

# The Stony Brook STXM Data Files

Benjamin Hornberger  
mailto:benjamin.hornberger@stonybrook.edu

April 28, 2004

## Contents

<b>1</b>	<b>Overview</b>	<b>2</b>
<b>2</b>	<b>Naming Conventions and Downloads</b>	<b>2</b>
<b>3</b>	<b>Reading STXM 4 Data Files – the read_sm Procedure</b>	<b>2</b>
3.1	The Procedure <code>read_sm_with_detcal.pro</code> . . . . .	3
3.2	Reading Spectra . . . . .	4
<b>4</b>	<b>The sm_par Structure</b>	<b>5</b>
4.1	Codes for Scan Devices . . . . .	10
4.2	<code>max_pointlist</code> , <code>n_pointlist</code> , <code>point_namestring</code> and <code>point_xystring</code> . . . . .	11
4.3	Bitmasks for ADC and Scaler Channel Settings . . . . .	11
4.4	Display Calculations – <code>disp_num_factors</code> and <code>disp_denom_factors</code> . . . . .	12
4.5	SGM (Monochromator) Calibration . . . . .	13
4.6	The Use of Byte Arrays . . . . .	13
<b>5</b>	<b>Writing STXM 4 Data Files from IDL – the Procedure write_sm.pro</b>	<b>14</b>
<b>6</b>	<b>Silicon Detector Calibration</b>	<b>14</b>
6.1	Calibration Model . . . . .	14
6.2	Calibration Procedures . . . . .	15
6.3	Known Issues . . . . .	15
6.4	Detector Calibration To Do List . . . . .	16
<b>7</b>	<b>The BSIF Common Block in IDL</b>	<b>16</b>
<b>8</b>	<b>The Structure of the netCDF File</b>	<b>17</b>
8.1	Dimensions . . . . .	18
8.2	Variables . . . . .	18
8.3	Global Attributes . . . . .	19
8.3.1	Storing Strings in Global Attributes . . . . .	20
<b>9</b>	<b>Data File Format Changelog</b>	<b>20</b>

## List of Tables

1	X-1A microscopes and short names . . . . .	2
2	The parameters in the <code>sm_par</code> structure . . . . .	5
3	Codes for scan devices . . . . .	11
4	The netCDF variables in our data files . . . . .	19

## 1 Overview

This document describes data files from the Stony Brook / NSLS Scanning Transmission X-ray Microscopes, and how to read and process them with IDL.

Mainly STXM IV data files are described. Information about older files might be added at some point. STXM V data files will be added here on in another document once the file format is in a final state.

## 2 Naming Conventions and Downloads

When an image is taken, the file will be stored under a name like

```
x1ais_20jan2004_0010.sm
```

The first part describes the microscope (see Table 1), the second part is obviously the date, and the third part the number of the image of that day. Of course the file could be renamed, but the original name will still be stored as `original_filename` in the `sm_par` structure in the file (see Sec. 4).

Every night all data files from the previous day are zipped and made available for download for all microscopes from

```
http://xray1.physics.sunysb.edu/data/data.php
```

All the beamline and microscope configuration files are also zipped every night and made available at the same web site under “X1A configuration/settings files”.

## 3 Reading STXM 4 Data Files – the `read_sm` Procedure

The procedure `read_sm.pro` is used to read STXM 4 data files from IDL. The calling sequence is as follows (some keywords are left out):

Table 1: X-1A microscopes and short names

Microscope	Short name
X-1A outboard STXM	x1aos
X-1A inboard STXM	x1ais
X-1A inboard cryo STXM	x1aic

```
IDL> read_sm, filename, sm_par, khz, col_dist_um, row_dist_um, $
      header_only=header_only, help=help, $
      display=display, xinch=xinch, print=print,$
      disp_file_name = disp_file_name, disp_name = disp_name
```

**filename** Name of the file to read in.

**sm\_par** Name of the `sm_par` structure to be returned from `read_sm`. This structure will contain all the necessary header information about the file. See Sec. 4.

**khz** Name of an array which contains the image (count rate in khz). The display which is read into the `khz` array is the one stored in `sm_par.disp_name` (the active display when the image was stored), unless specified by `disp_file_name` and `disp_name`.

**col\_dist\_um, row\_dist\_um** Names of arrays with the calibrated column / row positions for each pixel, to be returned from `read_sm`. Note that for spectral scans, `col_dist_um` is the monochromator wavelength in microns.

**header\_only** Set this keyword to read in the header only, and not the data.

**help** Display help.

**print** Print the image or spectrum. Works on Linux / Unix only.

**display** Display the image or spectrum.

**xinch** Image size for print.

**disp\_file\_name, disp\_name** Must be specified together. If you want to read another display into the `khz` array than the one which was active when the image was stored, you can specify a display file (e.g. `sm_gui_disp_settings_stxmi.dat` file – they are archived in the `x1a` configuration files), and any display name in that file (e.g. `Sum Inner+Outer (kHz)`). Then the `disp_num_factors` and `disp_denom_factors` from there will be used (see Sec. 4.4). That means, if you want that display, you would type

```
IDL> read_sm, filename, sm_par, khz, $
IDL> disp_file_name="sm_gui_disp_settings_stxmi.dat", $
IDL> disp_name="Sum Inner+Outer (kHz)"
```

The only (sometimes important) information which you won't get back from `read_sm` as parameter is the raw image data with all channels. That one will be read into the variable `image_data` in the BSIF common block (see Sec. 7).

