Object Oriented Device Driver Modules

GeoPIXE now has plugins that contain all the device dependent IDL code to handle specifics of list-mode data-streams, local metadata and miscellaneous local spectrum file formats. They are implemented using IDL Object techniques, with one new Object class per device handler, plus a Generic_device class, to handle common formats such as ASCII, and a Base_device, which is the super-class with methods that handle many common tasks for all devices.

All device driver source files have file-names of the form "XXX_device__define.pro" for device "XXX" (note the double underscore "__" before the "define") and when compiled they have names of the form "XXX_device__define.sav". These files should reside in the "interface" sub-directory to the main GeoPIXE directory. From here they will be detected and loaded automatically by GeoPIXE and used to populate all device droplists (e.g. in the "Sort EVT" and "Import Spectra" windows.

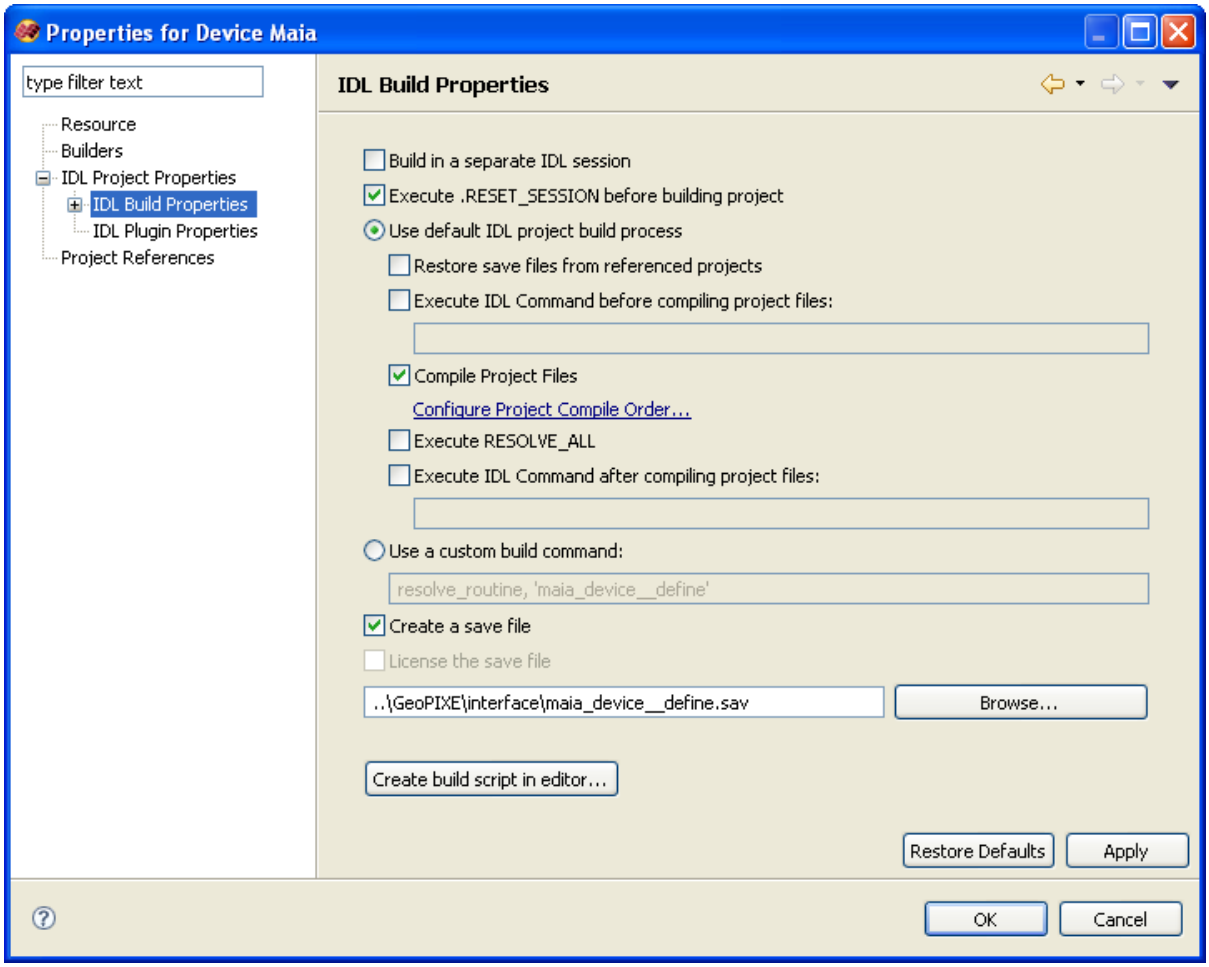
Contents

Device methods

Writing and using device methods, start by using features of existing device object define files as a guide. Do not base a new device object on the Base or Generic device code. Clone one of the existing "XXX_device__define.pro" files and rename all occurrences of "XXX" to the new device name. Make sure that this new name is unique. Store the files in the "interface" sub-directory to the main GeoPIXE directory.

