

Pixet 1.8.5 → 1.8.6

Online version: https://wiki.advacam.cz/wiki/Pixet_1.8.5_%E2%86%92_1.8.6

Contents

Pixet Python API	3
1. At a glance	3
2. Breaking changes — what to fix in your scripts	3
2.1 Check for exceptions, not return codes	3
2.2 Delete every destroy() and *RefInc() call	4
2.3 Device parameters: one generic accessor	4
2.4 Getters that became properties	4
2.5 Acquisition results	5
2.6 save() takes the data manager	5
2.7 Pixel masking: factory mask vs user mask	5
2.8 DACs: a property, and copy() instead of createCopyOfDacs()	6
2.9 Type casting: check the data format	7
2.10 Other renamed calls	7
2.11 pygui: widgets are classes	7
2.12 Constants became enum classes	8
3. What's new	10
Detector control	10
pygui	10
pyclustering	11
pyspectraimg	11
pyutils	11
pyftdi	12
Editor support	12
4. Removed modules	12
5. Upgrade checklist	13



Pixet Python API

This release modernises the Python API. Scripts written against 1.8.5 will need changes — the list of what to fix is in §2, with before/after for each item.

1. At a glance

- Errors are Python exceptions now. Calls raise a `PixetError` (a subclass by the kind of the error - `DeviceError`, `PixetFileError`, `PixetValueError`, ...) instead of returning -1.
- No more manual memory management. `destroy()` and the `*RefInc()` call variants are gone; objects are released for you.
- Common getters became properties — `dev.params`, `dev.pixcfg`, `dev.dacs`, `pixet.data_mgr`.
- Parameter access is type-free — one `getValue()` / `setValue()` pair instead of 39 typed accessors.
- Pixel masks are split into a factory mask, your own user mask, and a read-only combined view.
- `pygui` is a real widget toolkit — constructors instead of factory methods, plus tables, trees, scroll areas, date/time editors, a reworked plot, and Qt enums.
- Clustering and spectral imaging are proper modules with object APIs and progress callbacks.
- Type stubs ship with every module, so editors give you autocompletion and signature help (with the gaps noted in §3).
- Ten helper modules were removed (§4).

2. Breaking changes — what to fix in your scripts

2.1 Check for exceptions, not return codes

Failures now raise a `pypixet.PixetError`; bad arguments raise the usual `TypeError` / `ValueError`. The subclass tells the kind of the error - `DeviceError` (a failed device method, acquisition or communication), `PixetFileError` (also an `OSError`), `PixetTimeoutError` (also a `TimeoutError`), `PixetValueError` (also a `ValueError`) and `PixetNotFoundError` (also a `LookupError`). Every exception has code (an `ErrorCode` member) and message. The classes exist as soon as `pypixet` is imported. `pypixet.start()` and `pypixet.exit()` themselves raise plain `RuntimeError` — they run before the Pixet API is up.

1.8.5

```
rc = dev.doSimpleAcquisition(1, 0.1, pixet.PX_FTYPE_AUTODETECT, "out.pmf")
if rc < 0:
    print("acquisition failed:", pypixet.getLasError())
```

1.8.6

```
try:
    dev.doSimpleAcquisition(1, 0.1, "out.pmf")
except pypixet.PixetError as exc:
    print("acquisition failed:", exc)
```

`pixet.getLastError()` has been removed — the message is on the exception.

2.2 Delete every **`destroy()`** and **`*RefInc()`** call

1.8.5

```
frame = dev.lastAcqFrameRefInc()
data = frame.data()
frame.destroy()
```

1.8.6

```
frame = dev.lastAcqFrame()
data = frame.data()
```

Removed everywhere: `destroy()`, `destroyCE()`, `incRefCount()`, `decRefCount()`, `privateInterface()`, and the `RefInc` variants `acqDataRefInc`, `acqFrameRefInc`, `lastAcqDataRefInc`, `lastAcqFrameRefInc`, `lastAcqPixelsRefInc`, `lastAcqClustersRefInc`.

2.3 Device parameters: one generic accessor

`dev.parameters()` returned a parameter manager with 13 typed `create/get/set` families. It is replaced by the `dev.params` property and a single value API.

