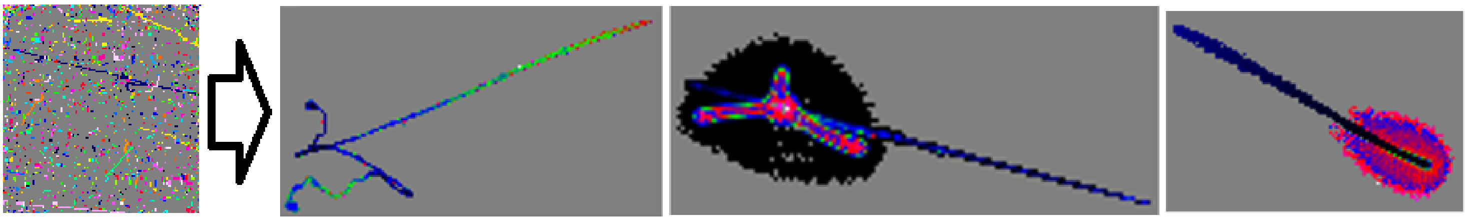
Binary Clustering API

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

Contents

Introduction

The Clustering is designed for easy work with clusters of the pixels. Typically used to determine energy and other parameters of high energy particles.



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, clusterimgapi.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 pxcInitialize+pxcGetIPixet initializing way used)

(on Linux equivalent SOs instead of DLLs)

- Other auxilliary files:
  - pixet.ini
  - device config file(s)
  - hwlibs and optional other device specific files (only for measuring)

See
[Pixet SDK: Auxilliary files]
[Files and directories of the Pixet and SDK]

The Clustering API basic workflow

This API is defined in the clusteringapi.h file.

Typical usage

1.a Load the Pixet core using the pxpClLoadPixetCore("pxcore.dll").

1.b Or normally start the application using the Pixet core API pxcInitialize() function, get the core pointer using pxcGetIPixet(), set the core pointer to the Clustering API using pxpClSetIPixet(iPixet);

2. Get the clustering instance handle using pxpClCreate(device_index);
3. Set-up the callbacks (not required).
4. Load the calibration (not required).
5. Start the measurement or repaly your data.
6. Use the data via one of NewClusters callbacks or from saved clog file.

(For 1.a/b init ways see examples: Initializing using the pxpClLoadPixetCore and Initializing from the active Pixet_core)

General functions

LoadPixetCore and UnloadPixetCore

pxpClLoadPixetCore

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

Definition

```
PXCLAPI int pxpClLoadPixetCore(const char* pxCoreLiPath);
```

Parameters

- pxCoreLiPath – path to the pxcore library

Return value

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

Example

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

pxpClUnloadPixetCore

Deinitializes and unloads the Pixet core.

Definition

```
PXCLAPI void pxpClUnloadPixetCore();
```

Parameters

- (no pars)

Return value

(void)

Example

```
pxpClUnloadPixetCore();
```

SetIPixet and GetIPixet

pxpClSetIPixet

Sets the internal Pixet API pointer. This is used when pxcore library is loaded separately in application. The use must pointer obtained via function pxcGetIPixet and should not load the pixet core with pxpClLoadPixetCore function.

Definition

```
PXCLAPI void pxpClSetIPixet(void* pixet);
```

Parameters

- pixet – The iPixet pointer from the pxcGetIPixet() function of the pxcore API.

Return value

(void)

Example

```
iPixet = pxcGetIPixet(); // Warning: Use the pxcGetIPixet from pxcapi.h,
                         // not pxpClGetIPixet from clusteringapi.h
if (iPixet==0) msgToList("pxcGetIPixet=NULL"); else msgToList("pxcGetIPixet OK");
pxpClSetIPixet(iPixet);
```

pxpClGetIPixet

Return internal Pixet 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 pxpClLoadPixetCore.

Definition

```
PXCLAPI void* pxpClGetIPixet();
```

Parameters

(no pars)

Return value
internal Pixet structure pointer or 0 if not set

Warning

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

Create and Free

pxpClCreate

Creates a new instance of clustering.