Building Device Object SAV files

When building SAV files for a new Device Object class, as with spectrum and image plugins, compile the main "XXX_device__define.pro" file and only resolve calls and routines that are local to your device and **NOT part of GeoPIXE** (i.e. your local routines and standard IDL). Avoid using "resolve_all". Then save the compiled routines to "XXX_device__define.sav" in the "interface" sub-directory to the main GeoPIXE directory. When the device object is used, all GeoPIXE routines will already be loaded as part of GeoPIXE and should not be in these SAV files.



Example of IDL 7.1 Project properties for the Maia device project. All Maia specific files are compiled but "resolve_all" is NOT used. The output goes in the "interface" directory.

Initialization

When all device classes are detected and loaded into GeoPIXE (all files of the form "xxx_device__define.sav" or "xxx_device__define.pro" in the "interface" sub-directory of the GeoPIXE main directory, excluding "BASE_DEVICE" and "GENERIC_DEVICE"), an Object reference is created for each using the IDL call "Obj_New('xxx_device')". This executes the "Init" method in the class.

init()	initialize the class definition and also defines the main object parameter struct in the xxx_device__define" routine. DO NOT RUN THIS METHOD (it is done by IDL when GeoPIXE creates each device object).
cleanup	this is run when an object is destroyed (DO NOT RUN THIS METHOD).

The init method defines the 'self' struct which contains the key parameters that define the properties of the device. They are set in a call to the base::init method from your device::init method. The parameters are:

Name	the unique name for this device (must end in "_device"
Title	device title (used in device droplists in GeoPIXE)
Ext	the file extension for raw list-mode data (if it is fixed)

Multi_files	1=flags that raw list-mode data occurs in multiple files
Multi_char	defines the character sequence that separates the raw data-file stub from the part that indicates the XY coordinates (if used)
Big_endian	1=flags that the raw list-mode data has Big Endian byte order
Vax_float	1=flags the use of VAX D floating format in the raw data
Start_adc	defines the number of the first detector ADC
Synchrotron	1=flags the use of this device for synchrotron data
Ionbeam	1=flags the use of this device for ion-beam data

The values for the following options can also be set-up in the device::init method.

self.options.scan.on = 1	scan sort options available for this class to be displayed as widgets in the "Sort" tab of the Sort EVT window (see below).
self.options.scan.ysize = 75	Y size of sort options box, when open

Similarly, the use of pileup, throttle and linearization can be flagged this way:

```
self.pileup.use = 1
self.throttle.use = 1
self.linear.use = 1
```

And, if the device can support the use of a Y lookup table to reduce processing times when extracting spectra from regions (only process multifile input files that are needed for the regions selected), this can be flagged this way:

```
self.header.scan.use_ylut = 1
self.header.scan.pYlut = ptr_new(/allocate_heap)
```

and a method 'trim_evt_files' will be needed. The Y lookup table is used with the extraction of spectra from regions in an image using the EVT button on the *Image Region* window. The 'trim_evt_files' method checks the Y lookup table against the regions selected and returns only the data file-names needed to provide data for these regions. In the Maia device, the Y LUT is just the first Y for each blog data segment file. Hence, only the Y values spanned by all regions is checked. It is possible to write more general lookup tables and methods that also check X.

For examples see supplied device IDL pro files to use as templates, such as:

Ion-beam:

FASTCOM_MPA3_DEVICE	for list-mode data in a single file
MPSYS_DEVICE	for a single list-mode file and example of the 'get_header_info' method to read header information from an associated metadata file.
LUND_KMAX_DEVICE	for multi-file list-mode data that includes import of some local spectra file formats

Synchrotron:

NSLS_HDF_DEVICE	for list-mode data in a single file.

