

Binary Spectral Imaging API

Online version: https://wiki.advacam.cz/wiki/Binary_Spectral_Imaging_API

Contents

Introduction	4
Requirements	4
Hardware	4
Software	4
The Spectral imaging API basic workflow	5
General functions	6
LoadPixelCore and UnloadPixelCore	6
pxpSiLoadPixelCore	6
pxpSiUnloadPixelCore	7
SetIPixel and GetIPixel	7
pxpSiSetIPixel	7
pxpSiGetIPixel	8
Create and Free	9
pxpSiLoadPixelCore	9
pxpSiFree	10
GetLastError	10
pxpSiGetLastError	10
Load calibration functions and test if calibration loaded	11
pxpSiLoadCalibrationFromDevice	11
pxpSiLoadCalibrationFromFiles	12
pxpSilsCalibrationLoaded	13
The measurement, processing and related	14
SetMeasParams, GetMeasParams and SetXrfCorrectionParams	14
pxpSiSetMeasParams	14
pxpSiGetMeasParams	15
pxpSiSetXrfCorrectionParams	15
StartMeasurement, ReplayData, IsRunning and Abort	16
pxpSiStartMeasurement	16
pxpSiReplayData	17
pxpSilsRunning	18
pxpSiAbort	19
BSTG files: pxpSiSaveToFile and pxpSiLoadFromFile	19
pxpSiSaveToFile	19
pxpSiLoadFromFile	20
Loading BSTG and using the data example	21
The messages, progress and related	21
Related declarations	21
SetMessageCallback and SetProgressCallback	22
pxpSiSetMessageCallback	22
pxpSiSetProgressCallback	23
ProcessedPixelsPerSecond and MeasuredPixelsPerSecond	23

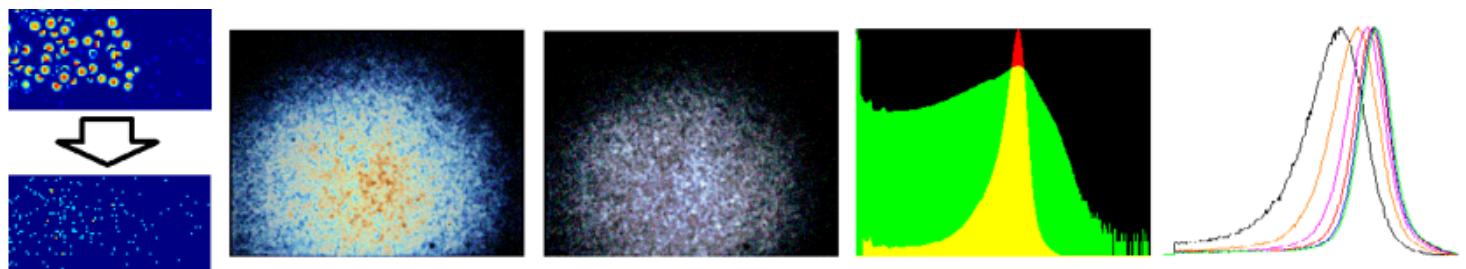


<i>pxpSiMeasuredPixelsPerSecond</i>	23
<i>pxpSiProcessedPixelsPerSecond</i>	24
Progress callback with px/sec example	24
Functions that using the output data	25
<i>pxpSiSpectrumSize</i>	25
<i>pxpSiSaveDataAsFramesToFile</i>	25
<i>pxpSiSaveDataAsSpectrumToFile</i>	26
<i>pxpSiGetFrameForEnergy</i>	27
<i>pxpSiGetFrameForEnergyRange</i>	27
<i>pxpSiGetGlobalSpectrum</i>	28
<i>pxpSiGetGlobalSpectrumInRect</i>	29
Examples	30
<i>Simple measuring and show the spectrum</i>	32
<i>Measuring (replay) and getFrameForEnergy</i>	33
<i>Measuring (replay) and getFrameForEnergyRange</i>	34
<i>Example results (getFrameForEnergyRange to RGB)</i>	35
Related	37



Introduction

The SpectralImg is designed for easy working with an energy spectra. It can work with previous saved data using the pxpCLReplayData or with physical device using the pxpCLStartMeasurement. Measured data will be clusterized, clusters will be divided into the required number of channels. Next you can use some frame-generating functions and data will be filtered by applied criteria and converted to pixels on output images. Or use graph-generating functions to generate graphs.



Spectral Imaging examples

Requirements

Hardware

This API requires computer with x86 compatible architecture (ARM can be on request). And connected some Advacam hardware with imaging chip. Timepix3, Timepix, etc or without measuring hardware if offline processing used.

Software

- 64bit Windows or Linux
- pxproc.dll, spectraimgapi.h (part of Pixel processing API package - on demand)

pxproc.lib (for linking only, Windows only)

- pxcore.dll (part of standard API package)

pxcore.lib (for linking only, Windows only, only if the pxclInitialize+pxcGetIPixel initializing way used)

(on Linux equivalent SOs instead of DLLs)

- Other auxilliary files:
 - pixel.ini
 - device config file(s) (both measuring or processing) or ABCT calibration files set (offline processing only)

- hwlibs and optional other device specific files (only for measuring)

See

- [Pixel SDK: Auxilliary files](#)
- [Files and directories of the Pixel and SDK](#)

The Spectral imaging API basic workflow

This API is defined in the spectraimgapi.h file.