### 3.1 The Procedure `read_sm_with_detcal.pro`

Note that you don't have to worry about this section if you are taking stacks with the proportional counter on the outboard STXM!

As explained in Sec. 6, the silicon detector calibration is not really optimally implemented yet, neither in the data acquisition programs (GUI and `sm_script`) nor in that the calibration information is not yet stored in the data file. This causes difficulties especially in the stack acquisition on the inboard STXM. For optimal calibration handling in stacks, we should

- store the full calibration information with the data file, and
- have some kind of auto-calibration during stacks (after every other scan or so).

As long as this is not working yet, there is the following workaround: For each scan taken by `sm_script` on the inboard STXM (TO BE CONFIRMED: TO WHICH SCANS DOES THAT REALLY APPLY?), the shutter is kept closed for the first scan line. The procedure `read_sm_with_detcal.pro` will then subtract the average from this first scan line from the rest of the image, the result of which is then returned into the `khz` array.

To use `read_sm_with_detcal`, you have to specifically do

```
IDL>.compile read_sm_with_detcal
```

Then, any call to the `read_sm` procedure will use the calibration-corrected version (only the filename is `read_sm_with_detcal.pro`, as opposed to `read_sm.pro`. The name of the procedure inside the file is still `read_sm`. That's why you have to compile, but then you can use `read_sm` to call the procedure).

This also applies to stacks: Compile the procedure as above, then you can use `stack_analyze` as known.

**Note the following problem:** Since the average of the dark signal (first scan line) is subtracted from the rest of the image in the end, but before already a full calibration, including dark current correction, with possibly wrong (at least old) calibration parameters is done, this actually can't be correct! We should fix this soon ...

## 3.2 Reading Spectra

Note that spectral scans have the same structure as image scans. The difference is that `col_device` is `EV` (the monochromator), the scan has only one row, and `col_dist_um` is the wavelength in microns.

Say you recorded an  $I_0$  spectrum in the file `i0file` and an  $I$  spectrum in the file `ifile`. You want to get optical density  $D = -\log(I/I_0)$ . To calculate and plot this, you can do

```
IDL> read_sm, i0file, sm_par_i0, khz_i0, col_dist_um
IDL> ev_i0 = 1.239842/col_dist_um ;; standard conversion from lambda to eV
IDL> read_sm, ifile, sm_par_i, khz_i, col_dist_um
IDL> ev_i = 1.239842/col_dist_um
IDL> khz_i = interpol(khz_i, ev_i, ev_i0, /quad) ;; remap to ev_i0 grid
IDL> d=-alog(khz_i/khz_i0) ;; optical density
IDL> plot, ev_i0, d, title="My Plot", $
IDL> xtitle="Energy (eV)", ytitle="Opt. Density"
```

If both images were taken over the same energy range with the same number of points, they will have the same energy scale and you can skip the line with `interpol`. To write the data out to an ASCII file that you can read in as `.csv` file in Excel, you can do

```
IDL> write_mapper, "file.csv", ev_i0, d
```

## 4 The `sm_par` Structure

The `sm_par` structure is returned by `read_sm.pro` and contains all the parameters about the data file which are interesting in addition to the actual data. Almost all the parameters are read from the global attributes in the netCDF file.

In the C++ code, the structure is declared in `sm_par.h`. In IDL, it can be initialized by `sm_ipar.pro`.

The parameters in the `sm_par` structure for STXM 4 data files are listed in Table 2.

Table 2: The parameters in the `sm_par` structure

Name	Data type [array size]	Example Value	Description
<code>filever</code>	long	1	File Version. See Sec. 9.
<code>progver</code>	long	1	Program Version. Not used.
<code>col_device</code>	long	0	Code for column scan device. See Sec. 4.1
<code>row_device</code>	long	0	Code for row scan device. See Sec. 4.1.
<code>strlen</code>	long	128	String length in characters
<code>adc_ports</code>	long	8	Number of ADC channels (used for silicon detector segments)
<code>scaler_ports</code>	long	8	Number of scaler channels (used for counting detectors and clock)
<code>dac_ports</code>	long	2	Number of DAC channels (used to drive the piezos)
<code>maxpointlist</code>	long	25	Maximum number of positions which can be stored with a description. See Sec. 4.2
<code>n_cols</code>	long	250	Number of columns (pixels per row). This parameter is not stored as global attribute in the netCDF file, but it is just read from the size of the data array.
<code>n_rows</code>	long	250	Number of rows (pixels per column). No global attribute in the netCDF file either.
<code>n_data</code>	long	9	Number of values which are stored per pixel. Usually 2 (clock plus proportional counter) or 9 (clock plus 8 silicon detector segments). No global attribute in the netCDF file either.
<code>prescan_pixels</code>	long	3	Number of pixels which are scanned before real data is collected (to let the scan devices accelerate properly).