NSLS_MCA_DEVICE	for multi-file synchrotron list-mode data that includes the ‘flux_scan’ method to look for ion chamber PVs and for import of some local spectra formats.
MAIA_DEVICE	for a multi-file list-mode format stream that includes ‘flux_scan’ as well as setting some extra variables local to the device through custom device widgets added to the Sort EVT window by the device and the use of Y lookup tables and the ‘trim_evt_files’ method.

Many of the devices use calls to Fortran DLL routines to speed up processing within the ‘read_buffer’ method, others just use IDL code. This choice will depend on whether the list-mode data stream has a simple recurring format, or has a more complex structure that demands fiddly processing. The former can make use of IDL vector processing, while the latter will most often be done more efficiently using Fortran or C code.

Base-device methods available

Some general purpose methods used by GeoPIXE are available in the BASE_DEVICE master-class to avoid writing these for each device class. These are:

name()	return name of device (e.g. "MAIA_DEVICE").
title()	return title string for this device (used in Sort EVT droplist)
extension()	return raw data file extension (if fixed, else ")
multi_files()	1=data organized in multiple files, else=0
multi_char()	character used to separate run name/number from numeric data-file series
big_endian()	1=flags data stored in raw data in Big Endian byte order, else=0
vax_float()	1=flags VAX D-floating variables as part of data header
start_adc()	# of the first detector ADC.
pileup()	1=flags the use of a pileup rejection file for this device
throttle()	1=f;ags the use of a Throttle mechanism for this device
linear()	1=flags a linearization correction table used for this device
ylut()	1=flags that this device can use and generate a lookup table of first Y for each member of a multi-file data series to speed up certain operations (e.g. spectra extract using EVT button on Image Regions window).
get_ylut()	retrieve the Y lookup table data from file.
write_ylut	write a Y lookup table file (called from the DA image sort).
use_bounds()	1=flags that this device may have a border of pixels that contain no data and no beam charge/flux that should be excluded.

Reading list-mode data to produce images and extract full spectra

All device object classes need to implement these methods, with a full parameter list (even if some parameters are not used).

read_setup()	will be called after each data file that is opened to setup internal device parameters (often as variables in common[1] blocks) needed for

[1] Common blocks are crude but have the advantage over fields within the object’s ‘self’ structure of being able to be passed by reference (by name) into subroutines that need to return updated values for these variables. Passing ‘self.var’ is only passed by value and cannot return a new value into ‘self.var’.

	reading data buffers, such as buffer size and device-specific buffer organization.
read_buffer()	called repeatedly to read buffers from the data file, process these to extract X,Y,E triplet data, tagged by detector channel STE, compress X,Y,E if needed, and optionally detect other information (e.g. flux/charge, energy tokens).
get_header_info()	interrogate the data files (usually prior to starting processing) for various details, such as scan size (physical size and/or pixel count), title, energy cal for detectors, etc. Called, for example, when new data files are selected in the Sort EVT window.

The above 3 methods (plus the init and cleanup methods) are the essential minimum set needed to be written for a new device class.

flux_scan()	scan the raw data-files for details of available ion chamber specifications (e.g. EPICS PVs) to provide for user selection, and select one to use, and the pre-amp sensitivity value and units.
trim_evt_files()	trim the list of files to only include files needed for the Y range seen in the region mask arrays (uses the information in the Y lookup table; used with EVT button on Image Regions window).

Device specific Import of spectra

Import of spectra is achieved in two modes: (i) using user-written IDL code to read local format spectrum files, and (ii) using the GeoPIXE "spec_evt" routine to scan through a list mode file(s) set to extract all data from all detectors, together with the X and Y projections of all events for each detector. The latter creates a new file with the same name as the input data file (or first file in a set) and the suffix "i.spec'.

Local spectrum formats are handled by local IDL routines. They are defined in the 'get_import_list()' method by setting up a parameter structure called 'opt' for each import format or mode (a template for 'opt' is given in the example device code). Just the following parameters in opt are needed for local spectrum import:

Name	a unique name for the import (usually created as the device name plus a few extra characters).
Title	A string to identify this import mode in the "Import Spectra" window.
In_ext	The file extension (if it's fixed) for the import spectral data.
Request	A string to use a title for the File-Requester pop-up window.
Device_name	The unique name for the device ('self.name')

The latter mode for extracting spectra from list-mode data-streams requires at least one more parameter:

Spec_evt	1= flags this import as an extract from list-mode file mode
Multifile	1= flags this as a multi-file set (defaults to not multifile)
Separate	the characters that separate file-name from XY tags (if used)
Raw	1= flags that the raw list-mode data can reside in a different file path to the output spectrum files created by this import.