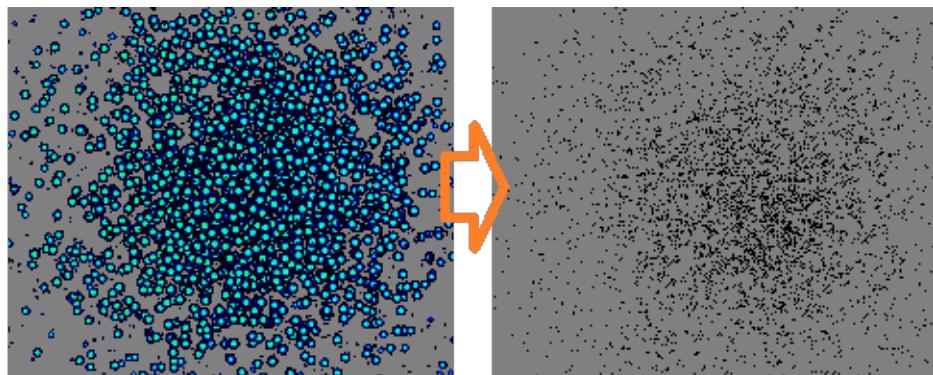
Typical usage:

Measuring, processing and use of the processed data

- 1.a Load the Pixel core using pxpSiLoadPixelCore("pxcore.dll");
- 1.b Or normally start the application using the Pixel core API pxclInitialize function, get the core pointer using pxcGetPixel(), set the core pointer to the Spectralmg API using pxpSiSetPixel(iPixel);
2. Get the Spectralmg instance using pxpSiCreate(device_index);
3. Set-up the callbacks (not required).
4. Load device calibration (not required).
5. Set the measurement parameters using the pxpSiSetMeasParams.
6. Set the X-ray fluorescence compensation parameters (not required) pxpSiSetXrfCorrectionParams.
- 7.a Start the measurement using pxpSiStartMeasurement.
- 7.b Or replay old data using the pxpSiReplayData function.
8. Wait for measurement and processing is complete (and display the progress) using while-pxpSilsRunning.
9. Use some get... method and use the processed data.
10. Deinitialize the Pixel core, if the pxclInitialize was used in step 1.

(For 1.a/b init ways see related Clustering API examples: [Binary Clustering API: Initializing using the pxpClLoadPixelCore](#) and [Binary Clustering API: Initializing from the active Pixel_core](#))





What do the Spectral imaging

Using BSTG files to save processing time

1. After processing is complete (end of waiting steps above), use the pxpSiSaveToFile(handle, "file.bstg") method.
2. Anytime later use the pxpSiLoadFromFile(handle, "file.bstg").
3. Use the pxpSiGetMeasParams to evaluate measure settings or set it to your program variables or GUI
4. Continue using the data as it was processed. The program now is in the step 9 of the list above.

General functions

LoadPixelCore and UnloadPixelCore

pxpSiLoadPixelCore

Loads the pixel core library (pxcore.dll/so). When the measurement with a device is intended the user has to either load pixel core with this function, or if the core is already loaded in the application (pxcore.dll/so was loaded separately), the pxpSiSetIPixel function has to be called.

Definition

```
PXSIAPI int pxpSiLoadPixelCore(const char* pxCoreLibPath);
```

Parameters

- pxCoreLibPath – path to the pxcore library

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Example

```
int rc = ppxSiLoadPixelCore("pxcore.dll");
errorToList("ppxSiLoadPixelCore", rc);
```

ppxSiUnloadPixelCore

Deinitializes and unloads the Pixel core.

Definition

```
PXSIAPI void ppxSiUnloadPixelCore();
```

Parameters

(no pars)

Return value

(void)

SetPixel and GetPixel

ppxSiSetPixel



Sets the internal Pixel API pointer. This is used when pxcore library is loaded separately in application. The use must pointer obtained via function pxcGetIPixel and should not load the pixel core with ppxSiLoadPixelCore function.

Definition

```
PXSIAPI void ppxSiSetIPixel(void* pixel);
```

Parameters

- **pixel** – The iPixel pointer from the [pxcGetIPixel\(\)](#) function of the pxcore API.

Return value

(void)

Example

```
iPixel = pxcGetIPixel(); // Warning: Use the pxcGetIPixel from pxcali.h,  
// not ppxSiGetIPixel from spectraimgapi.h  
if (iPixel==0) msgToList("pxcGetIPixel=NULL"); else msgToList("pxcGetIPixel OK");  
ppxClSetIPixel(iPixel);
```

ppxSiGetIPixel

Return internal Pixel structure pointer or 0 if not set.

Can be used if you want use functions of the pxcore API if the program was started using ppxSiLoadPixelCore.

Definition

```
PXSIAPI void* ppxSiGetIPixel();
```

Parameters

(no pars)



Return value

internal Pixel structure pointer or 0 if failed

Warning

Do not confuse this with the pxcGetIPixel() function of the pxcore API.

Create and Free

pxpSiLoadPixelCore

Definition

```
PXSIAPI sihandle_t pxpSiLoadPixelCore(int deviceIndex=SI_NO_DEVICE);
```

Parameters

- deviceIndex – index of the device this Spectralmg instance will manage.

If used offline (pxcore library not loaded), use SI_NO_DEVICE. The measurement will be not possible. Only replaying of data.

Return value

the handle of newly create instance of Spectra Imaging, or SI_INVALID_HANDLE if error

Note

If no device present, device with idx 0 is virtual file device. This is second way to offline use.

Example



```
siHandle = ppxSiCreate(0);
if (siHandle==CL_INVALID_HANDLE) msgToList("ppxSiCreate INVALID");
else msgToList("ppxSiCreate OK");
```

ppxSiFree

Frees the created instance of Spectralmg.

Definition

```
PXSIAPI int ppxSiFree(sihandle_t handle);
```

Parameters

- handle – Spectralmg handle received from function ppxSiCreate

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

GetLastError

ppxSiGetLastError

Gets the last error message - when a function returns error code. Gets either global message when handle = 0, or message for the specific Spectralmg instance. You can use this function or the message callback.

