Ho Group

STM Software

Code Manual

Ho Group
Clark Hall
Cornell University

Purpose

This document outlines the Ho group's STM software. It covers the overall software design and details how that design is implemented as code. This document is intended to serve two purposes. First, it should aid competent programmers in becoming familiar with the code in a reasonable amount of time. Second, it should serve as a reference guide for the details of the software implementation.

The reader should be familiar with the C programming language and the fundamentals of STM. Readers who are not familiar with the use of the STM software are advised to read the Ho Group STM Software User's Manual.

Revision History

This software is constantly being revised. The program is constantly evolving with the needs of the research group. Thus, it is important to note which version of the software is reflected in this document.

This document reflects the software as of: 01/01/98

Getting Started: Some General Information

The Ho group's STM software is written in the C programming language. It runs under Windows 3.2 (Win32). The software serves several important purposes. The software:

1. Provides a graphical user interface (GUI) for operating the STM.

2. Enables communication between a personal computer and the STM via digital i/o.

3. Handles data, storing it temporarily in memory and permanently in files.

4. Manipulates data for interpretation and presentation.

The next four segments of this document discuss how the proram accomplishes each of these major functions of the software. This document concludes with a reference of the C source files that make up the program. These four segments do not cover every detail of the STM software. Rather, they are intended to help the reader get started looking at and understanding the program.

Graphical User Interface (GUI)

The windows, dialog boxes, buttons, and other parts of the GUI are consistent with Microsoft Windows 3.2. A summary of relevant features of Win32 program design follows. For more detailed information, consult one of the many sources of information about programming for Win3x (One such resource is McCord's *Windows 3.1 Programmer's Reference*).

The entry point for all Windows 3x programs is the WinMain function. In the STM program, this function is can be found in the source file stm.c. The WinMain function performs some initialization and sets up the main application window. The main application window has five menus. These menus provide access to several dialog boxes which are used to control the STM and manipulate data. Many more dialog boxes may be accessed from these dialog boxes. This "nest" of dialog boxes constitutes the user interface for this program.

Windows is a message-driven operating system. The operating system sends messages to the windows and dialog boxes, carrying pertinent information such as "destroy this window" or "the user pressed the 'Scan' button." Each window and dialog box must process and respond to these messages. Usually, this is accomplished with a large switch statement. Messages for the main window in the STM software are handled by the function WndProc in stm.c. Messages for the various dialogs to messages are handled by relevant DialogProc's (for example, ScanDlg in scan.c).

From a design standpoint, the Windows-specific GUI implementation is a framework which the application is built around. The WndProc and DialogProc's handle messages and direct the program to STM-specific routines. Many of the source files are associated with individual dialog boxes. Some of the more important dialogs and there associated source files are:

scan dialog	scan.c
i/o control dialog	io_ctrl.c
data manipulation dialog	data.c

Each of these files contains a DialogProc which handles messages sent to the relevant dialog box.

It is important to distinguish between modal and modeless dialogs. Modal dialogs forbid the user to interact with any other windows or dialog boxes while a user is free to switch between modeless dialogs.

The dialog-boxes are designed with the application Dialog Editor.

Other routines that are closely tied to the GUI are the routines for updating the appearance of the dialog boxes, buttons, editable text, and other items on the screen. These routines are usually named repaint_*. There are also routines for enabling and disabling on screen controls with names of the form enable_*.

One important subtlety of the GUI implementation is the "out_smart" bug fix. A quick perusal of the code reveals the presence of the line

```
if( !out_smart) {...}
```

in many places. When the user adjusts a parameter that is associated with a decimal number (a voltage, for instance), it must be converted to a base-2, digital format. When dialog box receives a message indicating that such a parameter has been altered, the program converts the decimal representation of the parameter to a digital representation. The program updates the

editable text on the screen to reflect that change. However, when the editable text on the screen is changed, Windows once again sends a message to the dialog box indicating that the parameter has been adjusted. This could lead to an infinite loop in which the editable text is changed, leading to an internal change in representation, leading to the editable text being changed, etc. The out_smart lines alleviate this potential problem.

Digital i/o

As the user interacts with the GUI, the program must communicate the user's instructions to the STM. This is accomplished via the digital i/o.

The port(s) of the machine running the software should be connected to the 16-channel digital to analog converter which is, in turn, connected to the STM. Several Intel-specific routines are used to read and write data to and from the ports: inp, inpw, outp, outpw. Including conio.h allows access to these routines. The functions inp and outp allow communication of 8 bits at a time. outpw and inpw allow for 16 bits to be communicated at a time.

The state of the 16-channel digital to analog converter is stored in the array dac_data[] (declared in output.c):

```
unsigned int dac_data[16];
```

Each element of dac_data corresponds to a different channel. The channels are defined in dio.h. Some of them:

```
#define x_ch           2 /* fine x adjustment */
#define x_offset_ch    5 /* x offset (coarse x adjustment)*/
#define y_ch           3 /* fine y adjustment */
#define y_offset_ch    6 /* y offset (coarse y adjustment)*/
#define zo_ch          4
#define z_offset_ch    1
#define i_setpoint_ch  8
#define sample_bias_ch 0 /* sample bias (V)*/
```

Data may be sent out to the d-to-a converter with a short routine called dio_out. The code for that routine, from dio.c:

```
void dio_out(unsigned int ch,unsigned int n)
{
        outpw(cfg1,0x0000+out1);
        outpw(cfg1,CFG1_CONST+out1);
        outpw(porta,n+(ch << 12));
}
```

The first two lines reset the port. The third line actually sends the data. The first four bits are used to indicate which channel is being changed. The data is placed in the remaining 12 bits. The output dialog, whose DialogProc is in output.c, allows control at this level. dio_out is called using data and a channel selected by the user. This capability is included in the program for testing purposes and is seldom used during experiments.

dio_in_ch and dio_in_data combine to serve a similar role as dio_out, except for reading from rather than writing to the d-to-a converter. The input dialog, whose DialogProc is in input.c, serves a role analagous to the output dialog for reading data in from the d-to-a converter.

At a more sophisticated level, there are routines for adjusting x and y coordinates, changing the piezotube mode, adjusting the tip-sample bias, and so forth. These actions are available to the user via the i/o control dialog, whose DialogProc is defined in io_ctrl.c. The routines that

carry out these higher-level actions are in dio.c. Most of these routines have self-explanatory functions. For example, the routine bias() adjusts the tip-sample bias.

At the most sophisticated level, various actions that are accessible from the i/o control dialog are combined to perform complicated actions with the STM, the most important of which is scanning a sample surface. The routines relevant to scanning are in scan.c and scan2.c.