Table 2: The parameters in the `sm_par` structure (continued)

Name	Data type [array size]	Example Value	Description
<code>shutopen</code>	long	0	This flag is used from something like 2004-03-01 (to be confirmed) to denote that the shutter was closed for the first scan line. Used for automatic silicon detector calibration.
<code>bidirectional</code>	long	0	uni- or bidirectional scan
<code>clock_data_channel</code>	long	16384	Bitmask (here: 4000x) describing which channel is the clock (see Sec. 4.3)
<code>pixel_channels</code>	long	16639	Bitmask (here: 40FFx) describing which channels will be recorded for each pixel. The number of bits set must correspond to <code>n_data</code> . See Sec. 4.3.
<code>n_oneread_channels</code>	long	0	Number of channels which are read and stored only once per scan.
<code>oneread_channels</code>	long	0	Bitmask describing which channels will be recorded only once per scan. The number of bits set must correspond to <code>n_oneread_channels</code> .
<code>n_pointlist</code>	long	0	Actual number of points which has been stored with a description. See Sec. 4.2.
<code>disp_num_channels</code>	long	16639	Bitmask (here: 40FFx) describing which channels will go into the numerator for the display calculation (see Sec. 4.4 and 4.3).
<code>disp_denom_channels</code>	long	16384	Bitmask (here: 4000x) describing which channels will go into the denominator for the display calculation (see Sec. 4.4 and 4.3)
<code>disp_raw</code>	long	0	tells whether the display shows raw data from the scaler or ADC (1), i.e. an INT16 number, or kHz/Volts (0)
<code>col_pixel_um</code>	float	0.0503580	size of a column pixel in microns
<code>row_pixel_um</code>	float	0.0503580	size of a row pixel in microns
<code>x_center_um</code>	float	-87964.8	Center position of image in microns horizontally. STXM4: XSTG+XPZT. STXM5: interferometer position.
<code>y_center_um</code>	float	103.196	Center position of image in microns vertically. STXM4: YSTG+YPZT. STXM5: interferometer position.

Table 2: The parameters in the `sm_par` structure (continued)

Name	Data type [array size]	Example Value	Description
<code>z_center_um</code>	float	-18695.6	Center position of image in microns along the beam axis. STXM4: ZSTG+ZPZT. STXM5: interferometer position.
<code>x_pzt_um</code>	float	-15.8453	Position of X piezo in microns
<code>y_pzt_um</code>	float	-2.00354	Position of Y piezo in microns
<code>z_pzt_um</code>	float	0.000	Position of Z piezo in microns (non-existing)
<code>x_stg_um</code>	float	-87949.0	Position of X stage in microns
<code>y_stg_um</code>	float	105.00	Position of Y stage in microns
<code>z_stg_um</code>	float	-18695.6	Position of Z stage in microns
<code>x_det_um</code>	float	-227991.	Position of X detector stage in microns
<code>y_det_um</code>	float	706.000	Position of Y detector stage in microns
<code>z_det_um</code>	float	-42541.0	Position of Z detector stage in microns
<code>z_osa_um</code>	float	2142.45	Position of Z OSA stage in microns. Related to, but different from <code>zp_osa_z_um</code> .
<code>col_start_um</code>	float	-9.54055	Position in microns of horizontal scan start position. If a spectrum, <code>col_start_um</code> and <code>col_stop_um</code> are equal and <code>start_ev</code> and <code>stop_ev</code> indicate the energy range.
<code>col_stop_um</code>	float	-22.0797	Position in microns of horizontal scan stop position
<code>row_start_um</code>	float	4.29111	Position in microns of vertical scan start position
<code>row_stop_um</code>	float	-8.24803	Position in microns of vertical scan stop position
<code>start_ev</code>	float	526.997	Energy at which the scan started, in eV. <code>start_ev</code> and <code>stop_ev</code> will be equal for images and different for spectra.
<code>stop_ev</code>	float	526.997	Energy at which the scan stopped, in eV
<code>dwel_msec</code>	float	1.0000	Pixel dwell time in msec
<code>ring_gev</code>	float	2.80100	NLS electron storage ring energy in GeV
<code>ring_ma</code>	float	246.100	NLS electron storage ring current at end of scan in mA
<code>undulator_gap_mm</code>	float	36.1484	Undulator gap in mm

Table 2: The parameters in the `sm_par` structure (continued)

Name	Data type [array size]	Example Value	Description
<code>undulator_period_mm</code>	float	-1.99800	Undulator period in mm. Obviously wrong in this example file.
<code>ens_slit_um</code>	float	40.0000	Entrance slit width in microns. Zero if open.
<code>exs_slit_um</code>	float	38.0444	Exit slit width in microns. Zero if open.
<code>exsy_slit_um</code>	float	70.0000	Vertical exit slit width in microns. Zero if open.
<code>sgm_zero_order_steps</code>	float	11376.8	Zero order steps calibration for monochromator. See Sec. 4.5
<code>sgm_steps_per_angstrom</code>	float	-504.268	Steps per Angstrom calibration for monochromator. See Sec. 4.5
<code>zp_d_um</code>	float	80.0000	Zone plate diameter in microns
<code>zp_delta_nm</code>	float	30.0000	Zone plate outermost zone width in nanometers
<code>zp_stop_um</code>	float	35.0000	Zone plate central stop diameter in microns
<code>zp_osa_um</code>	float	30.0000	OSA diameter in microns
<code>zp_osa_z_um</code>	float	0.00000	Distance between OSA and zone plate in microns at <code>sm_par::align_focus_ev</code> . Given as the sum of <code>zp_osa_ideal_um</code> plus <code>align_zp_osa_extra_um</code> , as described in the alignment section.
<code>tilt_degrees</code>	float	0.00000	Tilt of the sample for tomography. For cryo, + is rotating upstream surface to face down.
<code>xpzt_nm_per_volt</code>	float	-10000.0	Calibration for X piezo in nm / volt
<code>ypzt_nm_per_volt</code>	float	-10000.0	Calibration for Y piezo in nm / volt
<code>zpzt_nm_per_volt</code>	float	0.0000	Calibration for Z piezo in nm / volt. Unused.
<code>xum_per_zum</code>	float	0.0000	Tilt of focusing axis relative to z-axis of motor for X.
<code>yum_per_zum</code>	float	0.0000	Tilt of focusing axis relative to z-axis of motor for Y.
<code>clock_hertz</code>	float	100000.	Clock frequency in Hertz
<code>det1_volt1</code>	float	0.0000	Voltage setting 1 for detector 1. Lost in the fog of time.
<code>det1_volt2</code>	float	0.0000	Voltage setting 2 for detector 1. Lost in the fog of time.