Definition

```
PXSIAPI int ppxSiGetLastError(clhandle_t handle, char* errorMsgBuffer, unsigned size);
```

Parameters



- handle – Spectralmg handle received from function pxpSiCreate
- errorMsgBuffer – output buffer where the error message will be stored
- size – size of the supplied errorMsgBuffer

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Example

```
// Using the pxpSiGetLastError in C++ CLR Windows app
const unsigned    cErrBufSize = 512;
// primary use to show function name, return code, last error message
void errorToList(const char* fName, int rc) {
    char errorMsg[cErrBufSize];
    char cMsg[cErrBufSize];
    pxpSiGetLastError(clHandle, errorMsg, cErrBufSize);
    if (rc!=0) {
        sprintf(cMsg, "%s %d err: %s", fName, rc, errorMsg);
    } else {
        sprintf(cMsg, "%s %d err: ---", fName, rc);
    };
    String^ sMsg = gcnew String(cMsg);
    msgToList(sMsg);
}
```

Load calibration functions and test if calibration loaded

pxpSiLoadCalibrationFromDevice

Loads the calibrations from the physically connected device. The device must support this feature. Minipix Tpx3 for example.

Definition

```
PXSIAPI int pxpSiLoadCalibrationFromDevice(sihandle_t handle);
```



Parameters

- handle – Spectralmg handle received from function pxpSiCreate

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

pxpSiLoadCalibrationFromFiles

Loads the calibration files (a,b,c,t files or a single xml device config file).

Definition

```
PXSIAPI int pxpSiLoadCalibrationFromFiles(sihandle_t handle, const char* filePaths);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- filePaths – Path to the file(s), variants:

The device xml config file

The four ABCT text files separated by the | symbol.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Example



```
rc = ppxSiLoadCalibrationFromFiles("mydetector.xml");
rc = ppxSiLoadCalibrationFromFiles("calibA.txt|calibB.txt|calibC.txt|calibT.txt");
```

ppxSilsCalibrationLoaded

Test if calibration is loaded.

Definition

```
PXSIAPI int ppxSiIsCalibrationLoaded(clhandle_t handle);
```

Parameters

- handle – Spectralmg handle received from function ppxSiCreate

Return value

Returns > 0 is loaded, = 0 not loaded, < 0 error

Example

```
int rc = ppxSiIsCalibrationLoaded(si);
if (rc<0) {
    print("ppxSiIsCalibrationLoaded: error %d", rc);
} else if (rc==0) {
    print("ppxSiIsCalibrationLoaded: Not loaded - output will NOT be calibrated (raw ToT
values)");
} else {
    print("ppxSiIsCalibrationLoaded: Loaded - output data will be calibrated (keV)");
}
```



The measurement, processing and related

SetMeasParams, GetMeasParams and SetXrfCorrectionParams

pxpSiSetMeasParams

Sets the parameters of the future measurement and processing

Definition

```
PXSIAPI int pxpSiSetMeasParams(clhandle_t handle, int spectFrom, int spectTo, double  
spectStep, bool maskNoisyPixels, bool doSubPixCorrection, bool correctXrf);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- spectFrom – minimal spectrum energy [keV]
- spectTo – maximal spectrum energy [keV]
- spectStep – Step width in the spectrum [keV] (float, but values significantly <1 may not work correctly).
- maskNoisyPixels – if measuring, periodically checks for noisy pixels and masks them
- doSubPixCorrection – if special filter should be applied to sub pixel images
- correctXrf – if the CdTe sensor internal fluorescence should be corected

Enable (true) or disable (false) XRF correction. Detected XRF energy will replaced by 25 keV.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Example

```
// set-up future spectrum measuring or processing to:  
// 500 to 1500 keV with steps 1.25, enable noisy masking and XRF correction  
int rc = pxpSiSetMeasParams(si, 500, 1500, 1.25, true, false, true);  
errorToList("pxpSiSetMeasParams", rc);
```



pxpSiGetMeasParams

Reads the processing parameters of the data in the Spectralmg instance memory.

(The parameters are pointers to output variables with the same meaning as the pxpSiSetMeasParams parameters)

Usesfull after loading a BSTG file.

Definition

```
PXSIAPI int pxpSiGetMeasParams(clhandle_t handle, int* spectFrom, int* spectTo, double* spectStep, bool* maskNoisyPixels, bool* doSubPixCorrection, bool* correctXrf);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- *spectFrom – pointer to store minimal spectrum energy [keV]
- *spectTo – pointer to store maximal spectrum energy [keV]
- *spectStep – pointer to store step width in the spectrum [keV] (float, but values significantly <1 may not work correctly).
- *maskNoisyPixels – pointer to store setting if measuring, periodically checks for noisy pixels and masks them
- *doSubPixCorrection – pointer to store setting if special filter should be applied to sub pixel images
- *correctXrf – pointer to store setting if the CdTe sensor internal fluorescence should be corected

Enable (true) or disable (false) XRF correction. Detected XRF energy will replaced by 25 keV.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

pxpSiSetXrfCorrectionParams

Sets the paramaters of the CdTe XRF correction in the future measurement and processing



Definition

```
PXSIAPI int pxpSiSetXrfCorrectionParams(clhandle_t handlee, double minVol, double maxVol,  
double toaDiff, bool remove);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- minVol – minimal energy volume of event to be considered as XRF
- maxVol – maximal energy volume of event to be considered as XRF
- toaDiff – maximal toa difference between primary and secondary XRF event
- remove – if XRF events should be removed (true) or assigned to original event (false)

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

StartMeasurement, ReplayData, IsRunning and Abort

pxpSiStartMeasurement

Starts measurement with the device for specified time and process the data. Only if the SI connected to the device. If calibration is loaded, energy values will be calibrated. Measurement works in the background. Use while-pxpSilsRunning() to wait for end, if need it.

