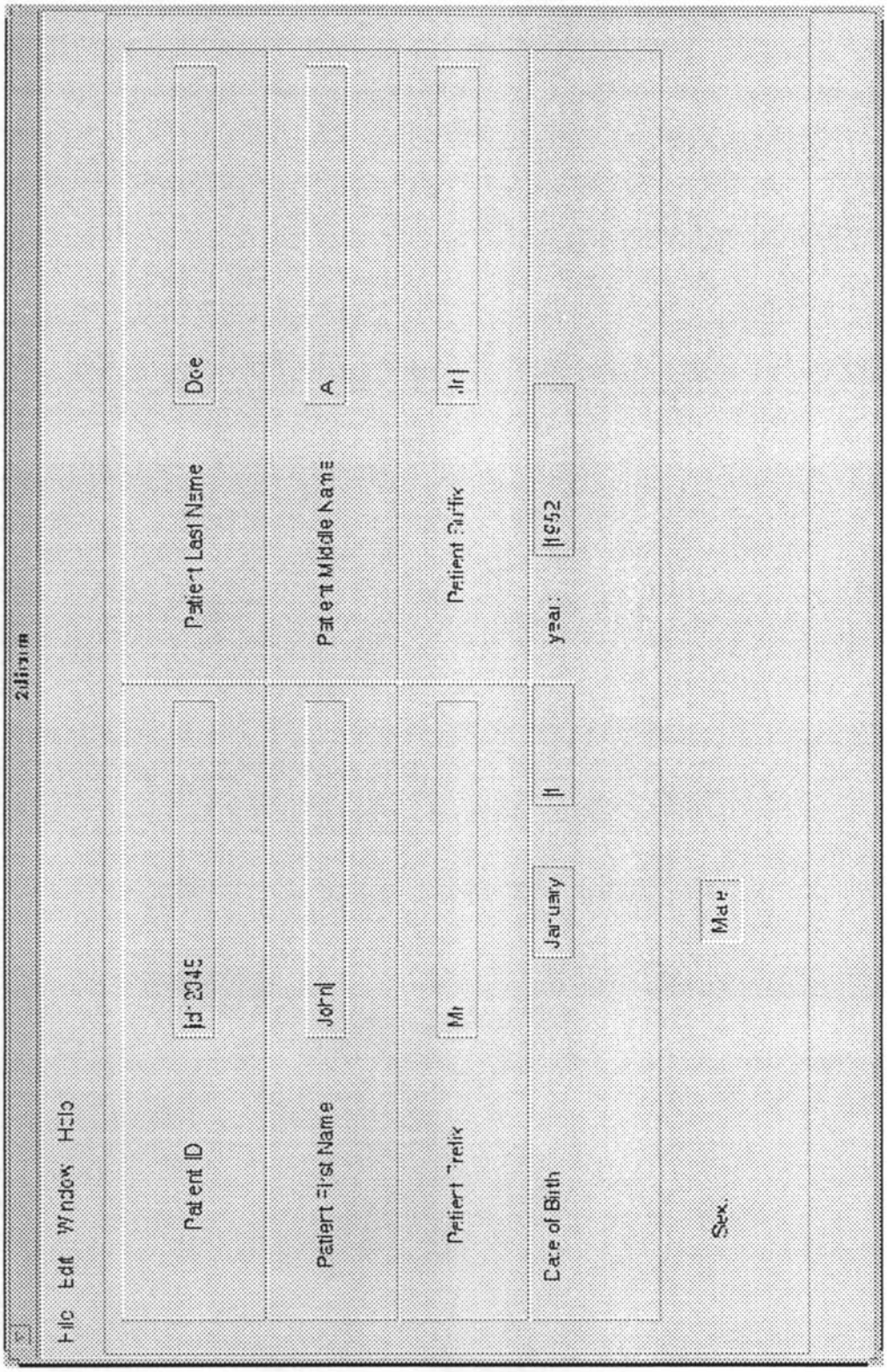
2Dicom developed as a part of my undergraduate research involving medical image format conversion. I chose to work with DICOM over the other available formats (namely, the three proprietary formats and another standard, Interfile) because of its well-defined header values and flexibility in which values are necessary in the files (thus saving disk space by leaving out unnecessary values). The standard is well-established within the medical imaging community and should serve as the basis for a number of new programs in the future. A C++ implementation of the program was planned during the summer but was later scrapped due to concerns over insufficient language standardization and toolkit compatibility. An ASCII terminal version of the program which converts ADAC images to an intermediate data structure was written by Sam Burgiss earlier this fall. The SUIT toolkit was selected for programming the GUI because of my previous experience with it. The current version is intended to demonstrate the program interface for presentation in November 1994.

Future Developments

The "Save As" option and Genesis compatibility are high priorities for future development. The file browser is currently based on a SUIT widget and will be modified to allow images from multiple source files to be stored as one study in the output DICOM file. I would also like to rewrite the GUI using another toolkit, such as Motif or OpenLook, which will improve the precision of placement of items within the window relative to the locations of other related items (and eliminate one of the bugs mentioned earlier). The DICOM values currently included represent a bare minimum of those values which are included within the standard, so future releases will hopefully incorporate other optional header values. An option permitting batch conversions at a later time would reduce the load on the system during peak hours. A feature permitting

conversion to some other popular image format (like GIF or TIFF) would make it easier to incorporate these images into documents and presentations created with common Macintosh or PC software. Finally, an option to view images before they are saved to disk could prevent some unnecessary conversions. If you have any other suggestions, you may reach me by e-mail at cruser@scanner.hosp.utk.edu.

Program Workspace



I must take this opportunity to thank Dr. Hedley W. Bond, Ph.D., for his support in this project. When I first approached him with the notion of performing undergraduate research in the Image Processing Lab, he readily accepted the idea, made the facilities of the lab available to me, provided knowledge of the need for medical image file conversion tools, and patiently answered my questions whenever I poked my head into his office or filled his electronic mailbox. Two other members of my committee whom I wish to thank for their time are Dr. Gary T. Smith, M.D., and Dr. Brad T. Vander Zanden, Ph.D., whose ideas were important in developing the project goals and and providing evaluation.

Two other individuals from the Image Processing Lab who provided valuble assistance are Mr. Sam Burgiss, who aided in code for ADAC image conversion, and Dr. Kent Hutson, M.D., who shared an interest in the conversion of medical image file formats. Dr. Ed Glass, M.D., graciously provided the original pet2peg code and thus the foundation for the Pet2PegTool program.

I also wish to thank Dr. Jack Reese, Ph.D., who provided advice and acted as liason between the UT campus and the hospital. Without the flexability of the College Scholars Program or his personal assistance this project would not have been possible.

Finally, I wish to thank my fiancee, Kristin Sullivan, who provided exceptional editing skills, encouragement, emotional support, and patience while I have worked to complete this project and apply to medical schools.

Clunie, D. *Medical Image FAQ* [Online]. September 1994. Available FTP: rtfm.mit.edu Directory: pub/usenet/news.answers/medical-image-faq Files: part1 part2 part3

Conway, J. & The University of Virginia. *The SUIT Version 2.3 Reference Manual* [Online]. August 1992. Available FTP: uvacs.cs.virginia.edu Directory: /pub/suit/distribution/sparc File: sparc.tar.Z

Mallinckrodt Institute of Radiology. *Central Test Node Software* [Online]. July 1993. Available FTP: ftp.xray.hmc.psu.edu Directory: dicom_software/Mallinckrodt File: V1.2.1.tar.Z

National Electrical Manufacturers Association (NEMA). *DICOM - Part 1: Introduction and Overview* [Online]. March 1992. Available FTP: ftp.xray.hmc.psu.edu Directory: /dicom_docs/dicom_3.0/word_hqx File: Part_01_Final_Text.hqx.Z

National Electrical Manufacturers Association (NEMA). *DICOM - Part 3: Information Object Definitions* [Online]. October 1993. Available FTP: ftp.xray.hmc.psu.edu Directory: /dicom_docs/dicom_3.0/word_hqx File: Part_03_Final_Text.hqx.Z

National Electrical Manufacturers Association (NEMA). *DICOM - Part 4: Service Class Specifications* [Online]. October 1993. Available FTP: ftp.xray.hmc.psu.edu Directory: /dicom_docs/dicom_3.0/word_hqx File: Part_04_Final_Text.hqx.Z

National Electrical Manufacturers Association (NEMA). *DICOM - Part 5: Data Structures and Encoding* [Online]. October 1993. Available FTP: ftp.xray.hmc.psu.edu Directory: /dicom_docs/dicom_3.0/word_hqx File: Part_05_Final_Text.hqx.Z

National Electrical Manufacturers Association (NEMA). *DICOM - Part 6: Data Dictionary* [Online]. October 1993. Available FTP: ftp.xray.hmc.psu.edu Directory: /dicom_docs/dicom_3.0/word_hqx File: Part_06_Final_Text.hqx.Z

National Electrical Manufacturers Association (NEMA). *DICOM - Part 10: Media Storage and File Format* [Online]. December 1993. Available FTP: ftp.xray.hmc.psu.edu Directory: /dicom_docs/drafts File: Part_10_Final_Draft_Let._Ball..hqx

Sun Microsystems, Inc. *SunView Programmer's Guide.* Sun Microsystems, Inc., 1990.

Todd-Pokropek, A., T. Cradduck, & F. DeConinck. "A file format for the exchange of nuclear medicine image data: a specification of Interfile version 3.3." *Nuclear Medicine Communications.* 13 (1992): 673-699.

University Hospital of Geneva. *PAPYRUS format specifications ver. 2.3.* Unite D'Imagerie Numerique/Hôpital Cantonal Universitaire de Geneve, 1991.

various authors. *DICOM* [Discussion, Online]. Available e-mail: USENET Newsgroup: comp.protocols.dicom

Source Code for

Pet2PegTool and 2Dicom

A Senior Research Project

Presented for the

Bachelor of Arts Degree

in College Scholars

The University of Tennessee, Knoxville

Marc Cruser

November 21, 1994

```
# Makefile for SunView pet2pegtool program
# by Marc Cruser, derived from pet2peg by Ed Glass
# 21 Nov 1994 UTMCK IPL

# change LIB_DIR to match the directory containing lib_toolkit.a
LIB_DIR = .

# change INC_DIR to match the directory containing the include
# files distributed with the ADAC toolkit
INC_DIR = .

PROG =    pet2pegtool
OBJS =    svfunctions.o pet2peg.o fill_peg_info.o matcount.o \
          validate_infile.o matrix.o error.o \
          write_matrix_data.o sortlist.o square_img.o
CFLAGS = -O  -I$(INC_DIR) $(SUNVIEW_FLAGS)
SUNVIEW_FLAGS =  -lsuntool -lsunwindow -lpixrect
ADAC_LIB = $(LIB_DIR)/lib_toolkit.a

pet2pegtool : $(OBJS)
        $(CC) $(CFLAGS) $(OBJS) $(ADAC_LIB) -lm -o $@
#       rm *.o
```

README

Purpose of Program

Pet2PegTool is a utility for use with medical imaging systems which work with both CTI and ADAC medical images. The program converts image files created on a CTI imaging device to a format readable on ADAC Pegasys workstations. Benefits include consolidation of sets of related images of different modalities and application of ADAC diagnosis software to CTI PET images.

Program Requirements

Pet2PegTool runs on ADAC Pegasys workstations running SunView. The program uses files included with the ADAC Programmer s Toolkit.

Installation

To extract the files from a 3.5" disk into the current directory, type tar xf /dev/rfd0. To extract the files from a tar file, type tar xf pet2pegtool.tar. This will create a new directory, pet2peg, that contains the program files. In this directory you will need to edit Makefile so it knows where to find the toolkit files on your system. Type make to create the executable file if it does not already exist. The cc compiler must be on your path in order to compile the program. Type pet2pegtool to run the program.

Running the Program

Opening Files

When Pet2PegTool begins, a file selection window appears on the screen. Type in the preferred directory and file filter, then press the "List" button to view the list of files. To select a file to convert, triple-click on one of the files listed in this box. If you wish to convert this file, click on "Prepare Conversion." You may also end the program by pressing the "Quit" button in this window.

Conversion Options Panel

If the selected file is in a recognizable CTI format, an option panel will appear over the file selection window. At the top of the panel is a message bar for communicating the current status of the conversion. Below this are the names of the source and output files. The default output filename is that of the source file with the extension ".adac." (If the file is immediately sent to the ADAC database, the output filename is irrelevant since it is removed during this process.) Specify whether the patient was scanned head-first or feet-first by clicking the value beside the "Orientation" label. If the source file contains static series, you may select a range of planes to convert. If it contains dynamic or gated images, options are available for converting a range of planes from a selected frame and for a range of frames from a specified plane. Check the box beside the preferred option if this choice is available. Next, you may select to reslice the image, an option recommended for transverse images. If the converted images are to be immediately added to the ADAC database, make sure this option has been selected. Below these options are a number of header fields which may be modified if needed. When the conversion options have been set, click on the "Convert" button to create the ADAC image file.

Known Bugs

This program will not work effectively with whole body projections. While these images can be processed, the result is typically not useful and somehow manages to include pieces of other images with its data set. Some images appear to be off-center after conversion. A "Notifier Error" message results when the option panel appears but does not seem to affect functionality. On some machines, the plane and frame selection fields appear to be askew.

Program History

The code for the original pet2peg program was written and modified by Ed Glass from 1989 to 1992. The SunView interface, enhanced dynamic image capabilities, and other functions (including some intended for whole body projections) were added by Marc Cruser as undergraduate research in November 1994.