Table 2: The parameters in the `sm_par` structure (continued)

Name	Data type [array size]	Example Value	Description
det2_volt1	float	0.0000	Voltage setting 1 for detector 2. Lost in the fog of time.
det2_volt2	float	0.0000	Voltage setting 2 for detector 2. Lost in the fog of time.
temp1_celsius	float	0.0000	Temperature reading 1 (end of scan). Used for cryo.
temp2_celsius	float	0.0000	Temperature reading 2. Used for cryo.
disp_min	float	0.0000	Minimum value for display settings
disp_max	float	0.0000	Maximum value for display settings
disp_gamma	float	0.0000	Gamma correction for display settings
disp_num_factors	float [adc_ports + scaler_ports + 1]		Numerator coefficients for display calculations. See Sec. 4.4
disp_denom_factors	float [adc_ports + scaler_ports + 1]		Denominator coefficients for display calculations. See Sec. 4.4
acd_voltsperbit	float [adc_ports]		Calibration for ADC in Volts / Bit
acd_voltsoffset	float [acd_ports]		Voltage offset for ADC
dac_voltsperbit	float [dac_ports]		Calibration for DAC in Volts / Bit
dac_voltsoffset	float [dac_ports]		Voltage offset for DAC
col_title	byte [strlen]		Title for column axis. See Sec. 4.6.
row_title	byte [strlen]		Title for row axis. See Sec. 4.6.
beamline	byte [strlen]	x1ais	Name of the microscope (rather than the beamline). See Sec. 4.6
system_time	byte [strlen]	(see on right)	String giving the system time when the image was taken (Thu Jan 29 10:37:42, 2004). See Sec. 4.6.
scientist	byte [strlen]	SW, BH	Name of the operating scientist(s). Can be specified when the file is stored. See Sec. 4.6.
sample	byte [strlen]		String describing the sample. Can be specified when the file is stored. See Sec. 4.6.
comments1	byte [strlen]		Arbitrary comment about the scan. Can be specified when the file is stored. See Sec. 4.6.
comments2	byte [strlen]		Arbitrary comment about the scan. Can be specified when the file is stored. See Sec. 4.6.

Table 2: The parameters in the `sm_par` structure (continued)

Name	Data type [array size]	Example Value	Description
<code>original_filename</code>	byte [strlen]	(see on right)	The original filename, so that it can be recovered even if the file was renamed later. See Sec. 2 and Sec. 4.6.
<code>zp_name</code>	byte [strlen]	Q2 old Ni series (Spector)	Name of the zone plate. See Sec. 4.6.
<code>temp1_name</code>	byte [strlen]		String describing the temperature reading 1. See Sec. 4.6.
<code>temp2_name</code>	byte [strlen]		String describing the temperature reading 2. See Sec. 4.6.
<code>scaler_divideby</code>	byte [scaler_ports]	0 0 0 0 0 0 0 0	For each scaler channel, 0 means scaler counts full number of pulses; 1 means scaler divides counts by 8 (to avoid overflow).
<code>adc_channel_names</code>	byte [adc_ports × strlen]	ADC0, ADC1, ...	Array of strings describing the names of the ADC channels. For segmented detector, could be named segment0, segment1, ... also. See Sec. 4.6.
<code>scaler_channel_names</code>	byte [scaler_ports × strlen]	Counts, Unused, ..., 10 MHz, 100 kHz	Array of strings describing the names of the scaler channels. We use only 3 scaler channels for the counter and two different clocks. See Sec. 4.6.
<code>point_namestr</code>	byte [max-pointlist × strlen]		Array of strings describing special positions in the image. See Sec. 4.2 and Sec. 4.6.
<code>point_xystr</code>	byte [max-pointlist × strlen]		Array giving the position of certain special locations in the image, as a string. See Sec. 4.2 and Sec. 4.6.
<code>disp_name</code>	byte [strlen]	Sum Inner+Outer (kHz)	String describing the display under which the file was stored. Particularly important when using the segmented detector. See Sec. 4.6.
<code>oneread_data</code>	int [adc_ports + scaler_ports]		Data from ADC and scaler which is stored only once per scan.

#### 4.1 Codes for Scan Devices

The codes for "col\_device" etc. are defined in `sm_par.h` and summarized in Table 3.