The scanning process is arbitrarily divided into two stages: pre-scanning and scanning. In broad terms, the algorithm for pre-scanning, from pre_scan() in scan.c, is:

(1) Set the tip-sample bias. (V)
(2) Set the tunneling current. (I)
(3) Set the piezo mode, in case it has not been set.
(4) Start dithering of channels 0 & 1, if necessary.
(5) Zero the fine x and fine y, i.e. move them to the center of the area to be imaged.
(6) "Synch Offsets" (In case the user forgot to hit "Send"). Set the offset x and offset y (coarse position controls) and the z offset.
(7) Move to the starting position (lower, left corner of area to be imaged, as it appears on screen). With the slow coordinate, overshoot the image edge and then move back. Do the same with the fast coordinate. This is intended to combat creep in the piezos.

Once the surface has been pre-scanned, a scan may begin. In broad terms, the algorithm for scanning, from scan in scan2.c, is:
(1) The tip moves up in the fast direction, taking measurements if appropriate.
(2) The data just read (in this_data) is stuffed into all_data[]
(3) For scan_dir == BOTH_DIR1, the tip is moved over one pixel. Or for scan_dir == BOTH_DIR2_POLAR, flip the bias. Otherwise, do nothing.
(4) The tip moves down in the fast direction, taking measurements if appropriate.
(5) Any data just read (in this_data) is stuffed into all_data.
(6) The tip is moved over one pixel in the slow direction.
(7) For scan_dir == BOTH_DIR2_POLAR, flip the bias. Otherwise, do nothing.
(8) Repeat until the the entire image is scanned.

If the scan_dir is BOTH_DIR1 (taking measurements in both directions to produce one composite image), then the tip will raster back and forth across the sample. For any other scan_dir, the tip will trace out a "comb-like" pattern: up,down,over,up,down,over,etc. During a scan, the program regularly checks to see if the ESC key has been pressed. If so, scanning is terminated.

Data Handling

Scanned STM data and scan settings are stored in structs of the type datadef. The form of the struct declaration from data.h is shown below.

```
typedef struct tagdatadef {

    // many, many fields

} datadef;
```

This struct has many, many data fields. Only a few are discussed here.

There are several data fields that relate to the geometry of the scan:

```
unsigned int  size;    /* length of one side in pixels; all 3D scans are
                                square */
unsigned int  x;       /* x coordinate of the center of scan region */
unsigned int  y;       /* y coordinate of the center of scan region */
unsigned int  z;
float         x_gain;
float         y_gain;
float         z_gain;
int           x_range;
int           y_range;
```

The scanned data is pointed to by:

```
float *ptr; /* where the data is stored */
```

Note that the data stored in a datadef may be from a three-dimensional (topographic) scan or a two-dimensional scan.

There are several fields that describe physical parameters:

```
struct commentdef sample_type;
struct commentdef dosed_type;
unsigned int  bias;    /* tip-sample bias */
int i_set_range;
int bias_range;
unsigned int amp_gain; /* The tunnelling current gain. NOTE: Setting the
                          tunnelling current gain in the program records,
                          but does not actually set, this value. The tunnelling
                          current gain must be set manually */
float tip_spacing;
float temperature;
```

These fields detail how the scan is performed:

```
READ_SEQ *read_seq;      /* array of reading sequences to carry out */
int read_seq_num;    /* number of reading sequences */
unsigned int  step;    /* number of bits moved between measurements */
int           scan_dir; /* scanning direction: FORWARD_DIR, BACKWARD_DIR,
                           BOTH_DIR1. BOTH_DIR2, etc */
int step_delay;          /* time to wait at a step before measurement */
int inter_step_delay; /* time to wait between inter_step movements
                         (movements without measurements) */
int inter_line_delay;
int digital_feedback; /* Boolean: whether or not the program's feedback
                         routine is used (in addition to electronics feedback */
int crash_protection;
int overshoot;           /* bitwise & with 0x1 to overshoot in fast dir,
                                  with 0x2 to overshoot in slow dir */
```

The following fields relate to reading data from and storing data in files.

```
int version;             /* specifies version of datadef struct */
char filename[FILE_NAME_SIZE];
char dep_filename[FILE_NAME_SIZE];
char full_filename[FULL_FILE_NAME_SIZE];
```

Additionally, there are fields that are specifically related to pulse data and spectroscopy data.

Here are several global, important instances of datadef's:

datadef *all_data[1000];
(declared in data.c) An array of up to 1000 sets of data taken on the most recent scan.

datadef *data;
(declared in data.c) The current data that the user has chosen to look at. Usually points to one of the datadef's in all_data[]

datadef **glob_data;
(declared in file.c) When data is loaded or saved, it is loaded into or saved from glob_data.

datadef *gendata[4]= {NULL, NULL, NULL, NULL};
(declared in image.c) Contains the four current images in the data manipulation 4x view.

datadef *scan_defaults_data[SCAN_NUM_DEFAULTS] (declared in scan.c) Four sets of scan settings are stored. The user can select between these using four radio buttons in the scan window. NOTE: In scan.h, SD is defined to be the current selected element of scan_defaults_data.

datadef's should be initialized with a call to alloc_data. Other routines for allocating and freeing memory associated with datadef's.

All variables that are related to time are given in microseconds.

Saving to and loading from files is accomplished with the various routines in file.c. More details about file.c are given in the file catalog at the end of this document.

Data Manipulation

Once a set of data has been acquired, the progam allows a user to manipulate the data so that it is suitable for analysis or presentation. Possible manipulations include: sumperimposing a grid over the image, taking the fourier transform of the data, subtracting one set of data from another and several other operations.

Code for the data manipulation dialog is in image.c. Several of the operations that can be carried out are coded in that file. The remainder of the operations are accessed via their own dialogs and are described separately (grid.c for superimposing grids is an example). Most of the data manipulations are straightforward mathematical operations on the data.

Code for operations that require more than one set of data obey the following convention: the index of the currently selected data is "a" and the index of the data indicated by pressing a button is "b". For example, the subtraction operation generates a set of data from gendata[a] - gendata[b]. To select data in quadrant 2 from data in quadrant 3, one would select quadrant 3 and then hit the "2" button in the arithmetic functions dialog.

For several of the operations, the program uses source files written by people outside the group. For example, ppmtogif.c.

File Reference

There are few comments in the Ho group STM software code. As a result, reading the code can be challenging. This portion of the manual is meant to serve as a reference for the *.c and *.h source files that make up the program. It is written as if it were a file full of comments.