Future Developments

I hope to find out what causes some of the images to appear off-center, and just what causes the whole body projections to become scrambled and mixed with other images. If you have any other suggestions, you may reach me by e-mail at cruser@scanner.hosp.utk.edu.

```
/* error.c - Error-handling routines for pet2pegtool
             included in all .c files.                */
#define SUNVIEW 1
#include "includes.h"

extern Frame main_frame;
extern Panel_item msg_item;

void Error(errmsg)
char *errmsg;
{
   Event alert_event;

#ifdef DEBUG
   fprintf(stderr, "Error: %s\n", errmsg);
#endif
   alert_prompt(main_frame, alert_event,
                ALERT_MESSAGE_STRINGS, "ERROR!", errmsg, 0,
                ALERT_BUTTON_NO, "Cancel", 0);
}

void Warning(warnmsg)
char *warnmsg;
{
   Event alert_event;

#ifdef DEBUG
   fprintf(stderr, "Warning: %s\n", warnmsg);
#endif
   alert_prompt(main_frame, alert_event,
                ALERT_MESSAGE_STRINGS, "Warning", warnmsg, 0,
                ALERT_BUTTON_NO, "OK",
                ALERT_NO_BEEPING, 1, 0);
}

void PopMessage(msg)
char *msg;
{
   Event alert_event;

#ifdef DEBUG
   fprintf(stderr, "%s\n", msg);
#endif
   alert_prompt(main_frame, alert_event,
      ALERT_MESSAGE_STRINGS, msg, 0,
      ALERT_BUTTON_NO, "OK",
      ALERT_NO_BEEPING, 1, 0);
}

void Message(msg)
char *msg;
{

#ifdef DEBUG
   fprintf(stdout, "%s\n", msg);
   fflush(stdout);
#endif
   panel_set(msg_item, PANEL_LABEL_STRING, msg, 0);

}
```

```
/*****************************************************************
*  int fill_peg_info (&mheader, &sheader, &demo, &extra, &data)
*  purpose:
*     fills pegasys demographics, extras, and data info.
*     fields from pet fields in Main_header and Image_subheader
*
*  returns: 0=ok
*           -1=error
*
*  24 Nov 92   written by Ed Glass
*  21 Nov 94   modified for SunView by Marc Cruser
*****************************************************************/

#define CTI 1
#define ADAC 1
#include "includes.h"

int fill_peg_info (mhptr, shptr, demoptr, extraptr, dataptr)
/* CTI pointers */
Main_header     *mhptr;
Image_subheader *shptr;
/* ADAC pointers */
DEMO_OBJ_PTR    demoptr;
EXTRA_OBJ_PTR   extraptr;
IP_OBJ_PTR      dataptr;
{
   int lname;
   word temp;

/* Fill IP_OBJ */
   dataptr->ip_xdim = (COORDINATE) shptr->dimension_1;
   dataptr->ip_ydim = (COORDINATE) shptr->dimension_2;
      /* CTI i*2 to ADAC unsigned short */
#ifdef DEBUG
   printf("xdim=%d   ydim=%d\n", dataptr->ip_xdim, dataptr->ip_ydim);
#endif

   dataptr->ip_sets = (SHORT) mhptr->num_gates;
      /* # sets of x,y,z - meaning open to question ? */
   dataptr->ip_tcol = 1;         /* true color flag=1 for now          */
   dataptr->ip_smin = 0.0;       /* minimum pixel value                */
   dataptr->ip_sfac = 1.0;       /* may change in pet2peg with dynamic data */
      /* scaling factor, guessed from other SPECTs ? */
   if (mhptr->file_type != 2) {  /* must be image file (2)             */
      Error("Source file is not an image file");
      return -1;
   }
   temp = shptr->data_type;  /* Must be byte (1) or int*2 (2) )        */
#ifdef DEBUG
   printf("fill_peg_info shptr->dtype = %i\n", temp);
#endif
   if (temp != DTYPE_BYTES && temp != DTYPE_I2) {
      Error("Source data must be byte or int*2.");
      return -1;
   }
   dataptr->ip_dat = (SHORT) (temp*8); /* Must be (SHORT) 8 or 16       */
   /* Still need to calculate # of planes in output set (ip_zdim), but
      note the num_bed_pos in Main_header is one less than counted number
      acquired and you simply can't rely on main header for the actual
      number of images. Use mat_list(). So, fill ip_zdim in main().     */
   /* Unfilled values:
      ip_zdim = # planes, calculate in pet2peg using matlist[]
      ip_smax = maximum pixel value, calculate in pet2peg
      ip_stot = total count of set(s), calculate in pet2peg
      ip_morg = memory origin, calculate in pet2peg                     */
#ifdef DEBUG
   printf("IP_OBJ filled\n");
#endif

/* Fill DEMO_OBJ */
   strncpy(demoptr->dm_name, mhptr->patient_name, NAME_LEN);
      /* patient name - ADAC=20, CTI=31 chars */
   lname = strlen(demoptr->dm_name);
   if (lname == 0 || lname == strspn(demoptr->dm_name, " ")) {
      strncpy(demoptr->dm_name, mhptr->original_file_name, 7);
      demoptr->dm_name[7] = 0;
   }
   strncpy(demoptr->dm_pid, mhptr->patient_id, PID_LEN);
      /* patient id - ADAC=11, CTI=15 chars */
   demoptr->dm_sex[0] = mhptr->patient_sex;
   demoptr->dm_sex[1] = '\0';
      /* patient sex - ADAC=1, CTI=1 char */
   demoptr->dm_age = (SHORT) atoi(mhptr->patient_age);
      /* patient age - ADAC=short, CTI=10 chars */
   demoptr->dm_high = (SHORT) atoi(mhptr->patient_height);
      /* patient height - ADAC=short, CTI=10 chars */
   demoptr->dm_wigh = (SHORT) atoi(mhptr->patient_weight);
      /* patient weight - ADAC=short, CTI=10 chars */
   sprintf(demoptr->dm_date,  "%04d%02d%02d",   mhptr->scan_start_year,
         mhptr->scan_start_month, mhptr->scan_start_day);
      /* scan date - ADAC=8 char, CTI=int's */
   sprintf(demoptr->dm_exmk, "%s", "92b500");
      /* unique exam key - special instruction for now ? (A.B.) */
   strncpy(demoptr->dm_proc, mhptr->study_description, 31);
      /* procedure - ADAC=36, CTI=32 chars */
   strncpy(demoptr->dm_rdr, mhptr->physician_name, DR_LEN);
      /* radiologist - ADAC=20, CTI=31 */
   sprintf(demoptr->dm_imag, "%s", "PT");
      /* modality - use "PT" */
   strncpy(demoptr->dm_hid, mhptr->facility_name, HID_LEN);
      /* hospital id - ADAC=20, CTI=20 chars */
   sprintf(demoptr->dm_atim, "%02d:%02d:%02d", mhptr->scan_start_hour,
         mhptr->scan_start_minute, mhptr->scan_start_second);
      /* acquisition start time - ADAC= 6 chars, CTI=int's */
   if (mhptr->num_gates > 1)
      sprintf(demoptr->dm_otyp, "%s", "GE"); /* GE for gated ECT */
   else
      sprintf(demoptr->dm_otyp, "%s", "DP"); /* DP for static ECT */
   sprintf(demoptr->dm_objk,  "%s", "00");
      /* unique object key, try "00" ? (A.B.) */
   strncpy(demoptr->dm_iid, mhptr->study_description, IID_LEN);
      /* image view id - ADAC=16, CTI=31 chars */
   /* Unfilled values:
      dm_upat = unique patient key, omit
      dm_time = dose administ. time, omit
      dm_adr  = attending physician, omit
      dm_ncrv = normaliz. curve file, omit
      dm_hcrv = histogram curve file, omit
      dm_parf = assoc. parent file, omit (A.B.)
      dm_size = file size in bytes, omit (A.B.)     */
#ifdef DEBUG
   printf("DEMO filled\n");
#endif

/* Fill EXTRA_OBJ */
   sprintf(extraptr->ex_devi, "%s%3d", "PET ", mhptr->system_type);
      /* imaging device name - SWITCH TO CASE STATEMENT */
   strncpy(extraptr->ex_devn, mhptr->node_id, 9);
      /* device serial num - ADAC=12, CTI=9 chars */
   sprintf(extraptr->ex_swv, "%s%3d", "PET V", mhptr->sw_version);
      /* software version - SWITCH TO CASE STATEMENT */
   strncpy(extraptr->ex_rad1, mhptr->radiopharmaceutical, RADPH_LEN);
      /* radiopharmaceutical - ADAC=16, CTI=32 chars */
   sprintf(extraptr->ex_isom,"%s","S");
```

```
   /* isotope img mode - set to "S" (A.B.) */
  sprintf(extraptr->ex_hifi, "%s", "H");
   /* head/feet-in orient. - use "H" */
  extraptr->ex_ew1c = (SHORT) 511;
   /* energy window center - positron energy=511 */
  extraptr->ex_ew1w = (SHORT)(mhptr->upr_true_thres - mhptr->lwr_true_thres);
   /* energy window width */
  extraptr->ex_spat = 1.0;
   /* spatial resolution - use 1.0 as of 12/14/92 (A.B.) */
  extraptr->ex_zoom = (FLOAT) shptr->recon_scale;
   /* zoom factor - same meaning for both     */
  extraptr->ex_slic = 10.0;
   /* slice thickness - use 10.0 as of 12/14/92 ? (A.B.) */
  extraptr->ex_vfr.v_whoami = 1;
   /* phase of study = 1 */
  extraptr->ex_tfr=(ULONG)
   (((float)((1+mhptr->num_bed_pos)*shptr->frame_duration))/64.0);
   /* time per frame - estimate assuming 64 stops */
  extraptr->ex_ttim = (ULONG)
   (((ULONG)(mhptr->num_bed_pos + 1))*shptr->frame_duration);
   /* total acquisition time */
  extraptr->ex_rint = (SHORT) 1000;
   /* R-R interval time - arbitrary value - is CTI in msec? */
  sprintf(extraptr->ex_rota, "%s", "+");
   /* dir of rotation - arbitrary value = clockwise */
  extraptr->ex_sang = 360;
   /* starting angle - default=360 */
  extraptr->ex_ndeg=360;
   /* degrees of rotation - assume 360 */
  temp = shptr->filter_code;
  if (temp >= 0 && temp <= 6)
   sprintf(extraptr->ex_ftyp, "%s", filters[temp]);
   /* filter type - value is 0-6 - filters[], code in scan_types.h ? */
  extraptr->ex_arot = 0;
   /* axis of rotation correction - default=0 */
  extraptr->ex_fcut = shptr->filter_params[0];
   /* filter cutoff frequency - should be item[0] */
  /* Unfilled values:
   ex_coll = collimator, omit
   ex_rad2 = 2nd radiopharmaceutical, omit
   ex_pori = patient orientation, omit (A.B.)
   ex_ucor = uniformity correction filename, omit
   ex_dos1 = dosage #1, omit
   ex_dos2 = dosage #2, omit
   ex_ew2c = energy window #2 center, omit
   ex_ew2w = energy window #2 width, omit
   ex_ew3c = energy window #3 center, omit
   ex_ew2w = energy window #3 width, omit
   ex_ew4c = energy window #4 center, omit
   ex_ew4w = energy window #4 width, omit
   ex_vfr.v_grp = list of assoc VFR's OBJK's, omit
   ex_vfr.v_dummy = unused
   ex_vfr.vnum  = # of VFR studies, omit
   ex_unused4[] = unused array
   ex_rrlo = R-R low tolerance, omit - no CTI data
   ex_rrhi = R-R high tolerance, omit - no CTI data
   ex_tolr = % of cycle imaged, omit - no CTI data
   ex_acyc = % of cycles accepted, omit - no CTI data
   ex_rcyc = % of cycles rejected, omit - no CTI data
   ex_edi  = end diastolic frame, omit - no CTI data
   ex_esy  = end systolic frame, omit - no CTI data
   ex_ejfr = approx EF ?, omit - no CTI data
   ex_etyp = continuous or step & shoot, omit
   ex_lims = limited recon start frame, omit
   ex_mmap = custom color map, omit
   ex_wwid = upper win gray shade, omit
   ex_wlvl = lower win gray shade, omit
   ex_cmap = assoc color map, omit
   ex_mimg = manipulated image flag, omit
   ex_rtyp = reconstruction type, omit
   ex_reoa = reorientation asimuth, omit
   ex_reoe = reorientation elev, omit
   ex_ford = filter order, omit - not specified in CTI
   ex_acof = atten coeff, omit - a matrix in CTI
   ex_rslt = program specific results area, omit        */

#ifdef DEBUG
   printf("IP_EXTRA filled\n");
#endif

   return 0;
}
```

```
/*************************************************************************
*   int Matcount.c
*   routine to determine number of image matrices in a study
*   that have a given frame and gate. Reads across bed positions and planes.
*   This returns the number of such matrix numbers found as int return value.
*   It also returns the int list mlist[], containing the matrix numbers
*   of the selected planes in ascending anatomic order or order of
*   descending bed position, then decreasing plane number within any
*   bed position - i.e. mlist[] should come out from feet -> head,
*   for patient placed head first in scanner.
*   Written by Ed Glass
*   Edited for SunView by Marc Cruser, 21 sep 94
*************************************************************************/
#define CTI 1
#include "includes.h"

int Matcount(fptr,frame,gate,mlist,lmax)
FILE *fptr;
int frame, gate, lmax, mlist[];

{
   int fra, bed2, blk, num_entry, i, err, pla, gat, dtyp;
   int isold, plamax, lastbed, num_beds, j;
   int bedlist[17];
   int nfree, nxtblk, nused, matnum;
   int dirbufr[MatBLKSIZE/4];
   int *low_ptr,*high_ptr;
   long offset;
   int swaw(),fseek(), fread();
   char bytebufr[MatBLKSIZE];
   void mat_sort();

   blk = MatFirstDirBlk;
   num_entry = 0;
   num_beds=0;
   plamax=0;   /* for highest plane number encountered   */
   lastbed=-1;
   bedlist[0]=-1;
   while(1) {
      offset=((long)(blk-1))*((long)MatBLKSIZE);
      if(fseek( fptr, offset, 0))   /* 0=from beginning */
      {
         Error("Error in attempt to fseek() file");
         return (-1);
       }
       err = fread( bytebufr, 1, MatBLKSIZE, fptr);
       if (err != MatBLKSIZE)
       {
          Error("matcount() Error in attempt to read() file");
          return (-1);
        }
        swab( bytebufr, dirbufr, MatBLKSIZE);
        swaw( dirbufr, dirbufr, MatBLKSIZE/2);
        /* nfree  = dirbufr[0]; Unused */
        nxtblk = dirbufr[1];
        /* nused  = dirbufr[3]; Unused */
        for (i=4; i<MatBLKSIZE/4; i+=4)
        {
            matnum = dirbufr[i];

/********* beds start at 0.   but planes, frames, and gates start at 1 ****************
/
/*********************below is layout of a matrix number*********************************
/
/*|dtyp|<-----gate------>|<----plane------------>|<---bed--->|<------frame----------->|*
/
/*31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0*
/
/*************** above are bit positions in a matrix number integer ********************/

            fra=(matnum&0xfff);      /* bits  0-11  (1st 12 bits)   */
            bed2=((matnum&0xf000)>>12);    /* bits 12-15    */
            pla=((matnum&0xff0000)>>16);    /* bits 16-23   */
            gat=((matnum&0x3f000000)>>24);   /* bits 24-29   */
            dtyp=((matnum&0xc0000000)>>30);   /* bits 30,31   */
#ifdef DEBUG
printf("frame %d   gate %d    bed %d    plane %d datyp %d\n",fra,gat,bed2,pla,dtyp);
#endif
            if (matnum && fra==frame && gat==gate)
            {
               if(num_entry < lmax)
                  (
                     mlist[num_entry]=matnum;   /* save it - a good one */
                     if(pla > plamax) plamax=pla;
                     num_entry++;
                     if(bed2 != lastbed)
                     {
                        for(isold=j=0;j<num_beds;j++) /* search list */
                        (
                           if(bed2==bedlist[j])
                              isold=1; /* exit if bed known */
                         )
                         if(!isold)      /* i.e. if new   */
                         (
                            bedlist[num_beds]=bed2;
                            num_beds++;
                          )
                       ) /* end if(bed2...)   */
                       lastbed=bed2;
                    ) /* end if(num_entry...)   */
                    else
                    (
                    Error("matcount(): list size exceeded");
                    return num_entry;
                  )
          )   /* end if(matnum...)   */
       /*   else
          (
             PopWarning("unwanted matnum, bed: %d   frame: %d    or gate: %d found",bed2,f
ra,gat);
          ) */
       )
       blk = nxtblk;
#ifdef DEBUG
      printf("the next dirblock is: %d\n",nxtblk);
#endif
      if (blk == MatFirstDirBlk) break;
   )   /* end while(1)   */
#ifdef DEBUG
printf("final accepted #beds = %d,  #planes = %d,  and largest plane # = %d\n",num_beds,n
um_entry,plamax);
#endif
/******* sort in ascending anatomic order or in order of descending bed_pos numbers, **/
/****** and then secondarily by descending order of plane number for any bed position */
   low_ptr=&mlist[0];
   high_ptr=&mlist[num_entry-1];
   mat_sort(low_ptr,high_ptr);
   return (num_entry);
}


/*************************************************************************
*   function mat_sort()
*   purpose:
```

```
*      sort an array of matnum's, placing them in descending order, in place.
*      sorts by bed position first, then plane number within any bed position.
*
*      ********************************
*      *** POINTER VERSION ************
*      ********************************
*   method:
*   use group splitting method developed by C. Hoare where the array is successively
*   split into two subgroups < and >= a pivot element of each group. The function
*   recursively calls itself until subgroups of size one element are achieved.
*
*   input:
*      low_ptr=pointer to bottom, top of integer array
*      high_ptr=pointer to top of integer array
*   output:
*      The array pointed to is sorted in ascending order, in place.
*
*   usage:
*    int *low_ptr,*high_ptr;
*         .
*         .
*         .
*    void quick_sort(low_ptr,high_ptr);
*
*   This function utilizes function partition(). See below.
*
* history:
*   5 October 1991   Ed Glass   quick_sort() - adapted from text for assignment #2
*   12 DECEMBER 1992 *   *      create mat_sort() from quick_sort()
*
*****************************************************************************/
void mat_sort(low_ptr,high_ptr)
int *low_ptr,
    *high_ptr;   /* low_ptr & high_ptr refer to address, not matnum   */
(
   int *piv_ptr;
   int *mpartition();  /* does the partitioning, returning integer pointer   */

   if(low_ptr < high_ptr) /* work if there's work to be done   */
   {
      piv_ptr=mpartition(low_ptr,high_ptr);
      mat_sort(low_ptr,piv_ptr-1);   /* sort lower set   */
      mat_sort(piv_ptr,high_ptr);   /* sort high set   */
   }
)
/*****************************************************************************
*   (int *) mpartition(low_ptr,high_ptr)
*
*   Goes with mat_sort(), q.v.
*
*   partition array of matnum's in descending order considering bed number
*   with highest priority, plane number second.
*   with top & bottom addresses pointed to by low_ptr & high_ptr
*   into two subsets, one < a pivot value and another >= that pivot value.
*      bed2=((matnum&0xf000)>>12);    * bits 12-15  = 4 bits *
*      pla=((matnum&0xff0000)>>16);    * bits 16-23  = 8 bits *
*
*   input:
*      low_ptr, high_ptr = pointers to integers in array of interest
*   output:
*      the bottom of upper partition is returned as pointer to integer
*
* history:
*   6 October 1991   Edwin Glass   written - from text for class assignment #2
*   12 Dec    1992  Edwin Glass   adapted partition() from class to mpartition()
*
*****************************************************************************/
```

```
/* #define   SWAP(type,x,y)   (type   temp=(x);(x)=(y),(y)=temp;)   swap macro   */
/* the macro below gives bed position highest priority ****************************/
/* by shifting old bits 12-15 to positions 24-27, 12 bits to left   *****************/
#define      VAL(i)   (((i)&0xff0000)+(((i)&0xf000)<<12))

int   *mpartition(low_ptr,high_ptr)
int   *low_ptr,*high_ptr;
(
   int l,h,vl,vh,vpivot,temp;
   int pivot = *(low_ptr + (high_ptr-low_ptr)/2); /* an integer value for comparison */
   vpivot=VAL(pivot);
   while(low_ptr <= high_ptr)   /* cont. until pointers cross */
   {
      l=*low_ptr; vl=VAL(l);
      h=*high_ptr;vh=VAL(h);
      while(vl > vpivot)   /* now actually compare values in array with pivot value */
      {
         low_ptr++;      /* and keep moving fwd. in array if belongs in high set   */
         l=*low_ptr; vl=VAL(l);
      }
      while(vh < vpivot)   /* check upper set as well */
      {
         high_ptr--;      /*   and move it downward when values belong in low set */
         h=*high_ptr;vh=VAL(h);
      }
   /********** if pointers have crossed (lowptr>highptr) you are done. Else swap ****/
      if(low_ptr <= high_ptr) /* if to here you have either crossed pointers or have */
/*    SWAP(int,*low_ptr++,*high_ptr--) legitimate values to exchange        */
      {
         temp=*low_ptr;
         *low_ptr++=*high_ptr;
         *high_ptr--=temp;
      }
   )
   return low_ptr;
)
```

```
/*****************************************************************
*
*  Convert matrix files to Pegasys files. Allow reslicing of planes
*  in z-direction to put PET images in SPECT-like format with z
*  spacing same as xy pixel size. Only one frame in PET images is
*  allowed in converted Pegasys files.
*
*  Usage: pet2peg infile outfile(null=list scans) transverse(0/1)
*         feet/head(0/1)first start_plane end_plane
*
*  Modifications:
*      1 Feb 1989: EMP : Changed the meaning of the Z-axis-reslice
*                        parameter.
*                  0 = no reslice.
*                  1 = calculate reslice factor from image.
*           For new section of code added, used new matrix
*           utilities written for the Sun environment.
*     21 Nov 1994  MCC : adapted for integration into SunView
*                  environment; added frame/gate selection
*                  capability; pads images to square dimensions.
*
*****************************************************************/
#define CTI 1
#define ADAC 1
#include "includes.h"

int stat_pet2peg (infilep, mheader, infile, outfile, transverse,
                  headfirst, spla, epla, frame, gate)
char *infile, *outfile;
int  transverse, headfirst, spla, epla, frame, gate;
FILE *infilep;
Main_header mheader;
{
    /* CTI variables */
    Image_subheader sheader;
    struct MatDir entry;

    /* ADAC variables */
    UBYTE     *inpmat_byptr;
    IP_OBJ    data, *data_ptr;
    DEMO_OBJ  demo;
    EXTRA_OBJ extra;
    UBYTE     *dest_ub, *dest_ubsv;
              /* unsigned char ptr. for 8 bit data, max 255 */
    UBYTE     *tmpmat_byptr, *mf_byptr, *mb_byptr, *src_ub;
    UBYTE     bmax, btemp;
    SHORT     *dest_ss, *dest_sssv;
              /* pointer for signed 16-bit data, max 32767 */
    SHORT     *mbptr, *mfptr, *src_ss;
              /* pointers to planes when interpolating along */
    SHORT     smax, stemp;
    FCNTL     outfile_struct;
              /* control structure for mapped file */
    ST_ERROR_HDR  error;
    ST_ERROR_HDR *err_ptr; /* for adac error structures */

    /* variables for pet2peg */
    float  qsv[NIMAGEMAX], scalef, zstep, ffwd, fback, pos;
    short int tmpmat[SIZE], inpmat[SIZE], matf[SIZE], matb[SIZE];
    int    row, col, i, k, nwrit, iback, ifwd, ilast, jump, jump2;
           /* a variety of counters */
    int    nimoutput, numpla, niminput;
           /* various numbers of images */
    long   matno, mheight, mwidth, size, newsize;
    float  fwdscale, backscale, dmax, dmin, dtot;
    int    bed, plane;
    int    sgates, egates;
    int    mlist[MLSIZE];  /* for list of matrix numbers */
    float  fb2, ff2, denom, denomax, ztoxratio, atof();
    int    err;  /* an error indication returned (0 = ok) */
    int    c0, dsize, exsize, iwidth, halfwidth, height, Matcount();
    int    reorder, reslice, status;
    int    squareflag;
    int    newwidth, newhalfwidth, newheight;
    char   errmsg[100];
    float  vaxftosunf();

    /* Determine number of images (total) in the input file. */
    /* numpla = total images selected for fixed frame & gate */
    numpla = Matcount(infilep, frame, gate, mlist, MLSIZE);
       /* mlist[] is ordered list: feet to head             */
    niminput = numpla;
       /* initialize niminput for now. may be changed below */

    /* allow only first gate of gated set, as adac uses ip_zdim for
       gating time dimension and then uses ip_sets for the number of
       planes. This confusion arose because previously, ip_zdim in
       planar gated was used for the sequntial images throughout the
       cardiac cycle. So, for gated spect, ip_zdim was kept as the
       gating variable, ip_sets for planes even though ip_zdim is used
       for the number of planes in nongated spect image sets.        */
    /* In a nutshell: ip_zdim = # planes in nongated,
                              = time dimension in gated.
                   ip_sets = # planes in gated.        */

    /* Initialize/clear ADAC data structs.  */
    dsize = sizeof(DEMO_OBJ);
    exsize = sizeof(EXTRA_OBJ);
    c0 = 0;
    memset(&demo, c0, dsize);   /* zero demo */
    memset(&extra, c0, exsize); /* zero extra */

    /* Initialize ADAC error structures. */
    err_ptr =&error;
    err_init_msg(err_ptr);
       /* initialize adac file ctl & error reporting */
    util_init_fcntl(&outfile_struct);

   /***          32-bit matrix number bit-position layout.          ***/
   /*|typ|-- gate ---|---- plane ----|- bed -|------- frame ---------|
     3 3 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
     1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 */

    /* Set matrix number. */
    plane = 1;
    bed = 0;
    matno = mat_numcod(frame, plane, gate, 0, bed);
#ifdef DEBUG
    printf("frame = %i, plane = %i, gate = %i, data_type = %i, bed = %i\n",
           frame, plane, gate, mheader.data_type, bed);
#endif
    /* use it to fill struct MatDir entry with start & end blocks  */
    status = mat_lookup(infilep, matno, &entry);

    /* Determine pixel size from subheader "sheader" as finished
       structure in sun2 format                                    */
    mat_read_image_subheader(infilep, entry.strtblk, &sheader);
       /* use "sheader" now for subh. - NOTE: needs to be set for each. */
    mwidth = sheader.dimension_1;
    mheight = sheader.dimension_2;

    /* Fill ADAC header structures as best possible. */

    data_ptr = &data;
```

```
status = fill_peg_info(&mheader, &sheader, &demo, &extra, &data);
/* note ip_zdim, ip_sets, ip_smax, ip_stot, ip_sfac, ex_slic still not filled */
if (status != 0) {
   fclose(infilep);
   return(-1);
}
/* data_type. allow only (2=SHORT=short int output or
   =1=UBYTE=unsigned char output)                          */
if ((sheader.data_type != DTYPE_BYTES) &&
    (sheader.data_type != DTYPE_I2)) {
   Error("Data in source file must be short int or unsigned char.");
   fclose(infilep);
   return(-1);
}

/* Set ratio between SLICE THICKNESS (?) and pixel (xy) size. */
ztoxratio = mheader.plane_separation / sheader.pixel_size;

/* If transverse: (1.) label this in view id,
                  (2.) (may)reorder output from inf. to superior,
                  (3.) reslice                                    */

/* Setting flags for transverse image reslicing.  */
if (transverse == 0) {    /* if NOT transverse set */
   /* No z-axis-reslice.  Only convert the original slices. */
   fflush(stdout);
   reslice=0;     /* do not reslice                          */
   ztoxratio=1.0; /* force this to avoid frame interpolation */
}
else { /* if IS transverse image set, reslice               */
   reslice = 1;
   sprintf(demo.dm_iid, "%s", "PET TRANSVERSE"); /* formerly just a prefix */
   /* determine the appropriate reslice factor from the matrix file  */
} /* end else for transverse settings */
reorder = (headfirst==1)? 1 : 0;    /* reorder if headfirst */
if (transverse == 0) reorder = 0; /* don't reorder if not transverse   */

/* note 1st gate is gate 1, but 1st bed is bed 0.
   start and end bed positions are determined only by default
   from planes selected                                      */

/* Determine # matrices to convert (niminput). */
niminput = epla - spla +1;
if (spla < 1 || spla > numpla || niminput > numpla || epla < spla
 || epla > numpla) {
   Error("Bad plane selection.");
   fclose(infilep);
   return(-1);
}

/* Set zstep, the increment used later to go through image locations. */
zstep = 1.0 / ztoxratio;
   /* zstep is xpxsize/zpxsize and is used to step through output slices */
   /* ztoxratio is ((cm.z/zpixel)/(cm.x/xpixel)) = (xpixel/zpixel). ******/
   /* Thus zstep =(zpixel/xpixel) is for stepping through indices in mlist[]. */

/* Determine # images output from this data set, considering reslicing. */
if (reslice != 0)
   nimoutput = (int) ((float)(niminput-1)/zstep) + 1;
else   /* when no reslicing */
   nimoutput=niminput;
data.ip_zdim = nimoutput;
