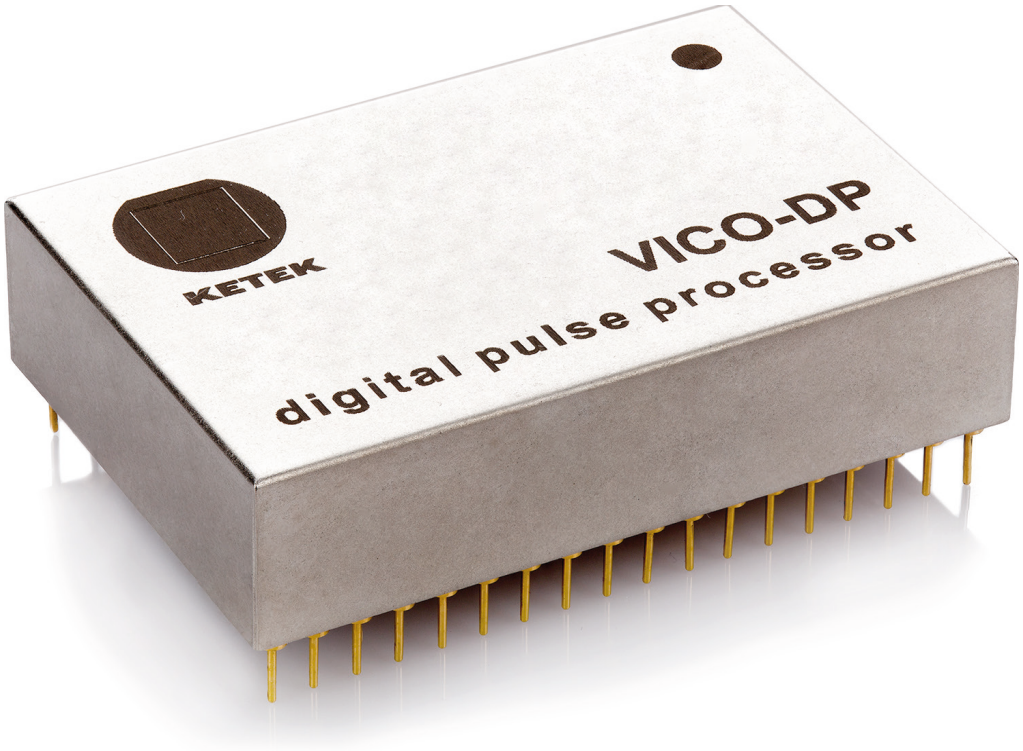
KETEK GmbH
Hofer Str. 3
81737 Munich
Germany
www.ketek.net
info@ketek.net
phone +49 89 673 467 70
fax +49 89 673 467 77

KETEK



Digital Pulse Processor
Handel DLL

Quick Start

DP
VICO

Table of Contents

This manual describes the DLL package which can be used for every KETEK system with the VICO-DP (DPP2) under Microsoft Windows operating system.

Revision history:

Rev. 1.1, October 2020: Board Operations List updated (*passthrough* parameter added)
Rev. 1.0, December 2017

1 Intended Audience

This document is intended for users of the KETEK VICO-DP (DPP2) hardware who would like to interface to it using the Handel driver library. This document assumes that users of the Handel driver library are familiar with the C programming language.

2 Conventions Used in this Document

- `This style is used to indicate source code.`
- CHECK_ERROR is a placeholder for user-defined error handling.
  (See the sample code for an example of how to implement such error handling in section 10)

3 Preliminary Details

3.1 Header Files

Before introducing the details of programming with the Handel API, it is important to discuss the relevant header files and other external details related to Handel. The following headers must be included:

- *handel.h* defines and imports Handel routines.
- *md_generic.h* contains the constants for e.g. setting logging levels.
- *handel_errors.h* error status constants.
- *handel_constants.h* contains the constants used for Handel calls.

3.2 Error Codes

A good programming practice with Handel is to compare the returned status value with XIA_SUCCESS and then deal with any returned errors before proceeding. All Handel routines (except for some of the debugging routines) return an integer value indicating success or failure.