The files are listed in alphabetical order. The list excludes *.h files which consist only of #define'd constants and function declarations. Additionally, functions of the form repaint_* and enable_* are not listed. The repaint_* functions are used to update editable texts, graphics, etc. on the screen. The enable_* functions are used to enable or disable buttons and other controls based on the value of a boolean status parameter. Finally, code written by people outside the group is not documented here (such as ppmtogif.c).

```
// **********
// anl.c
// **********

BOOL FAR PASCAL AnlDlg(HWND hDlg unsigned Message, WPARAM wParam,
                       LPARAM lParam)
// A big switch( Message) that handles messages sent to the analysis
// functions dialog, including those to various buttons and text fields
// (WM_COMMAND's).

float avg_z( SEL_REGION *region)
// returns the average value of z for all points within the region that
// the user has selected in the data manipulation window

float avg_z_2d( SEL_REGION *region1, SEL_REGION *region2)
// same as avg_z, except for two dimensional data

// **********
// coarse.c
// **********

BOOL FAR PASCAL CoarseDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                       LPARAM lParam)
// A big switch( Message) that handles messages sent to the coarse movement
// dialog, including various buttons and text fields ( WM_COMMAND's ).

// **********
// comment.c
// **********

BOOL FAR PASCAL CommentDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                       LPARAM lParam)
// Handles messages sent to the comment dialog
// Note that the comment dialog displays and allows editing of the global
// commentdef gcomment. So, gcomment should be appropriately defined before
// entering this dialog.
```

```
// **********
// comment2.c
// **********

// comment2 vs. comment
// comment2 is specific to the image manipulation dialog (image.c).
// comment2 is able to update itself based on which of the four images is
// selected. Also, comment2 is modeless while comment is modal.

BOOL FAR PASCAL Comment2Dlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// Handles messages sent to the comment2 dialog

// **********
// common.c
// **********

void CheckMsgQ()
// Routine for checking for Windows messages when we're hogging time
// (during scans, printing, etc.)

void init_listbox( HWND hDlg, int listboxid, LISTBOX *listbox)
// sets up listbox with appropriate items from listboxid

char *scan_freq_str( char *buf, unsigned int scale, unsigned int freq)
// looks up appropriate string, based on scale and freq

void fit_plane_simple( datadef *this_data, double *a, double *b, double *c)
// Fits a plane to the data in this_data using a least squares fit:
//                  z = a*x + b*y + c

void copy_data( datadef **dest, datadef **source)

void clear_area( HWND hDlg, int x1, int y1, int x2, int y2, COLORREF color)
// clears the rectangular area described by the coordinates and fills it
// with the specified color

float distance( float x1, float y1, float x2, float y2)

float interp_z( datadef *this_data, float row, float col)
// returns interpolated value of z at the point (row,col). for 3d data.

int bin_find( float *ptr, int find_min, int find_max, float goal)
// returns the index for which ptr[index] is nearest to goal
// looks for index in the range between find_min and find_max

char *bgr( datadef *this_data, float this_z, int *color32)

float linear_data( datadef *this_data, float row, float col)

int linear_data3( datadef *this_data, float row, float col)

int vert_int( float x1, float y1, float theta, float x2, float *y2)
```

```
int horiz_int( float x1, float y1, float theta, float *x2, float y2)

PRINT_EL *new_print_el( PRINT_EL **this_list, int type)

LOGPAL_EL *new_logpal_el( LOGPAL_EL **this_list, int index)

int new_count_el( COUNT_EL **this_list, int x, int y)

int remove_count_el( COUNT_EL **this_list, int x, int y)

void destroy_logpal( LOGPAL_EL **this_list)

void copy_logpal( LOGAPAL_EL **dest, LOGPAL_EL *src)

LOGPAL_EL *sort_logpal_els( LOGPAL_EL **this_list)

float float_bin_find( float *ptr, int find_min, int find_max, float goal)

float point_line_dist( float x1, float y1, float x2, float y2, float x3, float y3)

void calc_r_theta( GRID this_grid, float *r, float *theta)

void wait_cursor()
// Sets the cursor to the wait cursor

void arrow_cursor()
// Sets the cursor to the arrow cursor

void find_min_max( datadef *data, float *min_z, float *max_z)

int color_pal( GENPAL this_pal)

double calc_i_set( unsigned int i_setpoint, int i_set_range,
                                    unsigned int tip_gain)

void destroy_count( COUNT_DATA *list)

int index2d( datadef *this_data, int index)

void init_count_data( COUNT_DATA *this_data)
// Initializes the count_datadef pointed to by this_data

void copy_count( COUNT_DATA **dest, COUNT_DATA source)

void free_count( COUNT_DATA **this_data)

int rect_intersect_gen( float r1x1, float r1y1, float r1x2, float r1y2,
                                float r2x1, float r2y1, float r2x2, float r2y2 )

void paint_circles_gen( HDC hDC, int x_offset, int y_offset,
                                float pixel_size,
                                COUNT_EL *this_list, int size, COLORREF color)
```

```
// **********
// count.c
// **********

// Note that counting data is distinct from scanned data. Counting data
// is stored is structs of the type count_datadef, defined in data.h.

// important global variables
BOOL CountOn = FALSE; /* If CountOn==TRUE, clicking in image will add dot
                         and increment the total count for the current color */
COUNT_DATA count_data[4];   /* counting data for the 4x images in gendata[] */

BOOL FAR PASCAL CountDlg(HWND hDlg unsigned Message, WPARAM wParam,
                         LPARAM lParam)
// A big switch( Message) that handles messages sent to the counting
// dialog various buttons and text fields ( WM_COMMAND's ).

// This routine is responsible for handling the counting dialog. However,
// actual counting is done through the data manipulation dialog with calls
// to countit() in image.c.

void init_count()
// Initializes each of the four count_datadef's in count_data[]

// Arrows in the counting dialog are for shifting the entire set of counter
// dots vertically or horizontally. Internally, the program uses
// count_horiz_shift and count_vert_shift to do this
void count_horiz_shift( HWND hDlg, int shift)

void count_vert_shift( HWND hDlg, int shift)

// **********
// data.c
// **********

void alloc_data( datadef **dataptr, int type, int size, int x_type,
                 int y_type, int seq_size)
// Allocates memory and initializes fields in **dataptr

void alloc_data_ptrs( datadef **dataptr, int type, int size, int x_type,
                 int y_type, int seq_size)
// Allocates memory for (*dataptr)->ptr based on the type of data
// (topography, tunneling spectroscopy, etc).

void free_data( datadef *(*dataptr))
// frees dataptr and all ptrs within the struct

void free_data_ptrs( datadef *(*dataptr))
// frees all fields of dataptr struct that are not sub-structs, but are ptrs

void alloc_data_seq( datadef **dataptr, int seq_size)
// Allocates memory for (*dataptr)->read_seq and some other initialization.
```

```
void free_data_seq( datadef **dataptr)
// frees the read_seq field of dataptr

// **********
// dio.h
// **********

// define's

// input channels