Definition

```
PXCLAPI clhandle_t pxpClCreate(int deviceIndex=CL_NO_DEVICE);
```

Parameters

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

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

Return value
Returns the handle of newly create instance of clustering, or CL_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

```
clHandle = pxpClCreate(0);
if (clHandle==CL_INVALID_HANDLE) msgToList("pxpClCreate INVALID");
else msgToList("pxpClCreate OK");
```

pxpClFree

Frees the created instance of clustering

Definition

```
PXCLAPI int pxpClFree(clhandle_t handle);
```

Parameters

- handle – clustering handle

Return value

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

GetLastError

pxpClGetLastError

Definition

```
PXCLAPI int pxpClGetLastError(clhandle_t handle, char* errorMsgBuffer, unsigned size);
```

Parameters

- handle – clustering handle received from function pxpClCreate

- 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 CL_ERR_XXX code.

Example

```
// Using the pxpClGetLastError 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];
        pxpClGetLastError(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);
}
```

LoadCalibrationFromDevice and LoadCalibrationFromFiles

pxpClLoadCalibrationFromDevice

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

Definition

```
PXCLAPI int pxpClLoadCalibrationFromDevice(clhandle_t handle);
```

Parameters

- handle – clustering handle received from function pxpClCreate

Return value

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

Example

```
errorToList("pxpClLoadPixetCore", rc);
```

pxpClLoadCalibrationFromFiles

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

Definition

```
PXCLAPI int pxpClLoadCalibrationFromFiles(clhandle_t handle, const char* filePaths);
```

Parameters

- handle – clustering handle received from function pxpClCreate
- 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 CL_ERR_XXX code.

Example

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

StartMeasurement, ReplayData, IsRunning and Abort

pxpClStartMeasurement

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

Definition

```
PXCLAPI int pxpClStartMeasurement(clhandle_t handle, double acqTime, double measTime,
const char* outputFilePath);
```

Parameters

- handle – Clustering instance handle
- acqTime – acquisition time of a single frame / pixel measurement in seconds.

  – Primarily intended for frame-based only 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 supporting devices (Timepix3, no Timepix), this is the ToA limit. After exceeds, a ToA is resets and acqIndex in the newClusters... callbacks is incremented. acqTime=measTime can be used.

- measTime – total time of measurement in seconds. Use 0 to endless measurement (progress will always 100%).
- outputFilePath – 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 CL_ERR_XXX code.

Example

```
// measure for 60 seconds and save the clog.
int rc;
if (devType=="Tpx3") rc = pxpClStartMeasurement(handle, 60, 60, "out-files/test.clog");
else rc = pxpClStartMeasurement(handle, 0.01, 60, "out-files/test.clog");
errorToList("pxpClStartMeasurement", rc);
```

pxpClReplayData

Replays and process already measured data files (*.pmf, *.txt, *.t3r, *.t3pa, ...) and calls the corresponding callbacks. If calibration is loaded, energy values will be calibrated. If the output path defined and ending with .clog, cluster log will be saved.

Definition

```
PXCLAPI int pxpClReplayData(clhandle_t handle, const char* filePath, const char*
outputFilePath, bool blocking);
```

Parameters

- handle – Clustering instance handle
- filePath – full path to a data file to be replayed
- outputFilePath – output file where the process data (clusters) will be saved (*.clog). If saving not required, put "".
- blocking – whether the function will block until all clusters processed

Return value

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

Clog files note

For historical reasons, there are two CLOG formats. Tpx3 have one additional column. The pxpClReplayData 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.

pxpClIsRunning

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

Definition

```
PXCLAPI int pxpClIsRunning(clhandle_t handle);
```

Parameters

- handle – Clustering instance handle

Return value

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

pxpClAbort

Aborts the measurement or replaying of the data.

Definition

```
PXCLAPI int pxpClAbort(clhandle_t handle);
```

Parameters

- handle – Clustering instance handle

Return value

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

The callbacks

Typical callbacks orders

Measure

1. AcqStarted 0
2. Progress with finish false (0 to more occurance)
3. NewClusters or NewClustersWithPixels (0 to more occurance)
4. Progress with finish false (0 to more occurance)
5. AcqFinished 0
6. (repeats of 1-5 with Acq number 1 and more: depends of measTime/acqTime ratio and CPU usage)
7. Progress with finish true

Repaly

1. Message: err=0, Processing file filename...
2. Progress with finish false (0 to more occurance)
3. NewClusters or NewClustersWithPixels (0 to more occurance)
4. Progress with finish true

SetNewClusters... Callback functions

This are functions that where sets the new cluster callback. There are 2 variants ot the outputs: Clusters parameters list or Cluster list including pixels lists of each cluster. Only one of the callbacks NewClustersCallback or NewClustersWithPixelsCallback can be set. If was set both, only last is working.