Table 3: Codes for scan devices

Scan device	Short form	Code
X Piezo	XPZT	0
Y Piezo	YPZT	1
X Sample Stage	XSTG	2
Y Sample Stage	YSTG	3
Z Sample Stage	ZSTG	4
Monochromator	EV	5
X Detector Stage	XDET	6
Y Detector Stage	YDET	7
Z Detector Stage	ZDET	8
OSA Z Motor	ZOSA	9

#### 4.2 `max_pointlist`, `n_pointlist`, `point_namestring` and `point_xystring`

We have the option to store with the file a number of points (positions in the image) of particular interest with a description.

**`max_pointlist`** Maximum number of such positions.

**`n_pointlist`** Actual number of such position stored.

**`point_namestring`** Descriptions / annotations for these positions.

**`point_xystring`** Position (X, Y) in user units (microns) as a string.

Actually we don't have the option in the GUI to specify and describe these positions, so these entries in `sm_par` are not used.

#### 4.3 Bitmasks for ADC and Scaler Channel Settings

We have eight ADC and eight Scaler channels. In some cases we need a variable describing which of these channels is the clock, or which of these channels are stored in the data file etc. Therefore, 16-bit bitmasks are used. The lower 8 bits (the lower byte) stands for the ADC, the upper 8 bits (the upper byte) stands for the scaler. In binary form, such a bitmask might look like that:

```
0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1
```

This might describe which channels are stored in an Inboard STXM data file are recorded – all 8 ADC channels (lower byte), plus the 7th Scaler channel (100 kHz clock).

This binary number can easily be converted to hexadecimal form, which reads

```
40FFx
```

In data files, however, we can't store hexadecimal numbers as such, so that we have to convert it to a decimal number, which in this case is

```
16639
```

#### 4.4 Display Calculations – `disp_num_factors` and `disp_denom_factors`

How do we finally calculate the displayed image? If we want to display the count rate in kHz from a proportional counter signal, we will calculate

$$\text{kHz} = \frac{(\text{photon counts}) \times (\text{clock.hertz})}{(\text{clock counts}) \times 1000}. \quad (1)$$

Photon counts and clock signals are recorded in the scaler channels, and from the ADC channels we can calculate a count rate for each silicon detector segment (involving a rather complicated detector calibration). Therefore, we can calculate any display more abstractly from the eight ADC channels and the eight Scaler (S) channels as

$$\text{display} = \frac{a_0 + a_1 \cdot \text{ADC1} + \dots + a_8 \cdot \text{ADC8} + a_9 \cdot \text{S1} + \dots + a_{16} \cdot \text{S8}}{b_0 + b_1 \cdot \text{ADC1} + \dots + b_8 \cdot \text{ADC8} + b_9 \cdot \text{S1} + \dots + b_{16} \cdot \text{S8}}. \quad (2)$$

This enables us to calculate bright field, differential phase contrast and other images from the segmented silicon detector, but also the simple image from the proportional counter, as different "displays" from the ADC and Scaler's raw data. The arrays of 17 coefficients  $a_n$  for the numerator and  $b_n$  for the denominator of the display calculation, we call **`disp_num_factors`** and **`disp_denom_factors`**.

Of course, for the silicon detector, these coefficients will change after each detector calibration.

**Note** that specific `disp_num_factors` and `disp_denom_factors` are taken into account in the display calculation only if the corresponding bits in `disp_num_channels` and `disp_denom_channels` are set. Therefore, the `disp_factors` should be zero where the `disp_channels` are not set.

With every data file we store the name of the display which was active when the file was stored, along with all the parameters. All the available displays for the GUI are stored in the display settings file (`sm_gui_disp_settings_stxmi.dat` etc.). These files are saved every day in the x1a configuration files (See Sec. 2). Each of the displays specified there can be recalled in `read_sm` and read into the kHz array (see Sec. 3).

Two examples for display specifications in the display settings file are (the line break in the second example is only in this document because of the limited line length. In the real display file, it is just one line):

```
% Entry: Proportional Counter

disp_name "Proportional Counter (kHz)"
disp_raw 0
disp_num_channels 0100x
disp_denom_channels 0000x
disp_num_factors 0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,1.,0.,0.,0.,0.,0.,0.
disp_denom_factors 1.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.

%% Entry: "SiDet 4d+5d-6d-7d"

disp_name "SiDet 4d+5d-6d-7d"
disp_raw 0
disp_num_channels 407cx
```

```

disp_denom_channels 4000x
disp_num_factors -12.00018, 0, 0, -138.0481, 4183.5894, 4229.2378, -4673.7964, \
  -4154.4404, 0, 0, 0, 0, 0, 0, 0, 0.62077498, 0
disp_denom_factors -0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0099999998, 0

```

## 4.5 SGM (Monochromator) Calibration

The Spherical Grating Monochromator wavelength is calculated as

$$\lambda(\text{Angstroms}) = \frac{(\text{SGM position in steps}) - (\text{Zero Order Steps})}{(\text{Steps per Angstrom})} \quad (3)$$

## 4.6 The Use of Byte Arrays

We use byte arrays to store text information in the `sm_par` structure. The reason is that strings are not supported in the netCDF format. The netCDF format would support character arrays, but IDL doesn't support char variables. That's why we settled for byte arrays.