#define zi_ch          1  /* z input with feedback on */
#define i_in_ch        0  /* i input with feedback off */
#define tip_ch         0  /* tip channel during approach */

// output channels (also correspond to indices of dac_data[] )
#define x_ch           2       /* fine x coordinate */
#define x_offset_ch    5       /* coarse x coordinate */
#define y_ch           3       /* fine y coordingate */
#define y_offset_ch    6       /* coarse coordinate */
#define zo_ch          4
#define z_offset_ch    1
#define i_setpoint_ch  8
#define sample_bias_ch 0
#define test_ch        9       /* for testing purposes */

#define range_ch       12
#define AD_ch          13
#define mode_ch        14      /* piezo mode */
#define feedback_ch    14      /* feedback on/off */
#define hold_ch        14      /* having hold on is same as feedback off, v.v */
#define retract_ch     14
#define gain_ch        14      /* z offset gain: 0.1, 1, 10 */

// The following routines can be redefined, if the macro NO_DIO_CARD is defined,
// i.e. if you want to run w/o a dio card installed:

#define dio_out( CH, VAL)
#define dio_start_clock( TIME)
#define dio_wait_clock()

// **********
// dio.c
// **********

int dio_dither_status( int ch)
// Returns the dither status (1=on, 0=off) of the channel ch.

void dio_dither( int ch, int status)
// Sets the dither status of channel ch to status (1=on, 0=off).

void dio_set_registers()
// Initializes i/o registers.
```

```
void dio_init()
// Initializes 16 channels of electronics and i/o registers.

int get_range( int ch)
// returns 1 if range of channel ch is +-5 V, 2 if +-10 V

void set_range( int ch, int range)
// Resets the STM-range to +-10 V. If variable range is 1, then sets to +-5 V.

void dio_feedback( int on)
// Turns the feedback on/off based on boolean interpretation of parameter on.

unsigned int *dio_blk_setup( unsigned int ch, unsigned int wave,
                    unsigned int n, float wave_range, int dir)

void dio_out( unsigned int ch, unsigned int n)
// Sends n to channel ch via the output register.

// Functions dio_blk_free and dio_blk_out are only called from coarse.c
// and are of questionable usefulness.
void dio_blk_free( unsigned int *data)
void dio_blk_out( unsigned int n, unsigned int *data)

unsigned int *para_move_setup( unsigned int step, unsigned int size,
                                    unsigned int offset)
// This function is not called. It appears to have been copied as
// stair_move_setup(...), which does get called.

unsigned int para_size( unsigned int n)
// This function is not called. It appears to be superceded by stair_size.

void para_free( unsigned int *ptr)
// This function is not called.

unsigned int *stair_move_setup( unsigned int step, unsigned int offset)
// Sets up wave form for steps taken by piezos

unsigned int stair_size( unsigned int n)

void stair_free( unsigned int *ptr)

unsigned int *tip_setup( unsigned int xn, unsigned int zon,
                              unsigned int xstep, unsigned int zostep, int dir)
// Sets up wave form for steps taken by piezos during giant steps of tip approach

unsigned int *tip_zo_setup( unsigned int n)
// Sets up wave form for steps taken by piezos during baby steps of tip approach

void tip_free( unsigned int *data)

void dio_in_ch( unsigned int ch)

void dio_in_data( unsigned int *data)
```

```
double dio_read( unsigned int n)

double dtov( unsigned int data, int range)

double dtov_len( int data, int range)

unsigned int vtod( double v, int range)

char *dtob( unsigned int data)

unsigned int btod( char *str)

double in_dtov( unsigned int data)

int in_vtod( double data)

char *in_dtob( unsigned int data)
// data is a base-10 integer
// converts data to a string in binary
// Example data = 2177 is changed to "0000 1000 1000 0001"

void mode( int m)

void tip_offset( unsigned int data)

void move_to( unsigned int ch, unsigned int datai, unsigned int dataf)

void move_to_speed( unsigned int ch, unsigned int datai, unsigned int dataf,
                    int time, int digital, int Imin, int Imax,
                    int digital_feedback_max, int digital_feedback_reread)

void move_to_timed( unsigned int ch, unsigned int datai, unsigned int dataf,
                          int time)

unsigned int move_to_protect( unsigned int ch, unsigned int datai,
                  unsigned int dataf, int time, int crash_protection,
                  float limit_percent)

unsigned int move_to_protect2( unsigned int ch, unsigned int datai,
                  unsigned int dataf, int time, int crash_protection,
                  int digital, int Imin, int Imax, int digital_abort,
                  int force_it, int digital_feedback_max,
                  int digital_feedback_reread)
// datai, dataf are initial and final positions
// time is the delay (in microsecs) between each step during movement

void move_to2( unsigned int ch, unsigned int datai, unsigned int dataf,
                  unsigned int steps)

// ************************
// routines for setting gain on coordinate signals

void set_gain( unsigned int x_gain, unsigned int y_gain, unsigned int z_gain,
                  unsigned int z2_gain)
```

```
void set_x_gain( unsigned int x_gain)

void set_y_gain( unsigned int y_gain)

void set_z_gain( unsigned int z_gain)

void set_z2_gain( unsigned int z2_gain)

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


void hold( int hold_on)
// ...as in "sample and hold." Having the hold_on is the same thing as
// having the feedback off. This is only called in one instance, during the tip
// approach.

void retract( int retract_on)

void adc_delay()
// When reading data in from the STM, need to allow about 15 microsecs
// for completion of analog to digital conversion.

void tip_current( unsigned int data)

void bias( unsigned int data)
// set the tip-sample bias to be data

void ramp_bias( unsigned int data, int time, int skip, int bits)

void ramp_ch( unsigned int ch, unsigned int data, int time, int skip, int bits)

int dio_digital_feedback( int Imin, int Imax, int digital_feedback_max,
                                    int digital_feedback_reread)
// this routine monitors the current I and continues to run as long as
// I < Imin or I > Imax. digital_feedback_max is the maximum number of times
// that the routine will check the current before aborting. If digital_feeback_reread
// is nonzero, the routine will double check that Imin < I < Imax before exiting

unsigned int flipped_bias( unsigned int bias)

// **********
// dep.c
// **********

BOOL FAR PASCAL DepDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                    LPARAM lParam)
// A big switch(Message) that handles messages sent to the deposition
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL), various buttons
//  and text fields ( WM_COMMAND's ).
```

```
BOOL FAR PASCAL DepPulseDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch(Message) that handles messages sent to the pulse options
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL), various buttons
//  and text fields ( WM_COMMAND's ).

void dep_pulse2( HWND hDlg)

void dep_pulse3( HWND hDlg)

void set_data( datadef *this_data)

void get_data( datadef *this_data)

void dep_box( HWND hDlg)

void dep_lines( HWND hDlg)

void set_write_cond( HWND hDlg, unsigned int write_bias,
                        unsigned int write_i_setpoint)

void set_move_cond( HWND hDlg)

void set_scan_cond( HWND hDlg)