Related declarations

The functions described in this chapter use the following definitions:

```
// Definition of Cluster Pixel
typedef struct _PXPPixel {
    unsigned short x; // x coordinate of pixel
    unsigned short y; // y coordinate of pixel
    double toa; // time of arrival in nano seconds
    float energy; // energy of pixel in keV or ToT count if no calibration
} PXPPixel;

// Definition of cluster
```

```
typedef struct _PXPCluster {
    unsigned eventID; // event id to recognize coincidence events (same ID)
    float x; // x coordinate of cluster
    float y; // y coordinate of cluster
    float energy; // energy of cluster in keV
    double toa; // time of arrival
    unsigned short size; // size of cluster in pixels (number of pixels)
    float height; // maximal pixel value in cluster
    float roundness; // roundness (0 - 1, but to 2.7 in older versions)
    float linearity; // linearity (0 - 1)
    size_t borderpixcount; // The border pixel is hit, but not internal.
} PXPCluster;

// Definition of cluster with Pixel data
typedef struct _PXPClusterWithPixels {
    unsigned eventID; // event id to recognize coincidence events (same ID)
    float x; // x coordinate of cluster
    float y; // y coordinate of cluster
    float energy; // energy of cluster in keV
    double toa; // time of arrival
    unsigned short size; // size of cluster in pixels (number of pixels)
    float height; // maximal pixel value in cluster
    float roundness; // roundness (0 - 1, but to 2.7 in older versions)
    float linearity; // linearity (0 - 1)
    size_t borderpixcount; // The border pixel is hit, but not internal.
    PXPPixel* pixels;
} PXPClusterWithPixels;
```

roundness

Based on the ratio between the radius of the circle that would have S=number of pixels and the farthest pixel from the weighted center of the cluster.

linearity

Derived from the approximation of a rigid body and its distribution of mass around the main axis of rotation, aka as their ratio between the 1st and 2nd main axis of rotation (sometimes the main and minor are also used).

borderpixcount

The number of pixels of the border. The border pixel is hitted, but not internal. An internal pixel is a hitted pixel that has four hitted neighbors on both axes or more.

```
/// Callback when new clusters are measured/processed
/// Note: Only one of the new clusters callback can be set, either NewClustersCallback or
NewClustersWithPixelsCallback
/// @param clusters - array of new clusters
/// @param clusterCount - size of clusters array
/// @param acqIndex - index of curent acquisition
/// @param userData - data of the user that were set in set callback function
typedef void (*ClNewClustersCallback)(PXPCluster* clusters, size_t clusterCount, size_t
acqIndex, void* userData);

/// Callback when new clusters are measured/processed.
/// Note: Only one of the new clusters callback can be set, either NewClustersCallback or
NewClustersWithPixelsCallback
/// @param clusters - array of new clusters (with pixels)
/// @param clusterCount - size of clusters array
/// @param acqIndex - index of curent acquisition
/// @param userData - data of the user that were set in set callback function
typedef void (*ClNewClustersWithPixelsCallback)(PXPClusterWithPixels* clusters, size_t
clusterCount, size_t acqIndex, void* userData);
```

userData

See Binary core API: pxcRegisterAcqEvent

acqIndex

Acq number 1 and more: Depends of measTime/acqTime ratio and CPU usage.

pxpClSetNewClustersCallback

Callback when new clusters are measured/processed. Provides a statictics about clusters.

Definition

```
PXCLAPI int pxpClSetNewClustersCallback(clhandle_t handle, ClNewClustersCallback
callback, void* userData);
```

Parameters

- handle – Clustering instance handle
- callback – The callback function. The ClNewClustersCallback is defined in the clusteringapi.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 CL_ERR_XXX code.

Example

```
typedef struct {
    int d1;
    int d2;
} tUsrData;
tUsrData usrData = {123, 456};

void myNewClustersCallback(PXPCluster* clusters, size_t clusterCount, size_t acqIndex,
void*
userData) {
    tUsrData usrDat = (tUsrData*)userData;
    // do something with the usr data
    for(size_t n=0; n<clusterCount; n++) {
        // Do something with the clusters
    }
}

int rc = pxpClSetNewClustersCallback(myHandle, myNewClustersCallback, (void*)&usrData);
errorToList("pxpClSetNewClustersCallback", rc);
```

pxpClSetNewClustersWithPixelsCallback

Callback when new clusters are measured/processed. Provides a statistics and all pixel data.