1.8.5

```
bias = dev.parameters().get("BiasVoltage").getDouble()
dev.parameters().get("BiasVoltage").setDouble(200.0)
```

1.8.6

```
bias = dev.params.getValue("BiasVoltage")
dev.params.setValue("BiasVoltage", 200.0)
```

`getValue()` returns the natural Python type. If you need to know it, `dev.params.type("BiasVoltage")` returns an `int` you can compare against the new `pixet.DataType` enum (`Double`, `String`, ...), which is an `IntEnum` — `dev.params.type(name) == pixet.DataType.DOUBLE`. Blobs come back through `getBlob()`, and `isBlob()` tells you when to use it. `addValue(name, value, description='', dataType=-1)` replaces `addB00L` / `addI32` / `addString` and the rest.

2.4 Getters that became properties

<code>dev.parameters()</code>	→ <code>dev.params</code>
<code>dev.pixCfg()</code>	→ <code>dev.pixcfg</code>
<code>dev.dacs()</code>	→ <code>dev.dacs</code>
<code>pixet.dataMgr()</code>	→ <code>pixet.data_mgr</code>

Drop the parentheses — these are attributes, not calls. `dacs` keeps its name and only loses the parentheses; the rest of the DAC api changed as well, see §2.8.



2.5 Acquisition results

```
dev.lastAcqFrameRefInc() → dev.lastAcqFrame()
dev.lastAcqDataRefInc() → dev.lastAcqData()
dev.acqFrameRefInc(i)   → dev.acqFrame(i)
dev.acqDataRefInc(i)    → dev.acqData(i)
```

`doSimpleAcquisition()` has no `fileType` argument any more - the file type is always detected from the file name (`PX_FTYPE_AUTODETECT`) - and `fileName` defaults to `' '`, so `dev.doSimpleAcquisition(1, 0.1)` is enough for a measurement you only want in memory.

New on Timepix3 devices:

```
dev.lastAcqClustersRefInc() → dev.lastAcqClusters()
dev.lastAcqPixelsRefInc()   → dev.lastAcqPixels()
```

2.6 **save()** takes the data manager

1.8.5

```
frame.save("out.pmf", pixet.PX_FTYPE_AUTODETECT)
```

1.8.6

```
frame.save(pixet.data_mgr, "out.pmf", pixet.FileType.AUTODETECT)
```

`load()` is unchanged.

2.7 Pixel masking: factory mask vs user mask

The single mask was split into a factory mask (from the detector's calibration) and a user mask (yours), plus a read-only combined view. Every `mask*` name gained a `Factory` or `User` qualifier.

1.8.5

```
dev.pixCfg().mask(pixel, True)
n = dev.pixCfg().maskedCount()
m = dev.pixCfg().maskMatrix()
```

1.8.6

```
dev.pixcfg.maskUser(pixel, True)
n = dev.pixcfg.userMaskedCount()      # or combinedMaskedCount()
m = dev.pixcfg.userMaskMatrix()       # or combinedMaskMatrix()
```

`maskUser()` takes a linear pixel index, as `mask()` did; `maskUserRect(x, y, width, height, masked)` is the one that works in coordinates.

The same rule applies to the chip, row, column and rect variants (`maskUserChip`, `maskUserRow`, `maskUserColumn`, `maskUserRect`, `isUserMasked*`, `setUserMaskMatrix*`, ...) and to their `Factory` counterparts. Also new: `clearUserMask()`, `clearFactoryMask()`, `copyUserMask()`, `copyMasksFromPixCfg()`, `isCombinedMasked()`.

On Medipix2/Timepix pixel configs, the whole-matrix `modeMatrix()` / `setModeMatrix()` were removed; the per-chip `modeMatrixChip()` / `setModeMatrixChip()` and the `setMode*` calls are unchanged.

2.8 DACs: a property, and **`copy()`** instead of **`createCopyOfDacs()`**

1.8.5

```
dacs = dev.dacs()                # the DACs of the device
copy = dev.createCopyOfDacs()    # an independent set with the values of the device
...
copy.destroy()
```

1.8.6

```
dacs = dev.dacs                # a property, the same object on every access
copy = dacs.copy()             # an independent set with the values of 'dacs'
```

`dev.createCopyOfDacs()` no longer exists — `dev.dacs.copy()` replaces it, and `copy.copy(dev.dacs)` / `copy.deepcopy(dev.dacs)` do the same. Both hand back a set which already carries the values it was copied from and which nothing has to destroy. The difference is what is copied: `createCopyOfDacs()` always took the DACs of the device, while `copy()` takes the set you call it on — `dev.dacs_opm["CSM"].copy()` copies that operation mode. `copyFromDacs()` is still there for copying between two sets you already have.