static void dep_set_title( HWND hDlg)

// **********
// etest.c
// **********

BOOL FAR PASCAL EtestDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch( Message) that handles messages sent to the electronics
// test dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various
// buttons ( WM_COMMAND's ).

static void calc_digi_ch( HWND hDlg)

static void etest_ramp( HWND hDlg)

static void etest_test_input( HWND hDlg)

static void etest_test_output( HWND hDlg)

static void etest_calibration( HWND hDlg)

// **********
// file.c
// **********
```

```
// global variables related to files
char *current_dir_stm; /*...and others. Current paths */
char *initial_dir_stm; /*...and others. Initial paths */
char *current_file_stm; /*...and others. Current file name: MMDDYY#.ext */
int file_stm_count; /*..and others. Keeps track of # being used in filename */

void init_file( char *dir, char *filename, char *ext, int *count)
// Sets the filename by date: MMDDYY##.ext where the ## is determined by
// count. Count may be incremented in order to produce a unique filename.

BOOL file_open( HWND hWnd, HANDLE hInstance, int file_type, int allow_all,
                char* this_file)
// Allows user to select and open a file. Returns TRUE if successful.

BOOL file_save_as( HWND hWnd, HANDLE hInstance, int file_type)
// Allows user to save data. (if it's image data, glob_data in particular)
// Returns TRUE if successful.

int load_image_old( char *filename)
// Load an image that is "old format" (no magic & version).  Returns 0 if
// unsuccessful. If successful, returns non-zero value and places image
// data in glob_data datadef.

int load_image( char *filename, int compressit)
// Load an image by filename (a path+filename, really). Returns 0 if
// unsuccessful. If successful, returns non-zero value and places image
// data in glob_data datadef.

int load_image_fp( FILE *fp, int compressit))
// Load an image by file pointer.  Returns 0 if unsuccessful. If successful,
// returns non-zero value and places image data in glob_data datadef.
// Uses version number of file to determine what fields will be present.

int save_image( char *filename)
// Save glob_data as filename (a path+filename, really). Returns if
// successful.

int save_image_fp( FILE *fp)
// Save glob_data to fp file pointer.  Returns 1 if successful.
// Uses version number of file to determine what fields will be present.

void save_init( char *fname)
// Save important global variables to fname

void load_init( char *fname)
// (Mostly) initialization of global variables based on what's in fname

void init_dirs()
// Reads in directory paths from initialization file. Sets the corresponding
// global path variables (current_dir_stm, current_dir_pal, etc.).
```

```
// **********
// fine.c
// **********

BOOL FAR PASCAL FineDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch( Message) that handles messages sent to the fine movement
// dialog, including various buttons and text fields ( WM_COMMAND's ).

void pre_fine()

void post_fine()

// **********
// grid.c
// **********

// Two structs that are relevant to this dialog that are declared in data.h
//  are GRID AND GRID_LINE.

// The four grids that correspond to the 4X view in the data manipulation
// window are stored in the global variable:
GRID grids[4];

BOOL FAR PASCAL GridDlg(HWND hDlg unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch( Message) that handles messages sent to the grid
// dialog, including those to various buttons and text fields (WM_COMMAND's).

void copy_grid( GRID *dest, GRID *source)

void init_grids()
// Grid lines are initially hidden, have theta (from horizontal) = 0, have
// r (offset from bottom) = 1, and are separated by GRID_INIT_DIST

void grid_get_line( int line)

void grid_get_clip_line()

void grid_match3( int one, int two, int three)
// matches the column three grid's angle and separation to the
// column one and column two grids (presently, only called with parameters
// (0,1,2))

int grid_hidden( int this_image)
```

```
// **********
// heater.c
// **********

BOOL FAR PASCAL HeaterDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch( Message) that handles messages sent to the heater
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

// **********
// image.c
// **********

BOOL FAR PASCAL ImageDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch( Message) that handles messages sent to the Data Manipulation
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

// Note that the Data Manipulation dialog receives messages from other
// dialogs to handle the updating of the images. In a sense, the Data
// Manipulation dialog facilitates the functions of the Grid dialog,
// the Counting dialog, and the Selection Functions dialog.

void read_smooth_values( HWND hDlg, int reverse)

void image_sel_cut( SEL_REGION *region, datadef *dest)

void image_sel_copy( SEL_REGION *region, datadef **dest, datadef, **source)

void linearize( int num)

int fit_plane_region( int num, SEL_REGION *region, double *a, double *b,
                                                    double *c)

void fit_plane( int num, SEL_REGION *region)
// fits a plane to the region of gendata[num] of the form
//         z = ax + by - c
// and replaced the values of z in gendata[num] by the values
// of z relative to this plane

void take_log( int num)
// replaces the values of z in gendata[num] by base 10 log(z)

void project3D( int num)

void take_deriv( HWND hDlg, int num, int dir)

void invert( int num)
// replaces the values of z in gendata[num] by -z
```

```
void drift_comp( int number)
// maps square data onto a parallelogram to compensate for drift
// This algorithm comes from Sung-il Park's thesis

int rect_intersect( int image, RECT area)

void normalize( HWND hDlg)

void smooth( HWND hDlg, int num)

void calc_bitmap( int num)

void calc_bitmap_gen( datadef *this_image, unsigned char **this_bitmap,
                      int *size);

int time_index( int number, float t)

void im_do_scrolls( HWND hDlg, int number)

int floatcmp( float *f1, float *f2)

void free_dtrans( DTRANS *dtrans)

void read_dtrans( DTRANS *dtrans, char *filename)

void copy_filter( float dest[], float *src)

void paint_line( HWND hDlg, SEL_REGION *region)

void paint_rect( HWND hDlg, SEL_REGION *region)

void im_invert( HWND hDlg, SEL_REGION *region)

void paint_ellipse( HWND hDlg, SEL_REGION *region)

void paint_cross( HWND hDlg, SEL_REGION *region, HBITMAP cross)

void find_clip_rect( HWND hDlg, RECT *r)

void save_gif(char *fname);

void square_coords( SEL_POINT pt1, SEL_POINT pt2, int image)

void circle_coords( SEL_POINT pt1, SEL_POINT pt2, int image)

void inc_pt_dist( int x1, int *x2, int dist)

int pt_in_region( int x, int y, SEL_REGION *region)

int rev_size( int size)

int image2pixel( float image_x, float image_y, int *pixel_x, int *pixel_y,
                 int image)
```

```
static int draw_grid_line( HWND hDlg, float r, float theta, int this_image)

void notify_all()

void image_set_title( HWND hDlg)

int color_index( datadef *this_data, int row, int column)

void new_image( HWND hDlg)

void reset_image( HWND hDlg, int num)

static void image_cut_profile( SEL_REGION *region)

static void countit( HWND hDlg, int this_image, int num, SEL_POINT temp_pt)

// **********
// info.c
// **********

BOOL FAR PASCAL InfoDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                         LPARAM lParam)
// Handles messages sent to the information dialog