Definition

```
PXCLAPI int pxpClSetNewClustersWithPixelsCallback(clhandle_t handle,
ClNewClustersWithPixelsCallback callback, void* userData);
```

Parameters

- handle – Clustering instance handle
- callback – The callback function. The ClNewClustersWithPixelsCallback is defined in the clusteringapi.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 CL_ERR_XXX code.

Example

```
typedef struct {
    int d1;
    int d2;
} tUsrData;
tUsrData usrData = {123, 456};

void myNewClustersCallback(PXPClusterWithPixels* clusters, size_t clusterCount, size_t
acqIndex, void* userData) {
    tUsrData usrDat = (tUsrData*)userData;
    // do something with the usr data
    for(size_t n=0; n<clusterCount; n++) {
        // Do something with the clusters
    }
}

int rc = pxpClSetNewClustersCallback(myHandle, myNewClustersCallback, (void*)&usrData);
errorToList("pxpClSetNewClustersWithPixelsCallback", rc);
```

Other Set...Callback functions list

Related declarations

The functions described in this chapter use the following definitions:

```
/// 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 (*ClMessageCallback)(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 (*ClProgressCallback)(bool finished, double progress, void* userData);
```

```
/// Callback for when a single acquisition is started
/// @param acqIndex - index of the current acquisition
/// @param userData - data of the user that were set in set callback function
typedef void (*ClAcqStartedCallback)(int acqIndex, void* userData);

/// Callback for when a single acquisition is finished
/// @param acqIndex - index of the current acquisition
/// @param userData - data of the user that were set in set callback function
typedef void (*ClAcqFinishedCallback)(int acqIndex, void* userData);
```

userData
See Binary core API: pxcRegisterAcqEvent

acqIndex
Acq number 1 and more: Depends of measTime/acqTime ratio and CPU usage.

pxpClSetMessageCallback

Callback for messages and error messages returned from the API.

Definition

```
PXCLAPI int pxpClSetMessageCallback(clhandle_t handle, ClMessageCallback callback, void*
userData);
```

Parameters

- handle – Clustering instance handle
- callback – The callback function. The ClMessageCallback is defined in the clusteringapi.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 CL_ERR_XXX code.

Example

```
// message callback in the Windows CLR app
void ClMessageCallbackFn(bool error, const char* message, void* userData) {
        printToBuffer("ClMessageCallbackFn: err %d: %s", (int)error, message);
        if (error && acqWithClbRunning==1) acqWithClbRunning = 2;
}
```

pxpClSetProgressCallback

Callback for progress of an operation. Occurs every 1 second while measuring or processing.

Definition

```
PXCLAPI int pxpClSetProgressCallback(clhandle_t handle, ClProgressCallback callback,
void* userData);
```

Parameters

- handle – Clustering instance handle
- callback – The callback function. The ClProgressCallback is defined in the clusteringapi.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 CL_ERR_XXX code.

pxpClSetAcqStartedCallback

Callback for when a single acquisition is started.

Definition

```
PXCLAPI int pxpClSetAcqStartedCallback(clhandle_t handle, ClAcqStartedCallback callback,
void* userData);
```

Parameters

- handle – Clustering instance handle
- callback – The callback function. The ClAcqStartedCallback is defined in the clusteringapi.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 CL_ERR_XXX code.

pxpClSetAcqFinishedCallback

Callback for when a single acquisition is finished.

Definition

```
PXCLAPI int pxpClSetAcqFinishedCallback(clhandle_t handle, ClAcqFinishedCallback
callback, void* userData);
```

Parameters

- handle – Clustering instance handle
- callback – The callback function. The ClAcqFinishedCallback is defined in the clusteringapi.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.