`dacs.is_main` tells the DACs of the device (1) from a copy (0), and writing a copy leaves the device untouched until you send the set to it.

Setting values is unchanged: `dacs.setDac(index, chip, value)`, `dacs.setToDevice()`, `dacs.setDefault()`, `dev.setDacs(dacs)`, `dev.refreshDacs()`. `dacs.names()` returns the DAC names as a list of `str`, and `thresholdToDac()` / `dacToThreshold()` convert against the chip type of the device.

`dev.createCopyOfPixCfg()` went the same way — `dev.pixcfg.copy()`.

Medipix3 operation modes

1.8.5

```
dacs = dev.dacsMode(pixet.PX_MPX3_OPM_CSM)
copy = dev.createCopyOfDacsMode(pixet.PX_MPX3_OPM_CSM)
dev.setDacsMode(pixet.PX_MPX3_OPM_CSM, dacs)
```

1.8.6

```
dacs = dev.dacs_opm[pixet.Mpx30pMode.CSM]    # or dev.dacs_opm["CSM"]
copyDacs = dacs.copy()
copyDacs.setToDevice()                       # dev.setDacsMode(pixet.Mpx30pMode.CSM,
copyDacs)
```

`dacsMode()` and `createCopyOfDacsMode()` are removed. `dev.dacs_opm` is a sequence of the `IMpxDacs` of every operation mode - `len()`, iteration, `names()`, and an index which is the mode number or its name (case insensitive, the `PX_MPX3_OPM_` prefix optional). The object of a mode is cached, so `dev.dacs_opm[0]` is `dev.dacs_opm[0]`, and

modes which share one set of DACs in the device share the object. `dev.pixcfg_opm` is the same for the pixel configurations, in place of `pixCfgMode()` / `createCopyOfPixCfgMode()`.

2.9 Type casting: check the data format

The wrapper classes now form a real hierarchy — `IDevMpx3` derives from `IDevMpx` derives from `IDev`, `IMpxFrame` derives from `IData`. Most of the time no cast is needed at all: `dev.lastAcqFrame()` and `dev.acqFrame(i)` already hand you an `IMpxFrame`.

```
# 1.8.5
frame = data.asIMpxFrame()
if data.canCastToDataFormat(pixet.PX_DATAFORMAT_FRAME):
    ...
```

```
# 1.8.6
if data.dataFormat() == pixet.DataFormat.FRAME:
    ...
```

Removed: `asIData()`, `asIMpxFrame()`, `asIDev()`, `canCastToDataFormat()`.

The wrapper types themselves are not module attributes — there is no `pypixet.IMpxFrame` to pass to `isinstance()`. Test the data format as above, or compare `type(data).__name__` with `"IMpxFrame"`.

2.10 Other renamed calls

1.8.5	1.8.6
<code>frame.convertToChipByChip()</code>	<code>frame.chipByChipImage()</code>
<code>frame.ignoreMasked()</code>	<code>frame.setIgnoreMasked()</code>

The plugin API is reduced, not removed: `pixet.plugins()` still lists the loaded plugins and `IPlugin` keeps `id()`, `name()`, `execute()`, `params`, `registerEvent()` and `unregisterEvent()`. Gone from it are `pixet.addPlugin()`, `pixet.plugin(name)` and the `IMenuItem` type.

Also removed with no replacement in this release: the filter config-dialog hooks (`hasConfigGUI()`, `showConfigGUI()`), custom file-type registration (`dataMgr.registerFileType()` / `unregisterFileType()`), and the `IByteBuff` / `IRefByteBuff` buffer objects — buffer data is passed as plain Python lists.

2.11 **pygui**: widgets are classes

If you build panels with **pygui**, the factory methods are gone. Construct widgets directly.

```
# 1.8.5
app = pygui.App(sys.argv)
win = pygui.newMainWindow()
btn = win.newPushButton("Start")
plot = win.newPlot()
btn.setEventHandler(on_click)
```



1.8.6

```
app = pygui.Application(sys.argv)
win = pygui.MainWindow()
btn = pygui.PushButton("Start", win)
plot = pygui.Plot(win)
btn.eventFilter().setHandler(on_click)
```