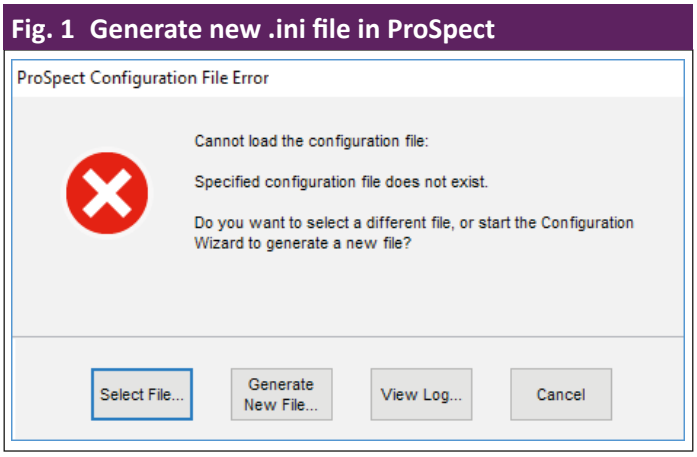
3.3 .INI Files

Besides the DLL package a hardware configuration file (.ini) is needed for the operation.

The simplest way to generate a proper .ini file is using the provided ProSpect software. Please connect the KETEK DPP2 system to a PC using the desired interface, either USB or RS232. Start the ProSpect software.

Upon the first start an error window will be displayed as the .ini file is missing. The user can generate this file using the option 'Generate New File' as shown below:

Fig. 1 Generate new .ini file in ProSpect



The .ini file is located in the following folder and can be copied into the user's project folder:
C:\Users\<CURRENT_USER>\AppData\Roaming\XIA LLC\ProSpect\KETEK DPP2_usb2.ini

In the next sections the function calls of the Handel DLL package will be described.

4 Initializing Handel

The first step in any program that uses Handel is to initialize the software library. Handel provides two routines to achieve this goal: *xiaInit()* and *xiaInitHandel()*. In fact, *xiaInit()* is nothing more then a wrapper around the following two functions: *xiaInitHandel()* and *xiaLoadSystem()*. The *xiaInit()* is the recommended function for the initialization of the system.

```
int status;

status = xiaInit("KETEK DPP2_usb2.ini");
CHECK_ERROR(status);
```

5 Starting the System

Once the initialization task has been completed, the next step is to "start the system." This process performs several operations including validating the hardware information supplied in the initialization file and verifying the specified communication interface (USB or RS-232). Calling *xiaStartSystem()* is straightforward:

```
status = xiaStartSystem();
CHECK_ERROR(status);
```

Once *xiaStartSystem()* has been called successfully, the system is ready to perform the standard DAQ operations such as starting a run, stopping a run, reading out the MCA and saving parameter information.

6 Configuring the DPP2 for Data Acquisition

The DPP2 stores all of its firmware and operating parameters on-board in non-volatile memory. Further the DPP2 is able to store 24 different peaking time configurations. For each of those additional measurement parameters (e.g. gap time) can be set and saved to a PARameter SET (**PARSET**) in memory.

In addition to the PARSETs, there are two other sets: GENeral SETs (**GENSET**) and GLOBal SETs (**GLOBSET**). The DPP2 contains a total of 5 GENSETs. The parameters in the GENSETs are gain and spectrum related. There is a single GLOBSET which contain values specific to the detector preamplifier, debugging and run control.

A key component of configuring the DPP2 is choosing values for the various PARSETs and GENSETs, and saving these values to non-volatile memory. Not all of the PARSET/GENSET parameters map directly to acquisition values in Handel. The fact is you only need to modify a small subset of the total number of available parameters in order to adjust the DPP2 for your system, if necessary.

NOTE
Every KETEK system with DPP2 is preconfigured. Thus adjusting of the PARSET/GENSET parameters is not mandatory.

In general your application should only need to swap between the different PARSET/GENSET entries. Furthermore, on power-up, the DPP2 remembers which PARSET/GENSET was used last and loads it into memory so there is no need to track PARSET/GENSET in your application.

In case some of the parameters needs to be readjusted proceed like in the following example. In this example, we want to configure the DPP2 with the following settings (please also refer to the VICO-DP manual for more information on those parameters):

- PARSET number: 0
- GENSET number: 0
- Medium bin width granularity
- 4k spectrum
- Trigger threshold: 25
- Base gain: 5.700