If it is a single parameter, the size of the byte array is `strlen` (128), and you can easily convert it back to a string:

```
IDL> col_title = string(sm_par.col_title)
```

If the parameter is actually an array of  $n$  strings, the byte array will be  $(n \times \text{strlen})$  bytes (characters) long. You can do something like

```

IDL> adc_ports = sm_par.adc_ports
IDL> strlen = sm_par.strlen
IDL> adc_channel_names = strarr(adc_ports)
IDL> for i = 0, (adc_ports-1) do adc_channel_names[i] = $
IDL> string(sm_par.adc_channel_names[i*strlen:(i*strlen+strlen-1)])

```

Actually, it is even easier to do this:

```

IDL> adc_channel_names = $
  string(reform(sm_par.adc_channel_names, sm_par.strlen, sm_par.adc_ports))

```

In STXM 5, the parameters will be stored directly as strings in the `sm_par` structure. They are converted from byte arrays to strings when they are read from the file.

**Note:** As of 2004-04-23, we have two IDL functions to do the conversion between strings / string arrays and byte arrays. `bytarr_to_strarr.pro` will convert a byte array of  $(n \times \text{strlen})$  length into a string array with  $n$  entries, or a scalar string if the byte array is `strlen` bytes long. `strarr_to_bytarr.pro` will convert a string or a string array with  $n$  entries to a byte array with  $(n \times \text{strlen})$  elements. The calling sequence is

```

IDL> string_array = bytarr_to_strarr(byte_array, strlen)
IDL> byte_array = strarr_to_bytarr(string_array, strlen)

```

For instance, to read the `adc_channel_names` and the `zp_name`, you could do

```

IDL> print, bytarr_to_strarr(sm_par.adc_channel_names, sm_par.strlen)
ADC0 ADC1 ADC2 ADC3 ADC4 ADC5 ADC6 ADC7
IDL> print, bytarr_to_strarr(sm_par.zp_name, sm_par.strlen)
Stein2000-005

```

## 5 Writing STXM 4 Data Files from IDL – the Procedure `write_sm.pro`

If you happened to wish to write a STXM 4 data file from IDL, you can use the procedure `write_sm`. Obviously this will rarely be the case, since normally data files are written by the scan program. The calling sequence is as follows:

```
IDL> write_sm, filename, sm_par, col_dist_um, row_dist_um, help=help
```

**filename** The name of the data file to be written.

**sm\_par** The `sm_par` structure as described in Sec. 4 which contains the global attributes to be written to the data file. It is not clear to me what happens if this `sm_par` structure doesn't match the format described in Sec. 4. There is no test in `write_sm` to check the number of parameters or anything else.

**col\_dist\_um, row\_dist\_um** Arrays which contain the absolute positions for the pixels in columns and rows. It will be checked that the number of elements in these arrays correspond to `sm_par.n_cols` and `sm_par.n_rows`, as well as to the number of columns and rows in the `image_data` array (see below).

**help** Will display help.

As you realize, there is no data array passed to `write_sm`. Instead, the array `image_data` will be read from the BSIF common block (see Sec. 7). It will be checked that the dimensions of that array correspond to `sm_par.n_cols`, `n_rows` and `n_data`.

## 6 Silicon Detector Calibration

Other than the proportional counter, which really counts incident photons (electric pulses), the segmented silicon detector works like a photodiode whose current is integrated during the pixel dwell time. The integrated charge will translate to a voltage which is read by the ADC. Moreover, even without incident X-rays, the leakage current of the silicon chip will lead to a certain signal. Therefore, a rather involved detector calibration is necessary. The theoretical background of the detector calibration can be found in Michael Feser's thesis [1, Chap. 3.4], while we review the practical aspects here.

### 6.1 Calibration Model

The X-ray flux (Photons per second) onto one detector segment is calculated as follows:

$$\Phi = \frac{F U_{\text{out}} - U_0 - C I_{\text{dark}} t_{\text{dwell}}}{C t_{\text{dwell}} - t_{\text{dead}}} \quad (4)$$

[1, Eq. 3.9], where

- $F$  is the photon-charge conversion factor (number of photons to generate a certain charge). *E.g.*,  $41.7 \text{ fC}^{-1}$  for 540 eV X-rays in silicon (the material of our detector chip),

- $C$  is the electronics calibration constant (output voltage of the instrumentation amplifier per input charge),
- $U_{\text{out}}$  is the output voltage of the detector channel as measured by the ADC,
- $U_0$  is a voltage offset (no physical meaning, but comes out as a linear fit parameter in the calibration),
- $I_{\text{dark}}$  is the dark (leakage) current of the silicon chip,
- $t_{\text{dwell}}$  is the pixel dwell time of the microscope, and
- $t_{\text{dead}}$  is the dead time of the detector as determined by the setting of the  $S3$  pulse width on the Quantum Composers timing module.

Moreover, crosstalk between the eight electronics channels has to be taken into account by an  $8 \times 8$  Crosstalk-Matrix  $C_{mn}$  [1, Chap. 3.4.6]. Actually, the crosstalk correction has to be applied before the calibration.

## 6.2 Calibration Procedures

While all other calibration parameters are constant, the dark current can change due to ambient conditions like temperature, etc. The full calibration (including crosstalk correction) can be done with the IDL routine `det_cal.pro`, while pure dark current corrections can also be done by clicking the “Calibrate Now” button in the microscope GUI.

What both of these methods do is calculating the calibration parameters from a set of dark scans and translating this into sets of `disp_num_factors` and `disp_denom_factors` (see Sec. 4.4) for a predefined number of different displays (segment combinations). These settings are stored in the `display_settings` file (e.g., `sm_gui_display_settings_stxmi_sidet.dat`). This display settings file can be specified when starting the GUI (to determine the displays available in the menu), or when using `read_sm` (to read a specific display into the `khz` array).