- All 24 newXxx() factory methods are replaced by constructors of the same name minus new — except newButton(), which was an alias of newPushButton(); use PushButton.
- setEventHandler() / setFilterEvents() / updatefilterEvents() / getfilterEvents() are replaced by eventFilter(), which gives the one EventFilter object of the widget - handler() / setHandler(), events() / setEvents() and isEnabled() / setEnabled() are its methods.
- App() became the Application(argv) constructor.
- The remaining application module functions moved onto Application as static methods — appExists() is Application.exists(), and processEvents(), getMainWindow(), screens(), screenGeometry(), setCursor(), cursorShape(), beep() keep their names (pygui.Application.processEvents(), ...). loadFromUiFile() / loadFromUiString() stay module functions.
- The pygui.app module attribute is gone — keep the Application object you construct.
- WPlot is now Plot. setRangeZ() was removed; layers, lines, scatters, highlights and the axes are returned as objects instead of integer indices. A line, a bar line and a highlight are added on the layer (plot.addLayer().addLine(), plot.layer(0).addHighlight()), and the index based duplicates of the object methods are gone (setLineColor(), setLineWidth(), setScatter*(), setHighlight*(), rangeX/Y(), setRangeX/Y(), setAutoRangeX/Y(), setScaleTypeX/Y(), the counts, clearLayer/Line/Scatter(), showLine/Scatter()). plot.setLineColor(0, 0, 'red') is plot.layer(0).addLine('red'), plot.setRangeX(0, 10) is plot.axisX.setRange(0, 10). On the layer setLineData(), addLinePoint(), replaceLinePoint() and addScatterPoint() keep taking indices.
- MpxFrame.setAutoResize() was removed.

2.12 Constants became enum classes

The PX_*, PLUGIN_* and PLGERR_* constants are no longer attributes of pixet. Each group is an enum class of the pypixet module, pixet holds the same classes (pypixet.FileType is pixet.FileType). The member name is the constant without its group prefix, in upper case (PX_FTYPE_PNG → FileType.PNG, PX_CE_INIT_DEVICE → CeCmd.INIT_DEVICE).

1.8.5

```
frame.save("out.png", pixet.PX_FTYPE_PNG)
```

1.8.6

```
frame.save("out.png", pixet.FileType.PNG)
```

IntEnum / IntFlag members still compare equal to the old numbers and StrEnum members to the old strings, so a plain int or str is accepted wherever a member is.

Constants	Enum class	Example
PX_PAR_*	ParamFlag	PX_PAR_READONLY → ParamFlag.READONLY