data.ip_sets = 1;    /* 1 FRAME ONLY! */

/* get new dimensions for padding to square image */
squareflag = newsize = 0;
if (!( (mwidth == mheight) && ((mwidth == 128) || (mwidth == 256)) )) {
   if ((mwidth > 128) || (mheight > 128)) {
      squareflag = TRUE;
      newsize = 256*256;
      newwidth = newheight = 256;
      newhalfwidth = 128;
      data.ip_xdim = 256;
      data.ip_ydim = 256;
   }
   else if ((mwidth > 64) || (mheight > 64)) {
      squareflag = TRUE;
      newsize = 128*128;
      newwidth = newheight = 128;
      newhalfwidth = 64;
      data.ip_xdim = 128;
      data.ip_ydim = 128;
   }
   else squareflag = FALSE;
}

/* Create output image without pixel values. */
if (err = util_create_img(outfile, &demo, data_ptr, &extra, data.ip_xdim,
 data.ip_ydim, data.ip_zdim, data.ip_dat, data.ip_sets, data.ip_tcol,
 &outfile_struct, err_ptr)) {
   Error("Error mapping output file.");
   fclose(infilep);
   return(-1);
}

/* Set output pointers. */
/* these, as opposed to input ptrs, don't have to be updated as step through */
switch (data.ip_dat) {
   case UB_DAT:   /* 8-bit data UB_DAT is #defined as "8"  */
#ifdef DEBUG
      printf("data is  8 bit *******\n");
#endif
      dest_ub = (UBYTE *)data.ip_morg;   /* starting point (memory origin) */
      dest_ubsv = dest_ub; /* save value for later scaling */
      smax = 0;
      break;
   case SS_DAT:   /* 16-bit data SS_DAT is #defined as "16" */
#ifdef DEBUG
      printf("data is  16 bit *******\n");
#endif
      dest_ss = (SHORT *)data.ip_morg;
      dest_sssv = dest_ss; /* save value for later scaling */
      smax = 0;
      break;
   default:
      Error("Improper data type attempted.");
      fclose(infilep);
      return(-1);
      break;
}  /* end switch */

/* Adjust counters to backwards mlist. */
/* determine initial position pos for 1st plane sought */
/* recall matcount reads in everything in reverse order from that on disk */
/* i.e. mlist[] has planes from feet up (if headfirst), whereas
   reverse (head down) from disk */
/* note that pos is in reference to zero-based mlist[] index, whereas
   spla & epla are in reference to 1-based plane number index on disk in
   opposite direction. This situation arose because of their needs
   for Matcount().   ************/
if (reorder) {  /* reorder = 1 is default in mlist[] */
   pos = numpla - epla;  /* ranges from 0 to near or at spla */
}
else {  /* if not reordering */
```

```
      pos = numpla - spla;  /* then we must go backwards because of mlist[] */
      zstep =- zstep;
   }
#ifdef DEBUG
   printf("number of planes of original PET data is: %d **************\n", numpla);
   printf("numpla=%d   niminput=%d   nimoutput=%d\n", numpla, niminput, nimoutput);
#endif

/* MAIN incremental loop through output image generation by stepping slices */
   for (nwrit =0, ilast =-10000; nwrit < nimoutput; nwrit++, pos +=zstep) {
      /* step by slice;  pos = slice location */
      /* Must READ IMAGES IN PAIRS because may index into whole new space */
      if ((int)pos != ilast) {  /* If not last img, read in a new pair.   */
         if (reorder) { /* reorder=default for fwd indexing through mlist */
            iback = (int)pos;   /* trailing index, going fwd. in mlist[]  */
            ifwd = (iback <= (numpla-2)) ? iback+1 : iback;
               /* ifwd = leading index unless iback is final matrix       */
         }
         else { /* Do not reorder; move downhill in mlist[] indices       */
            ifwd = (int)pos;          /* =end...-1  */
            if (ifwd < 0)  ifwd = 0;
            iback = ifwd +1;          /* =end+1...0 */
            if (iback > numpla-1)  iback = numpla-1;
         }  /* iback is the plane behind ifwd      */
         ilast = (int)pos;  /* to know when have entered another index window */

         /* First read fwd image, later backward image.
            To read image: get matno, byte#, subheader, then image. */
         matno = mlist[ifwd];
         status = mat_lookup(infilep, matno, &entry); /* 1=success, 0=fail */
            /* mat_lookup calls mat_rdblk to position infilep. */
         mat_read_image_subheader(infilep, entry.strtblk, &sheader);
            /* infilep now points to data, after subheader */

         sprintf(errmsg, "Reading plane %2i...", nwrit+1);
         Message(errmsg);

         fwdscale = sheader.quant_scale; /* set new fwd. scale   */
         if (fwdscale == 0.0)  fwdscale = 1.0;

         iwidth = sheader.dimension_1;  /* img pixel dimensions */
         halfwidth = iwidth/2;
         height = sheader.dimension_2;
         /* require first image's size for entire set: ONLY 1 SIZE IN VOLUME */
         if (mwidth != iwidth || mheight != height) {
            sprintf(errmsg, "Cannot convert plane %i. Matrix width incompatible.",
                    ifwd);
            Error(errmsg);
            fclose(infilep);
            return(-1);
         }

         size = height*iwidth;
         if (size > SIZE || size <=0) {  /* SIZE is the max allowed */
            Error("Illegal image size encountered.");
            return(-1);
         }

         /* Read one img into inpmat array at a time as short ints */
         err = fread(inpmat, 2, size, infilep);
         if (err != size) {
            sprintf(errmsg, "Cannot read plane %i.", ifwd);
            Error(errmsg);
            fclose(infilep);
            return(-1);
         }
         /* Convert VAX to 6800 format by swapping byte order. */
         if (data.ip_dat == SS_DAT)      /* swab() for 16 bit data */
            swab(inpmat, tmpmat, size*2); /* swab(from,to,nbytes) needs size in bytes */
         else {  /* don't swab() for 8 bit data, just xfer to work buffer */
            tmpmat_byptr = (UBYTE *)&tmpmat[0]; /* redirect pointers */
            inpmat_byptr = (UBYTE *)&inpmat[0];
            for (k = 0; k < size; k++)
               *tmpmat_byptr++ = *inpmat_byptr++;
         }
         if (squareflag)
            square_img(matf, tmpmat, &newsize, iwidth, height);
         else {
            newsize = size;
            newwidth = iwidth;
            newhalfwidth = iwidth/2;
            newheight = height;
            for (k = 0; k < size; k++)
               matf[k] = tmpmat[k];
         }

      /* Now read the backward image...
         To read image: get matno, byte#, subheader, then image. */
      matno = mlist[iback];
      status = mat_lookup(infilep, matno, &entry);
      mat_read_image_subheader(infilep, entry.strtblk, &sheader);

      /* More scaling...? */
      backscale = sheader.quant_scale; /* set new backwd. scale */
      if (backscale == 0.0)  backscale = 1.0;

      /* read one image at a time into array inpmat[] as short ints */
      err = fread(inpmat, 2, size, infilep);
         /* read 2 bytes per item, size items, on infilep */
      if (err != size) {
         sprintf(errmsg, "Cannot read plane %i.", iback);
         Error(errmsg);
         fclose(infilep);
         return(-1);
      }

      /* Convert VAX to 6800 format by swapping byte order. */
      if (data.ip_dat == SS_DAT)
         swab(inpmat, tmpmat, size*2); /* swab(from,to,nbytes) requires size in bytes */
      else {  /* don't swab() for 8 bit data, just xfer to work buffer */
         tmpmat_byptr = (UBYTE *)&tmpmat[0];   /* redirect pointers   */
         inpmat_byptr = (UBYTE *)&inpmat[0];
         for (k = 0; k < size; k++)
            *tmpmat_byptr++ = *inpmat_byptr++;
      }
      if (squareflag)
         square_img(matb, tmpmat, &newsize, iwidth, height);
      else {
         newsize = size;
         newwidth = iwidth;
         newhalfwidth = iwidth/2;
         newheight = height;
         for (k = 0; k < size; k++)
            matb[k] = tmpmat[k];
      }

   } /* end if(..ilast...); end conditions for bringing in new pair image planes */

   if (reorder)
      ffwd = pos - (float)iback; /* weight for leading plane */
   else
      ffwd = (1.0 - (pos - (int)pos));
   fback = 1.0 - ffwd; /* note switching of directions for scaling */
```

```
   fb2 = backscale*fback;
   ff2 = fwdscale*ffwd;
   denom = fb2 + ff2; /* this is quant_scale fctr.new plane */
   denomax = (denom > denomax) ? denom : denomax;
   qsv[nwrit] = denom; /* save it for later scaling of output image */

   /* Generate an interpolated plane -
      Rescale only by scale factor for each plane.  */
   switch (data.ip_dat) {
      case UB_DAT:  /* 8-bit data */
         mf_byptr = (UBYTE *)&matf[0]; /* redirect pointers */
         mb_byptr = (UBYTE *)&matb[0]; /* redirect pointers */
         for (k = 0; k < newsize; k++) {
            btemp = (UBYTE)((((float)(*mb_byptr++))*fb2 +
                    ((float)(*mf_byptr++))*ff2) /denom);
            bmax = (btemp > bmax) ? btemp : bmax;
            *dest_ub++ = (btemp < 0) ? 0 : btemp;
         }
         break;
      case SS_DAT:  /* 16-bit data */
         mfptr =&matf[0]; /* redirect pointers */
         mbptr =&matb[0]; /* redirect pointers */
         for (k = 0; k < newsize; k++) {
            stemp = (SHORT) ((((float)(*mbptr++))*fb2 +
                    ((float)(*mfptr++))*ff2) /denom);
            *dest_ss++ = (stemp < 0) ? 0 : stemp;
            smax = (stemp > smax) ? stemp : smax;
         }
         break;
      default:
         Warning("Illegal data type encountered.");
         break;
      }  /* end switch for data type */
   }  /* end main incremental for-loop through all created planes */

   jump = newwidth - 1;
   jump2 = newwidth - newhalfwidth;

#ifdef DEBUG
printf("after for-loop: newwidth = %i, newhalfwidth = %i.\n\n",
       newwidth, newhalfwidth);
   printf("scaling required:  max found from input = %d  *******\n",smax);
   printf("Scaling in progress - please standby...\n");
   if (headfirst == 0) printf("Image inversion (right/left) also in progress\n");
   printf("spla, epla, reorder = %3d %3d %3d\n", spla,epla,reorder);
   printf(" reslice,transv,headfirst = %3d %3d %2d\n",reslice,transverse,headfirst);
   printf("jump, jump2, halfwidth = %d %d %d\n",jump,jump2,newhalfwidth);
#endif

   Message("Scaling images ...");
   /* Scale images to max allowed by ADAC: 4095 for int, 255 byte. */
   switch (data.ip_dat) {
      case UB_DAT:  /* 8-bit data */
         for (i = 0; i < nwrit; i++) {  /* loop images and scale fctr.s */
            scalef = 255.*qsv[i] / (denomax*(float)smax);
            if (headfirst ==1)   /* if head first, don't flip R-L.   */
               for (k = 0; k < newsize; k++)  /* loop inside each image */
                  *dest_ubsv++ *= scalef;
                     /* point back to beginning & scale all pixels */
            else {  /* else when feet first, must R-L.invert image */
               for (row = 0; row < newheight; row++) {
                  src_ub = dest_ubsv + jump;
                  for (col = 0; col < newhalfwidth; col++) {
                     btemp = scalef*(*dest_ubsv);
                     *dest_ubsv++ = scalef*(*src_ub);
                     *src_ub-- = btemp;
                  } /* end loop across columns of row   */
                  dest_ubsv += jump2;   /* advance to next row   */
               } /* end loop  down rows */
            } /* end else for case when feetfirst, requiring R-L.inversion */
         } /* end loop for(i...)   over images   */
         break;

      case SS_DAT:  /* 16-bit data */
         for (i = 0; i < nwrit; i++) {  /* loop images and scale fctr.s */
#ifdef DEBUG
printf("scaling #%i of %i...\n", i, nwrit);
#endif
            scalef = 4095.*qsv[i] / (denomax*(float)smax);
            if (headfirst==1)   /* if head first, don't flip R-L. */
               for (k = 0; k < newsize; k++)   /* loop inside each image */
                  *dest_sssv++ *= scalef;
                     /* point back to beginning & scale all pixels */
            else {  /* else when feet first, must R-L.invert image */
               for (row = 0; row < newheight; row++) {
                  src_ss = dest_sssv + jump;
                  for (col = 0; col < newhalfwidth; col++) {
                     stemp = scalef*(*dest_sssv);
                     *dest_sssv++ = scalef*(*src_ss);
                     *src_ss-- = stemp;
                  } /* end loop across columns of row */
                  dest_sssv += jump2;   /* advance pointer to next row   */
               } /* end loop down rows */
            }  /* end else for case when feetfirst, requiring R-L.inversion */
         }  /* end loop for(i...)   over images   */
      break;

   default:
      break;
   } /* end switch for scaling output image */

   /* Clean up and quit. */
   /* Fill in min, max, total counts for IP_OBJ of data. */
   ip_minmax(data_ptr, &dmin, &dmax, &dtot, err_ptr);

#ifdef DEBUG
   printf("after ip_minmax: dmin=%g  dmax=%g   dtot=%g\n",
          dmin,dmax,dtot);
#endif

   util_save_img(data_ptr, &outfile_struct, err_ptr);
   sprintf(errmsg, "%d images written to disk.", niminput);
   Message(errmsg);
   return(1);
} /* end stat_pet2peg */

/*****************************************************************/
/*****************************************************************/

int dyn_pet2peg (infilep, mheader, outfile, headfirst,
                 startframe, endframe, startgate, endgate, planenum)
int  headfirst, startframe, endframe, startgate, endgate, planenum;
char *outfile;
FILE *infilep;
Main_header mheader;
{
   /* CTI variables */
   Image_subheader sheader;
   struct MatDir entry;

   /* ADAC variables */
   UBYTE     *inpmat_byptr;
   IP_OBJ    data, *data_ptr;
   DEMO_OBJ  demo;
```

```
   EXTRA_OBJ extra;
   UBYTE     *dest_ub, *dest_ubsv;
             /* unsigned char ptr for 8 bit data, max 255 */
   SHORT     *dest_ss, *dest_sssv;
             /* pointer for signed 16-bit data, max 32767 */
   FCNTL     outfile_struct;
             /* control structure for mapped file */
   ST_ERROR_HDR  error;
   ST_ERROR_HDR *err_ptr; /* for adac error structures */

   /* pet2peg variables */
   int numpla, niminput, nimoutput, nwrit;
   int frame, gate, bed, plane;
   long matno, mheight, mwidth, size;
   int i, j, n, status, err;
   char errmsg[100];
   int c0;
   float zstep, ztoxratio;
   float dmin, dmax, dtot;
   short inmat[SIZE], tmpmat[SIZE];
   int iwidth, halfwidth, height, ifwd;

   /* my variables */
   int k, numimgs, datatype;
   struct MatDir framelist[MLSIZE];  /* all frames/gates with specified plane */
   struct Matval matval;
   struct MatDir mlist[MLSIZE];
   float scale;
   short maxval;

   /* matlist puts image matrices into mlist and returns # of imgs */
   numimgs = mat_list(infilep, mlist, MLSIZE);
   if (numimgs < 1) {
      Error ("This does not appear to be an image file.");
      fclose(infilep);
      return(-1);
   }
   for (i = 0, j = 0; i < numimgs; i++) {
      mat_numdoc(mlist[i].matnum, &matval);
      if (matval.plane  == planenum) {
         framelist[j++] = mlist[i];
#ifdef DEBUG
         printf("   got framelist[%i] (hex) %x.\n", j-1, framelist[j-1].matnum);
#endif
      }
      fflush(stdout);
   }  /* now have all imgs with given plane in framelist */

   sortlist(framelist, j);
   for (i = 0; i < j; i++)
#ifdef DEBUG
      printf("plane %i list[%i] = %x.\n", planenum, i, framelist[i].matnum);
#endif

  /***          32-bit matrix number bit-position layout.          ***/
  /*|typ|-- gate ---|---- plane ----|- bed -|------- frame ---------|
     3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
     1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 */

   /* Set matrix number. */
   bed = 0;  datatype = 0;
   matno = mat_numcod(startframe, planenum, startgate, datatype, bed);
#ifdef DEBUG
   printf("   frame %i, plane %i, gate %i, data_type %i, bed %i\n",
          startframe, planenum, startgate, datatype, bed);
#endif
   /* use it to fill struct MatDir entry with start & end blocks  */
```

```
status = mat_lookup(infilep, matno, &entry);

/* Determine pixel size from subheader "sheader" as finished
   structure in sun2 format */
mat_read_image_subheader(infilep, entry.strtblk, &sheader);
   /* use "sheader" now for subh. - NOTE: needs to be set for each. */
mwidth = sheader.dimension_1;
mheight = sheader.dimension_2;

/* Initialize/clear ADAC data structs.  */
c0 = 0;
memset(&demo, c0, sizeof(DEMO_OBJ));   /* zero demo */
memset(&extra, c0, sizeof(EXTRA_OBJ)); /* zero extra */

/* Initialize ADAC error structures. */
err_ptr =&error;
err_init_msg(err_ptr);
   /* initialize ADAC file ctl & error reporting */
util_init_fcntl(&outfile_struct);

/* Fill ADAC header structures as best possible. */
data_ptr = &data;
status = fill_peg_info(&mheader, &sheader, &demo, &extra, &data);
/* note ip_zdim, ip_sets, ip_smax, ip_stot, ip_sfac, ex_slic still not filled */
if (status != 0) {
   fclose(infilep);
   return(-1);
}
/* data_type. allow only 2=SHORT= short int output or
                         1=UBYTE= unsigned char output                    */
if ((sheader.data_type != DTYPE_BYTES) &&
    (sheader.data_type != DTYPE_I2)) {
   Error("Data in source file must be short int or unsigned char.");
   fclose(infilep);
   return(-1);
}

 /* Determine # matrices to convert (niminput). */
if (endframe > startframe) /* multiframe img */
   niminput = endframe - startframe + 1;
else if (endgate > startgate) /* gated image */
   niminput = endgate - startgate + 1;
else {
   Error ("Re-enter images to convert.");
   fclose(infilep);
   return(-1);
}
if (startframe < 1 || startgate < 1) {
   Error("Cannot convert frames or gates before #1.");
   fclose(infilep);
   return(-1);
}
if (endframe > mheader.num_frames || endgate > mheader.num_gates) {
   Error("Cannot convert images after last frame or gate.");
   fclose(infilep);
   return(-1);
}

ztoxratio = mheader.plane_separation / sheader.pixel_size;
zstep = 1.0 / ztoxratio;
   /* Set zstep, the increment used later to go through image locations. */
   /* zstep is xpxsize/zpxsize and is used to step through output slices */
   /* ztoxratio is ((cm.z/zpixel)/(cm.x/xpixel)) = (xpixel/zpixel). ******/
   /* Thus zstep =(zpixel/xpixel) is for stepping through indices in mlist[]. */

nimoutput = niminput;
data.ip_zdim = nimoutput;
```

```
   data.ip_sets = 1;    /* 1 FRAME ONLY! */
   maxval = 0;

   /* Create output image without pixel values. */
   if (err = util_create_img(outfile, &demo, data_ptr, &extra, data.ip_xdim,
    data.ip_ydim, data.ip_zdim, data.ip_dat, data.ip_sets, data.ip_tcol,
    &outfile_struct, err_ptr)) {
      Error("Error mapping output file.");
      fclose(infilep);
      return(-1);
   }

   /* Set output pointers. */
   switch (data.ip_dat) {
      case UB_DAT:   /* 8-bit data UB_DAT is #defined as "8"  */
#ifdef DEBUG
         printf("dyn_pet2peg: data is  8 bit.\n");
#endif
         dest_ub = (UBYTE *)data.ip_morg;   /* starting point (memory origin) */
         dest_ubsv = dest_ub; /* save value for later scaling */
         break;
      case SS_DAT:   /* 16-bit data SS_DAT is #defined as "16" */
#ifdef DEBUG
         printf("dyn_pet2peg: data is  16 bit.\n");
#endif
         dest_ss = (SHORT *)data.ip_morg;
         dest_sssv = dest_ss; /* save value for later scaling */
         break;
      default:
         Error("Improper data type attempted.");
         fclose(infilep);
         return(-1);
         break;
   }

   for (i = 0; i < niminput; i++) {
      if (endgate > 1)
         sprintf(errmsg, "Reading gate %2i...", niminput+1);
      else
         sprintf(errmsg, "Reading frame %2i...", niminput+1);
      Message(errmsg);

      matno = framelist[i].matnum;
      status = mat_lookup(infilep, matno, &entry); /* 1 = success, 0=fail */
      mat_read_image_subheader(infilep, entry.strtblk, &sheader);

      iwidth = sheader.dimension_1;  /* img pixel dimensions */
      halfwidth = iwidth/2;
      height = sheader.dimension_2;

      /* require first image's size for entire set: ONLY 1 SIZE IN VOLUME */
      if (mwidth != iwidth || mheight != height) {
         sprintf(errmsg, "Cannot convert frame %i. Matrix width incompatible.",
                 i);
         Error(errmsg);
         fclose(infilep);
         return(-1);
      }
      size = height*iwidth;
      if (size > SIZE || size <=0) {  /* SIZE is the max allowed */
         Error("Illegal image size encountered.");
         return(-1);
      }

      /* Read one img into input array at a time as short ints */
      err = fread(inmat, 2, size, infilep);
      if (err != size) {
         sprintf(errmsg, "Cannot read frame %i.", i);
         Error(errmsg);
         fclose(infilep);
         return(-1);
      }

      /* Convert VAX to 6800 format by swapping byte order. */
      /* also retrieve the largest intensity value of image */
      if (data.ip_dat == SS_DAT)      {  /* swab() for 16 bit data */
         swab(inmat, tmpmat, size*2);   /* swab(from,to,nbytes) requires size in bytes */
         for (k = 0; k < size; k++) {
            if (tmpmat[k] < 0)  tmpmat[k] = 0;
            maxval = (maxval > tmpmat[k]) ? maxval : tmpmat[k];
            *dest_ss++ = (SHORT) tmpmat[k];
         }
      }
      else {  /* don't swab() for 8 bit data, just xfer to work buffer */
         for (k = 0; k < size; k++) {
            tmpmat[k] = (inmat[k] > 0) ? inmat[k] : 0;
            maxval = (maxval > tmpmat[k]) ? maxval : tmpmat[k];
            *dest_ub++ = (UBYTE) tmpmat[k];
         }
      }
   } /* end for-loop */

   Message("Scaling images...");
   switch (data.ip_dat) {
      case UB_DAT:
         scale = 255./(float)maxval;
         for (i = 0; i < niminput; i++)
            for (k = 0; k < size; k++)
               *dest_ubsv++ = (UBYTE) (scale * (float)*dest_ubsv);
         break;
      case SS_DAT:
         scale = 4095./(float)maxval;
         for (i = 0; i < niminput; i++)
            for (k = 0; k < size; k++)
               *dest_sssv++ = (SHORT) (scale * (float)*dest_sssv);
         break;
      default:
         break;
   }
   fflush(stdout);

   data.ip_sfac = scale;

   /* Fill in min, max, total counts for IP_OBJ of data. */
   ip_minmax(data_ptr, &dmin, &dmax, &dtot, err_ptr);

#ifdef DEBUG
   printf("after ip_minmax: dmin=%g  dmax=%g   dtot=%g\n",
          dmin,dmax,dtot);
#endif

   util_save_img(data_ptr, &outfile_struct, err_ptr);
   sprintf(errmsg, "%d images written to disk.", niminput);
   Message(errmsg);
   return(1);
} /* end dyn_pet2peg */
```

```
/* sortlist.c - functions for sorting a matrix list by plane number.
   Part of pet2pegtool program, called from dyn_pet2peg(). MCC     */

#include "matrix.h"

void sortlist(mlist, num)
struct MatDir mlist[];
int num;
{
   int i, j;

   for (i=0; i < num; i++)
      for(j = num-1; j > i; --j)
         if (mlist[j-1].matnum > mlist[j].matnum)
            swap_matdir(&mlist[j-1], &mlist[j]);
}

swap_matdir(a, b)
struct MatDir *a, *b;
{
   struct MatDir c;

    c= *a;
   *a= *b;
   *b=  c;
}
```

```
/* square_img(newimg, oldimg, size, x1, y1)
   This function takes addresses for input and output image arrays,
   the address of the original size of the image, and the original
   width and height as arguments.  The output is returned via the
   newimg address and consists of all the points in the original
   matrix stored in the center of a new square one which has sides
   of length 256 or 128 depending on the data.
   The purpose of this addition to pet2peg.c was to alleviate some
   of the display problems encountered on ADAC workstations when
   viewing converted whole body projections which have odd,
   unequal width and height dimensions, but it has had little effect.
*/
#include <stdio.h>
#define SIZE 256*256

void square_img(newimg, oldimg, size, x1, y1)
short newimg[SIZE], oldimg[SIZE], x1, y1;
long *size;
{
   int x, y, xleft, xright, ytop, ybottom, width, height;
   long pixelnum = 0;

#ifdef DEBUG
   printf("old size = %li, ", *size);
#endif
   if ((x1 == y1) && ((x1 == 128) || (x1 == 256)))
      return;

   if ((x1 > 128) || (y1 > 128)) { /* 256x256 grid */
      xleft = 128 - x1/2;   /* set boundaries */
      xright = xleft + x1;
      ytop = 128 - y1/2;
      ybottom = ytop + y1;
      width = height = 256;
      *size = 256*256;
   }
   else if ((x1 > 64) || (y1 > 64)) { /* 128x128 grid */
      xleft = 64 - x1/2;    /* set boundaries */
      xright = xleft + x1;
      ytop = 64 - y1/2;
      ybottom = ytop + y1;
      width = height = 128;
      *size = 128*128;
   }
   else      /* you're on your own */
      return;

   for (y = 0; y < ytop; y++)
      for (x = 0; x < width; x++)
         newimg[x, y] = 0;

   for (y = ytop; y < ybottom; y++) {
      for (x = 0; x < xleft; x++)
         newimg[x, y] = 0;
      for (x = xleft; x < xright; x++)
         newimg[x, y] = oldimg[pixelnum++];
      for (x = xright; x < width; x++)
         newimg[x, y] = 0;
   }

   for (y = ybottom; y < height; y++)
      for (x = 0; x < width; x++)
         newimg[x, y] = 0;

#ifdef DEBUG
   printf("x*y = %li, pixelnum = %li.\n", x*y, pixelnum);
#endif
```

```
#define SUNVIEW 1
#define CTI 1
#include "includes.h"

Frame      main_frame, option_frame;
Panel      control_panel, option_panel;
Textsw     list_sw;
Panel_item dir_item, fname_item, image_item, newfilename_item,
           msg_item;
Icon icon;
static struct direct **filelist = NULL; /* list of files */
static short fileindex = -1; /* index for file list */
static int numfiles;     /* # files in file list */

void ls_proc(),
     list_proc(),
     Option_panel_proc(),
     Quit_proc(),
     Convert_proc(),
     Cancel_option_panel_proc(),
     Edit_header_proc();
void list_int_proc();
char *Get_selection();
int StrSearch();

#define MAX_PATH_LEN 100
#define MAX_FILENAME_LEN 50
#define MAX_FILE_LEN 50

static short icon_image[] = {
#include "pet2pegtool.icon"
};
mpr_static(pet2pegtool, 64, 64, 1, icon_image);

main(argc, argv)
int argc;
char **argv;
{
   icon = icon_create(ICON_IMAGE, &pet2pegtool, 0);
   main_frame = window_create(NULL, FRAME,
               FRAME_ARGS, argc, argv,
               FRAME_ICON, icon,
               FRAME_LABEL, "CTI PET to Pegasys ADAC Image Converter",
               0);
   Init_control_panel();
   window_fit(main_frame);
   window_main_loop(main_frame);
   exit(0);
}

Init_control_panel()
{
   char *getwd();
   char current_dir[MAX_PATH_LEN];
   char cmdstr[200], *tmplistfile;
   FILE *tmplistfp;

   control_panel = window_create(main_frame, PANEL,
                   0);
   dir_item = panel_create_item(control_panel, PANEL_TEXT,
                               PANEL_VALUE_DISPLAY_LENGTH, 35,
                               PANEL_LABEL_STRING, "Dir: ",
                               PANEL_VALUE, "/900",
                               /* PANEL_VALUE, getwd(current_dir), */
                               0);
   fname_item = panel_create_item(control_panel, PANEL_TEXT,
               PANEL_ITEM_X, ATTR_COL(0),
               PANEL_ITEM_Y, ATTR_ROW(1),
               PANEL_VALUE_DISPLAY_LENGTH, 35,
               PANEL_VALUE, "*.img",
               PANEL_LABEL_STRING, "File:",
               0);
   panel_create_item(control_panel, PANEL_BUTTON,
      PANEL_ITEM_X,   ATTR_COL(0),
      PANEL_ITEM_Y,   ATTR_ROW(2),
      PANEL_LABEL_IMAGE, panel_button_image(control_panel, "List", 0, 0),
      PANEL_NOTIFY_PROC, ls_proc,
      0);
/*
   panel_create_item(control_panel, PANEL_BUTTON,