// **********
// infodep.c
// **********

BOOL FAR PASCAL InfodepDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                            LPARAM lParam)
// Handles messages sent to the deposition information dialog

// **********
// infospc.c
// **********

BOOL FAR PASCAL InfospcDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                            LPARAM lParam)
// Handles messages sent to the this dialog

// **********
// input.c
// **********

BOOL FAR PASCAL InputDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// Handles messages sent to the input dialog.
```

```
// **********
// io_ctrl.c
// **********

BOOL FAR PASCAL IOCtrlDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                           LPARAM lParam)
// A big switch( Message) that handles messages sent to the i/o control
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

// The i/o control dialog gives the user direct control over just about every
// facet of the digital i/o, including fine and coarse postion, feedback,
// tip retraction and so on. An important note, radio buttons exist for
// choosing the tunnelling current gain. Selecting these radio buttons records
// but does not actually change the tunnelling current gain. This must be done
// manually.

// **********
// lockin.c
// **********

BOOL FAR PASCAL LockinDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                           LPARAM lParam)
// A big switch( Message) that handles messages sent to the lockin
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

// **********
// logpal.c
// **********

BOOL FAR PASCAL LogpalDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                           LPARAM lParam)
// A big switch( Message) that handles messages sent to the logical pallette
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

static void remove_logpal_el( HWND hDlg, LOGPAL_EL *remove_el)

// **********
// masspec.h & masspec.c
// **********

// Mass spectroscopy was superceded by tunneling spectroscopy. Little or none
// of the code in these files is ever compiled or executed.
```

```
// **********
// oscill.c
// **********

BOOL FAR PASCAL OscillDlg(HWND hDlg unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the oscilloscope
// dialog, including scrolling (WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

// **********
// output.c
// **********

// important global variables
unsigned int dac_data[16];   /* Reflects the current state of the electronics.
                                The channels that correspond to the indices
                                of dac_data are defined in dio.h */

BOOL FAR PASCAL OutputDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the output
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

// **********
// pal.c
// **********

BOOL FAR PASCAL PalDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the palettes
// dialog, including various buttons ( WM_COMMAND's ).

void animate_pal( HWND hDlg, int del);

void apply_gamma( unsigned char *dacbox);

void MakeColorLUT(int *pLut,     /* array to hold LUT */
                  double gamma); /* exponent of stretch */

int spindac( int direction, int step, unsigned char *dacbox);
// Originally, this shifted the palette by step. Repeated calls to this would
// "spin" the colors of the image in a manner that some might call "trippy."
// It is currently broken, but a call of spindac(0,1, ...) will stuff the
// dacbox into the Windows logical palette.
```

```
void ramp( int colindex, int mincol, int maxcol, int start, int end
        unsigned char* dacbox)
// for RGB, colindex = 0 is R, 1 is G, 2 is B
// fills in the colindex (R,G, or B) segment from index start to end with
// shades between mincol and maxcol

void set_Plasma_palette(unsigned char *dacbox);
// fills in dacbox with shades of colors red->orange->yellow->green->blue

void get_rand_color( unsigned char *color)

void get_rand_high_color( unsigned char *color)

void set_random_pal( unsigned char *dacbox);

void set_random_one_pal( unsigned char *dacbox);

void histogram()

void equalize( datadef *this_data, float min_z, float max_z, float *fhist)

// **********
// pre_scan.c
// **********

BOOL FAR PASCAL PrescanDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                        LPARAM lParam)
// A big switch( Message) that handles messages sent to the Prescan 1
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

BOOL FAR PASCAL Prescan2Dlg( HWND hDlg, unsigned Message, WPARAM wParam,
                        LPARAM lParam)
// A big switch( Message) that handles messages sent to the Prescan 2
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

int auto_z_below( int target)
// Adjusts the z-offset signal so that it is just below target. Note that
// the feedback electronics will maintain a constant tunneling current,
// so that actual distance bewteen the tip and sample will be unchanged.

// Important calls to auto_z_below and auto_z_above are made just before
// switching the gain (see WM_COMMAND, SCAN_SCALE_001, etc.). This prevents
// the tip from crashing.

int auto_z_above( int target)
// Adjusts the z-offset signal so that it is just above target. Note that
// the feedback electronics will maintain a constant tunneling current,
// so that actual distance bewteen the tip and sample will be unchanged.

void read_sample_list( SAMPLELIST *list, char *fnam, char *first_name,
                        char *first_dir)
```

```
void free_sample_list( SAMPLELIST *list)

int isdir( char *name)

unsigned int calc_i_setpoint( double i_set, int i_set_range,
                              unsigned int tip_gain)

static int z_read()

static int check_crash_protection( HWND hDlg)

static void check_digital_feedback( HWND hDlg)

static void copy_seq( READ_SEQ *dest, READ_SEQ *src)

static void set_dumb_seq( READ_SEQ *this_seq)

static void add_sequence()

static void insert_sequence()

static void del_sequence()

int num_data()
// Returns the number of read sequences that record a channel of data

static void summary( HWND hDlg)

// **********
// print.c
// **********

BOOL FAR PASCAL IPrintDlg(HWND hDlg unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the printing
// dialog, including scrolling (WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

void print( HWND hDlg)

int place_image( PRINT_EL *this_el, int image_number, FILE *fp)

void postscript_preamble( FILE *fp)

void postscript_postamble( FILE *fp)

void write_image( HWND hDlg, PRINT_EL *this_image, int image_number,
                  FILE *fp, int this_output_color)

void print_bit( FILE *fp, int this_bit)

void flush_row( FILE *fp)
```

```
void finish_print_bit( FILE *fp)

void start_print_bit( FILE *fp, int num)

void clear_maybe()

void select_next( int multi_select)

void clear_selection()

int maybe_select( POINT pt)

int pt_in_image( PRINT_EL *this_el, POINT pt)

int pt_in_rect( int x1, int y1, int x2, int y2, POINT pt)

int coord_in_rect( float x, float y, float x1, float y1, float x2, float y2)

void selection_squares( HDC hDC, int x1, int y1, int x2, int y2)

void selection_square( HDC hDC, int x, int y)

void count_selected()

void print_status( HWND hDlg, char *mes)

static float pof_bw( float z, float gamma)

static float pof_grey( float z, float gamma)

static float pof_color( int color32, float gamma)

int line_side( GRID this_grid, int x, int y)

static void write_circles( PRINT_EL *this_el, FILE *fp)