PX_ATTR_*	ParamAttr	PX_ATTR_READONLY [] ParamAttr.READONLY
PX_PAR_VALUE_*	ParamChange	PX_PAR_VALUE_CHANGED [] ParamChange.CHANGED
PX_PARAM_*	PixetParam	PX_PARAM_APIVERSION [] PixetParam.APIVERSION
PX_DATAFORMAT_*	DataFormat	PX_DATAFORMAT_FRAME [] DataFormat.FRAME
PX_FTYPE_*	FileType	PX_FTYPE_AUTODETECT [] FileType.AUTODETECT
PX_DATAEVENT_*	DataEventType	PX_DATAEVENT_BEFORE_SAVE [] DataEventType.BEFORE_SAVE
PX_FRAMETYPE_*	FramePixelType	PX_FRAMETYPE_I16 [] FramePixelType.I16
PX_FRAMESAVE_*	SaveFlag	PX_FRAMESAVE_BINARY [] SaveFlag.BINARY
PX_MFFORMAT_*	MultiFrameFormat	PX_MFFORMAT_BINARY_MULTIFRAME [] MultiFrameFormat.BINARY_MULTIFRAME
PX_DATAFILTER_*	DataFilterType	PX_DATAFILTER_NORMAL [] DataFilterType.NORMAL
PX_DEVINFO_VENDOR_*	Vendor	PX_DEVINFO_VENDOR_IEAP [] Vendor.IEAP
PX_DEVINFO_TYPE_*	DevModel	PX_DEVINFO_TYPE_DUMMY [] DevModel.DUMMY
PX_DEVTYPE_*	DevType	PX_DEVTYPE_MPX2 [] DevType.MPX2
PX_DEVSTATUS_*	DevStatus	PX_DEVSTATUS_OK [] DevStatus.OK
PX_EVENT_*	EventName	PX_EVENT_BUSY_CHANGED [] EventName.BUSY_CHANGED
PX_PAR_EVENT_VALUE_CHANGED	EventName	PX_PAR_EVENT_VALUE_CHANGED [] EventName.PARAMETER_VALUE_CHANGED
PX_EVENT_XRAY_STATUS_*	XrayEventName	PX_EVENT_XRAY_STATUS_XRAYON [] XrayEventName.XRAYON
PX_MSGTYPE_*	LogMsgType	PX_MSGTYPE_INFO [] LogMsgType.INFO
PX_ACQTYPE_*	AcqType	PX_ACQTYPE_FRAMES [] AcqType.FRAMES
PX_ACQMODE_*	AcqMode	PX_ACQMODE_NORMAL [] AcqMode.NORMAL
PX_REPEAT_*	RepeatFlags	PX_REPEAT_OVERWRITE_SERIES [] RepeatFlags.OVERWRITE_SERIES
PX_CHIPTYPE_*	ChipType	PX_CHIPTYPE_MXR [] ChipType.MXR
PX_PIXCFG_MXR, ... (chip types)	PixCfgType	PX_PIXCFG_MXR [] PixCfgType.MXR
PX_PIXCFG_ALL, ... (config items)	PixCfgPart	PX_PIXCFG_ALL [] PixCfgPart.ALL
PX_THLFLG_*	ThlFlag	PX_THLFLG_NONE [] ThlFlag.NONE
PX_MPX3_OPM_*	Mpx3OpMode	PX_MPX3_OPM_SPM_1CH [] Mpx3OpMode.SPM_1CH
PX_MPX3_CNT_*	Mpx3Counter	PX_MPX3_CNT_LOW [] Mpx3Counter.LOW
PX_MPX3_CNTD_*	Mpx3Depth	PX_MPX3_CNTD_12B [] Mpx3Depth.BITS_12
PX_MPX3_COLB_*	Mpx3ColBlock	PX_MPX3_COLB_ALL [] Mpx3ColBlock.ALL
PX_MPX3_ROWb_*	Mpx3RowBlock	PX_MPX3_ROWb_ALL [] Mpx3RowBlock.ALL
PX_MPX3_GAIN_*	Mpx3Gain	PX_MPX3_GAIN_SUPERHIGH [] Mpx3Gain.SUPERHIGH
PX_TPX3_OPM_*	Tpx3OpMode	PX_TPX3_OPM_TOATOT [] Tpx3OpMode.TOATOT
PX_TPX3_GRAYCNT_*	Tpx3GrayCounter	PX_TPX3_GRAYCNT_AUTO [] Tpx3GrayCounter.AUTO
PX_TPX3_FRAMETYPE_*	Tpx3FrameType	PX_TPX3_FRAMETYPE_SUM [] Tpx3FrameType.SUM
PX_TPX2_OPM_*	Tpx2OpMode	PX_TPX2_OPM_TOT10_TOA18 [] Tpx2OpMode.TOT10_TOA18
PX_STGTYPE_*	SettingsFormat	PX_STGTYPE_INI [] SettingsFormat.INI
PLUGIN_ID_*	PluginId	PLUGIN_ID_DEV_FEEDER [] PluginId.DEV_FEEDER
PLUGIN_EVENT_*	PluginEvent	PLUGIN_EVENT_INITIALIZED [] PluginEvent.INITIALIZED
PLGERR_*	PluginErrorId	PLGERR_NO_ERROR [] PluginErrorId.NO_ERROR
PXERR_* (pxerrors.h)	ErrorCode	PXERR_FILE_OPEN [] ErrorCode.FILE_OPEN



PX_MXR_* (Medipix2 MXR DACs)	MxrDacs	PX_MXR_IKRUN [] MxrDacs.IKRUN
PX_TPX_* (Timepix DACs)	TpxDacs	PX_TPX_IKRUN [] TpxDacs.IKRUN
PX_MPX3_* (Medipix3 DACs)	Mpx3Dacs	PX_MPX3_TH0 [] Mpx3Dacs.TH0
PX_TPX3_* (Timepix3 DACs)	Tpx3Dacs	PX_TPX3_IBIAS_PREAMP_ON [] Tpx3Dacs.IBIAS_PREAMP_ON
PX_TPX2_* (Timepix2 DACs)	Tpx2Dacs	PX_TPX2_PREAMP_ON [] Tpx2Dacs.PREAMP_ON
PX_TPX2_DACOUT_*	Tpx2DacOuts	PX_TPX2_DACOUT_DOUT_PREAMP_ON [] Tpx2DacOuts.DOUT_PREAMP_ON

A class is created only when a method, an argument or a struct field uses it. No method uses PX_DFUID_*, PX_EXT_*, PX_METADATA_*, PX_SENSOR_TYPE_*, PX_TPXMODE_*, PX_MPX2_DOUTPUT_*, PX_DEVMPIX2_CALIB_*, PX_TPX3_VCO_*, PX_PIXETTYPE_* and PX_STATUS_*, so they are no longer available in Python - use the plain value.