Definition

```
PXSIAPI int pxpSiStartMeasurement(clhandle_t handle, double acqTime, double measTime,  
const char* outputPath, bool processData);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- acqTime – acquisition time of a single frame / pixel measurement in seconds.



Primary for frame-based devices (Medipixes, Timepix, no Tpx3): This is single frame time. Use a short enough time to prevent clusters overlapping. Too short time can cause too many losses between frames.

On data-driven devices (Timepix3, no Timepix), this is the ToA limit. After exceeds, ToA is resets and acqIndex in the internal callbacks is incremented. acqTime=measTime can be used.

- measTime – total time of measurement in seconds. Use 0 to endless measurement (progress will always 100%).
- outptuFilePath – output file where the process data (clusters) will be saved (*.clog). If saving not required, put "".
- processData – if the measured data should be processed online (clustering, filtering)

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Example

```
// measure for 100 secs, output file OFF, process the data ON
rc = pxpSiStartMeasurement(siHandle, 100, 100, "", true); // tpx3 version: acq=meas or
acq<meas
// pxpSiStartMeasurement(siHandle, 0.1, 100, "", true); // tpx/mpx version: acq safe time
errorToList("pxpSiStartMeasurement", rc);
```

pxpSiReplayData

Definition

```
PXSIAPI int pxpSiReplayData(clhandle_t handle, const char* filePath, const char*
outputFilePath);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- filePath – full path to a data file to be replayed
- outptuFilePath – output file where the process data (clusters) will be saved (*.clog). If saving not required, put "".



Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Note

Clog files: For historical reasons, there are two CLOG formats. Tpx3 have one additional column. The pxpSiReplayData cannot replay CLOG form Tpx3. Use the T3PA instead the CLOG. The T3PA files can be generated by data-driven measuring in the pxcore API.

Example

```
//    replay data from a file
rc = pxpSiReplayData(siHandle, "testdata.t3pa", ""); // tpx3 version
//    pxpSiReplayData(siHandle, "testdata.clog", ""); // tpx/mpx version
errorToList("pxpSiReplayData", rc);
```

pxpSilsRunning

Test if the SI is running.

Definition

```
PXSIAPI int pxpSiIsRunning(clhandle_t handle);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate

Return value

Returns 1 if running, 0 = not running, < 0 error.

Example



```
// Commandline app:  
while(ppxSiIsRunning(siHandle)) {  
    // do something while measuring or replaying  
}  
  
// Windows app:  
int rc = ppxSiIsRunning(siHandle);  
if (rc>=0) labelStatus->Text = rc ? "Running..." : "(stopped)";  
else errorToList("ppxSiIsRunning", rc);
```

ppxSiAbort

Aborts the measurement or replaying of the data.

Definition

```
PXSIAPI int ppxSiAbort(clhandle_t handle);
```

Parameters

- handle – Spectralmg handle received from function ppxSiCreate

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

BSTG files: ppxSiSaveToFile and ppxSiLoadFromFile

These functions allow you to save CPU time: Save the processed data with all settings and reuse it in future.

ppxSiSaveToFile



Saves the measurement, processing results and settings to a file

Definition

```
PXSIAPI int pxpSiSaveToFile(clhandle_t handle, const char* filePath);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- filePath – full path to a measurement will be saved (*.bstg)

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

pxpSiLoadFromFile

Loads the saved measurement including settings and processed data from a file

Definition

```
PXSIAPI int pxpSiLoadFromFile(clhandle_t handle, const char* filePath);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- filePath – full path to the saved measurement file (*.bstg)

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.



Loading BSTG and using the data example

```

int          rc;
int      spectFrom = -1;
int      spectTo = -1;
double     spectStep = -1;
bool maskNoisyPixels, doSubPixCorrection, correctXrf;
double  *data;
size_t   w=256, h=256;

rc = pxpSiLoadFromFile(siHandle, "testfile.bstg");
errorToList("pxpSiLoadFromFile", rc);

rc = pxpSiGetMeasParams(siHandle, &spectFrom, &spectTo, &spectStep, &maskNoisyPixels,
&doSubPixCorrection, &correctXrf);
errorToList("pxpSiGetMeasParams", rc);

msgToList(String::Format("From {0}, To {1}, Step {2}, Mask {3}, SubPix {4}, XRF {5}",
spectFrom, spectTo, spectStep, maskNoisyPixels, doSubPixCorrection, correctXrf));

data = (unsigned*)malloc(w * h * sizeof(double));
if (data==NULL) { msgToList("malloc error!"); return; }

// pxpSiGetFrameForEnergy(handle, energyIndex, sumFrame, normalize, zoom, frameData, w, h);
rc = pxpSiGetFrameForEnergy(siHandle, 1, false, false, false, data, &w, &h);
errorToList("pxpSiGetFrameForEnergy", rc);
if (rc<0) { free(data); return; }

// normalize data and show the image with caption (internal function of the example
project)
viewFrame(data, String::Format("GetFrameForEnergy {0} keV", 40 + 1*20));
free(data);

```

The messages, progress and related

Related declarations

```

/// Callback for messages and error messages returned from the SDK
/// @param error - true if error message
/// @param message - text of the message

```



```
/// @param userData - data of the user that were set in set callback function
typedef void (*SiMessageCallback)(bool error, const char* message, void* userData);

/// Callback for progress of an operation
/// @param finished - true if operation finished
/// @param progress - percentage of progress 0 - 100
/// @param userData - data of the user that were set in set callback function
typedef void (*SiProgressCallback)(bool finished, double progress, void* userData);
```

userData

See [Binary core API: pxcRegisterAcqEvent](#)

SetMessageCallback and SetProgressCallback

pxpSiSetMessageCallback

Callback for messages and error messages returned from the API.

Definition

```
PXSIAPI int pxpSiSetMessageCallback(clhandle_t handle, SiMessageCallback callback, void*
userData);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- callback – The callback function. The SiMessageCallback is defined in the spectraimgapi.h.
- userData – A pointer to the user data that was set in the registration of the callback function. Useful if the callback is registered multiple times, for example.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.



pxpSiSetProgressCallback

Callback for progress of an operation. Occurs every 1 second while measuring or processing. If measuring and processing, occurs twice.