static void write_circles_list( COUNT_DATA *count_data, int num,
                                float pixel_size, FILE *fp)

static void write_grid( PRINT_EL *this_el, FILE *fp)

int intersection_pts( float r, float theta, float *x1, float *y1,
                      float *x2, float *y2, int image_size)

static int write_grid_line( float r, float theta, int image_size,
                            float print_size, FILE *fp)

void re_link()

static void remove_print_el( PRINT_EL **print_list, PRINT_EL *remove_el)

void delete_selected()

static void copy_print_el( PRINT_EL **this_list, PRINT_EL *this_el)
```

```
static void copy_selected()

static void paste_seleceted()

void delete_all()

void print_char( FILE *fp, int c)

void get_char_bitmap( HWND hDlg, LOGFONT *font, char chr, char **bitmap,
                      FONTBITMAPINFO *fbinfo)

static void write_text( HWND hDlg, HDC hDC, PRINT_EL *this_el)

static void place_char( char *bitmap, float x, float y, int bitmap_x,
                        int bitmap_y, STM_COLOR color, FILE *fp)

static void write_arrow( HWND hDlg, PRINT_EL *this_el, FILE *fp)

static void prune()

void init_arrow_heads()

static PRINT_EL *add_image( int num)

static void reset_current()

static int cur_obj_type( int type)

static float max_print_x_origin()

static float max_print_y_origin()

static void print_do_scrolls( HWND hDlg)

static void print_do_x_scroll( HWND hDlg)

static void print_do_y_scroll( HWND hDlg)

static int draw_grid_line( HDC hDC, float r, float theta, int left,
                           int bottom, int right, int top, int x_offset,
                           int y_offset, float pixel_size)

static void repaint_grids( HDC hDC, PRINT_EL *this_el)

// **********
// scan.h
// **********

// preprocessor define (for convenience)
#define SD scan_defaults_data[scan_current_default]
```

```
// **********
// scan.c
// **********

// important global variables
float scan_x_gain;      /* for fine x signal; can be 0.1, 1, or 10 */
float scan_y_gain;      /* for fine y signal; can be 0.1, 1, or 10 */

datadef *scan_defaults_data[SCAN_NUM_DEFAULTS]

BOOL FAR PASCAL ScanDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                                LPARAM lParam)
// A big switch( Message) that handles messages sent to the scan dialog,
// including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields (WM_COMMAND's).

void calc_gains( unsigned int *x_gain, unsigned int *y_gain,
                 unsigned int *z_gain, unsigned int *z2_gain)

void calc_step_delay()
// sets step delay for the current scan default data (SD), based on scanning
// scale and scanning frequency

float calc_initial_time()

float calc_overshoot_fast_time()

float calc_overshoot_slow_time()

float calc_total_time()

float calc_seq_time()

float calc_time( int lines_left)

void sharpen( HWND hDlg)

void init_bitmap_data()
// if the current data ( in global datadef *data) is valid, fills in scan bitmap
// accordingly

int find_size( int size)

void scan_status( HWND hDlg, char *text, float num)

float calc_z_gain( unsigned int scan_scale)

static void scan_set_title( HWND hDlg)
// Sets the title of the scan dialog according to the filename of the current data
```

```
void set_scan_defaults( datadef *this_data)
// Sets fields in datadef pointed to by this_data to default values

int pre_scan( HWND hDlg)
// pre_scan performs the following tasks, in this order:
//      (1) Set the tip-sample bias. (V)
//      (2) Set the tunneling current. (I)
//      (3) Set the piezo mode (just to be sure).
//      (4) Start dithering of channels 0 & 1, if necessary.
//      (5) Zero the fine x and fine y, i.e. move them to the center of the
//          area to be imaged.
//      (6) "Synching Offsets". In case the user forgot to hit "Send."
//          Set the offset x and offset y (coarse position controls) and
//          the z offset.
//      (7) Move to the starting position (lower, left corner of area to be
//          imaged, as it appears on screen). With the slow coordinate,
//          overshoot the image edge and then move back. Do the same with the
//          fast coordinate.


// **********
// scan2.c
// **********

void scan( HWND hDlg)
// Basic scanning routine. Directs digital i/o and updates on-screen image.
// This routine assumes that pre_scan has already been called to set V & I,
// to set the x and y offset to the appropriate position, and to set the fine
// x and y to the lower left corner of the area to be imaged.
// The scan follows these steps:
//      (1) The tip moves up in the fast direction, taking measurements
//          if appropriate.
//      (2) The data just read (in this_data) is stuffed into all_data
//      (3) For scan_dir == BOTH_DIR1, the tip is moved over one pixel.
//          Or for scan_dir == BOTH_DIR2_POLAR, flip the bias. Otherwise, do
//          nothing.
//      (4) The tip moves down in the fast direction, taking measurements
//          if appropriate.
//      (5) Any data just read (in this_data) is stuffed into all_data.
//      (6) The tip is moved over one pixel in the slow direction.
//      (7) For scan_dir == BOTH_DIR2_POLAR, flip the bias. Otherwise, do
//          nothing.
//      (8) Repeat until the the entire image is scanned.
// The program regularly checks to see if the ESC key has been pressed. If so,
// scanning is terminated.
// Note that for all possible values of scan_dir except BOTH_DIR1, the scan
// does not actually raster across the image. Rather, it follows a "comb-like"
// pattern: up, back, over, up, back, over, etc.
```

```
static void update_line( int num, int back)
// Updates one row or column of bitmap_data in memory. SetDIBitsToDevice(...)
// must be called to actually update the image on-screen.

static void set_scan_parameters( int num)
// Updates all fields in all_data[num] to reflect the line just scanned.


// **********
// scan2.h
// **********

// Several routines are defined here. Preprocessor #define's are used instead
// of procedures in order to maximize performance (These routines  get invoked
// a lot!).

#define SET_DATA(DATA)
// OLD. Currently unused.

#define READ_Z()
// For each of the read sequences for this scan, turn feedback on/off, wait,
// dither, and record measurements as appropriate.

#define READ_Z_OLD()
// OLD. Currently unused.

#define Z_CALC_MINMAX()

#define Z_CRASH_PROTECT()

#define Z_CRASH_PROTECT_BACK()

#define DO_DIGITAL_FEEDBACK()
// Waits for signal to be between Imin and Imax. Calls dio_digital_feedback.

// **********
// sel.c
// **********

BOOL FAR PASCAL SelDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                        LPARAM lParam)
// A big switch( Message) that handles messages sent to the Selection Functions
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// ( WM_COMMAND's ).

int StartSelection( HWND hWnd, POINT ptCurrent, LPRECT lpSelectRect,
                    int fFlags)

int UpdateSelection( HWND hWnd, POINT ptCurrent, LPRECT lpSelectRect,
                    int fFlags)

int EndSelection( POINT ptCurrent, LPRECT lpSelectionRect)
```

```
int ClearSelection( HWND hWnd, LPRECT lpSelectRect, int fFlags)

int find_coords( SEL_POINT *pt, LPARAM lParam)

// **********
// sharp.c
// **********

BOOL FAR PASCAL SharpDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the Sharpen Tip dialog,
// including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// ( WM_COMMAND's ).