      PANEL_LABEL_IMAGE, panel_button_image(control_panel,
                        "Edit Header", 0, 0),
      PANEL_NOTIFY_PROC, Edit_header_proc,
      0);
*/
   panel_create_item(control_panel, PANEL_BUTTON,
      PANEL_LABEL_IMAGE, panel_button_image(control_panel,
                         "Prepare Conversion", 0, 0),
      PANEL_NOTIFY_PROC, Option_panel_proc,
      0);
   panel_create_item(control_panel, PANEL_BUTTON,
      PANEL_LABEL_IMAGE, panel_button_image(control_panel, "Quit", 0, 0),
      PANEL_NOTIFY_PROC, Quit_proc,
      0);
   window_fit(control_panel);

   /* Initialize file list */
   tmplistfile = tmpnam(NULL);
   sprintf(cmdstr, "cd %s", panel_get_value(dir_item));
   system(cmdstr);
   sprintf(cmdstr, "ls %s > %s", panel_get_value(fname_item),
           tmplistfile);
   system(cmdstr);

   list_sw = window_create(main_frame, TEXTSW,
             TEXTSW_READ_ONLY, TRUE,
             TEXTSW_DISABLE_LOAD, TRUE,
             TEXTSW_DISABLE_CD, TRUE,
             TEXTSW_MENU, NULL,
             TEXTSW_FILE, tmplistfile,
             WIN_ROWS, 20,
             WIN_COLUMNS, (int)window_get(main_frame, WIN_COLUMNS),
             WIN_X, (int)window_get(control_panel, WIN_X),
             WIN_BELOW, control_panel,
             0);
   window_fit_width(main_frame);
/*
   list_proc();
   notify_interpose_event_func(list_sw, list_int_proc, NOTIFY_SAFE);
*/
   remove(tmplistfile);
}

void list_proc()
{
   extern alphasort();
   char dirstr[MAX_PATH_LEN];
   char tmpstr[100];
   int ind1, ind2, entry;
   int i, j, len, winsize;

   strcpy(dirstr, (char *)panel_get_value(dir_item));
   numfiles = scandir(dirstr, &filelist, NULL, alphasort);
   winsize = (int)window_get(list_sw, WIN_COLUMNS) - 5;
```

```
   if (numfiles > 0) {
      for (i = 0; i < numfiles; i++) {
         len = strlen(filelist[i]->d_name);
         sprintf(tmpstr, "%s", filelist[i]->d_name);
         for (j = len; j < winsize; j++)
            tmpstr[j] = ' ';
         textsw_insert(list_sw, tmpstr, strlen(tmpstr));
      }
      entry = (numfiles > 1) ? 1 : 0;

      ind1 = textsw_index_for_file_line(list_sw, entry);
      ind2 = ind1 + winsize;
      textsw_set_selection(list_sw, ind1, ind2, 1);

      fileindex = entry;
      panel_set(fname_item,
         PANEL_VALUE, filelist[fileindex]->d_name,
         0);
      textsw_possibly_normalize(list_sw, 0);
   }
   else
      Error("No files found in current directory.");
}

Init_option_panel(filename)
char *filename;
{
   int framenum, planenum, gatenum, n;
   int col, row, status;
   char newfilename[MAX_FILENAME_LEN], tmpstr[200];
   char srcfilestr[MAX_FILENAME_LEN+20], srcfileimgstr[50];
   char endplanestr[4], endframestr[4], endgatestr[4];
   Main_header mheader;
   Panel_item prev;
   FILE *infilep;

   strncpy(newfilename, filename, n = StrSearch(filename, '.'));
   newfilename[n] = 0;
   strcat(newfilename, ".adac");

   /* Prep input file ptr for reading values to edit */
   infilep = fopen(filename, "r");
   if (infilep == NULL) {
      Error("Cannot open input file.");
      return;
   }
   status = mat_read_main_header(infilep, &mheader);
   fclose(infilep);
   if (strlen(mheader.study_description) == 0)
      sprintf(mheader.study_description, "%s", "PET IMAGE");

   option_frame = window_create(main_frame, FRAME,
      FRAME_NO_CONFIRM, TRUE,
      FRAME_SHOW_LABEL, FALSE,
      /* FRAME_SHOW_SHADOW, FALSE, */
      WIN_ROWS, 20,
      WIN_HEIGHT, 300,
      0);
   option_panel = window_create(option_frame, PANEL,
      WIN_PERCENT_HEIGHT, 10,
      0);

   msg_item = panel_create_item(option_panel, PANEL_MESSAGE,
      PANEL_LABEL_STRING, "Enter conversion options, then press Convert.",
      0);
   row = col = 1;
   sprintf(tmpstr, "Source File:  %s", filename);
   panel_create_item(option_panel, PANEL_MESSAGE,
      PANEL_LABEL_STRING, tmpstr,
      PANEL_ITEM_X, ATTR_COL(col),
      PANEL_ITEM_Y, ATTR_ROW(row++),
      0);
   panel_create_item(option_panel, PANEL_TEXT,
      PANEL_LABEL_STRING, "Output File: ",
      PANEL_VALUE_DISPLAY_LENGTH, 30,
      PANEL_VALUE_STORED_LENGTH, MAX_FILENAME_LEN,
      PANEL_VALUE, newfilename,
      PANEL_ITEM_X, ATTR_COL(col),
      PANEL_ITEM_Y, ATTR_ROW(row++),
      0);
   panel_create_item(option_panel, PANEL_CYCLE,
      PANEL_LABEL_STRING, "Orientation: ",
      PANEL_CHOICE_STRINGS,
         "Head-first, no flipping of images",
         "Feet-first, laterally flip images", 0,
      PANEL_ITEM_X, ATTR_COL(col),
      PANEL_ITEM_Y, ATTR_ROW(row++),
      0);
   row++;

   sprintf(tmpstr, "Source file has %i planes, %i frame(s), %i gate(s).",
          mheader.num_planes, mheader.num_frames, mheader.num_gates);
   panel_create_item(option_panel, PANEL_MESSAGE,
      PANEL_LABEL_STRING, tmpstr,
      PANEL_ITEM_X, ATTR_COL(col),
      PANEL_ITEM_Y, ATTR_ROW(row++),
      0);

   if (mheader.num_frames > 1) { /* Dynamic image options */
      sprintf(endplanestr, "%i", mheader.num_planes);
      sprintf(endframestr, "%i", mheader.num_frames);
      prev = panel_create_item(option_panel, PANEL_CHOICE,
         PANEL_LAYOUT, PANEL_VERTICAL,
         PANEL_CHOICE_STRINGS, "Convert planes ", "Convert frames ", 0,
         PANEL_ITEM_X, ATTR_COL(col),
         PANEL_ITEM_Y, ATTR_ROW(row),
         0);
      row += 3;
      panel_create_item(option_panel, PANEL_TEXT,
         PANEL_VALUE_DISPLAY_LENGTH, 2,
         PANEL_VALUE, "1",
         PANEL_ITEM_X, ATTR_COL(col+18),
         PANEL_ITEM_Y, (int)panel_get(prev, PANEL_ITEM_Y),
         0);
      panel_create_item(option_panel, PANEL_TEXT,
         PANEL_LABEL_STRING, " through ",
         PANEL_VALUE_DISPLAY_LENGTH, 2,
         PANEL_VALUE, endplanestr,
         PANEL_ITEM_X, ATTR_COL(col+20),
         PANEL_ITEM_Y, (int)panel_get(prev, PANEL_ITEM_Y),
         0);
      panel_create_item(option_panel, PANEL_TEXT,
         PANEL_LABEL_STRING, " of frame ",
         PANEL_VALUE_DISPLAY_LENGTH, 2,
         PANEL_VALUE, endframestr,
         PANEL_ITEM_X, ATTR_COL(col+32),
         PANEL_ITEM_Y, (int)panel_get(prev, PANEL_ITEM_Y),
         0);

      panel_create_item(option_panel, PANEL_TEXT,
         PANEL_VALUE_DISPLAY_LENGTH, 2,
         PANEL_VALUE, "1",
         PANEL_ITEM_X, ATTR_COL(col+18),
```

```
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_CHOICE_Y, 1),
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " through ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, endframestr,
        PANEL_ITEM_X, ATTR_COL(col+20),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_CHOICE_Y, 1),
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " of plane ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, "1",
        PANEL_ITEM_X, ATTR_COL(col+32),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_CHOICE_Y, 1),
        0);
}
else if (mheader.num_gates > 1) {   /* Gated image options */
    sprintf(endplanestr, "%i", mheader.num_planes);
    sprintf(endgatestr, "%i", mheader.num_gates);
    prev = panel_create_item(option_panel, PANEL_CHOICE,
        PANEL_LAYOUT, PANEL_VERTICAL,
        PANEL_CHOICE_STRINGS, "Convert planes ", "Convert gates  ", 0,
        PANEL_ITEM_X, ATTR_COL(col),
        PANEL_ITEM_Y, ATTR_ROW(row += 2),
        0);
    row += 3;
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, "1",
        PANEL_ITEM_X, ATTR_COL(col+18),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_ITEM_Y),
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " through ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, endplanestr,
        PANEL_ITEM_X, ATTR_COL(col+20),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_ITEM_Y),
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " of gate  ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, endframestr,
        PANEL_ITEM_X, ATTR_COL(col+32),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_ITEM_Y),
        0);

    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, "1",
        PANEL_ITEM_X, ATTR_COL(col+18),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_CHOICE_Y, 1),
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " through ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, endframestr,
        PANEL_ITEM_X, ATTR_COL(col+20),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_CHOICE_Y, 1),
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " of plane ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, "1",
        PANEL_ITEM_X, ATTR_COL(col+32),
        PANEL_ITEM_Y, (int)panel_get(prev, PANEL_CHOICE_Y, 1),
        0);
}
else { /* Static image -> less options */
    sprintf(endplanestr, "%i", mheader.num_planes);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, "Convert planes ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_ITEM_X, ATTR_COL(col),
        PANEL_ITEM_Y, ATTR_ROW(row),
        PANEL_VALUE, "1",
        0);
    panel_create_item(option_panel, PANEL_TEXT,
        PANEL_LABEL_STRING, " through ",
        PANEL_VALUE_DISPLAY_LENGTH, 2,
        PANEL_VALUE, endplanestr,
        PANEL_ITEM_X, ATTR_COL(col+18),
        PANEL_ITEM_Y, ATTR_ROW(row++),
        0);
    row++;
}

panel_create_item(option_panel, PANEL_TOGGLE,
    PANEL_CHOICE_STRINGS, "Transverse source image - needs reslicing", 0,
    PANEL_TOGGLE_VALUE, 0, FALSE,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_ROW(row++),
    0);
panel_create_item(option_panel, PANEL_TOGGLE,
    PANEL_CHOICE_STRINGS, "Add to ADAC database", 0,
    PANEL_TOGGLE_VALUE, 0, TRUE,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_ROW(row++),
    0);
row++;

panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Patient ID         : ",
    PANEL_VALUE_DISPLAY_LENGTH, 12,
    PANEL_VALUE_STORED_LENGTH, 12,
    PANEL_VALUE, mheader.patient_id,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Patient Name       : ",
    PANEL_VALUE_DISPLAY_LENGTH, 16,
    PANEL_VALUE_STORED_LENGTH, 16,
    PANEL_VALUE, mheader.patient_name,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
sprintf(tmpstr, "%c", mheader.patient_sex);
panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Patient Sex        : ",
    PANEL_VALUE_DISPLAY_LENGTH, 2,
    PANEL_VALUE_STORED_LENGTH, 2,
    PANEL_VALUE, tmpstr,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Patient Age        : ",
    PANEL_VALUE_DISPLAY_LENGTH, 10,
    PANEL_VALUE_STORED_LENGTH, 10,
    PANEL_VALUE, mheader.patient_age,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
```

```
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Patient Height     : ",
    PANEL_VALUE_DISPLAY_LENGTH, 10,
    PANEL_VALUE_STORED_LENGTH, 10,
    PANEL_VALUE, mheader.patient_height,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Patient Weight     : ",
    PANEL_VALUE_DISPLAY_LENGTH, 10,
    PANEL_VALUE_STORED_LENGTH, 10,
    PANEL_VALUE, mheader.patient_weight,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Referring physician: ",
    PANEL_VALUE_DISPLAY_LENGTH, 20,
    PANEL_VALUE_STORED_LENGTH, 20,
    PANEL_VALUE, mheader.physician_name,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Attending physician: ",
    PANEL_VALUE_DISPLAY_LENGTH, 20,
    PANEL_VALUE_STORED_LENGTH, 20,
    PANEL_VALUE, mheader.operator_name,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Radiopharmaceutical: ",
    PANEL_VALUE_DISPLAY_LENGTH, 16,
    PANEL_VALUE_STORED_LENGTH, 16,
    PANEL_VALUE, mheader.radiopharmaceutical,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Hospital ID        : ",
    PANEL_VALUE_DISPLAY_LENGTH, 20,
    PANEL_VALUE_STORED_LENGTH, 20,
    PANEL_VALUE, mheader.facility_name,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  panel_create_item(option_panel, PANEL_TEXT,
    PANEL_LABEL_STRING, "Study Description  : ",
    PANEL_VALUE_DISPLAY_LENGTH, 32,
    PANEL_VALUE_STORED_LENGTH, 32,
    PANEL_VALUE, mheader.study_description,
    PANEL_ITEM_X, ATTR_COL(col),
    PANEL_ITEM_Y, ATTR_COL(row++),
    0);
  row++;

  (void) panel_create_item(option_panel, PANEL_BUTTON,
    PANEL_LABEL_IMAGE, panel_button_image(option_panel,
                       "Cancel", 0,0),
    PANEL_NOTIFY_PROC, Cancel_option_panel_proc,
    PANEL_LABEL_X, ATTR_COL(0),
    PANEL_LABEL_Y, ATTR_ROW(row),
    0);
  (void) panel_create_item(option_panel, PANEL_BUTTON,
    PANEL_LABEL_IMAGE, panel_button_image(option_panel,
                       "Convert", 0,0),
    PANEL_NOTIFY_PROC, Convert_proc,
    PANEL_LABEL_X, ATTR_COL(10),
    PANEL_LABEL_Y, ATTR_ROW(row),
    0);

}

void ls_proc()
{
  static char previous_dir[MAX_PATH_LEN];
  char *current_dir, *tmplistfile;
  char cmdstring[200], tempstr[100];

  tmplistfile = tmpnam(NULL);

  current_dir = (char *) panel_get_value(dir_item);
  if (strcmp(current_dir, previous_dir)) {
    chdir(current_dir);
    sprintf(cmdstring, "cd %s", current_dir);
    system(cmdstring);
    strcpy(previous_dir, current_dir);
  }
  strcpy(tempstr, panel_get_value(fname_item));
  if (strchr(tempstr, '*') || strchr(tempstr, '?'))
    sprintf(cmdstring, "ls %s > %s",tempstr, tmplistfile);
  else /* a filename, not a filter */
    sprintf(cmdstring, "ls *.img > %s", tmplistfile);
  system(cmdstring);

  window_set(list_sw, TEXTSW_FILE, tmplistfile, 0);
}

void Quit_proc()
{
  window_destroy(main_frame);
}

void Edit_header_proc()
{
  Error("Function Edit_header_proc not implemented yet.");
}

void Option_panel_proc()
{
  char *filename, errmsg[100];
  short status;

  if (strlen(filename = Get_selection()) == 0)
    return;

  if (option_frame == NULL) {
    Init_option_panel(filename);
    window_fit(option_panel);
    window_fit(option_frame);
    window_main_loop(option_frame);
  }
}

void Convert_proc()
{
  char infile[MAX_FILENAME_LEN], outfile[MAX_FILENAME_LEN];
  char tmpstr[30], db_string[MAX_FILENAME_LEN+10];
  int transverse, headfirst, startpla, endpla, in_dbase, err;
  int framenum, startframe, endframe, planenum, gatenum;
  int startgate, endgate;
```

```
   short status, planetoggle;
   FILE *infilep;
   Panel_item next;
   Main_header mheader;

   strcpy(infile, (char *)panel_get_value(fname_item));
   /* Open the input file for reading, get main header */
   if (err = Validate_infile(infile)) {
      Error("Invalid source filename.");
      return;
   }
   infilep = fopen(infile, "r");
   if (infilep == NULL) {
      Error("Cannot open input file.");
      return;
   }
   status = mat_read_main_header(infilep, &mheader);
   if (mheader.file_type != FTYPE_IMAGE) {
      Error("Source file is not a CTI image file!");
      fclose(infilep);
      return;
   }

   startframe = endframe = startgate = endgate = 1;
   framenum = gatenum = planenum = 1;

   /* Retrieving values for conversion */
   next = panel_get(option_panel, PANEL_FIRST_ITEM); /* message str */
   next = panel_get(next, PANEL_NEXT_ITEM); /* src filename */
   sprintf(infile, "%s", (char *)panel_get_value(next));
   next = panel_get(next, PANEL_NEXT_ITEM);  /* output filename */
   sprintf(outfile, "%s", (char *)panel_get_value(next));
   next = panel_get(next, PANEL_NEXT_ITEM);  /* orientation */
   headfirst = 1 - (int) panel_get_value(next);
   next = panel_get(next, PANEL_NEXT_ITEM);  /* imgs in src */


   if (mheader.num_frames > 1) {  /* Dynamic image */
#ifdef DEBUG
printf("working on dynamic options...\n");
#endif
      next = panel_get(next, PANEL_NEXT_ITEM);  /* plane-toggle */
      planetoggle = (int) panel_get_value(next);
      next = panel_get(next, PANEL_NEXT_ITEM); /* 1st plane */
      startpla = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* last plane */
      endpla = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* frame # */
      framenum = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* 1st frame */
      startframe = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* last frame */
      endframe = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* plane # */
      planenum = atoi((char *) panel_get_value(next));
   }
   else if (mheader.num_gates > 1) {  /* Gated image */
#ifdef DEBUG
printf("working on gated options...\n");
#endif
      next = panel_get(next, PANEL_NEXT_ITEM);  /* plane-toggle */
      planetoggle = (int) panel_get_value(next);
      next = panel_get(next, PANEL_NEXT_ITEM); /* 1st plane */
      startpla = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* last plane */
      endpla = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* gate # */
      gatenum = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* 1st gate */
      startframe = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* last gate */
      endframe = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* plane # */
      planenum = atoi((char *) panel_get_value(next));
   }
   else {  /* Static image */
#ifdef DEBUG
printf("working on static options...\n");
#endif
      next = panel_get(next, PANEL_NEXT_ITEM); /* 1st plane */
      sprintf(tmpstr, "%i", (char *) panel_get_value(next));
      startpla = atoi((char *) panel_get_value(next));
      next = panel_get(next, PANEL_NEXT_ITEM); /* last plane */
      sprintf(tmpstr, "%i", (char *) panel_get_value(next));
      sscanf(tmpstr, "%i", &endpla);
      endpla = atoi((char *) panel_get_value(next));
      planetoggle = 0;
   }

   next = panel_get(next, PANEL_NEXT_ITEM); /* transverse toggle */
   transverse = (int) panel_get_value(next);
   next = panel_get(next, PANEL_NEXT_ITEM); /* dbase toggle */
   in_dbase = (int) panel_get_value(next);

   /* Editing header values */
   next = panel_get(next, PANEL_NEXT_ITEM); /* patient id */
   sprintf(mheader.patient_id, "%s", (char*)panel_get_value (next));
   next = panel_get(next, PANEL_NEXT_ITEM); /* patient name */
   sprintf(mheader.patient_name, "%s", (char*)panel_get_value (next));
   next = panel_get(next, PANEL_NEXT_ITEM); /* patient sex */
   sprintf(tmpstr, "%1.1s", (char*)panel_get_value (next));
   mheader.patient_sex=tmpstr[0];
   next = panel_get (next, PANEL_NEXT_ITEM); /* patient age */
   sprintf(mheader.patient_age, "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* patient height */
   sprintf(mheader.patient_height, "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* patient weight */
   sprintf(mheader.patient_weight, "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* referring physician */
   sprintf(mheader.physician_name, "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* attending physician */
   sprintf(mheader.operator_name, "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* radiopharmaceutical */
   sprintf(mheader.radiopharmaceutical,  "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* hospital id */
   sprintf(mheader.facility_name, "%s", (char*)panel_get_value (next));
   next = panel_get (next, PANEL_NEXT_ITEM); /* study description */
   sprintf(mheader.study_description,  "%s", (char*)panel_get_value (next));

#ifdef DEBUG
   printf("planetoggle = %i\n", planetoggle);
   printf("   infile = %s \toutfile = %s \ttransverse = %i \theadfirst = %i\n",
          infile, outfile, transverse, headfirst);
   printf("   startpla = %i \tendpla = %i framenum = %i \tgatenum = %i\n",
          startpla, endpla, framenum, gatenum);
   printf("   startframe = %i \tendframe = %i \tplanenum = %i\n",
          startframe, endframe, planenum);
   printf("   startgate = %i \tendgate = %i\n\n", startgate, endgate);
#endif

   if (planetoggle == 0)
      err = stat_pet2peg(infilep, mheader, infile, outfile, transverse,
                         headfirst, startpla, endpla, framenum, gatenum);
   else
```

```
      err = dyn_pet2peg(infilep, mheader, outfile, headfirst,
                        startframe, endframe, startgate, endgate, planenum);

   fclose(infilep);
   if ((in_dbase == TRUE) && (err > 0)) {
      Message("Adding images to database...");
      sprintf(db_string, "add_to_db -e %s > Pet2peg.log", outfile);
      system(db_string);
      remove("Pet2peg.log");
   }

   if (err > 0) { /* success */
      window_destroy(option_frame);
      option_frame = NULL;
   }
   else
      Message("File conversion aborted.");
}

void Cancel_option_panel_proc()
{
   window_destroy(option_frame);
   option_frame = NULL;
}

char *Get_selection()
{
   static char filename[MAX_FILENAME_LEN];
   Seln_holder holder;
   Seln_request *buffer;
   int lastchar;

   holder = seln_inquire(SELN_PRIMARY);
   buffer = seln_ask(&holder, SELN_REQ_CONTENTS_ASCII, 0, 0);
   strncpy(filename, buffer->data + sizeof(Seln_attribute), MAX_FILENAME_LEN);



   if ((filename[lastchar = (strlen(filename) - 1)] == '\n') ||
    (filename[lastchar] == ' '))
      filename[lastchar] = 0;  /* remove eol/space character */
   panel_set_value(fname_item, filename);
   return(filename);
}


/*****************************************************************/

int StrSearch (string, ch)
char string[], ch;
/* StrSearch Arguments: string of letters & a character to find;
   StrSearch Returns: location of character in string, if present,
                      else -1.                                    */
 {
    int i = 0;

    while (string[i] != 0) {
       if (string[i] == ch)
            return (i);
            i++;
       }
    return (-1);
 }

/************************************************/

void list_int_proc(frame, event, arg, type)
Frame frame;
Event *event;
Notify_arg arg;
Notify_event_type type;
{
   int entry, ind1, ind2, topline, bottomline, length;

   switch(event_id(event)) {
      case MS_LEFT:
      case LOC_DRAG:
         entry = event_y(event)/PIXEL_SEPARATION; /* get line# */
         textsw_file_lines_visible(list_sw, &topline, &bottomline);
            /* get top & bottom indices */
         entry = entry + topline;  /* compensate for offset into window */
     /*    if (entry >= no_entries)   if invalid line, fix it /
            entry = no_entries - 1;
*/
         /* select the entry */
         ind1 = textsw_index_for_file_line(list_sw, entry);
         ind2 = ind1 + (TEXTSW_WIDTH - 5);
#ifdef DEBUG
         printf("top = %i, bottom = %i, WIDTH=%i, ind1 = %i, ind2 = %i\n",
                topline, bottomline, TEXTSW_WIDTH, ind1, ind2);
#endif
         textsw_set_selection(list_sw, ind1, ind2, 1);
         break;
      default:
         notify_next_event_func(frame, event, arg, type);
   }
}

void get_dirstr(dirstr)
char *dirstr;
{
   char tmpstr[MAX_PATH_LEN], filestr[MAX_FILE_LEN];
   static char olddirstr[MAX_PATH_LEN];
   char ch;
   int dirlen, filelen, i;

   strcpy(dirstr, (char *)panel_get_value(dir_item));
   strcpy(filestr, (char *)panel_get_value(fname_item));

   (void) getwd(olddirstr);

   if ((dirlen = strlen(dirstr)) > 0) { /* if dir specified */
      if ((filelen = strlen(filestr) > 0)){
         /* combine for dir-filter string */
         if (dirstr[dirlen-1] == '/')
            sprintf(tmpstr, "%s%s", dirstr, filestr);
         else
            sprintf(tmpstr, "%s/%s", dirstr, filestr);
      }
      else {  /* just return directory */
         if (dirstr[dirlen-1] == '/')
            sprintf(tmpstr, "%s%s", dirstr, filestr);
         else
            sprintf(tmpstr, "%s/%s", dirstr, filestr);
      }
   }
   if (dirlen == 0)  /* no directory specified */
      if ((filelen = strlen(filestr) > 0))
         /* just return filter */
         strcpy(tmpstr, filestr);
      else
         /* use prev dir */
         strcpy(tmpstr, olddirstr);
}
```

```
/*********************************************
*       int Validate_infile(infile)
*
*       return = 0 = ok (.img, .wbp)
*                1 = bad filename found
*
*********************************************/
#include        <string.h>

int Validate_infile(filename)
char    *filename;

{
static  char    simg[5]={".img"};
static  char    swbp[5]={".wbp"};

if(strstr(filename,simg)==NULL &&
strstr(filename,swbp)==NULL)
        return 1;       /* error if of neither
        type    */
else
        return 0; /* ok, extension correct */
}
```

```
#define CTI 1
#include "includes.h"

int
write_matrix_data(fptr, strtblk, nblks, dptr, dtype)
FILE *fptr;
int strtblk,
    nblks,
    dtype;
char *dptr;
{
        int err,
            i;

        switch (dtype)
        {
            case 1:     /* byte format...no
                         * translation necessary */
                mat_wblk(fptr, strtblk, dptr, nblks);
                break;
            case 2:     /* Vax I*2 */
                err = mat_write_idata(fptr, strtblk, dptr, 512 * nblks);

                if (err < 0)
                        return (-1);
                break;
            case 4:     /* Vax R*4 */
                mat_write_fdata(fptr, strtblk, dptr, 512 * nblks);
                break;
            case 5:     /* IEEE R*4 */
                mat_wblk(fptr, strtblk, dptr, nblks);
                break;
            case 6:     /* 68K I*2 */
                mat_wblk(fptr, strtblk, dptr, nblks);
                break;
            case 7:     /* 68K I*4 */
                mat_wblk(fptr, strtblk, dptr, nblks);
                break;
            default:    /* something
                         * else...treat as Vax
                         * I*2 */
                mat_write_idata(fptr, strtblk, dptr, 512 * nblks);
                break;
        }
        return (OK);
}
```

```
/* includes.h - include file for all components of pet2pegtool
 *              by Marc Cruser, derived from code by Ed Glass  */

/*
#define DEBUG
*/

#define CTI_DIR "/900"
#define MAX_PATH_LEN 100
#define MAX_FILENAME_LEN 50
#define MAX_FILE_LEN 50

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/dir.h>

#ifdef SUNVIEW
#include <memory.h>
#include <sys/file.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/textsw.h>
#include <suntool/icon_load.h>
#include <suntool/seln.h>
#include <suntool/alert.h>

#define PIXEL_SEPARATION 16
#define TEXTSW_WIDTH 45
#endif

#ifdef CTI
#include "matrix.h"
#include "header_defs.h"
#endif

#ifdef ADAC
#include "fcntl_toolkit.h"
#include "scan_types.h"
#include "adac_types.h"
#include "img_objs.h"
#include "demo_objs.h"
#include "extra_objs.h"
#include "err_struct.h"
#include "adac_err.h"
#endif

#define LMAX      1024
#define WIDTH     256
#define HEIGHT    256
#define SIZE      WIDTH*HEIGHT
#define HSIZE     256
#define NIMAGEMAX 310  /* enough for 10 beds, 31 images each ungated */
#define MLSIZE    1000 /* max size for mlist[] of matno's */
```

```
# Makefile for 2dicom program using the SUIT X-Windows toolkit
# by Marc Cruser, 21 Nov 1994 UTMCK IPL

PROG = 2dicom

# this denotes where the SUIT distribution lives
SUIT      =/home/Se/cruser/suit/sparc
XLIBS     =/lib
X_INC_DIR =/usr/openwin/share/include/X11

CC      =       gcc
LDLIBS  =       -lsuit -lsrgp -lX11 -lm
LDFLAGS =       -L$(SUIT)/lib -L$(XLIBS)

#DEBUG  =       -g
OPTIMIZE =      -O

# Remove the static flag to dynamically link, use less disk space
STATIC= -static

CFLAGS  =       $(STATIC) -fpcc-struct-return  -Wall \
               -I$(X_INC_DIR) -I$(SUIT)/include $(DEBUG) $(OPTIMIZE)

COMPILE.c=$(CC) $(CFLAGS) $(CPPFLAGS) -c

2dicom: suitfunctions.o cti2dcm.o  adac2dicom.o
        $(CC) $(CFLAGS) $(LDFLAGS) -o $(PROG) suitfunctions.o \
           cti2dcm.o matrix.o write_matrix_data.o \
           matcount.o  adac2dicom.o $(LDLIBS)
        strip $(PROG)
########################################################################

clean:
#       rm -f *.o core 2dicom
```