Definition

```
PXSIAPI int pxpSiSetProgressCallback(clhandle_t handle, SiProgressCallback callback,  
void* userData);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- callback – The callback function. The SiProgressCallback is defined in the spectraimgapi.h.
- userData – A pointer to the user data that was set in the registration of the callback function. Useful if the callback is registered multiple times, for example.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

ProcessedPixelsPerSecond and MeasuredPixelsPerSecond

pxpSiMeasuredPixelsPerSecond

Returns the number of measured pixels per seconds.

Definition

```
PXSIAPI double pxpSiMeasuredPixelsPerSecond(clhandle_t handle);
```

Parameters



- handle – Spectralmg handle received from function pxpSiCreate

Return value

if positive, measured pixel count per second, if negative error code (SI_ERR_XXX)

pxpSiProcessedPixelsPerSecond

Returns the number of processed pixels per seconds.

Definition

```
PXSIAPI double pxpSiProcessedPixelsPerSecond(clhandle_t handle);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate

Return value

if positive, processed pixel count per second, if negative error code (SI_ERR_XXX)

Progress callback with px/sec example

```
void ProgressCallbackFn(bool finished, double progress, void* userData) {  
    char *fin = "(working)", *finF = "(finished)";  
    static int cnt = 0;  
  
    double mpps = pxpSiMeasuredPixelsPerSecond(siHandle);
```



```
double ppps = pxpSiProcessedPixelsPerSecond(siHandle);
if (finished) fin = finF;
sprintf(clbStatText, "Progress: %s, cnt %d, prog %.2f %%, measured: %.2f px/s,
processed: %.2f px/s", fin, cnt, progress, mpps, ppps);
cnt++;
// acquisition and processing both generating this callback
// (double cnt per new percents while measuring, single while offline processing)
if (finished) {
    cnt = 0; // This not occurs if error occured and process failed.
}
}
```

Functions that using the output data

pxpSiSpectrumSize

Returns spectrum size

Definition

```
PXSIAPI int pxpSiSpectrumSize(clhandle_t handle);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate

Return value

if positive, processed pixel count per second, if negative error code (SI_ERR_XXX)

pxpSiSaveDataAsFramesToFile

Saves generated spectra images (for each energy) to frame files.

Definition



```
PXSIAPI int pxpSiSaveDataAsFramesToFile(clhandle_t handle, const char* filePath, bool oneFile);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- fielPath – full path of a base file name.

Each frame file will have and number suffix corresponding to the energy bin.

- oneFile – whether the images should be saved into a single multi frame file (*.pmf)

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

pxpSiSaveDataAsSpectrumToFile

Saves generated spectra images as a list of spectra for each pixel (file with 65536 spectras at 65536 lines)

Definition

```
PXSIAPI int pxpSiSaveDataAsSpectrumToFile(clhandle_t handle, const char* filePath);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- filePath – full path to the file where spectra will be saved

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.



pxpSiGetFrameForEnergy

Gets the data of the image (frame) for a selected energy index or sum image if selected.

Definition

```
PXSIAPI int pxpSiGetFrameForEnergy(clhandle_t handle, unsigned energyIndex, bool sumFrame, bool normalize, int zoom, double* frameData, size_t* width, size_t* height);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- energyIndex – index of the energy bin to get the frame. If sum frame selected, this is ignored.
- sumFrame – if sum frame from all energy bins should be returned
- zoom – zoom factor (1, 2, or 3) for sub pixel frame. Only available when sum frame is selected
- normalize – whether the image should be normalized by the spectrum
- frameData – buffer where the frame will be saved
- *width [in/out] – pointer to the width of the frame.^{*note}
- *height [in/out] – pointer to the height of the frame.^{*note}

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Note

*Width and height are for energy bin frame and single detector = 256.

If SumFrame used and with zoom factor, then 256 * 2^zoom.

pxpSiGetFrameForEnergyRange

Gets the data of the image (frame) for a selected energy range (frame is sum of frames for each energy bin).

Definition

```
PXSIAPI int pxpSiGetFrameForEnergyRange(clhandle_t handle, unsigned energyIndexFrom,
```



```
unsigned energyIndexTo, bool normalize, double* frameData, size_t* width, size_t* height);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- energyIndexFrom – first energy index in the range
- energyIndexTo – last energy index in the range
- normalize – whether the image should be normalized by the spectrum
- *frameData – buffer where the frame will be saved
- *width [in/out] – pointer to the width of the frame.^{*note}
- *height [in/out] – pointer to the height of the frame.^{*note}

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

Note

*Width and height are for energy bin frame and single detector = 256.

pxpSiGetGlobalSpectrum

Gets the global energy spectrum

Definition

```
PXSIAPI int pxpSiGetGlobalSpectrum(clhandle_t handle, unsigned* data, double* step,  
size_t* size);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- *data – data buffer where the spectrum will be stored
- *step [out] – pointer to store step of the spectra
- *size [in/out] – pointer to the size of the supplied buffer.