Only four values stay plain attributes of `pixet`: `PX_PIXET_VERSION` [] `pixet.VERSION`, `PX_CHIP_ALL` [] `pixet.CHIP_ALL`, `PX_TPX3_PIXSAVE_FILTERDUMMY` [] `pixet.TPX3_PIXSAVE_FILTERDUMMY` and `PX_TPX3_PIXADD_UPDATE_PREVIEW` [] `pixet.TPX3_PIXADD_UPDATE_PREVIEW`.

3. What's new

Detector control

- `pixet.devicesTpx2()` joins the existing per-type device lists.
- Device log events: `dev.registerLogEvent()` / `unregisterLogEvent()`.
- `registeredEvents()` lists what `registerEvent()` has registered on an object - one `EventItem` per registration, with the read only members `name`, `callbackId` and `func` (the registered callable). On `IDev`, `IDev2`, `IPixet`, `IParamSet` and `IPlugin` - everything that registers events. Log events have their own handler and are not in the list.
- Constants are enum classes. `pixet.PX_FTYPE_AUTODETECT` and friends were invisible to `dir()` and to editors. They are now members of enum classes of the module - `pypixet.FileType.AUTODETECT`, `pypixet.DataType.DOUBLE` - listed in the stub, so autocompletion finds them, and the methods return members (`dev.deviceType()` is `<DevType.TPX3: 3>`).

pygui

New widgets and types: `Table`, `TreeWidget`, `ScrollArea`, `Frame`, `DateEdit`, `TimeEdit`, `DateTimeEdit`, `BoxLayout`, `FormLayout`, and the value types `Point`, `Size`, `Rect`, `Color`. Qt constants are exposed as proper enums — `CursorShape`, `WindowType`, `EventType`, `AligmentFlag`, `ScrollBarPolicy`, `LayoutSizeConstraint`, `FrameShape`, `FrameShadow` and more.

`Plot` gains per-object layers, lines, scatters, highlights and axes (`PlotAxis` on `axisX`, `axisY`, `secAxisX`, `secAxisY`), log/linear axis scales, colour maps, the nested `ScaleType` / `AxisType` / `AxisOrientation` / `LabelType` / `ColorMap` enums, and `setLineData()` for updating a curve in one call. `MpxFrame` / `MpxFramePanel` gain `data()`, `setGamma()`, zoom control (`setZoom()`, `setZoomRect()`, `zoomFactor()`, `zoomOut()`), selection rectangles and a `toolTipCallback` member, plus the nested `SizeType` / `MarkType` enums. `ComboBox` and `PropertyTreeView` are much more complete.

The dialog module functions are unchanged — `showMessage()`, `showWarning()`, `showError()`, `showQuestion()`, `inputText()`, `showItemSelection()`, `getOpenFileName()`, `getSaveFileName()`, `getSaveFileNameAndFilter()`, `getExistingDirectory()` all keep their 1.8.5 names and signatures.

pyclustering

A real module rather than a plugin shim, with `Pixel` and `Cluster` value types:

```
import pypixet
import pyclustering

pixet = pypixet.start()
device = pixet.devices()[0]

c = pyclustering.Clustering(device)
c.newClustersCb = lambda clusters, acqIndex: print(len(clusters), "clusters")
c.loadCalibrationFromDevice()
c.startMeasurement(1.0, 60.0, "run.clog")
```

`Clustering(device)` takes the device the measurement runs on; without it only `replayData()` works — `startMeasurement()` and `loadCalibrationFromDevice()` need one. The methods take positional arguments only, keyword arguments raise `TypeError`.