In STXM 5, the detector calibration will write the raw, basic calibration parameters (not `disp_factors`) into some configuration file, and they will be stored with each data file. The procedure `read_stxm5.pro` will calibrate the signal from these parameters.

## 6.3 Known Issues

The following things have to be considered:

- After clicking the “Calibrate Now” button in the GUI, the program has to be closed and re-opened to read in the new display settings file.
- Currently, the calibration parameters (`disp_num_factors` and `disp_denom_factors`) are stored in the data file **only** for the currently active display. That means, if you want to use any other display for that data file later, you need the corresponding display settings file (see Sec. 4.4). Since the display settings file is overwritten with each detector calibration, you manually have to make a backup copy (on the Linux command line) of the display settings file **before** you do the detector calibration, and you have to keep track of which display settings file belongs to a certain scan. For the backup, you can do something like (don’t break the line when you are really doing it)

```
[micros@x1a-stxmi]$ cp /mnt/x1a/sm_gui_disp_settings_stxmi_sidet.dat \  
/mnt/x1a/sm_gui_disp_settings_stxmi_sidet_19mar2004_incl0020.dat
```

if this display settings file is valid for all data files up to and including `x1ais_19mar2004_0020.dat`. Since the whole `/mnt/x1a` directory is zipped and made available as download (see Sec. 2), you can retrieve it from there.

- I hope this paragraph is correct: If scans are done from `sm_script` (stacks!), the display settings to be stored with the data file are read from the `sm_scan` configuration file (e.g., `sm_scan_stxmi_sidet.dat`). I am currently not aware of any procedure which would make sure that the display parameters in that file are adapted after a new detector calibration. That means that for stacks (on inboard STXM), we are maybe still using the calibration info from the first-ever detector calibration. Also see Sec. 3.1.

## 6.4 Detector Calibration To Do List

Obviously the situation described in the previous section is not what one would expect. At some point,

- we will implement storing the full calibration info (either the `disp_num_factors` and `disp_denom_factors` for all displays, or, more likely, the raw calibration parameters) in the data file,
- we should make the GUI reread the calibration parameters after a calibration (so that no restart is necessary),
- we should implement some auto-calibration procedure for stacks.

## 7 The BSIF Common Block in IDL

Common blocks in IDL are a set of variables which can be read and written from all routines which have the common block defined (and from the command line, if you specify the common block there). Up to STXM 4 data files, some data is read into the BSIF common block and can be processed further from there. The most important data, however, is passed as parameters anyway.

Common blocks are dangerous though, because many procedures can manipulate the variables, and you don't always notice it. Also, to understand what's going on, you need to know the internals of the procedures, which is not good programming style. That's why starting from STXM 5 data files, we don't use common blocks any more.

The BSIF common block is defined in the IDL routine `bsif_common.pro`. Every routine which is supposed to use the common block variables has to run `@bsif_common` in the beginning, and if you want to use it on the command line, you have to run that command manually or include it into your startup file.

The following variables are defined in the BSIF common block:

**version** BSIF file format version number - not currently used, allows for future changes in file format.

**first\_data\_record** First data record in file. This field is set by `WRITE_BSIF` and is not needed by any IDL routines.

**n\_rows, n\_cols** Number of rows and columns in image.

**n\_data** Number of data values at each pixel.

**x\_normal, y\_normal** Flags signaling top/bottom, left/right order. Not used.

**rotated** Flag signaling 90 degree rotation. Not used.

**x\_start, x\_stop** X range coordinates in user units (microns).

**y\_start, y\_stop** Y range in user units (microns).

**x\_dist, y\_dist** Arrays with calibrated units.

**data\_type** Data type.

**compression\_type** Data compression type. Not used.

**data\_min, data\_max** Range of data. Not used.

**image\_title** Title of image (actually filename).

**x\_title** Title of X axis. Not used.

**y\_title** Title of Y axis. Not used.

**data\_title** Titles of data values (array).

**user\_buffer** User defined buffer (unused).

**image\_data** The image data (n\_cols, n\_rows, n\_data) (raw).

As you can see, most of the data can also be found in the `sm_par` structure or from parameters passed from `read_sm`. The only important variable from the BSIF common block (for STXM 4 files) is `image_data`. For STXM 5 data files, we will pass that as parameter from `read_sm` as well.

## 8 The Structure of the netCDF File

This sections describes how the data files are built up internally. Usually you won't have to bother about that. It's just reference information for people who have to deal with how `read_sm`, `write_sm` and other procedures work internally.

We use the netCDF data format for STXM data files. Information can be found at

<http://www.unidata.ucar.edu/packages/netcdf/>

IDL has a lot of built-in functions to deal with netCDF files. Information can be found in the help-index under `netcdf`, and all the functions start with `ncdf_`.

To open and read from a netCDF file in IDL, do

```
IDL> id=ncdf_open(datafile,/nowrite)
```

You will have to refer to `id` on all the following commands. When you are done with the file, you should close it:

```
IDL> ncdf_close, id
```

The following are stored in the netCDF file: *dimensions*, *variables* and *global attributes*. You can get information about the datafile by doing the following:

```
IDL> info=ncdf_inquire(id)
IDL> help,info,/str
** Structure<152b0d0>, 4 tags, length=16, data length=16, refs=1:
  NDIMS          LONG          3
  NVAR           LONG          3
  NGATTS         LONG          97
  RECDIM         LONG          -1
```

which gives you the number of dimensions, the number of variables and the number of global attributes. We won't worry about RECDIM.