README

Purpose of Program

2Dicom was written to convert medical image files used within the UTMC IPL to the DICOM imaging standard. The program will be able to receive as input those image files which have been stored in ADAC, CTI, and Genesis formats. This conversion should make the collection of medical image files in the IPL easier to manage and increase compatibility with future imaging equipment which will likely have built-in DICOM compatibility.

Program Requirements

The conversion program runs on Sun Sparc workstations running X-Windows.

Installation

The file currently lies in the directory /home/Se/cruser/project/dicom. Type 2dicom to run the program. (In a full implementation of the program, extraction from a tar file and modification of the Makefile will also be necessary.)

Running the Program

Opening Files

From the File menu, select "Open ADAC," "Open CTI," or "Open Genesis." This creates a file browser from which you can change directories or select files with a double click of the mouse. If the program can recognize the file as an ADAC, CTI, or Genesis image, the header data from the file will be converted to the DICOM format. The Patient Data Window will appear in the program workspace with the data ready to edit if necessary. If you wish to view or edit the study-specific header information, select "Study Data" from the Edit menu. The two items under the Edit menu will toggle between these two displays, and the contents in each of these windows will be stored in memory when the window is closed.

Saving Files

To save the current set of images as a DICOM file, select "Save As" from the File menu. (In a later version of the program you will be prompted for a filename, and the image data will be transferred to disk. This option is currently not available.)

Additional Features

The header information that define values necessary for the display of the images can be viewed by selecting "Image Header" from the Window menu. These values cannot be changed. (This is necessary to preserve image integrity.) To print out the header information for the current image, select "Print Header" from the File menu. Under the Help menu you can select options which provide information about the program or this README file. To end the program, select "Quit" from the File menu.

Known Bugs

As mentioned earlier, the program does not yet save the information to a DICOM file, but this will be remedied in early 1995. Some large CTI files (mostly dynamic images) cause the program to crash when they are read. A feature which allows the user to remove the "Image Header" display via the Window menu as well as from the "Done" button makes the program unstable. Resizing the program window can disorient some fields in the "Patient Data" and "Study Data" windows.

Program History

2Dicom developed as a part of my undergraduate research involving medical image format conversion. I chose to work with DICOM over the other available formats (namely, the three proprietary formats and another standard, Interfile) because of its well-defined header values and flexibility in which values are necessary in the files (thus saving disk space by leaving out unnecessary values). The standard is well-established within the medical imaging community and should serve as the basis for a number of new programs in the future. A C++ implementation of the program was planned during the summer but was later scrapped due to concerns over insufficient language standardization and toolkit compatibility. An ASCII terminal version of the program which converts ADAC images to an intermediate data structure was written by Sam Burgiss earlier this fall. The SUIT toolkit was selected for programming the GUI because of my previous experience with it. The current version is intended to demonstrate the program interface for presentation in November 1994.