Also: `replayData()`, `abort()`, `isRunning()`, `loadCalibrationFromFiles()`, `isCalibrationLoaded()`, and the callbacks `messageCb`, `progressCb`, `newClustersWithPixelsCb`, `acqStartedCb`, `acqFinishedCb`.

pyspectraimg

Replaces the flat `si*` functions with a `SpectraImg` object and a `ProcParameters` settings object:

```
import pyspectraimg

si = pyspectraimg.SpectraImg(dev)
si.progressCb = lambda finished, pct: print(f"{pct:.0f}%")
si.loadCalibrationFromDevice()
si.startMeasurement(1.0, 60.0, "run.h5", True)
img = si.getFrameForEnergyRange(10, 40, True)
```

The device argument is mandatory — `SpectraImg` reads the matrix size and the chip layout from it. As in `pyclustering`, the methods take positional arguments only.

Plus `getGlobalSpectrum()`, `getGlobalSpectrumInRect()`, `saveSumFrame()`, `saveDataAsFramesToFile()`, `setXrfParams()`, `processedPixelsPerSecond()`, and XRF correction options on `ProcParameters`.

pyutils

- New `linearRegression(y, x)`.

- Results are named tuples now, so `result.mean` works alongside `result[0]`: `SpectrumPeak`, `GerfFitResult`, `FitPeakResult`, `PerformToTfitResult`, `LinearRegressionResult`, `ClustersWithParams`.
- `calibGetSpectrumPeak()` was renamed to `calibSpectrumPeak()`.
- `performToTfit()` no longer takes a `stddev_out` argument — the value is in the result.

pyftdi

The module-level functions now raise `RuntimeError` with the FTDI error text instead of returning a status code, and `open()` closes a device left open by an earlier `open()`. The `Ftdi` class — which already existed in 1.8.5 and lets you drive several devices at once — keeps its status-code return values, so its methods are unchanged.

1.8.5

```
rc = pyftdi.open("FT4232H", 115200)
if rc != 0:
    ...
```

1.8.6

```
try:
    pyftdi.open("FT4232H", 115200)
except RuntimeError as exc:
    ...
```

Editor support

Every module ships a `.pyi` type stub — `pypixet`, `pygui`, `pyutils`, `pyclustering`, `pyspectraimg`, `pyftdi`, `pyhid`. Editors and type checkers pick these up automatically and give you completion, parameter hints and type checking against the API.

The stubs are generated, and two of them are behind the sources in this release: `pypixet.pyi` lists only a few of `IPixet`'s methods and still declares the removed `getLastError()`, and `pygui.pyi` omits the static methods of `Application`. The modules themselves are correct — use `help()` / `dir()` when a stub looks incomplete.

4. Removed modules

Module	Replacement
<code>pymeasutils</code>	<code>pyspectraimg</code> — same functionality, object API
<code>pybhfilt</code> , <code>pyffilt</code>	The filters themselves are unchanged; reach them through <code>pixet.filter_mgr</code> (<code>filters()</code> , <code>filterByName()</code> , <code>addFilter()</code>)
<code>pypxproc</code>	No direct replacement
<code>pydevcontrol</code>	No direct replacement
<code>pyequalization</code>	No direct replacement
<code>pyfitpix</code> , <code>pyminipix</code> , <code>pymodupix</code>	Device-specific helpers; use the standard <code>IDev</code> / <code>IDevMpx</code> API
<code>pypyscripting</code>	No direct replacement

The underlying Pixet features are unaffected — it is the thin Python wrapper modules that were retired. If you depend on



one of these, contact support before upgrading so we can cover your use case.

5. Upgrade checklist

1. Wrap acquisition and configuration calls in `try / except pypixet.PixetError`; remove `-1` checks and `pypixet.getLastError()`.
2. Delete every `destroy()` call and rename `*RefInc()` to the plain name.
3. Replace `dev.parameters().get(name).getX()` with `dev.params.getValue(name)`.
4. Drop the parentheses from `dev.dacs()`, `dev.pixCfg()`, `pixet.dataMgr()`, `pixet.filterMgr()`.
5. Replace `dev.createCopyOfDacs()` with `dev.dacs.copy()` (and `dev.dacsMode(m)` with `dev.dacs_opm[m]`).
6. Add `pixet.data_mgr` as the first argument to `save()`.
7. Qualify mask calls with `User` or `Factory`.
8. Drop `asIMpxFrame()` — the acquisition getters already return `IMpxFrame` — and replace `canCastToDataFormat()` with a `data.dataFormat()` comparison.
9. In `pygui` code, swap `newXxx()` factories for constructors and `setEventHandler()` for `eventFilter().setHandler()`.
10. Replace the `pixet.PX_*` constants with the enum class members (§2.12).
11. Check §4 for any removed module you import.