Use_IC	1= flags the need to pop-up the flux selection window to allow IC PVs to be selected (or scanned in the data using the 'flux_scan' method).
Xr	default X range
Yr	default Y range

Some parameters are probably only needed for the Maia device, unless needed for your device too:

Use_linear	1= request linearization file
Use_pileup	1= request pileup file
Use_throttle	1= request throttle file
Use_tot	1= collect time-over-threshold (ToT) spectral data too

The method 'get_import_list()' is called by the *Import Spectra* window to build the selection lists and tables of devices for Synchrotron and Ion-beam local data formats. When one of the local spectrum file imports is selected, the method 'import_spec()' is called on the selected device object class to execute some local routine to read the spectrum data. If an extraction from list-mode import is selected, GeoPIXE handles this internally using 'spec_evt'. The spectrum reading routines should create a GeoPIXE spectrum struct using a call to "define(/spectrum)" and return a pointer to this struct (or an array of these structs for multiple spectra import).

Device specific parameters

If the device has some device specific parameters that need to be set-up for processing and read/written along with image and region data, etc. to/from disk, then use this facility to define widgets to gather info about parameter options and manage them. If you don't need them do not define these methods, then a default (no action) method will be used in the "BASE_DEVICE" master-class. These options are set-up in widgets that appear in the Sort EVT window options box on the Sort tab. The parameters live in the class 'self' struct, defined in the device class 'init' method, and are handled using these methods:

render_options	draws widgets needed in supplied parent base (on Sort tab)
read_options	called when images and regions are read from disk to read the device specific options into the object self struct.
write_options	called when images and regions are written to disk to write the device specific options from the object self struct.
options_legend()	Returns a formatted string array to be added to the image history list in the Image History window to show device specific parameters.
set_options	explicitly pass these options parameters into the object. This should only be used internally to your object code to set option parameters.
get_options()	Explicitly get options parameters from Maia object. This should only be used internally to your object code to get option parameters.

The following two are used with Options widgets, called by GeoPIXE to the device object, but are handled by the Base super-class:

show_sort_options()	flags that this device has sort options to display.
get_sort_ysize()	returns number of Y pixels needed for device options fields.

To define and use addition device specific Options parameters, set these parameters in your 'init' method code. The size needed is set-up in the device Init method using:

self.options.scan.on = 1	scan sort options available for this class to be displayed as widgets in the "Sort" tab of the Sort EVT window .
self.options.scan.ysize = 75	Y size of sort options box, when open

The code in the render_options for creating options widgets for the Maia device, which also calls some OnRealize routines and an event handler, can be used as a model for new device options fields.

Device Object programming notes

List-mode File organization

List-mode files can be organized in a number of ways:

1. Single data stream containing detector energy (E) data as well as pixel coordinates (XY). May also contain flux data. For single long files, the method 'read_setup' sets up initial parameters (perhaps reading a header or some associated file) and then sequential large buffers are read and processed by method 'read_buffer'.
2. Multiple files holding a single logical data stream. In this case, the method 'read_setup' opens each file and sets up initial parameters (perhaps reading a header) and then for each file sequential large buffers are read and processed by method 'read_buffer'.
3. Pixel spectra files, one for each pixel, that contain E spectra, and perhaps other information (e.g. energy calibration, dead-time, flux IC count, IC preamp sensitivity and range, ...). The initial opening of each pixel file is done by method 'read_setup' and then 'read_buffer' fills in the variables E, X, Y to return the pixel data to be processed.
4. Line of pixels (e.g. NSLS_NETCDF device, with data for each line stored).

The main job of the 'read_buffer" method is to fill in the vectors E for energy of each event and X,Y, which are the scan coordinates for the corresponding event and STE, which contains the index of the detector for each event. The variable N contains the length of these vectors. If a particular E,X,Y,STE occurs many times, either list them (in any order) or set MULTIPLE to the count for the repeat. This is useful for presenting a spectrum in a list-mode form, by using Multiple to record the counts in a channel of the spectrum for a detector (e.g. "NSLS_MCA_DEVICE", "HASYLAB_FIO_DEVICE").