Future Developments

The "Save As" option and Genesis compatibility are high priorities for future development. The file browser is currently based on a SUIT widget and will be modified to allow images from multiple source files to be stored as one study in the output DICOM file. I would also like to rewrite the GUI using another toolkit, such as Motif or OpenLook, which will improve the precision of placement of items within the window relative to the locations of other related items (and eliminate one of the bugs mentioned earlier). The DICOM values currently included represent a bare minimum of those values which are included within the standard, so future releases will hopefully incorporate other optional header values. An option permitting batch conversions at a later time would reduce the load on the system during peak hours. A feature permitting conversion to some other popular image format (like GIF or TIFF) would make it easier to incorporate these images into documents and presentations created with common Macintosh or PC software. Finally, an option to view images before they are saved to disk could prevent some unnecessary conversions. If you have any other suggestions, you may reach me by e-mail at cruser@scanner.hosp.utk.edu.

```
/* adac2dicom.c - This file includes functions which convert ADAC data
   to a DICOM format. Written by Sam Burgiss, edited by Marc Cruser
   21 Nov 1994, UTMC IPL.                                          */

#define DICOM
#include "includes.h"
#include "adacfunctions.h"

int fread_int(FILE *fp, int offset, int c_s_l)
   /*************************************************************************
   The fread_int function reads either a char, short, or int from an input file
   at a specified offset.
   Input:  fp                          fp points to the file which will be read
                                       from
           offset                      position of data to be read
           c_s_l                       indicates type of data to be read
                                       0 for char, 1 for short, and 2 for int
   Ouput:  The fucntion returns the data which is read from the file.
   *************************************************************************/
   {
   char let;
   short short_num;
   int num;
   fseek(fp,(long int)offset,0);
   if(c_s_l==0)
      {
      fread(&let,sizeof(char),1,fp);
      return (int)let;
      }
   else if(c_s_l==1)
      {
      fread(&short_num,sizeof(short),1,fp);
      return (int)short_num;
      }
   else if(c_s_l==2)
      {
      fread(&num,sizeof(int),1,fp);
      return num;
      }
   else return (0);
   }

void fread_string(FILE *fp, int offset, int ln, char *string)
   /*************************************************************************
   The fread_string function reads a string from an input file at a specified
   offset.
   Input:  fp                          fp points to the file which will be read
                                       from
           offset                      position of data to be read
           ln                          length of string to be read
           string                      pointer where the data read will be stored
   Ouput:  string will point to the data read after the functions execution.
   *************************************************************************/
   {
   fseek(fp,(long int)offset,0);
   fgets(string,ln+1,fp);
   }

float fread_float(FILE *fp, int offset)
   /*************************************************************************
   The FREAD_FLOAT function reads a float from an input file at a specified
   offset.
   Input:  fp                          fp pointes to the file which will be read
                                       from
           offset                      position of the data to be read
   Output:  The function returns the number read from the file.
   *************************************************************************/
   {
   float num;
   fseek(fp,(long int)offset,0);
   fread(&num,sizeof(float),1,fp);
   return num;
   }

void read_adac(adac_data hdr, char filename[], int *num_headers)
   /*************************************************************************
   The READ_FILE function reads the header data from an ADAC image file
   Input:  string containing file name and path, int pointer;
   Output:  The fucntion returns a structure of the type data which contains
            the header information read from the file, and the offset to
            the image data.
   *************************************************************************/
   {
   FILE *fp;
   int pos=10;
   int min_offset=99999;
   int done;
   char temp_char;
   short keynum;
   short offset;

      fp=fopen(filename,"rb");
   while (pos < min_offset)
      {
      keynum=(short)fread_int(fp,pos,1);
      offset=(short)fread_int(fp,pos+4,1);
      if (0)
         printf("keynum:  %d   offset:  %d  pos:  %d  min_offset:  %d\n",
            keynum,offset,pos,min_offset);
      if (keynum==1)
         fread_string(fp,offset,20,hdr.pat_name);
      else if(keynum==2)
         fread_string(fp,offset,11,hdr.pat_id);
      else if(keynum==3)
         hdr.pat_sex=(char)fread_int(fp,offset,0);
      else if(keynum==4)
         hdr.pat_age=(short)fread_int(fp,offset,1);
      else if(keynum==5)
         hdr.pat_height=(short)fread_int(fp,offset,1);
      else if(keynum==6)
         hdr.pat_weight=(short)fread_int(fp,offset,1);
      else if(keynum==76)
         fread_string(fp,offset,6,hdr.pat_key);
      else if(keynum==109)
         fread_string(fp,offset,8,hdr.birth_date);
      else if(keynum==7)
         fread_string(fp,offset,8,hdr.acq_date);
      else if(keynum==8)
         fread_string(fp,offset,8,hdr.dose_admin_time);
      else if(keynum==9)
         fread_string(fp,offset,8,hdr.exam_key);
      else if(keynum==10)
         fread_string(fp,offset,36,hdr.exam_pro);
      else if(keynum==11)
         fread_string(fp,offset,20,hdr.referring_phy);
      else if(keynum==12)
         fread_string(fp,offset,20,hdr.attending_phy);
      else if(keynum==13)
         fread_string(fp,offset,2,hdr.imaging_modality);
      else if(keynum==14)
         fread_string(fp,offset,20,hdr.institution_name);
      else if(keynum==82)
         fread_string(fp,offset,10,hdr.nrml_crv_filenm);
      else if(keynum==15)
```

```
   fread_string(fp,offset,20,hdr.histog_crv_filenm);
else if(keynum==16)
   fread_string(fp,offset,10,hdr.acq_start_time);
else if(keynum==17)
      fread_string(fp,offset,2,hdr.data_type);
else if(keynum==18)
   fread_string(fp,offset,16,hdr.image_viewid);
else if(keynum==83)
   fread_string(fp,offset,3,hdr.object_key);
else if(keynum==75)
   fread_string(fp,offset,20,hdr.assos_parent_file);
else if(keynum==19)
   fread_string(fp,offset,10,hdr.imaging_device_name);
else if(keynum==20)
   fread_string(fp,offset,12,hdr.device_serial_number);
else if(keynum==21)
   fread_string(fp,offset,6,hdr.collimator_used);
else if(keynum==22)
   fread_string(fp,offset,8,hdr.software_version_num);
else if(keynum==23)
   fread_string(fp,offset,16,hdr.radiopharm_1);
else if(keynum==24)
   hdr.dosage_1=(short)fread_int(fp,offset,1);
else if(keynum==25)
   fread_string(fp,offset,16,hdr.radiopharm_2);
else if(keynum==26)
   hdr.dosage_2=(short)fread_int(fp,offset,1);
else if(keynum==27)
   hdr.isotope_imaging_mode=(char)fread_int(fp,offset,0);
else if(keynum==28)
   hdr.energy_win_1_center=(short)fread_int(fp,offset,1);
else if(keynum==29)
   hdr.energy_win_1_width=(short)fread_int(fp,offset,1);
else if(keynum==30)
   hdr.energy_win_2_center=(short)fread_int(fp,offset,1);
else if(keynum==31)
   hdr.energy_win_2_width=(short)fread_int(fp,offset,1);
else if(keynum==32)
   hdr.energy_win_3_center=(short)fread_int(fp,offset,1);
else if(keynum==33)
   hdr.energy_win_3_width=(short)fread_int(fp,offset,1);
else if(keynum==34)
   hdr.energy_win_4_center=(short)fread_int(fp,offset,1);
else if(keynum==35)
   hdr.energy_win_4_width=(short)fread_int(fp,offset,1);
else if(keynum==36)
   fread_string(fp,offset,6,hdr.pat_orientation);
else if(keynum==110)
   hdr.dir_orientation=(char)fread_int(fp,offset,0);
else if(keynum==37)
   hdr.spatial_res=fread_float(fp,offset);
else if(keynum==38)
   hdr.slice_thickness=fread_float(fp,offset);
else if(keynum==39)
   hdr.x_dim_of_data=(short)fread_int(fp,offset,1);
else if(keynum==40)
   hdr.y_dim_of_data=(short)fread_int(fp,offset,1);
else if(keynum==41)
   hdr.z_dim_of_data=(short)fread_int(fp,offset,1);
else if(keynum==86)
   hdr.num_sets_imgs=(short)fread_int(fp,offset,1);
else if(keynum==42)
   hdr.pixel_depth=(short)fread_int(fp,offset,1);
else if(keynum==85)
   hdr.true_color_flag=(char)fread_int(fp,offset,0);
else if(keynum==43)
   fread_string(fp,offset,20,hdr.uniformity_cor);
else if(keynum==44)
   hdr.acq_zoom_factor=fread_float(fp,offset);
else if(keynum==45)
   hdr.tot_counts=fread_float(fp,offset);
else if(keynum==46)
   hdr.time_frame=fread_int(fp,offset,2);
else if(keynum==47)
   hdr.tot_aqu_time=fread_int(fp,offset,2);
else if(keynum==48)
   hdr.max_val_frame_set=fread_float(fp,offset);
else if(keynum==49)
   hdr.min_val_frame_set=fread_float(fp,offset);
else if(keynum==87)
   hdr.curr_scale_factor=fread_float(fp,offset);
else if(keynum==50)
   hdr.rr_interval_time=(short)fread_int(fp,offset,1);
else if(keynum==112)
   hdr.rr_low_tol_time=(short)fread_int(fp,offset,1);
else if(keynum==113)
   hdr.rr_high_tol_time=(short)fread_int(fp,offset,1);
else if(keynum==51)
   hdr.per_cycle_imaged=(short)fread_int(fp,offset,1);
else if(keynum==52)
   hdr.num_cycles_accepted=(short)fread_int(fp,offset,1);
else if(keynum==53)
   hdr.num_cycles_rejected=(short)fread_int(fp,offset,1);
else if(keynum==54)
   hdr.approx_end_dias_frame=(short)fread_int(fp,offset,1);
else if(keynum==55)
   hdr.approx_end_sys_frame=(short)fread_int(fp,offset,1);
else if(keynum==56)
   hdr.approx_ef=fread_float(fp,offset);
else if(keynum==57)
   hdr.starting_angle=(short)fread_int(fp,offset,1);
else if(keynum==58)
   hdr.deg_of_rotation=(short)fread_int(fp,offset,1);
else if(keynum==59)
   hdr.dir_of_rotation=(char)fread_int(fp,offset,0);
else if(keynum==60)
   hdr.reorientation_type=(char)fread_int(fp,offset,0);
else if(keynum==61)
   hdr.lim_recon_start_frame=(short)fread_int(fp,offset,1);
else if(keynum==62)
   hdr.upper_win_gray_shade=(short)fread_int(fp,offset,1);
else if(keynum==63)
   hdr.lower_lvl_gray_shade=(short)fread_int(fp,offset,1);
else if(keynum==64)
   hdr.assoc_color_map=(short)fread_int(fp,offset,1);
else if(keynum==65)
   fread_string(fp,offset,20,hdr.cust_color_map);
else if(keynum==66)
   hdr.manipulated_image=(char)fread_int(fp,offset,0);
else if(keynum==67)
   hdr.axis_rot_cor=(short)fread_int(fp,offset,1);
else if(keynum==68)
   hdr.reorient_azimuth=(short)fread_int(fp,offset,1);
else if(keynum==69)
   hdr.reorient_elev=(short)fread_int(fp,offset,1);
else if(keynum==70)
   fread_string(fp,offset,16,hdr.filter_type);
else if(keynum==71)
   hdr.filter_order=(short)fread_int(fp,offset,1);
else if(keynum==72)
   hdr.filter_cutoff_freq=fread_float(fp,offset);
else if(keynum==73)
   fread_string(fp,offset,4,hdr.reconstr_type);
else if(keynum==74)
```

```
      hdr.attenuation_coef=fread_float(fp,offset);
    pos=pos+6;
    if (offset<min_offset)
      min_offset=offset;
    )
  pos=10;
  min_offset=999999;
  done=0;
  while (!done)
    (
    temp_char=(char)fread_int(fp,pos,0);
    if (temp_char=='C')
      (
      temp_char=(char)fread_int(fp,pos+1,0);
      if(temp_char=='A')
        (
        temp_char=(char)fread_int(fp,pos+2,0);
        if(temp_char=='L')
          (
          pos=pos+5;
          done=1;
          )
        )
      )
    pos++;
    )
  fread_string(fp,pos,6,hdr.calb);

  fseek(fp,(long int)8,0);
  fread(num_headers, sizeof(unsigned char), 1, fp);

#ifdef DEBUG
printf("hdr.pat_name = %s.\n", hdr.pat_name);
printf("hdr.pat_id = %s.\n", hdr.pat_id);
printf("hdr.birth_date = %s.\n", hdr.birth_date);
printf("hdr.radiopharm_1 = %s.\n", hdr.radiopharm_1);
printf("*num_headers = %i.\n", *num_headers);
#endif

  return hdr;
  )

void make_dicom_hdr(header_data dicom_hdr, adac_data adac_hdr,
                    char *adac_file_name, int offset)
  /*************************************************************************
  The MAKE_DICOM_HDR function converts an adac header structure into a dicom
  header structure
  Input:  adac_hdr            a structure of type data which contains adac
                              header info
          adac_file_name      file name including path of adac file
          offset              offset of image data in adac file
  Output:  This funcution returns a header_data structure which contains
           header info in a form that can easily be converted into a dicom file.
  *************************************************************************/
  (
  int temp_int;

  sprintf(dicom_hdr.patient_name,"%s^^^^",adac_hdr.pat_name);
  strncpy(dicom_hdr.patient_id,adac_hdr.pat_id,7);
  strncpy(dicom_hdr.patient_birthday,adac_hdr.birth_date,8);
  dicom_hdr.patient_sex=adac_hdr.pat_sex;
  strncpy(dicom_hdr.study_date,adac_hdr.acq_date,8);
  *(dicom_hdr.study_time)=*(adac_hdr.acq_start_time);
  *(dicom_hdr.study_time+1)=*(adac_hdr.acq_start_time+1);
  *(dicom_hdr.study_time+2)=*(adac_hdr.acq_start_time+3);
  *(dicom_hdr.study_time+3)=*(adac_hdr.acq_start_time+4);
  sprintf(dicom_hdr.ref_physician,"%s^^^^",adac_hdr.referring_phy);
  *(dicom_hdr.modality)='N';
  *(dicom_hdr.modality+1)='M';
  temp_int=(int)(strtol(adac_hdr.object_key,NULL,36));
  sprintf(dicom_hdr.series_uid,"%d",temp_int);
  dicom_hdr.series_num=temp_int;

  sprintf(dicom_hdr.frame_ref_uid,"%d",temp_int);

  sscanf(adac_hdr.calb,"%f",&(dicom_hdr.pixel_spacing));
  dicom_hdr.pixel_spacing=dicom_hdr.pixel_spacing*10;
  sscanf(adac_hdr.calb,"%f",&(dicom_hdr.slice_thickness));
  dicom_hdr.slice_thickness=dicom_hdr.slice_thickness*10;
  dicom_hdr.rows=adac_hdr.x_dim_of_data;
  dicom_hdr.columns=adac_hdr.y_dim_of_data;
  dicom_hdr.pixel_depth=adac_hdr.pixel_depth;
  dicom_hdr.high_bit=1; /******   Verify this value!!      *******/
  dicom_hdr.pixel_representation=0;
  if ((adac_hdr.data_type[0]=='D')||(adac_hdr.data_type[0]=='G'))
    (
    dicom_hdr.num_frames=adac_hdr.z_dim_of_data;
    dicom_hdr.num_planes=1;
    )
  else
    (
    dicom_hdr.num_frames=1;
    dicom_hdr.num_planes=adac_hdr.z_dim_of_data;
    )
  dicom_hdr.frame_time=adac_hdr.time_frame/1000.0;
  strncpy(dicom_hdr.radionucleotide,adac_hdr.radiopharm_1,17);
#ifdef DEBUG
  printf("dicom_hdr.patient_name = %s.\n", dicom_hdr.patient_name);
  printf("dicom_hdr.patient_id = %s.\n", dicom_hdr.patient_id);
  printf("dicom_hdr.patient_birthday = %s.\n", dicom_hdr.patient_birthday);
  printf("dicom_hdr.radionucleotide = %s.\n", dicom_hdr.radionucleotide);
#endif

  dicom_hdr.img_list=(img_list_element *)malloc(sizeof(img_list_element));
  strcpy(dicom_hdr.img_list->filepath,adac_file_name);
  dicom_hdr.img_list->blksize=(long)(adac_hdr.x_dim_of_data*adac_hdr.y_dim_of_data
                                     *adac_hdr.z_dim_of_data);
  dicom_hdr.img_list->offset=(long)(offset);
  dicom_hdr.img_list->next=NULL;

  strncpy(dicom_hdr.study_uid,adac_hdr.acq_date,8);
  dicom_hdr.study_uid[8]='\0';
  sprintf(dicom_hdr.frame_ref_uid,"%d",temp_int);
  sprintf(dicom_hdr.sop_class_uid,"%d",temp_int);
  sprintf(dicom_hdr.sop_inst_uid,"%d",temp_int);
  )
```

```
/* cti2dcm.c - has files which convert CTI images to intermediary
   DICOM data structure. Written by Marc Cruser, 21 Nov 1994      */

#define CTI
#define DICOM
#include "includes.h"

extern header_data output;

/*********************************************************/
int cti2struct (char *filename, FILE *infilep)
{
   Main_header mheader;
   Image_subheader sheader;
   struct MatDir entry, mlist[500];
   long matno;
   struct tm *time_date;
   time_t now;
   char tempstr[100];
   int byear, frame, gate, status, numpla;

   /* Determine number of images (total) in the input file  */
   /* numpla = total images selected for fixed frame & gate */
   frame = gate = 1;
   /* matlist puts image matrices into mlist and returns # of imgs */
   numpla = mat_list(infilep, mlist, MLSIZE);

   /* Read main header & check file type                    */
   status = mat_read_main_header(infilep, &mheader);
   if (mheader.file_type != FTYPE_IMAGE || numpla < 1) {
      Error("File is not an image file!");
      fclose(infilep);
      return(-1);
   }

   /* Read subheader */
   matno = 1 + (1<<16) + (1<<24); /* matnum for 1st sheader */
   status = mat_lookup(infilep, matno, &entry);
   mat_read_image_subheader(infilep, entry.strtblk, &sheader);

   /**** Set header values ****/
   strcpy(output.patient_id, mheader.patient_id);

   now = time(NULL);
   time_date = localtime(&now);
   sscanf(mheader.patient_age, "%s", tempstr);
   byear = 1900 + time_date->tm_year - atoi(tempstr);
   sprintf(output.patient_birthday, "%i0101", byear);

   if (mheader.patient_sex == 'U')
      output.patient_sex = 'O';
   else
      output.patient_sex = mheader.patient_sex;

   /* Study Date */
   sprintf(tempstr, "%i%02i%02i", mheader.scan_start_year,
           mheader.scan_start_month, mheader.scan_start_day);
   if (strlen(tempstr) < 8)
      sprintf(output.study_date, "19%s", tempstr);
   else
      strcpy(output.study_date, tempstr);

   /* Study Time */
   sprintf(tempstr, "%02i%02i", mheader.scan_start_hour,
           mheader.scan_start_minute);
   strcpy(output.study_time, tempstr);

   strcpy(output.modality, "PT");

   /* Getting a series number based on isotope in filename */
   output.series_num = 1; /* for now */

   output.pixel_spacing = sheader.pixel_size;

   output.slice_thickness = sheader.slice_width;

   output.rows = sheader.dimension_1;
   output.columns = sheader.dimension_2;

   switch(mheader.data_type) { /* Check these values! */
      case DTYPE_BYTES:
         output.pixel_depth = 16;
         output.high_bit = 16;
         output.pixel_representation = 1;
         break;
      case DTYPE_I2:
         output.pixel_depth = 16;
         output.high_bit = 16;
         output.pixel_representation = 1;
         break;
      case DTYPE_I4:
         output.pixel_depth = 32;
         output.high_bit = 32;
         output.pixel_representation = 1;
         break;
      case DTYPE_VAXR4:
         output.pixel_depth = 32;
         output.high_bit = 32;
         output.pixel_representation = 1;
         break;
      case DTYPE_SUNFL:
         output.pixel_depth = 32;
         output.high_bit = 1;
         output.pixel_representation = 1;
         break;
      case DTYPE_SUNIN:
         output.pixel_depth = 16;
         output.high_bit = 1;
         output.pixel_representation = 1;
         break;
      default:
         output.pixel_depth = 16;
         output.high_bit = 16;
         output.pixel_representation = 1;
         break;
   }

   output.num_frames = (mheader.num_frames < 1) ? mheader.num_gates
                                : mheader.num_frames;
   output.num_planes = mheader.num_planes;
/*
   output.frame_pointer = (unsigned short) 0;
*/

      sprintf(tempstr, "%s^^^^", mheader.patient_name);
      strcpy(output.patient_name, tempstr);

      sprintf(tempstr, "%s^^^^", mheader.physician_name);
      strcpy(output.ref_physician, tempstr);

      strcpy(output.radionucleotide, mheader.radiopharmaceutical);

   /* UID fillers */
   strcpy(output.study_uid, output.study_date);
```

```
    strcpy(output.series_uid, output.study_date);
    strcpy(output.frame_ref_uid, output.study_date);
    strcpy(output.sop_class_uid, output.study_date);
    strcpy(output.sop_inst_uid, output.study_date);

    return(0);
}
```

```
/* suitfunctions.c - These functions provide an X-Windows interface to the
   DICOM conversion program via the SUIT toolkit developed at Univ of Va.
   This program will convert ADAC, CTI, and Genesis file formats to the
   current DICOM standard (based on the Mallinkrodt software for now).
   The program as of 21 Nov 1994 converts the header data for ADAC and
   CTI files to a DICOM-ready data structure.
   Program written by Marc Cruser at the UTMC IPL, 21 Nov 1994.
   Some portions written by Sam Burgiss and CTI.                          */

#define SUIT
#define CTI
#define ADAC
#define DICOM
#include "includes.h"

char monthlist[12][10] = {