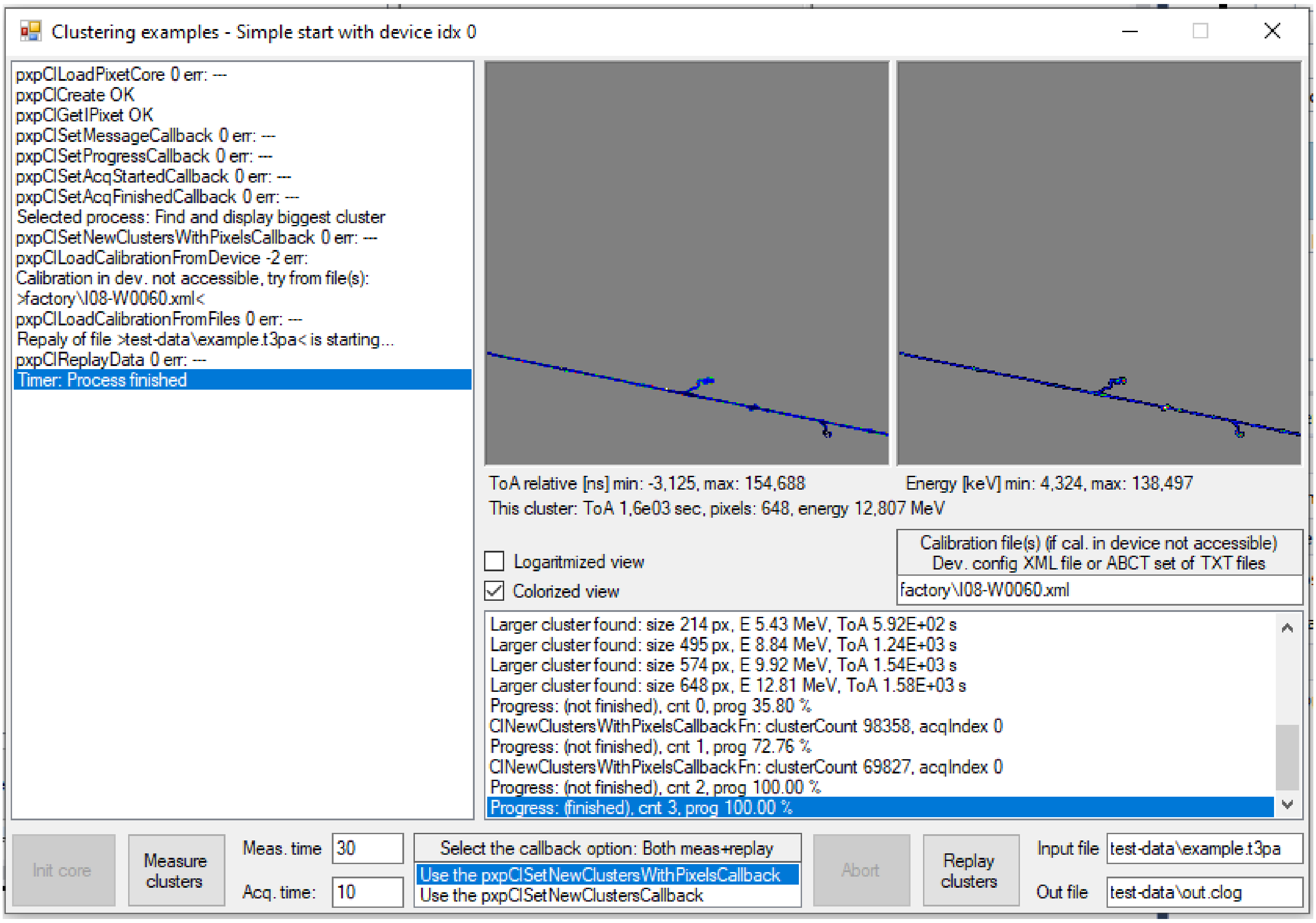
Return value

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

Examples

This chapter containing parts of example projects clustering-1 and clustering-2 from the AdvacamAPIexamples collection. The projects are almost the same, differing only in the initialization and the ability to select devices in the second version.

Download the AdvacamAPIexamples package
https://advacam.com/examples/AdvacamAPIexamples.rar
(MS Visual Studio 2017 Solution with C++ projects of Windows CLR programs)



```
pxpClLoadPixetCore 0 err: ---
pxpClCreate OK
pxpClGetIPixet OK
pxpClSetMessageCallback 0 err: ---
pxpClSetProgressCallback 0 err: ---
pxpClSetAcqStartedCallback 0 err: ---
pxpClSetAcqFinishedCallback 0 err: ---
Selected process: Find and display biggest cluster
pxpClSetNewClustersWithPixelsCallback 0 err: ---
pxpClLoadCalibrationFromDevice -2 err:
Calibration in dev. not accessible, try from file(s):
>factory\I08-W0060.xml<
pxpClLoadCalibrationFromFiles 0 err: ---
Repaly of file >test-data\example.t3pa< is starting...
pxpClReplayData 0 err: ---
Timer: Process finished
```