Header Info

Devices can provide a 'get_header_info' method to return details read in from a data file header, or some other associated metadata file or database. Often this method provides details of scan size (pixel count), energy calibration, scan size (mm) and beam energy. The full range of parameters that can be returned and usefully applied by GeoPIXE are given in the 'header' part of the device's 'self' variable (defined in the BASE device). The header part of the 'self' struct contains the following variables and default values:

header: {header_devicespec, $

```
charge:         0.0, $          ; beam charge (uC)
energy:         0.0, $          ; beam energy (MeV: ion-beam, keV: SXRF)

scan: {scan_devicespec, $
        on:             0, $            ; scan active
        x_pixels:       0, $            ; # X pixels
        y_pixels:       0, $            ; # Y pixels
        x:              0.0, $          ; X absolute origin (mm)
        y:              0.0, $          ; Y absolute origin (mm)
        x_mm:           0.0, $          ; X size of scan (mm)
        y_mm:           0.0, $          ; Y size of scan (mm)
        mode:           0, $            ; scan mode (0 XY, 1 Xstep, 3 Ystep)
        n_steps:        0, $            ; number of steps per pixel
        step_size:      0.0, $          ; step size in step mode
        sort_mode:      0, $            ; sort mode (0 XY image, 1 Traverse)
        dwell:          0.0, $          ; dwell time (ms)
        use_ylut:       0, $            ; able to filter file list based on Y LUT
        pYlut:          ptr_new(/allocate_heap)}, $   ; Y lookup table

px_coords:      ptr_new(), $        ; X coords array
py_coords:      ptr_new(), $        ; Y coords array
x_coord_units:      '', $           ; X coord units
y_coord_units:      '', $           ; Y coord units
detector:       intarr(geopixe_max_adcs), $  ; detector_types **
title:              '', $           ; comment/title string
cal:            replicate( {cal_devicespec, on:0, a:0.0, b:0.0, units:''},
                        geopixe_max_adcs), $
error:              0 } $           ; error flag
```

** Detector types (starting at 0) in order: 'PIXE', 'PIGE', 'RBS', 'ERDA', 'STIM', 'CHARGE', 'DEADTIME', 'SXRF', 'SEM' (e.g. SXRF = 7).

Flux set-up

The “Sort EVT” window in GeoPIXE has a tab for setting up flux parameters. If an indirect flux/charge measurement is used, via an EPICS PV for example, then a button appears to scan the data for PVs, which are labels associated with ion chamber data used for flux measurement. This button calls the method ‘flux_scan’ in many devices to read in PV or other Tag strings that can be used to select the correct parameter for IC counter, and often PVs for preamp sensitivity. If sensitivity is not found, then some numerical sensitivity droplists are provided.

This panel sets the contents of the ‘flux_ic” variable that is passed into the ‘read_setup’ method. Typically, this sets the sensitivity of the IC count, often using the variable ‘nsls_flux_scale’ (= flux_ic.val * flux_ic.unit). Internally, the flux in a pixel becomes NSLS_IC, which is IC count times this factor (e.g. see NSLS_NetCDF_device__define.pro).

The flux_ic structure:

```
{ mode:0, pv:'', val:0.0, unit:0.0, conversion:1., use_dwell:0, dwell:1.0}
```

where mode 0 no flux IC (ion beam, no IC data)

	1	use IC PV
	2	use conversion only (no PV)
pv		user selected EPICS PV string to be used for flux IC
val		pre-amp sensitivity value
unit		pre-amp sensitivity unit (scale)
conversion		conversion from flux count to charge (uC)
use_dwell		use the dwell time with flux count-rate to build flux count per pixel
dwell		dwell time in a pixel (ms)

Flux and Dead-time

When building images, zero ‘flux’ and ‘dead_fraction’ variables are passed to the imaging routine and on to the ‘read_setup’ and ‘read_buffer’ device object methods as arrays of the same size as the image area. These arrays need to be filled by the device, either:

1. Read in whole flux (and dead_fraction) arrays in ‘read_setup’ (e.g. APS and SLS should be done this way).
2. Read in flux for each pixel as it is processed (e.g. in NSLS and Hasylab devices this sets variable NSLS_IC), which is then used to set flux array value for current pixel in ‘read_buffer’.
3. Typically, the data stream may supply an IC count. This is converted to flux count by scaling by the product “flux_ic.val * flux_ic.unit”, which comes from the flux PV selection pop-up window (the product is variable ‘nsls_flux_scale’ in many devices).
4. Dead-fraction may be determined for some devices from live and real time for each pixel (e.g. for ‘NSLS_MCA’ device in ‘read_setup’ for each pixel file and then assigned in ‘read_buffer’).