   "January", "February", "March", "April", "May", "June",
   "July", "August", "September", "October", "November", "December");

SUIT_object top_obj, menu_obj, patient_obj, study_obj;
SUIT_object month_obj, sex_obj, readme_obj, modality_obj;
SUIT_object browser_obj;
short patientwin_toggle = 0;
short studywin_toggle = 0;

header_data output; /* DICOM header data structure for currently open file*/
FILE *infilep;      /* input file pointer */
char input_filename[200];


void main(int argc, char *argv[])
{
   SUIT_init(argv[0]);
   SUIT_clearScreen();
   top_obj = SUIT_createBulletinBoard("top object");
   SUIT_setViewport(top_obj, VIEWPORT,
      SUIT_mapToParent(top_obj, 0.0, 0.0, 1.0, 1.0));
   SUIT_deluxeSetEnumString(top_obj, BORDER_TYPE, "fancy motif", GLOBAL);
   SUIT_performRedisplay();

   CreateMenu();

   SUIT_beginStandardApplication();
}

/********** Initialization functions **************/

void CreateMenu()
{
   SUIT_object next_obj;

#ifdef DEBUG
   printf("got to CreateMenu\n");
#endif
   menu_obj = SUIT_createMenuBar("top menu");
   SUIT_deluxeSetFont(menu_obj, FONT,
      GP_defFont("helvetica", "", 14), GLOBAL);

   next_obj = SUIT_createPullDownMenu("file menu");
   SUIT_setText(next_obj, LABEL, "File");
   SUIT_addChildToObject(menu_obj, next_obj);
   SUIT_addToMenu(next_obj, "Open ADAC", CreateADACBrowser);
   SUIT_addToMenu(next_obj, "Open CTI", CreateCTIBrowser);
   SUIT_addToMenu(next_obj, "Open Genesis", OpenGenesis);
   SUIT_addToMenu(next_obj, "Save As", SaveAs);
   SUIT_addToMenu(next_obj, "Print Header", PrintHeader);
   SUIT_addToMenu(next_obj, "Quit", Quit);

   next_obj = SUIT_createPullDownMenu("edit menu");
   SUIT_setText(next_obj, LABEL, "Edit");
   SUIT_addChildToObject(menu_obj, next_obj);
   SUIT_addToMenu(next_obj, "Patient Data", TogglePatientWin);
   SUIT_addToMenu(next_obj, "Study Data", ToggleStudyWin);

   next_obj = SUIT_createPullDownMenu("window menu");
   SUIT_setText(next_obj, LABEL, "Window");
   SUIT_addChildToObject(menu_obj, next_obj);
   SUIT_addToMenu(next_obj, "Image Header", ToggleHeaderWin);

   next_obj = SUIT_createPullDownMenu("help menu");
   SUIT_setText(next_obj, LABEL, "Help");
   SUIT_addChildToObject(menu_obj, next_obj);
   SUIT_addToMenu(next_obj, "Read Me", ToggleReadMe);
   SUIT_addToMenu(next_obj, "About", About);

   SUIT_addChildToObject(top_obj, menu_obj);
   SUIT_setViewport(menu_obj, VIEWPORT,
      SUIT_mapToParent(menu_obj, 0.0, 0.95, 0.2, 1.0));
}

/********** Menu Item functions ******************/

void CreateADACBrowser(SUIT_object menu_obj)
/* Creates SUIT-browser. Needs to be modified for
   multiple file selection. */
{
   char dir[200];

   if (patientwin_toggle || studywin_toggle)
      switch (SUIT_askWithCancel ("Save current file?",
            "Yes", "No")) {
         case REPLY_BUTTON1:
            SaveAs(menu_obj);
            break;
         case REPLY_BUTTON2:
            break;
         case REPLY_CANCEL:
         default:
            return;
            break;
      }

#ifdef DEBUG
printf("Creating Browser...\n");
#endif
   if (getwd(dir) == 0)  {
      SUIT_inform("Cannot access directory.");
      return;
   }
   browser_obj = SUIT_createFileBrowser(
                 "ADAC File Browser",
                 dir,                    /* current directory */
                 "Open ADAC File",
                 "Select image file to convert",
                 OpenADAC);
   SUIT_setText(browser_obj, FILE_FILTER, "*.img");
   SUIT_centerObjectOnScreen(browser_obj);
   SUIT_bringToFront(browser_obj);
}


void CreateCTIBrowser(SUIT_object menu_obj)
/* Creates SUIT-browser. Needs to be modified for
   multiple file selection. */
```

```
{
   char dir[200];

   if (patientwin_toggle || studywin_toggle)
      switch (SUIT_askWithCancel ("Save current file?",
              "Yes", "No")) {
         case REPLY_BUTTON1:
            SaveAs(menu_obj);
            break;
         case REPLY_BUTTON2:
            break;
         case REPLY_CANCEL:
         default:
            return;
            break;
      }

#ifdef DEBUG
printf("Creating Browser...\n");
#endif
   if (getwd(dir) == 0)  {
      SUIT_inform("Cannot access directory.");
      return;
   }
   browser_obj = SUIT_createFileBrowser(
                 "CTI File Browser",
                 dir,                    /* current directory */
                 "Open CTI File",
                 "Select image file to convert",
                 OpenCTI);
   SUIT_setText(browser_obj, FILE_FILTER, "*.img");
   SUIT_centerObjectOnScreen(browser_obj);
   SUIT_bringToFront(browser_obj);
}

void OpenADAC(SUIT_object browser_obj)
{
   adac_data adac_input, *adac_input_ptr;
   header_data *output_ptr;
   int num_hdrs;
   char errstr[100];

#ifdef DEBUG
printf("in OpenADAC...  ");
#endif
   strcpy(input_filename, SUIT_getText(browser_obj, CURRENT_VALUE));

   SUIT_destroyObject(browser_obj);

   if ((infilep = fopen(input_filename, "r")) == NULL) {
      sprintf(errstr, "Cannot open file %s.", input_filename);
      SUIT_inform(errstr);
      return;
   }

   adac_input_ptr = &adac_input;
   output_ptr = &output;
   read_adac(adac_input_ptr, input_filename, &num_hdrs);
   make_dicom_hdr(output_ptr, adac_input, input_filename, num_hdrs*1024);
   fclose(infilep);

   if (patientwin_toggle == 0) {
      if (studywin_toggle == 1) {
         studywin_toggle = 0;
         SUIT_destroyObject(study_obj);
         month_obj = NULL;
      }
      patientwin_toggle = 1;
      CreatePatientWin();
   }
   else {
      SUIT_destroyObject(patient_obj);
      CreatePatientWin();
   }

}


void OpenCTI(SUIT_object browser_obj)
{
   char errstr[100];
   int status;

#ifdef DEBUG
printf("in OpenCTI...  ");
#endif
   strcpy(input_filename, SUIT_getText(browser_obj, CURRENT_VALUE));

   SUIT_destroyObject(browser_obj);

   if ((infilep = fopen(input_filename, "r")) == NULL) {
      sprintf(errstr, "Cannot open file %s.", input_filename);
      SUIT_inform(errstr);
      return;
   }
   status = cti2struct(input_filename, infilep);

   fclose(infilep);
   if (status != 0) {
      sprintf(errstr, "Cannot convert file %s.", input_filename);
      Error(errstr);
   }

   if (patientwin_toggle == 0) {
      if (studywin_toggle == 1) {
         studywin_toggle = 0;
         SUIT_destroyObject(study_obj);
         month_obj = NULL;
      }
      patientwin_toggle = 1;
      CreatePatientWin();
   }
   else {
      SUIT_destroyObject(patient_obj);
      CreatePatientWin();
   }

}

void OpenGenesis(SUIT_object obj)
{
   SUIT_inform("This option will read a Genesis image for conversion");
}

void SaveAs(SUIT_object obj)
{
   SUIT_inform("Use this option to save as a DICOM file");
}

void PrintHeader(SUIT_object obj)
{
   char tmpname[100], tmpstr[100], name[64], year[8], month[8], day[8];
   FILE *tmpfile;
```

```
    SUIT_inform("Printing Image Header");

    tmpnam(tmpname);
    if ((tmpfile  = fopen(tmpname, "w")) == 0) {
       Error("Cannot open temporary file");
       return;
    }

    fprintf(tmpfile, "Image Header for %s\n\n", input_filename);
    GetName(output.patient_name, 1, name);
    fprintf(tmpfile, "Patient Last Name:     %s\n", name);
    GetName(output.patient_name, 2, name);
    fprintf(tmpfile, "Patient First Name:    %s\n", name);
    GetName(output.patient_name, 3, name);
    fprintf(tmpfile, "Patient Middle Name:   %s\n", name);
    GetName(output.patient_name, 4, name);
    fprintf(tmpfile, "Patient Name Prefix:   %s\n", name);
    GetName(output.patient_name, 5, name);
    fprintf(tmpfile, "Patient Name Suffix:   %s\n", name);
    fprintf(tmpfile, "Patient ID:            %s\n", output.patient_id);
    GetDate(output.patient_birthday, year, month, day);
    fprintf(tmpfile, "Patient Birth Date:    %s %s, %s\n", month, day, year);
    GetSex(output.patient_sex, name);
    fprintf(tmpfile, "Patient Sex:           %s\n\n", name);

    GetDate(output.study_date, year, month, day);
    fprintf(tmpfile, "Patient Study Date:    %s %s, %s\n", month, day, year);
    GetName(output.ref_physician, 1, name);
    fprintf(tmpfile, "Physician Last Name:   %s\n", name);
    GetName(output.ref_physician, 2, name);
    fprintf(tmpfile, "Physician First Name:  %s\n", name);
    GetName(output.ref_physician, 3, name);
    fprintf(tmpfile, "Physician Middle Name: %s\n", name);
    GetName(output.ref_physician, 4, name);
    fprintf(tmpfile, "Physician Name Prefix: %s\n", name);
    GetName(output.ref_physician, 5, name);
    fprintf(tmpfile, "Physician Name Suffix: %s\n", name);
    fprintf(tmpfile, "Study Time:            %c%c:%c%c\n",
           output.study_time[0], output.study_time[1],
           output.study_time[2], output.study_time[3]);
    GetModality(output.modality, name);
    fprintf(tmpfile, "Modality:              %s\n", name);
    fprintf(tmpfile, "Radionucleotide:       %s\n\n", output.radionucleotide);

    fprintf(tmpfile, "Series Number:         %i\n", output.series_num);
    fprintf(tmpfile, "Number of Rows:        %i\n", output.rows);
    fprintf(tmpfile, "Number of Columns:     %i\n", output.columns);
    fprintf(tmpfile, "Number of Frames:      %i\n", output.num_frames);
    fprintf(tmpfile, "Number of Planes:      %i\n", output.num_planes);
    fprintf(tmpfile, "Pixel Depth:           %i\n", output.pixel_depth);
    fprintf(tmpfile, "High Bit:              %i\n", output.high_bit);
    fprintf(tmpfile, "Pixel Representation:  %i\n", output.pixel_representation);
    fprintf(tmpfile, "Pixel Spacing:         %f\n", output.pixel_spacing);
    fprintf(tmpfile, "Slice Thickness:       %f\n", output.slice_thickness);
    fprintf(tmpfile, "Time per Frame:        %f\n", output.frame_time);

    fclose(tmpfile);
    sprintf(tmpstr, "lpr %s", tmpname);
    system(tmpstr);
    sprintf(tmpstr, "rm %s", tmpname);
    system(tmpstr);
}

void Quit (SUIT_object obj)
{
    SUIT_done(SAVE_SUI_FILE, EXIT_APPLICATION);
}
```

```
void TogglePatientWin(SUIT_object obj)
{
    if (patientwin_toggle == 0) {
       if (studywin_toggle == 1) {
          SaveStudyWin();
          studywin_toggle = 0;
          SUIT_destroyObject(study_obj);
          month_obj = NULL;
       }
       patientwin_toggle = 1;
       CreatePatientWin();
    }
    else {
       SavePatientWin();
       patientwin_toggle = 0;
       SUIT_destroyObject(patient_obj);
       month_obj = NULL;
       SUIT_paintObject(top_obj);
    }
}

void ToggleStudyWin(SUIT_object obj)
{
    if (studywin_toggle == 0) {
       if (patientwin_toggle == 1) {
          SavePatientWin();
          patientwin_toggle = 0;
          SUIT_destroyObject(patient_obj);
          month_obj = NULL;
       }
       studywin_toggle = 1;
       CreateStudyWin();
    }
    else {
       SaveStudyWin();
       studywin_toggle = 0;
       SUIT_destroyObject(study_obj);
       month_obj = NULL;
       SUIT_paintObject(top_obj);
    }
}

void ToggleHeaderWin(SUIT_object obj)
{
    static SUIT_object hdr_obj;
    static short toggle = 0;

    if (toggle == 0) {
       toggle = 1;
       hdr_obj =  CreateHeaderWin();
    }
    else {
       toggle = 0;
       SUIT_destroyObject(hdr_obj);
       hdr_obj = NULL;
    }
}

void ToggleReadMe(SUIT_object obj)
{
    static short toggle = 0;

    if (toggle == 0) {
       toggle = 1;
       CreateReadMe();
    }
```

```
    else {
        toggle = 0;
        SUIT_destroyObject(readme_obj);
        readme_obj = NULL;
    }
}

void About (SUIT_object obj)
{
    SUIT_inform(
        "UTMC Medical Image File Converter\nVersion 0.01 - Interface Demo");
}


/*** Object Creation functions ***/

void CreatePatientWin()
{
    SUIT_object next_obj;
    char newname[64], tmpstr[10];
    char year[10], month[10], day[10];
    void SetMonth(SUIT_object);

    patient_obj = SUIT_createBulletinBoard("patient data window");

    next_obj = CreateLabelTypeInBox("Patient ID");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.8, 0.5, 1.0));
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, output.patient_id);

    next_obj = CreateLabelTypeInBox("Patient Last Name");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.5, 0.8, 1.0, 1.0));
    GetName(output.patient_name, 1, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateLabelTypeInBox("Patient First Name");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.6, 0.5, 0.8));
    GetName(output.patient_name, 2, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateLabelTypeInBox("Patient Middle Name");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.5, 0.6, 1.0, 0.8));
    GetName(output.patient_name, 3, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateLabelTypeInBox("Patient Prefix");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.4, 0.5, 0.6));
    GetName(output.patient_name, 4, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateLabelTypeInBox("Patient Suffix");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.5, 0.4, 1.0, 0.6));
    GetName(output.patient_name, 5, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateDateBox("Date of Birth");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.2, 1.0, 0.4));
    strncpy(tmpstr, output.patient_birthday, 8);
    tmpstr[8] = 0;
    GetDate(tmpstr, year, month, day);
    SUIT_setText(SUIT_name("Date of Birth: year"), CURRENT_VALUE, year);
    SUIT_setText(month_obj, LABEL, month);
    SUIT_setText(SUIT_name("Date of Birth: day"),  CURRENT_VALUE, day);

    next_obj = SUIT_createLabel("Sex:");
    SUIT_addChildToObject(patient_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.0, 0.15, 0.2));

    sex_obj = SUIT_createButton("sex button", PressSexButton);
    SUIT_addChildToObject(patient_obj, sex_obj);
    SUIT_setViewport(sex_obj, VIEWPORT,
        SUIT_mapToParent(sex_obj, 0.15, 0.0, 0.5, 0.2));
    GetSex(output.patient_sex, newname);
    SUIT_setText(sex_obj, LABEL, newname);

    SUIT_addChildToObject(top_obj, patient_obj);
    SUIT_setViewport(patient_obj, VIEWPORT,
        SUIT_mapToParent(patient_obj, 0.0, 0.0, 1.0, 0.85));

    SUIT_setBoolean(patient_obj, SHRINK_TO_FIT, TRUE);
    SUIT_centerObjectOnScreen(patient_obj);
}

void CreateStudyWin(void)
{
    SUIT_object next_obj;
    char newname[64], tmpstr[64];
    char month[10], day[10], year[10];

#ifdef DEBUG
    printf("creating study window\n");
#endif
    study_obj = SUIT_createBulletinBoard("study data window");

    next_obj = CreateDateBox("Date of Study");
    SUIT_addChildToObject(study_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.8, 1.0, 1.0));
    strncpy(tmpstr, output.study_date, 8);
    tmpstr[8] = 0;
    GetDate(tmpstr, year, month, day);
    SUIT_setText(SUIT_name("Date of Study: year"), CURRENT_VALUE, year);
    SUIT_setText(month_obj, LABEL, month);
    SUIT_setText(SUIT_name("Date of Study: day"),  CURRENT_VALUE, day);

    next_obj = CreateLabelTypeInBox("Physician Last Name");
    SUIT_addChildToObject(study_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.0, 0.6, 0.5, 0.8));
    GetName(output.ref_physician, 1, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateLabelTypeInBox("Physician First Name");
    SUIT_addChildToObject(study_obj, next_obj);
    SUIT_setViewport(next_obj, VIEWPORT,
        SUIT_mapToParent(next_obj, 0.5, 0.6, 1.0, 0.8));
    GetName(output.ref_physician, 2, newname);
    SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, newname);

    next_obj = CreateLabelTypeInBox("Pixel Spacing");
```

```
   SUIT_addChildToObject(study_obj, next_obj);
   SUIT_setViewport(next_obj, VIEWPORT,
      SUIT_mapToParent(next_obj, 0.0, 0.4, 0.5, 0.6));
   SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, sprintf(tmpstr, "%f",
      output.pixel_spacing));

   next_obj = CreateLabelTypeInBox("Time per Frame");
   SUIT_addChildToObject(study_obj, next_obj);
   SUIT_setViewport(next_obj, VIEWPORT,
      SUIT_mapToParent(next_obj, 0.5, 0.4, 1.0, 0.6));
   SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, sprintf(tmpstr, "%f",
      output.frame_time));

   next_obj = CreateLabelTypeInBox("Radionucleotide");
   SUIT_addChildToObject(study_obj, next_obj);
   SUIT_setViewport(next_obj, VIEWPORT,
      SUIT_mapToParent(next_obj, 0.0, 0.2, 0.5, 0.4));
   SUIT_setText(SUIT_getChild(next_obj, 1), CURRENT_VALUE, output.radionucleotide);

   next_obj = SUIT_createLabel("Modality:");
   SUIT_addChildToObject(study_obj, next_obj);
   SUIT_setViewport(next_obj, VIEWPORT,
      SUIT_mapToParent(next_obj, 0.0, 0.0, 0.2, 0.2));

   modality_obj = SUIT_createButton("modality button", PressModalityButton);
   SUIT_addChildToObject(study_obj, modality_obj);
   GetModality(output.modality, newname);
   SUIT_setText(modality_obj, LABEL, newname);
   SUIT_setViewport(modality_obj, VIEWPORT,
      SUIT_mapToParent(modality_obj, 0.2, 0.0, 0.5, 0.2));

   SUIT_addChildToObject(top_obj, study_obj);
   SUIT_setViewport(study_obj, VIEWPORT,
      SUIT_mapToParent(study_obj, 0.0, 0.0, 1.0, 0.85));

   SUIT_setBoolean(study_obj, SHRINK_TO_FIT, TRUE);
   SUIT_centerObjectOnScreen(study_obj);
}

SUIT_object CreateHeaderWin()
{
   SUIT_object board, obj;
   char hdrtxt[500];

   sprintf(hdrtxt, "   Image Data for %s\n\n", input_filename);
   sprintf(hdrtxt, "%sRows:\t\t%i\n", hdrtxt, output.rows);
   sprintf(hdrtxt, "%sColumns:\t\t%i\n\n", hdrtxt, output.columns);
   sprintf(hdrtxt, "%sPixel Depth:\t%i\n", hdrtxt, output.pixel_depth);
   sprintf(hdrtxt, "%sHigh Bit:\t\t%i\n", hdrtxt, output.high_bit);
   sprintf(hdrtxt, "%sPixel Representation:\t%i\n\n",
         hdrtxt, output.pixel_representation);
   sprintf(hdrtxt, "%sNumber of Frames:\t%i\n", hdrtxt, output.num_frames);
   sprintf(hdrtxt, "%sNumber of Planes:\t%i\n", hdrtxt, output.num_planes);

   board = SUIT_createBulletinBoard("header board");

   obj = SUIT_createTextBox("header box", hdrtxt);
   SUIT_setInteger(obj, NUMBER_OF_LINES, 12);
   SUIT_addChildToObject(board, obj);
   SUIT_setViewport(obj, VIEWPORT,
      SUIT_mapToParent(obj, 0.0, 0.2, 1.0, 1.0));

   obj = SUIT_createButton("Header button", QuitBox);
   SUIT_setText(obj, LABEL, "Dismiss");
   SUIT_addChildToObject(board, obj);
   SUIT_setViewport(obj, VIEWPORT,
      SUIT_mapToParent(obj, 0.0, 0.0, 1.0, 0.2));

   SUIT_addChildToObject(top_obj, board);
   SUIT_setViewport(board, VIEWPORT,
      SUIT_mapToParent(board, 0.2, 0.2, 0.8, 0.8));
   SUIT_centerObjectOnScreen(board);
   SUIT_bringToFront(board);

   return(board);
}

void CreateReadMe (void)
{
   SUIT_object obj;

#ifdef DEBUG
      printf("creating readme\n");
#endif
      readme_obj = SUIT_createBulletinBoard("Read Me board");

      obj = SUIT_createTextEditor("Read Me viewer", NULL);
      SUIT_setText(obj, CURRENT_VALUE,
         SUIT_textOfFile("README"));
      SUIT_setBoolean(obj, READ_ONLY, TRUE);
      SUIT_addChildToObject(readme_obj, obj);
      SUIT_setViewport(obj, VIEWPORT,
         SUIT_mapToParent(obj, 0.0, 0.2, 1.0, 1.0));

      obj = SUIT_createButton("Read Me button", QuitBox);
      SUIT_setText(obj, LABEL, "Dismiss");
      SUIT_addChildToObject(readme_obj, obj);
      SUIT_setViewport(obj, VIEWPORT,
         SUIT_mapToParent(obj, 0.0, 0.0, 1.0, 0.2));

      SUIT_addChildToObject(top_obj, readme_obj);
      SUIT_setViewport(readme_obj, VIEWPORT,
         SUIT_mapToParent(readme_obj, 0.2, 0.2, 0.8, 0.8));
      SUIT_centerObjectOnScreen(readme_obj);
      SUIT_bringToFront(readme_obj);
}

SUIT_object CreateLabelTypeInBox(char namestr[])
/* This function is the building block for Patient
   and Study Window creation.                        */
{
   SUIT_object board, box, label;
   char tmpstr[100];

   board = SUIT_createBulletinBoard(namestr);

   sprintf(tmpstr, "%s: label", namestr);
   label = SUIT_createLabel(tmpstr);
   SUIT_addChildToObject(board, label);
   SUIT_setText(label, LABEL, namestr);
   SUIT_setViewport(label, VIEWPORT,
      SUIT_mapToParent(label, 0.0, 0.0, 0.4, 1.0));
   SUIT_centerInParent(label, 0.2, 0.5);

   sprintf(tmpstr, "%s: box", namestr);
   box = SUIT_createTypeInBox(tmpstr, Dummy);
   SUIT_addChildToObject(board, box);
   SUIT_setViewport(box, VIEWPORT,
      SUIT_mapToParent(box, 0.45, 0.0, 1.0, 1.0));
   SUIT_centerInParent(board, 0.6, 0.5);

   return board;
}
```

```
SUIT_object CreateDateBox(char namestr[])
/* Creates date fields for birthdate and study date. */
{
   SUIT_object board, obj;
   char tmpstr[100];

#ifdef DEBUG
printf("creating date box for %s\n", namestr);
#endif
   board = SUIT_createBulletinBoard(namestr);

   sprintf(tmpstr, "%s: label", namestr);