## 8.1 Dimensions

The dimensions describe the dimensions of the variables (data) in the file. They have a name and a size. All the possible dimensions are declared here, and later for each variable you will say how many and which dimensions it has. Above you saw that our data files have three dimensions, so let's check out dimensions 0 to 2:

```
IDL> ncdf_diminq,id,0,name0,size0
IDL> print,name0 & print,size0
n_cols
      250
IDL> ncdf_diminq,id,1,name1,size1
IDL> print,name1 & print,size1
n_rows
      250
IDL> ncdf_diminq,id,2,name2,size2
IDL> print,name2 & print,size2
n_data
      9
```

As you can see, this particular data file had 250 x 250 pixels and 9 channels stored per pixel (the clock plus 8 silicon detector channels).

## 8.2 Variables

netCDF variables contain the actual data. Variables have a name, a datatype, dimensions and attributes. We don't use the attributes for the variables though. Find out about the variables:

```
IDL> var0=ncdf_varinq(id,0)
IDL> help,var0,/struct **
Structure<152cc20>, 5 tags, length=44, data length=44, refs=1:
  NAME          STRING      'image_data'
```

Table 4: The netCDF variables in our data files. NDIMS = Number of dimensions, NATTS = Number of attributes. The numbers in the *Dimensions* column refer to the dimension IDs, see Sec. 8.1.

Variable ID	Name	Datatype	NDIMS	NATTS	Dimensions
0	image_data	short	3	0	0,1,2
1	col_dist_um	float	1	0	0
2	row_dist_um	float	1	0	1

```

DATATYPE      STRING      'SHORT'
NDIMS          LONG              3
NATTS          LONG              0
DIM            LONG      Array[3]
IDL> print,var0.dim
           0           1           2
IDL> var1=ncdf_varinq(id,1)
IDL> help,var1,/struct **
Structure<152b290>, 5 tags, length=36, data length=36, refs=1:
  NAME          STRING      'col_dist_um'
  DATATYPE      STRING      'FLOAT'
  NDIMS          LONG              1
  NATTS          LONG              0
  DIM            LONG      Array[1]
IDL> print,var1.dim
           0
IDL> var2=ncdf_varinq(id,2)
IDL> help,var2,/struct **
Structure<152d158>, 5 tags, length=36, data length=36, refs=1:
  NAME          STRING      'row_dist_um'
  DATATYPE      STRING      'FLOAT'
  NDIMS          LONG              1
  NATTS          LONG              0
  DIM            LONG      Array[1]
IDL> print,var2.dim
           1

```

The variables are summarized in Table 4. Variable 0 contains the actual image data, is of datatype *short* (16-bit integer from scaler or ADC) and has three dimensions (columns, rows, channels per pixel).

Variables 1 and 2 contain arrays with the actual positions in columns and rows in microns, respectively. They are of data type *float*, and have one dimension (column or row, respectively).

### 8.3 Global Attributes

The global attributes can store additional information about the data. We use it to store many additional parameters which are or might be of interest. Every global attribute corresponds to an

entry in the `sm_par` structure, and specific information about the various attributes can be found there (see Sec. 4).

The only exceptions are `n_cols`, `n_rows` and `n_data`, which are parameters in the `sm_par` structure but are not stored as global attributes. They will be read directly from the properties of the data array into the `sm_par` structure.

Global attributes are global to the data file. Variables can also have attributes in netCDF, but we don't use them.

Global attributes have a name, a datatype and a value, of course. The number of global attributes and therefore the range of attribute IDs was found above. To get the name of an attribute with a specific ID (here: 20), do the following:

```
IDL> attname=ncdf_attname(id,/global,20) & print, attname
col_pixel_um
```

Once you have the name, you can get more info on the data type and the length (number of elements: 1 for scalars, more for arrays):

```
IDL> attinfo=ncdf_attinq(id,/global,'col_pixel_um') & help, attinfo, /struct
** Structure <152de30>, 2 tags, length=16, data length=16, refs=1:
DATATYPE      STRING      'FLOAT'
LENGTH        LONG                1
```

And of course you can get the value itself:

```
IDL> ncdf_attget,id,/global,'col_pixel_um',col_pixel_um & print, col_pixel_um
0.0503580
```

### 8.3.1 Storing Strings in Global Attributes

The netCDF format doesn't allow strings to be stored in the file. Instead, we use byte arrays of a specific length (`STRLEN`, normally 128). In STXM 4, these parameters are even written into the `sm_par` structure as byte arrays. If you want to read them, you have to follow the instructions in Sec. 4.6. In STXM 5, `read_stxm5.pro` will convert them right after they are read, so that they are stored in the `sm_par` structure as strings.

## 9 Data File Format Changelog

This section is supposed to log all data file format changes and their consequences. Normally, with each change we should increment the filever, but actually we are still stuck at filever 1 as of March 2004.

## References

- [1] M. Feser, *Scanning Transmission X-ray Microscopy With a Segmented Detector*, PhD thesis, Department of Physics and Astronomy, Stony Brook University, 2002.