If buffer size doesn't match error occurs and size will contain correct size.
Correct spectrum size can be obtained via pxpSiSpectrumSize function.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.

pxpSiGetGlobalSpectrumInRect

Gets the global energy spectrum in selected frame rectangle. Pixel pos. conditions are <= and >=

Definition

```
PXSIAPI int pxpSiGetGlobalSpectrumInRect(cchandle_t handle, unsigned x1, unsigned y1,  
unsigned x2, unsigned y2, unsigned* data, double* step, size_t* size);
```

Parameters

- handle – Spectralmg handle received from function pxpSiCreate
- x1 – left coordinate of the rectangle
- y1 – top coordinate of the rectangle
- x2 – right coordinate of the rectangle
- y2 – bottom coordinate of the rectangle
- *data – data buffer where the spectrum will be stored
- *step [out] – pointer to store step of the spectra
- *size [in/out] – pointer to the size of the supplied buffer.

If buffer size doesn't match error occurs and size will contain correct size.
Correct spectrum size can be obtained via pxpSiSpectrumSize function.

Return value

0 if successful, otherwise the return value is a SI_ERR_XXX code.



Examples

This chapter containing parts of example project Spectralmg from the AdvacamAPlexamples collection.

Download the AdvacamAPlexamples package

<https://advacam.com/examples/AdvacamAPlexamples.rar>

(MS Visual Studio 2017 Solution with C++ projects of Windows CLR programs)

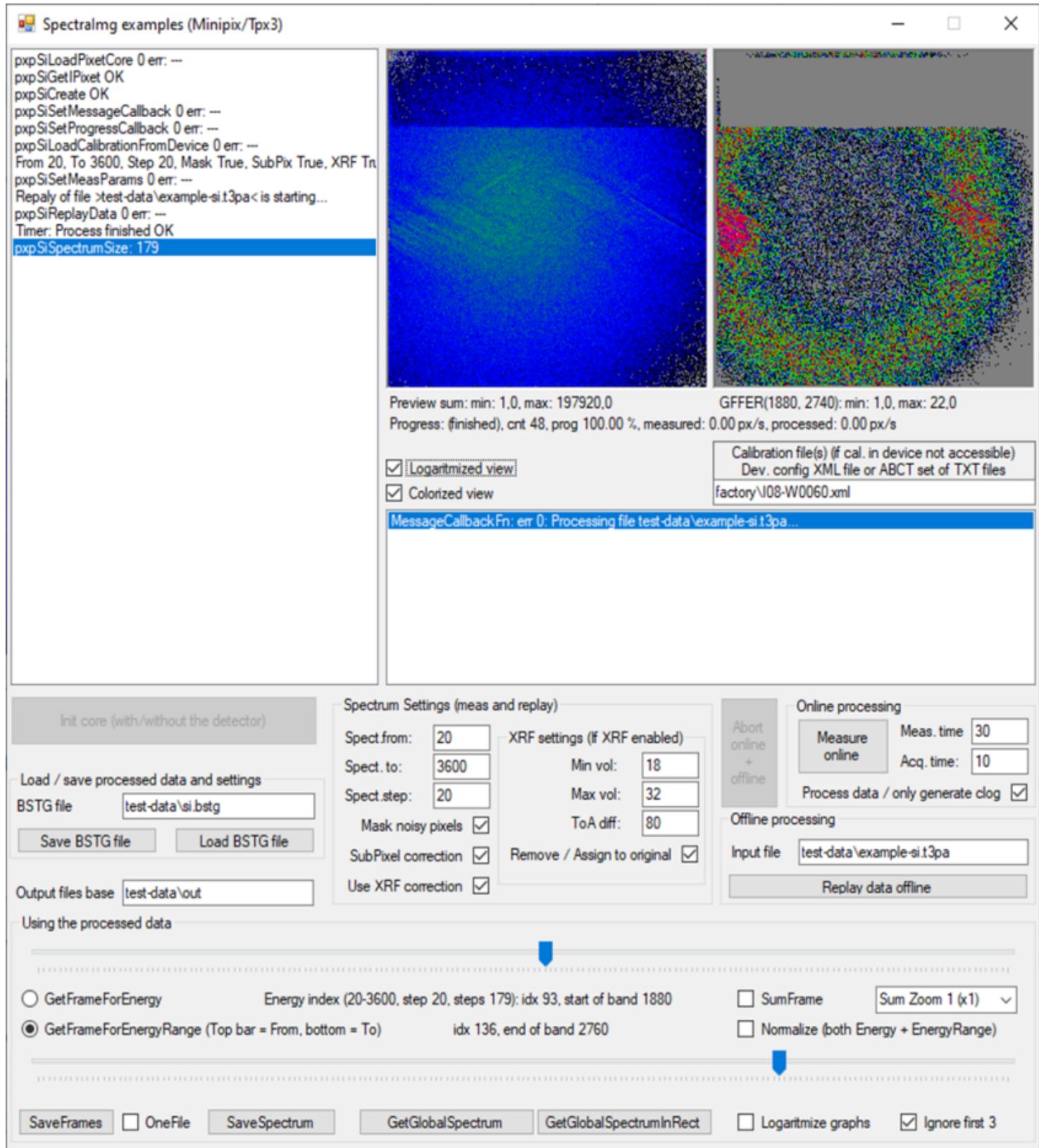
Note

To understand 2 initializing ways you can see the Clustering initialization examples:

[Binary Clustering API: Initializing using the pxpClLoadPixelCore](#)

[Binary Clustering API: Initializing from the active Pixel core](#)