ToA relative [ns] min: -3,125, max: 154,688
Energy [keV] min: 4,324, max: 138,497
This cluster: ToA 1,6e03 sec, pixels: 648, energy 12,807 MeV

☐ Logaritmized view
☑ Colorized view

Calibration file(s) (if cal. in device not accessible)
Dev. config XML file or ABCT set of TXT files: factory\I08-W0060.xml

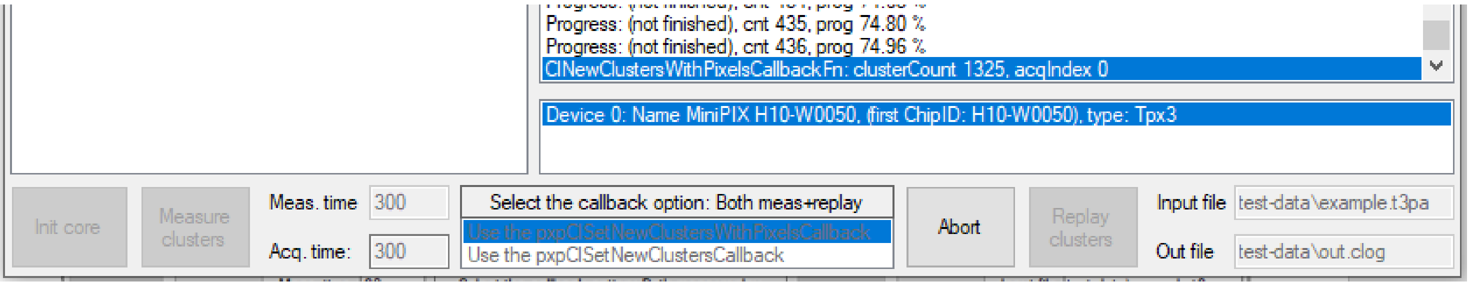
```
Larger cluster found: size 214 px, E 5.43 MeV, ToA 5.92E+02 s
Larger cluster found: size 495 px, E 8.84 MeV, ToA 1.24E+03 s
Larger cluster found: size 574 px, E 9.92 MeV, ToA 1.54E+03 s
Larger cluster found: size 648 px, E 12.81 MeV, ToA 1.58E+03 s
Progress: (not finished), cnt 0, prog 35.80 %
ClNewClustersWithPixelsCallbackFn: clusterCount 98358, acqIndex 0
Progress: (not finished), cnt 1, prog 72.76 %
ClNewClustersWithPixelsCallbackFn: clusterCount 69827, acqIndex 0
Progress: (not finished), cnt 2, prog 100.00 %
Progress: (finished), cnt 3, prog 100.00 %
```

Meas. time: 30
Acq. time: 10

Select the callback option: Both meas+replay
Use the pxpClSetNewClustersWithPixelsCallback
Use the pxpClSetNewClustersCallback

Input file: test-data\example.t3pa
Out file: test-data\out.clog

The clustering-1 screenshot



```
Progress: (not finished), cnt 435, prog 74.80 %
Progress: (not finished), cnt 436, prog 74.96 %
ClNewClustersWithPixelsCallbackFn: clusterCount 1325, acqIndex 0
```