   obj = SUIT_createLabel(tmpstr);
   SUIT_setText(obj, LABEL, namestr);
   SUIT_addChildToObject(board, obj);
   SUIT_setViewport(obj, VIEWPORT,
      SUIT_mapToParent(obj, 0.0, 0.5, 0.1, 1.0));

   sprintf(tmpstr, "%s button", namestr);
   if (strcmp(namestr, "Date of Birth") == 0)
      month_obj = SUIT_createButton(tmpstr, PressBirthMonthButton);
   else
      month_obj = SUIT_createButton(tmpstr, PressStudyMonthButton);
   SUIT_addChildToObject(board, month_obj);
   SUIT_setText(month_obj, LABEL, "March");
   SUIT_setViewport(month_obj, VIEWPORT,
      SUIT_mapToParent(month_obj, 0.21, 0.5, 0.4, 1.0));

   sprintf(tmpstr, "%s: day", namestr);
   obj = SUIT_createTypeInBox(tmpstr, Dummy);
   SUIT_addChildToObject(board, obj);
   SUIT_setViewport(obj, VIEWPORT,
      SUIT_mapToParent(obj, 0.41, 0.5, 0.5, 1.0));

   sprintf(tmpstr, "%s: year label", namestr);
   obj = SUIT_createLabel(tmpstr);
   SUIT_addChildToObject(board, obj);
   SUIT_setText(obj, LABEL, "year:");
   SUIT_setViewport(obj, VIEWPORT,
      SUIT_mapToParent(obj, 0.51, 0.5, 0.6, 1.0));

   sprintf(tmpstr, "%s: year", namestr);
   obj = SUIT_createTypeInBox(tmpstr, Dummy);
   SUIT_addChildToObject(board, obj);
   SUIT_setViewport(obj, VIEWPORT,
      SUIT_mapToParent(obj, 0.61, 0.5, 0.75, 1.0));

   SUIT_setBoolean(board, SHRINK_TO_FIT, TRUE);

   return board;
}

SUIT_object CreateDropList(char *labelstr, char list[][10], int numitems,
                           SUIT_callbackFunctionPtr function)
{
   SUIT_object radio_obj;
   int i;
   char tmpstr[100];

#ifdef DEBUG
printf("in CreateDropList for %s...\n", labelstr);
#endif
   sprintf(tmpstr, "%s radio", labelstr);
   radio_obj = SUIT_createRadioButtons(tmpstr, function);
   SUIT_addChildToObject(top_obj, radio_obj);

   for (i = 0;  i < numitems; i++)
      SUIT_addButtonToRadioButtons(radio_obj, list[i]);

   sprintf(tmpstr, "%s button", labelstr);
   strcpy(tmpstr, SUIT_getText(SUIT_name(tmpstr), LABEL));
   SUIT_pressThisRadioButton(radio_obj, tmpstr);
   SUIT_setViewport(radio_obj, VIEWPORT,
      SUIT_mapToParent(radio_obj, 0.4, 0.2, 0.8, 0.6));

   SUIT_setBoolean(radio_obj, SHRINK_TO_FIT, TRUE);
   SUIT_bringToFront(radio_obj);

   return radio_obj;
}

/********** Save Data functions *****************/

void SavePatientWin(void)
/* Function called whenever a Patient window is closed. */
{
   char tmpstr[100];
   int year, month, day;

#ifdef DEBUG
printf("Saving Patient Window...\n");
#endif
   sprintf(output.patient_name, "%s^%s^%s^%s^%s",
      SUIT_getText(SUIT_name("Patient Last Name: box"), CURRENT_VALUE),
      SUIT_getText(SUIT_name("Patient First Name: box"), CURRENT_VALUE),
      SUIT_getText(SUIT_name("Patient Middle Name: box"), CURRENT_VALUE),
      SUIT_getText(SUIT_name("Patient Prefix: box"), CURRENT_VALUE),
      SUIT_getText(SUIT_name("Patient Suffix: box"), CURRENT_VALUE));
   year = atoi(SUIT_getText(SUIT_name("Date of Birth: year"), CURRENT_VALUE));
   month = GetMonthNum(SUIT_getText(month_obj, LABEL));
   day = atoi(SUIT_getText(SUIT_name("Date of Birth: day"), CURRENT_VALUE));
   sprintf(output.patient_birthday, "%4i%02i%02i", year, month, day);
   output.patient_sex = *SUIT_getText(sex_obj, LABEL);
   if (output.patient_sex == 'U')  output.patient_sex = 'O';

#ifdef DEBUG
printf("name = %s.\n", output.patient_name);
printf("birthday = %s.\n", output.patient_birthday);
printf("sex = %c.\n", output.patient_sex);
#endif
}

void SaveStudyWin(void)
/* Function called whenever a Study window is closed. */
{
   char tmpstr[100];
   int year, month, day;

#ifdef DEBUG
   printf("Saving Study Window...\n");
#endif

   year = atoi(SUIT_getText(SUIT_name("Date of Study: year"), CURRENT_VALUE));
   month = GetMonthNum(SUIT_getText(month_obj, LABEL));
   day = atoi(SUIT_getText(SUIT_name("Date of Study: day"), CURRENT_VALUE));
   sprintf(output.study_date, "%4i%02i%02i", year, month, day);

   sprintf(output.ref_physician, "%s^%s^%s^%s^%s",
      SUIT_getText(SUIT_name("Physician Last Name: box"), CURRENT_VALUE),
      SUIT_getText(SUIT_name("Physician First Name: box"), CURRENT_VALUE),
      "", "", "MD");
   sscanf(SUIT_getText(SUIT_name("Pixel Spacing: box"), CURRENT_VALUE), "%f",
         &output.pixel_spacing);
   sscanf(SUIT_getText(SUIT_name("Time per Frame: box"), CURRENT_VALUE), "%f",
```

```
        &output.frame_time);
   strcpy(output.radionucleotide,
        SUIT_getText(SUIT_name("Radionucleotide: box"), CURRENT_VALUE));

   strcpy(tmpstr, SUIT_getText(modality_obj, CURRENT_VALUE));
   if (strcmp(tmpstr, "CT") == 0)
      strcpy(output.modality, "CT");
   else if (strcmp(tmpstr, "PET") == 0)
      strcpy(output.modality, "PT");
   else if (strcmp(tmpstr, "Nuc Med") == 0)
      strcpy(output.modality, "NM");

#ifdef DEBUG
printf("study_date = %s.\n", output.study_date);
printf("name = %s.\n", output.ref_physician);
printf("pixel spacing = %f.\n", output.pixel_spacing);
printf("frame time = %f.\n", output.frame_time);
printf("Radionucleotide = %s.\n", output.radionucleotide);
printf("modality = %s.\n", output.modality);
#endif
}

/******** Button Action functions **************/

void PressBirthMonthButton(SUIT_object button_obj)
{
   SUIT_object obj;

   SUIT_setBoolean(button_obj, DISABLED, TRUE);
   obj = CreateDropList("Date of Birth", monthlist, 12, SetMonth);
}

void PressStudyMonthButton(SUIT_object button_obj)
{
   SUIT_object obj;

   SUIT_setBoolean(button_obj, DISABLED, TRUE);
   obj = CreateDropList("Date of Study", monthlist, 12, SetMonth);
}

void PressSexButton(SUIT_object button_obj)
{
   SUIT_object obj;
   char sexlist[3][10] = { "Male", "Female", "Unknown");

   SUIT_setBoolean(button_obj, DISABLED, TRUE);
   obj = CreateDropList("Sex", sexlist, 3, SetSex);
}

void PressModalityButton(SUIT_object button_obj)
{
   SUIT_object obj;
   char modlist[3][10] = { "CT", "PET", "Nuc Med");

   SUIT_setBoolean(button_obj, DISABLED, TRUE);
   obj = CreateDropList("Modality", modlist, 3, SetModality);
}

/********* Set DropList functions **************/

void SetMonth(SUIT_object radio_obj)
{
   char month[10];

   strcpy(month, SUIT_getEnumString(radio_obj, CURRENT_VALUE));
   SUIT_setText(month_obj, LABEL, month);
   SUIT_setBoolean(month_obj, DISABLED, FALSE);
   SUIT_destroyObject(radio_obj);
   SUIT_performRedisplay();
}

void SetSex(SUIT_object radio_obj)
{
   char sex[10];

   strcpy(sex, SUIT_getEnumString(radio_obj, CURRENT_VALUE));
   SUIT_setText(sex_obj, LABEL, sex);
   SUIT_setBoolean(sex_obj, DISABLED, FALSE);
   SUIT_destroyObject(radio_obj);
   SUIT_performRedisplay();
}

void SetModality(SUIT_object radio_obj)
{
   char modality[10];

   strcpy(modality, SUIT_getEnumString(radio_obj, CURRENT_VALUE));
   SUIT_setText(modality_obj, LABEL, modality);
   SUIT_setBoolean(modality_obj, DISABLED, FALSE);
   SUIT_destroyObject(radio_obj);
   SUIT_performRedisplay();
}

/********* Get DICOM Header Data functions **************/

void GetName(char dcmstr[], int whichpart, char newname[])
{
   char oldstr[64], newstr[64];
   int i, j, count, len;

   count =  i = j = 0;
   len = strlen(dcmstr);

   while ((count < whichpart) && (i < len)) {
      if (dcmstr[i] == '^') {
         count++;
         i++;
         newstr[j] = 0;
         strcpy(oldstr, newstr);
         j = 0;
      }
      else {
         newstr[j++] = dcmstr[i++];
      }
   }
   if (i == len)
      newstr[j] = 0;
   strcpy(newname, newstr);
}

void GetDate(char dcmstr[], char year[], char month[], char day[])
{
   strncpy(year, dcmstr, 4);
   year[4] = 0;

   if (dcmstr[4] == '0') {
      month[0] = dcmstr[5];
      month[1] = 0;
   }
   else {
      month[0] = dcmstr[4];
      month[1] = dcmstr[5];
      month[2] = 0;
   }
```

```
   strcpy(month, monthlist[atoi(month)-1]);

   if (dcmstr[6] == '0') {
      day[0] = dcmstr[7];
      day[1] = 0;
   }
   else {
      day[0] = dcmstr[6];
      day[1] = dcmstr[7];
      day[2] = 0;
   }
}

int GetMonthNum(char *month)
{
   int num = 0;
   int cmp = 1;

   while ((cmp != 0) && (num < 12)) {
      cmp = strcmp(monthlist[num], month);
      num++;
   }
   return(num);
}

void GetSex(char sex, char string[])
/* nothing kinky */
{
   if (sex == 'M')
      strcpy(string, "Male");
   else if (sex == 'F')
      strcpy(string, "Female");
   else
      strcpy(string, "Unknown");
}

void GetModality(char dcmstr[], char string[])
{
   if (strcmp(dcmstr, "CT") == 0)
      strcpy(string, "CT");
   else if (strcmp(dcmstr, "PT") == 0)
      strcpy(string, "PET");
   else if (strcmp(dcmstr, "NM") == 0) /* SPECT */
      strcpy(string, "Nuc Med");
}

/********* Various Utility functions **************/

void QuitBox (SUIT_object obj)
{
   SUIT_object parent;

   parent = SUIT_getParent(obj);
   SUIT_destroyObject(parent);
   parent = NULL;
}

void Error (char errstr[])
{
   char msg[100];

#ifdef DEBUG
printf("Error: %s\n", errstr);
#endif

   sprintf(msg, "Error!\n%s", errstr);
   SUIT_inform(msg);
   SUIT_done(DO_NOT_SAVE_SUI_FILE, EXIT_APPLICATION);
}

void Dummy(SUIT_object obj)
{
   SUIT_inform("This function has not been implemented yet.");
}
```

```
/* adac2dicom.h - These structure definitions are used by functions which
   convert ADAC images to DICOM format. By Sam Burgiss, Fall 1994.        */

typedef struct list_element
        {
        char *file_name;
        char *pat_name;
        unsigned char num_headers;
        char *mr_num;
        char *exam_pro;
        char *img_view;
        struct list_element *next_element;
        } list_element;

typedef struct adac_data
        {
        char pat_name[21];   /*usually last name then first name separated by a comma*/
        char pat_id[12];     /*mr number*/
        char pat_sex;        /*M=male  F=female*/
        short int pat_age;   /*usually not filled in*/
        short int pat_height;/*usually not filled in*/
        short int pat_weight;/*usually not filled in*/
        char pat_key[7];     /*usually not filled in*/
        char birth_date[9];  /*yyyymmdd*/
        char acq_date[9];    /*yyyymmdd*/
        char dose_admin_time[9];   /*usually not filled in*/
        char exam_key[9];          /*first four characters of file name*/
        char exam_pro[37];         /*study name*/
        char referring_phy[21];    /*usually not filled in*/
        char attending_phy[21];    /*usually not filled in*/
        char imaging_modality[2];  /*NM=nuclear medicine*/
        char institution_name[21];
        char nrml_crv_filenm[11];  /*usually not filled in*/
        char histog_crv_filenm[21];/*usually not filled in*/
        char acq_start_time[11];   /*hh:mm:ss  in 24 hr. time*/
        char data_type[3];         /*first character:  D=dynamic G=gated S=static*/
        char image_viewid[17];     /*series name*/
        char object_key[4];        /*last two characters of file name*/
        char assos_parent_file[21];/*usually not filled in*/
        char imaging_device_name[11];
        char device_serial_number[13];
        char collimator_used[7];
        char software_version_num[9];/*usually not filled in*/
        char radiopharm_1[17];
        short int dosage_1;
        char radiopharm_2[17];
        short int dosage_2;
        char isotope_imaging_mode;
        short int energy_win_1_center;
        short int energy_win_1_width;
        short int energy_win_2_center;
        short int energy_win_2_width;
        short int energy_win_3_center;
        short int energy_win_3_width;
        short int energy_win_4_center;
        short int energy_win_4_width;
        char pat_orientation[7];
        char dir_orientation;
        float spatial_res;     /*usually 1 but this is incorect should be equal to CALB*
/
        float slice_thickness; /*usually 10 but this is incorrect should be equal to CAL
B*/
        short int x_dim_of_data;
        short int y_dim_of_data;
        short int z_dim_of_data;/*can be # of frames or planes depending on study*/
        short int num_sets_imgs;
        short int pixel_depth;  /*number of bits of each data element*/
        char true_color_flag;
        char uniformity_cor[21];
        float acq_zoom_factor;
        float tot_counts;
        int time_frame;         /*time per frame*/
        int tot_aqu_time;
        float max_val_frame_set;
        float min_val_frame_set;
        float curr_scale_factor;
        short int rr_interval_time;
        short int rr_low_tol_time;
        short int rr_high_tol_time;
        short int per_cycle_imaged;
        short int num_cycles_accepted;
        short int num_cycles_rejected;
        short int approx_end_dias_frame;
        short int approx_end_sys_frame;
        float approx_ef;
        short int starting_angle;
        short int deg_of_rotation;
        char dir_of_rotation;
        char reorientation_type;
        short int lim_recon_start_frame;
        short int upper_win_gray_shade;
        short int lower_lvl_gray_shade;
        short int assoc_color_map;
        char cust_color_map[21];
        char manipulated_image;
        short int axis_rot_cor;
        short int reorient_azimuth;
        short int reorient_elev;
        char filter_type[17];
        short int filter_order;
        float filter_cutoff_freq;
        char reconstr_type[5];
        float attenuation_coef;
        char calb[7];       /*cm per pixel*/
        } adac_data;
```

```
/* adacfunctions.h
   This file contains the gcc-compatable function declarations
   for the part of the program for DICOM conversion which deal
   with ADAC file conversion. These functions were created by
   Sam Burgiss during his work at the UTMC IPL in Fall 1994. */

int fread_int(FILE *, int, int);
void fread_string(FILE *, int, int, char *);
float fread_float(FILE *, int);
void read_adac(adac_data, char [],int *);
void make_dicom_hdr(header_data, adac_data, char *, int);
```

```
/* convert_dcm.h - header data for program to convert from CTI/ADAC to DICOM */

#include <time.h>

/* Defined constants */
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifndef MAXPATHLENGTH
#define MAXPATHLENGTH 100
#endif
#ifndef MAXFILELENGTH
#define MAXFILELENGTH 40
#endif
#ifndef MAXPATLENGTH
#define MAXPATLENGTH 40
#endif


/* Structure definitions */

typedef struct img_list_element {
   char     filepath[MAXPATHLENGTH]; /* NULL-terminated */
   short    blksize;
   long     offset;
   struct img_list_element *next;
}  img_list_element;

typedef struct header_data {
   char     patient_name[64];     /* LastName^FirstNames^MiddleName^Prefix^SuffixNULL */
   char     patient_id[64];       /* NULL-terminated */
   char     patient_birthday[9];  /* yyyymmdd - no NULL */
   char     patient_sex;          /* M F O */
   char     study_date[9];        /* yyyymmdd - no NULL */
   char     study_time[5];        /* hhmm, 24hr - no NULL */
   char     ref_physician[64];    /* LastName^FirstNames^MiddleName^Prefix^SuffixNULL */
   char     modality[3];          /* no NULL */
   int      series_num;
   float    pixel_spacing;        /* in mm */
   float    slice_thickness;      /* in mm */
   short    rows;
   short    columns;
   short    pixel_depth;          /* # bits per pixel */
   short    high_bit;             /* which bit has greatest significance */
   short    pixel_representation; /* 0 for unsigned, 1 if signed */
   int      num_frames;
   short    num_planes;
   float    frame_time;           /* in seconds */
   char     radionucleotide[64];
   struct img_list_element *img_list;

   char     study_uid[64];        /* for now use date; study = a day's images */
   char     series_uid[64];       /* for now use the series number */
   char     frame_ref_uid[64];
   char     sop_class_uid[64];
   char     sop_inst_uid[64];
}  header_data;

/* Function Declarations */
int cti2struct(char *, FILE *);
```

```
/* functions.h - include file for temporary DICOM conversion
   interface functions - MCC */

void CreateMenu(void);

void CreateADACBrowser(SUIT_object);
void CreateCTIBrowser(SUIT_object);
void OpenADAC(SUIT_object);
void OpenCTI(SUIT_object);
void OpenGenesis(SUIT_object);
void OpenFORMAT(SUIT_object);
void SaveAs(SUIT_object);
void PrintHeader(SUIT_object);
void Quit(SUIT_object);
void About(SUIT_object);

void TogglePatientWin(SUIT_object);
void ToggleStudyWin(SUIT_object);
void ToggleHeaderWin(SUIT_object);
void ToggleReadMe(SUIT_object);

void CreatePatientWin(void);
void CreateStudyWin(void);
SUIT_object CreateHeaderWin(void);
void CreateReadMe(void);

SUIT_object CreateLabelTypeInBox(char *);
SUIT_object CreateDateBox(char *);
SUIT_object CreateDropList(char *, char [][], int,
            SUIT_callbackFunctionPtr);

void SavePatientWin(void);
void SaveStudyWin(void);

void PressBirthMonthButton(SUIT_object);
void PressStudyMonthButton(SUIT_object);
void PressSexButton(SUIT_object);
void PressModalityButton(SUIT_object);

void SetMonth(SUIT_object);
void SetSex(SUIT_object);
void SetModality(SUIT_object);

void GetName(char *, int, char *);
void GetDate(char *, char *, char *, char *);
int  GetMonthNum(char *);
void GetSex(char, char *);
void GetModality(char *, char *);

void QuitBox(SUIT_object);
void Error(char *);
void Dummy(SUIT_object);
```

```
/* includes.h - include file for all components of DICOM conversion
 *              program by MCC.  */
/* #define DEBUG */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <sys/types.h>
#include <sys/dir.h>

#ifdef SUIT
#include "suit.h"
#include "functions.h"
#endif

#ifdef SUNVIEW
#include <memory.h>
#include <sys/file.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/textsw.h>
#include <suntool/icon_load.h>
#include <suntool/seln.h>
#include <suntool/alert.h>
#define PIXEL_SEPARATION 16
#define TEXTSW_WIDTH 45
#endif

#ifdef CTI
#include "matrix.h"
#include "header_defs.h"
#endif

#ifdef ADAC
#include "fcntl_toolkit.h"
#include "scan_types.h"
#include "adac_types.h"
#include "img_objs.h"
#include "demo_objs.h"
#include "extra_objs.h"
#include "err_struct.h"
#include "adac_err.h"
#endif

#ifdef DICOM
#include "convert.h"
#include "adac2dicom.h"
#include <sys/stat.h>
#include <sys/param.h>
#include <math.h>
#endif

#define LMAX       1024
#define WIDTH      256
#define HEIGHT     256
#define SIZE       WIDTH*HEIGHT
#define HSIZE      256
#define NIMAGEMAX 310  /* enough for 10 beds, 31 images each ungated */
#define MLSIZE     1000 /* max size for mlist[] of matno's */

#define MAXPATHLENGTH 100
#define MAXPATLENGTH 40
#define MAXFILELENGTH 40
#define MAXMRLENGTH 11
#define MAXEXAMPROLENGTH 37
#define MAXIMGVIEWLENGTH 17
```

```
/* convert_dcm.h - header data for program to convert from CTI/ADAC to DICOM */

#include <time.h>

/* Defined constants */
#ifndef TRUE
#define TRUE 1
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifndef MAXPATHLENGTH
#define MAXPATHLENGTH 100
#endif
#ifndef MAXFILELENGTH
#define MAXFILELENGTH 40
#endif
#ifndef MAXPATLENGTH
#define MAXPATLENGTH 40
#endif


/* Structure definitions */

typedef struct img_list_element {
   char     filepath[MAXPATHLENGTH]; /* NULL-terminated */
   short    blksize;
   long     offset;
   struct img_list_element *next;
}  img_list_element;

typedef struct header_data {
   char     patient_name[64];     /* LastName^FirstNames^MiddleName^Prefix^SuffixNULL */
   char     patient_id[64];       /* NULL-terminated */
   char     patient_birthday[9];  /* yyyymmdd - no NULL */
   char     patient_sex;          /* M F O */
   char     study_date[9];        /* yyyymmdd - no NULL */
   char     study_time[5];        /* hhmm, 24hr - no NULL */
   char     ref_physician[64];    /* LastName^FirstNames^MiddleName^Prefix^SuffixNULL */
   char     modality[3];          /* no NULL */
   int      series_num;
   float    pixel_spacing;        /* in mm */
   float    slice_thickness;      /* in mm */
   short    rows;
   short    columns;
   short    pixel_depth;          /* # bits per pixel */
   short    high_bit;             /* which bit has greatest significance */
   short    pixel_representation; /* 0 for unsigned, 1 if signed */
   int      num_frames;
   short    num_planes;
   float    frame_time;           /* in seconds */
   char     radionucleotide[64];
   struct img_list_element *img_list;

   char     study_uid[64];        /* for now use date; study = a day's images */
   char     series_uid[64];       /* for now use the series number */
   char     frame_ref_uid[64];
   char     sop_class_uid[64];
   char     sop_inst_uid[64];
}  header_data;

/* Function Declarations */
int cti2struct(char *, FILE *);
```