The Spectralmg examples screenshot

Simple measuring and show the spectrum

```

int rc;
size_t spSiz;
unsigned *data;

rc = ppxSiLoadCalibrationFromDevice(siHandle);
errorToList("ppxSiLoadCalibrationFromDevice", rc)
if (rc!=0) msgToList("Calibration failed! - ToT counts will be instead of energies");

// from 40, to 4040, step 20 keV, mask noisy pixels ON, subpix corr in sum OFF, XRF corr
OFF
rc = ppxSiSetMeasParams(siHandle, 40, 4040, 20, true, false, false);
errorToList("ppxSiSetMeasParams", rc);

// measure for 100 secs, output file OFF, proces the data ON
rc = ppxSiStartMeasurement(siHandle, 100, 100, "", true); // tpx3 version: acq=meas or
acq<meas
// ppxSiStartMeasurement(siHandle, 0.1, 100, "", true); // tpx/mpx version: acq safe time
errorToList("ppxSiStartMeasurement", rc);

// wait for measure complete
while(ppxSiIsRunning(siHandle)) System::Threading::Thread::Sleep(100);

// number of elements in the spectrum = (4040-40)/20 in this example
spSiz = ppxSiSpectrumSize(siHandle);
if (spSiz>=0) msgToList("ppxSiSpectrumSize: "+ spSiz.ToString());
else { errorToList("ppxSiSpectrumSize", spSiz); return; }

data = (unsigned*)malloc(spSiz * sizeof(unsigned));
if (data==NULL) { msgToList("malloc error!"); return; }

// get the spectrum to the "data" array
rc = ppxSiGetGlobalSpectrum(siHandle, data, &spectStep, &spSiz);
errorToList("ppxSiGetGlobalSpectrum", rc);
if (rc<0) { free(data); return; }

// show the graph (internal function of the example project)
graph(data, spSiz, gcnew Pen(Color::Black, 3.0), GRAPH_clear);
free(data);

```



Measuring (replay) and getFrameForEnergy

```

int rc;
size_t w=256, h=256;
double *data;

rc = ppxSiLoadCalibrationFromDevice(siHandle);
errorToList("ppxSiLoadCalibrationFromDevice", rc)
if (rc!=0) msgToList("Calibration failed! - ToT counts will be instead of energies");

// from 40, to 4040, step 20 keV, mask noisy pixels ON, subpix corr in sum OFF, XRF corr
OFF
rc = ppxSiSetMeasParams(siHandle, 40, 4040, 20, true, false, false);
errorToList("ppxSiSetMeasParams", rc);

// measure for 100 secs, output file OFF, proces the data ON
rc = ppxSiStartMeasurement(siHandle, 100, 100, "", true); // tpx3 version: acq=meas or
acq<meas
// ppxSiStartMeasurement(siHandle, 0.1, 100, "", true); // tpx/mpx version: acq safe time
errorToList("ppxSiStartMeasurement", rc);

// replay data from file – alternative to ppxSiStartMeasurement
// rc = ppxSiReplayData(siHandle, "testdata.t3pa", ""); // tpx3 version
// rc = ppxSiReplayData(siHandle, "testdata.clog", ""); // tpx/mpx version
// errorToList("ppxSiReplayData", rc);

// wait for measure complete
while(ppxSiIsRunning(siHandle)) System::Threading::Thread::Sleep(100);

data = (unsigned*)malloc(w * h * sizeof(double));
if (data==NULL) { msgToList("malloc error!"); return; }

// ppxSiGetFrameForEnergy(handle, energyIndex, sumFrame, normalize, zoom, frameData, w, h);
rc = ppxSiGetFrameForEnergy(siHandle, 1, false, false, false, data, &w, &h);
errorToList("ppxSiGetFrameForEnergy", rc);
if (rc<0) { free(data); return; }

// normalize data and show the image with caption (internal function of the example
project)
viewFrame(data, String::Format("GetFrameForEnergy {0} keV", 40 + 1*20));
free(data);

```



Measuring (replay) and getFrameForEnergyRange

```

int rc;
size_t w=256, h=256;
double *data;

rc = ppxSiLoadCalibrationFromDevice(siHandle);
errorToList("ppxSiLoadCalibrationFromDevice", rc)
if (rc!=0) msgToList("Calibration failed! - ToT counts will be instead of energies");

// from 40, to 4040, step 20 keV, mask noisy pixels ON, subpix corr in sum OFF, XRF corr
OFF
rc = ppxSiSetMeasParams(siHandle, 40, 4040, 20, true, false, false);
errorToList("ppxSiSetMeasParams", rc);

// measure for 100 secs, output file OFF, proces the data ON
rc = ppxSiStartMeasurement(siHandle, 100, 100, "", true); // tpx3 version: acq=meas or
acq<meas
// ppxSiStartMeasurement(siHandle, 0.1, 100, "", true); // tpx/mpx version: acq safe time
errorToList("ppxSiStartMeasurement", rc);

// replay data from file alternative
// rc = ppxSiReplayData(siHandle, "testdata.t3pa", ""); // tpx3 version
// rc = ppxSiReplayData(siHandle, "testdata.clog", ""); // tpx/mpx version
// errorToList("ppxSiReplayData", rc);

// wait for measure complete
while(ppxSiIsRunning(siHandle)) System::Threading::Thread::Sleep(100);

data = (unsigned*)malloc(w * h * sizeof(double));
if (data==NULL) { msgToList("malloc error!"); return; }

// ppxSiGetFrameForEnergy(handle, energyIndex, sumFrame, normalize, zoom, frameData, w, h);
rc = ppxSiGetFrameForEnergy(siHandle, 1, false, false, false, data, &w, &h);

// ppxSiGetFrameForEnergyRange(handle, From, To, normalize, frameData, width, height);
rc = ppxSiGetFrameForEnergyRange(siHandle, 100, 120, false, data, &w, &h);
errorToList("ppxSiGetFrameForEnergyRange", rc);
if (rc<0) { free(data); return; }

// normalize data and show the image with caption (internal function of the example
project)
viewFrame(data, String::Format("GetFrameForEnergyRange {0}-{1} keV", 40 + 100*20, 40 +