// **********
// stm.c
// **********

// One routine is defined here. Preprocessor #define is used instead
// of a  procedure in order to maximize performance (This routine gets invoked
// a lot!).

#define READ_DEP_CLOCK()

int PASCAL WinMain( HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow)
// WinMain is the entry point for the application ( Like main() in standard
// C programs ). Creates the application's window, allocates memory,
// initializes variables, and sets up the message loop.

long FAR PASCAL WndProc( HWND hWnd, unsigned Message,
                         WPARAM wParam, LPARAM lParam)
// A big switch( Message) that handles messages sent to the application's
// window, including standard Windows stuff ( like WM_CREATE and WM_DESTROY )
// and menu commands that create our dialogs ( WM_COMMAND'S ).

void set_smooth_vals()
// Initializes smooth_s and smooth_l.

void calibrate( HWND hWnd)

// **********
// spec.c
// **********

BOOL FAR PASCAL SpecDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                         LPARAM lParam)
// A big switch( Message) that handles messages sent to the spectroscopy
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

static void calc_integral()

static void calc_part_integral()
```

```
static void find_background()

void calc_didv()

static void repaint_bias( HWND hDlg, unsigned int bias)

static void set_data( datadef *this_data)

static void get_data( datadef *this_data)

int sp_data_valid( int ch)

static int init_spec( HWND hDlg, int ch, int size, int type)

static void sp_status( HWND hDlg, int pass, char *status)

static void spec_set_title( HWND hDlg)

static void sp_update( HWND hDlg, int value)

// **********
// specopt.c
// **********

BOOL FAR PASCAL SpecOptDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                            LPARAM lParam)
// A big switch( Message) that handles messages sent to the spec. opt.
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

static void sp_enable_abs_rel( HWND hDlg)

static void recalc_track_limits()

void sp_calc_scan()

// **********
// subtract.c
// **********

BOOL FAR PASCAL SubDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                        LPARAM lParam)
// A big switch( Message) that handles messages sent to the arithmetic
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

void subtract_data( int a, int b, float sub_x, float sub_y, float sub_z)
// Calculates gendata[a]-gendata[b] and puts the result in gendata[3]
// Does nothing if gendata[a] and gendata[b] are not of the same type

void average_data( int a, int b, float sub_x, float sub_y, float sub_z)
// Calculates gendata[a]+gendata[b]/2 and puts the result in gendata[3]
// Does nothing if gendata[a] and gendata[b] are not of the same type
```

```
void didv( int a, int b, float sub_x, float sub_y, float sub_z)
// Calculates (dI/dV)/(I/V) for two 2D graphs. The data in gendata[a] should
// be dI/dV (selected data). The data in gendata[b] should be I/V (button pressed)
// Result is put in gendata[3]. Does nothing if current image is 3D.

static datadef *create_single_step( datadef *this_data)
// Returns a copy of this_data that has been altered to correspond to a step size
// of one bit.

void average_1_data( int num, float x)
// Averages every x data points for a 2D graph gendata[num]. Does nothing if
// current image is 3D.

void smooth_1_data( int num, float x)
// Smooths data by averaging each data point with x neighboring data points for a
// 2D graph gendata[num]. Does nothing if current image is 3D.

// **********
// summary.c
// **********

BOOL FAR PASCAL SummaryDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                            LPARAM lParam)
// Displays information from the datadef glob_data
// Handles messages sent to the summary dialog

// **********
// tip.c
// **********

BOOL FAR PASCAL TipDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                        LPARAM lParam)
// A big switch( Message) that handles messages sent to the tip approach
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// and text fields ( WM_COMMAND's ).

void tip_approach( HWND hDlg, unsigned int accel)
// Calculates the parabolic waveform for giant steps and then cycles through
// giant steps/baby steps until tunnelling

void giant_steps( unsigned int xn, unsigned int zon, unsigned int *data)
//  During a giant step, the three outer piezos rotate the sample
//   with a constant accleration. Then, the three piezos stop and are lowered.
// The sample, still rotating, falls down toward the tip.

unsigned int baby_steps( HWND hDlg, unsigned int n, unsigned int *data)
//  During a baby step, the z voltage of the outer piezos, z0, is adjusted
//  by small amounts. During baby steps the tunneling current is measured.
//  If the tunneling current is greater than the minimum tunneling current,
//   tip approach is complete.

void tip_step( HWND hDlg, int dir)
```

```
void pre_tip()

void post_tip()

// **********
// trace.c
// **********

BOOL FAR PASCAL TraceDlg( HWND hDlg, unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the trace
// dialog, including scrolling (WM_HSCROLL, WM_VSCROLL) and various buttons
// ( WM_COMMAND's ).

static int trace_in_range( datadef* this_data, int pos)

static void new_trace_el( TRACE_EL **this_list, float width)

static void remove_last_trace_el( TRACE_EL **this_list)

static void destroy_trace_list( TRACE_EL **this_list)

static void trace_save_width( TRACE_EL *this_list, char *ext)

static void trace_bin( TRACE_EL *bin, TRACE_EL *list, float width)

static void trace_save_binned( TRACE_BIN bin, char *ext)

static void trace_save_bin()

// **********
// track.c
// **********

// important global variable
int tracking_mode; /* indicates track for max current or min current */

BOOL FAR PASCAL TrackDlg(HWND hDlg unsigned Message, WPARAM wParam,
                         LPARAM lParam)
// A big switch( Message) that handles messages sent to the tracking
// dialog, including scrolling (WM_VSCROLL) and various buttons
// ( WM_COMMAND's ).

static void recalc_track_limits()
// Calculates the tracking limits in terms of x,y coordinates from limtits
// in terms of bits
```

```
void track( int track_auto_auto, int high_limit, int low_limit,
            int track_avg_data_pts, int tracking_mode, int step_delay,
            int inter_step_delay, int track_limit_x_min,
            int track_limit_x_max, int track_limit_y_min,
            int track_limit_y_max);
// track looks for a max (min) in the tunneling current. The current is
// measured at a particular x,y pixel and at each of the eight neighboring
// pixels. The tip is then moved to the pixel with the highest (lowest)
// tunneling current. The process is repeated.

// NOTE that the local variable tracking_mode is different than the global
// one. The global variable is set to either TRACK_MAX or TRACK_MIN. The local
// variable is essentially a boolean, set to non-zero if in TRACK_MAX mode

// **********
// vscale.c
// **********

BOOL FAR PASCAL VscaleDlg(HWND hDlg unsigned Message, WPARAM wParam,
                          LPARAM lParam)
// A big switch( Message) that handles messages sent to the tracking
// dialog, including scrolling (WM_VSCROLL) and various buttons
// ( WM_COMMAND's ).

// Vertical scale allows the vertical scale of a 2D graph to be rescaled. All
// controls are disabled if the current image is 3D.
```