Device 0: Name MiniPIX H10-W0050, (first ChipID: H10-W0050), type: Tpx3

Meas. time: 300

Acq. time: 300

Select the callback option: Both meas+replay

Use the pxpClSetNewClustersWithPixelsCallback

Use the pxpClSetNewClustersCallback

Input file: test-data\example.t3pa

Out file: test-data\out.clog

The clustering-2 difference

Initializing using the pxpClLoadPixetCore

```
// (simplest with blind using of the device with index 0)

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

clHandle = pxpClCreate(0);
if (clHandle==CL_INVALID_HANDLE) msgToList("pxpClCreate INVALID");
else msgToList("pxpClCreate OK");

// error messages
rc = pxpClSetMessageCallback(clHandle, ClMessageCallbackFn, NULL);
errorToList("pxpClSetMessageCallback", rc);

// progress %, is finished?
rc = pxpClSetProgressCallback(clHandle, ClProgressCallbackFn, NULL);
errorToList("pxpClSetProgressCallback", rc);

// occurs at every single acquisition is started
rc = pxpClSetAcqStartedCallback(clHandle, ClAcqStartedCallbackFn, NULL);
errorToList("pxpClSetAcqStartedCallback", rc);

// occurs at every single acquisition is finished
rc = pxpClSetAcqFinishedCallback(clHandle, ClAcqFinishedCallbackFn, NULL);
errorToList("pxpClSetAcqFinishedCallback", rc);

// arriving data containing properties of the clusters with all their pixels
rc = pxpClSetNewClustersWithPixelsCallback(clHandle, ClNewClustersWithPixelsCallbackFn,
NULL);
errorToList("pxpClSetNewClustersCallback", rc);
```

Initializing from the active Pixet core

```
msgToList("Initializing the Pixet core...");
btnInit->Enabled = 0;
int rc = pxcInitialize();
errorToList("pxcInitialize", rc);

// (omited test of device count is > 0)

iPixet = pxcGetIPixet();
if (iPixet==0) msgToList("pxcGetIPixet=NULL"); else msgToList("pxcGetIPixet OK");
// Warning: Use the pxcGetIPixet from pxcapi.h, not pxpClGetIPixet from clusteringapi.h

pxpClSetIPixet(iPixet);

// (omited feeding of the device list)

deviceIndex = listDevices->SelectedIndex;
msgToList("Selected device index: " + deviceIndex.ToString());

clHandle = pxpClCreate(deviceIndex);
if (clHandle==CL_INVALID_HANDLE) msgToList("pxpClCreate INVALID");
else msgToList("pxpClCreate OK");

// (omited setting up the callbacks)
```

Using the NewClusters callback

```
// Callback "new clusters" (without each pixel data) - The process: Display some statistics
void ClNewClustersCallbackFn(PXPCluster* clusters, size_t clusterCount, size_t acqIndex,
void*
userData) {
    printToBuffer("clusterCount %d, acqIndex %d", (int)clusterCount, (int)acqIndex);

    float maxClE = 0.0, maxClH = 0.0;
    unsigned short maxClsiz = 0;
    for (unsigned ci = 0; ci<(unsigned)clusterCount; ci++) {
        if (clusters[ci].energy>maxClE) maxClE = clusters[ci].energy;
        if (clusters[ci].height>maxClH) maxClH = clusters[ci].height;
        if (clusters[ci].size>maxClsiz) maxClsiz = clusters[ci].size;
    }
    printToBuffer("^ max single cluster size %d px, max energy %.2f MeV, max height %.2f
MeV",
maxClsiz, maxClE/1000.0, maxClH/1000.0);
}
```

Using the NewClustersWithPixels callback

```
// Callback "Clusters with pixels" - The process: Find and display biggest cluster
void ClNewClustersWithPixelsCallbackFn(PXPClusterWithPixels* clusters, size_t clusterCount,
size_t acqIndex, void* userData) {
     printToBuffer("ClNewClustersWithPixelsCallbackFn: clusterCount %d, acqIndex %d",
(int)clusterCount, (int)acqIndex);

     float maxClE = 0.0;
     for (unsigned ci = 0; ci<(unsigned)clusterCount; ci++) {
             if (clusters[ci].energy>maxClE) maxClE = clusters[ci].energy;
             if (clusters[ci].size>maxClusterSizeFound) {
                     maxClusterSizeFound = clusters[ci].size;
                    maxSizedClusterEnergy = clusters[ci].energy;
                    maxSizedClusterToa = clusters[ci].toa;
                    printToBuffer("Larger cluster found: size %d px, E %.2f MeV, ToA %.2E
s",
                clusters[ci].size, clusters[ci].energy/1000.0,
clusters[ci].toa/1000000000.0);
                    for (unsigned n = 0; n<cDevPixels; n++) {
                            rawData1[n] = 0; rawData2[n] = 0;
                    }
                    for (unsigned pi = 0; pi<(unsigned)clusters[ci].size; pi++) {
                            unsigned adr = clusters[ci].pixels[pi].x + cDevXsize *
clusters[ci].pixels[pi].y;
                            rawData1[adr] = clusters[ci].pixels[pi].toa - clusters[ci].toa;
                            rawData2[adr] = clusters[ci].pixels[pi].energy;
                    }
                    newDataToView = true;
            }
    }
}
```

Related

- Pixet SDK
- Files and directories: Main directory of the API-using programs