```



```
120*20));  
free(data);
```

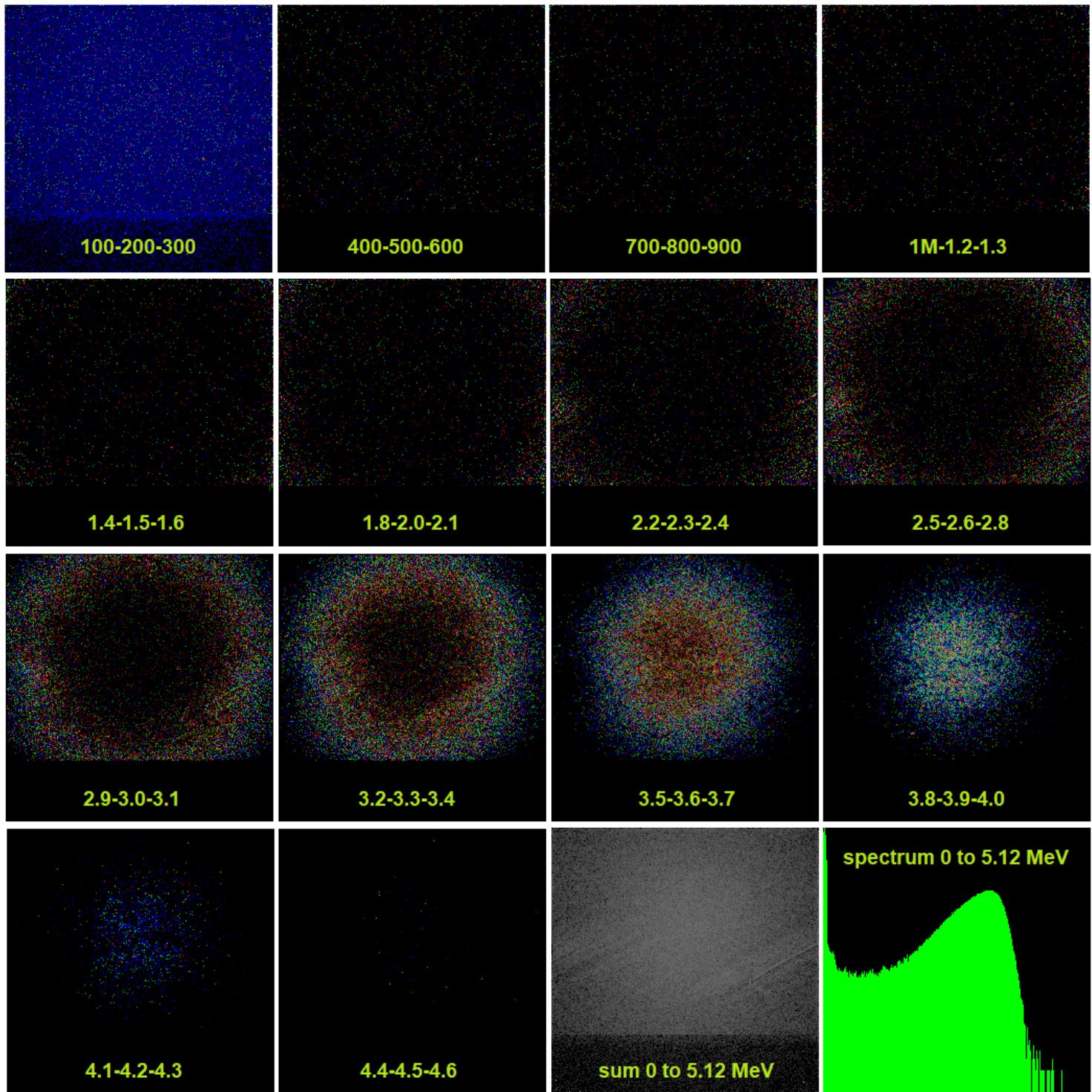
Example results (getFrameForEnergyRange to RGB)

²⁴¹Am from smoke detector, located 3 mm above the chip, MinipixTpx3, CdTe 2 mm, acq. time 45 seconds. Settings used:

```
pxpSiSetMeasParams(0, 5120, 20, 1, 0, 0) // (from, to, step, maskNP, doSPC, XRF)
```

- Each RGB color channel in images was generated from sum of 5 energy ranges (100 keV), using the pxpSiGetFrameForEnergyRange method, log2(val+0.5) applied and normalized to 0-255. Order is blue-green-red.
- The blue at the first image is gamma byproduct 59.5 keV and noise in the first band. The images with MeV ranges shows a 5.48 MeV alpha particles attenuated in air.
- Summary frame shows all hits in the set range. It was get using the pxpSiGetFrameForEnergy method with sumFrame=true.
- A “scratches” on the picture: Spectralmg is very sensitive to small differences in px sensitivity.
- The spectrum on last image was get using the pxpSiGetGlobalSpectrum method and processed like as pixels data, include using log2. A gaps on the right are caused by no hits.





The Spectralmg examples results

Note

The images collection are from the [Python API commandline examples](#)

A different imaging method is used here, where three adjacent energy ranges are converted to RGB image channels.

Related

- [Pixel SDK](#)
- [Files and directories: Main directory of the API-using programs](#)