Now the parameters will be set accordingly and saved into the DPP2 memory. Please note that after setting an acquisition value (or values) that you "apply" the new value with a call to *xiaBoardOperation(),* as shown below:

```
double setGENSET = 0;
double setPARSET = 0;
double binWidth = 1.0;
double nMCA = 4096.0;
double threshold = 25.0;
double gain = 5.700;


//Select PARSET and GENSET you want to modify
status = xiaSetAcquisitionValues(0, "parset", (void *)&setPARSET);
CHECK_ERROR(status);
status = xiaSetAcquisitionValues(0, "genset", (void *)&setGENSET);
CHECK_ERROR(status);


//Modify some acquisition values
status = xiaSetAcquisitionValues(0, "mca_bin_width", (void *)&binWidth);
CHECK_ERROR(status);
status = xiaSetAcquisitionValues(0, "number_mca_channels", (void *)&nMCA);
CHECK_ERROR(status);
status = xiaSetAcquisitionValues(0, "trigger_threshold", (void *)&threshold);
CHECK_ERROR(status);
status = xiaSetAcquisitionValues(0, "gain", (void *)&gain);
CHECK_ERROR(status);


//Apply changes to parameters
status = xiaBoardOperation(0, "apply", (void *)&ignored);


//Now these parameters will be saved to the GENSET = 0 and PARSET = 0
double currentGENSET;
double currentPARSET;
```

```
/* Check GENSET and PARSET which are currently selected. Should be no. 0 as
those have been selected in the previous step */
status = xiaGetAcquisitionValues(0, "genset", &currentGENSET);
CHECK_ERROR(status);
printf("Current GENSET = %lf\n", currentGENSET);

status = xiaGetAcquisitionValues(0, "parset", &currentPARSET);
CHECK_ERROR(status);
printf("Current PARSET= %lf\n", currentPARSET);


/* Save to GENSET/PARSET at the same position as the previously selected
GENSET/PARSET (in this example position 0)*/
unsigned short saveGENSET = (unsigned short) setGENSET;
unsigned short savePARSET = (unsigned short) setPARSET;


/* To be sure that only the previously selected GENSET/PARSET will be
overwritten */
if ( currentPARSET == setPARSET && currentGENSET == setGENSET )
   {
      status = xiaBoardOperation(0, "save_genset", &saveGENSET);
      CHECK_ERROR(status);
      status = xiaBoardOperation(0, "save_parset", &savePARSET);
      CHECK_ERROR(status);
   }
```

IMPORTANT

As described in the above example the procedure of changing the parameters should be done in three steps:

1. Select the GENSET/PARSET you want to modify
2. Modify the GENSET/PARSET parameters
3. Save the GENSET/PARSET parameters at the **same position** as selected in 1.

Otherwise there is a risk that the parameters from one GENSET/PARSET could be overwritten with those from another GENSET/PARSET

7 Selecting a Peaking Time

In the previous example we simply saved our configuration to PARSET 0 (as well as GENSET 0) without any explanation of what PARSET 0 corresponds to. Each of the PARSETs are equal to a different peaking time. To discover what peaking times are available (see also Tab. 4 on page 16), use the following code:

```
int i;
double *peakingTimes = NULL;

//Print out the current peaking times
status = xiaBoardOperation(0, "get_number_pt_per_fippi", &numberPeakingTimes);
CHECK_ERROR(status);
```

```
peakingTimes = (double *)malloc(numberPeakingTimes * sizeof(double));
CHECK_MEM(peakingTimes);

status = xiaBoardOperation(0, "get_current_peaking_times", peakingTimes);
CHECK_ERROR(status);

for (i = 0; i < MAX_PTS; i++)
{
   printf("peaking time %d = %lf\n", i, peakingTimes[i]);
}
```

where i corresponds to the PARSET responsible for that peaking time. To switch to peaking time/PARSET i, simply do:

```
double parset = (double)i;
status = xiaSetAcquisitionValues(0, "parset", (void *)&parset);
CHECK_ERROR(status);
```

8 Controlling the DPP

Once the DPP2 is properly configured, it is ready to begin data acquisition tasks. This section discusses starting a run, stopping a run, reading out the MCA spectrum and, lastly, configuring the DPP for preset length runs.

8.1 Starting and Stopping a Run

The Handel interface to starting and stopping the run are two simple routines: `xiaStartRun()` and `xiaStopRun()`. Both routines require a detector channel number (like `xiaSetAcquisitionValues()`) as their first argument, while `xiaStartRun()` also requires an unsigned short that determines if the MCA is to be cleared when the run is started. To start a run with the MCA cleared, run for 5 seconds and then stop the run, the following code may be used:

```
int status;
unsigned short clearMCA = 0;
status = xiaStartRun(0, clearMCA);
CHECK_ERROR(status);
/* Windows API call. You can use whatever routine is available on your plat-
form to wait. */
Sleep(5000);
status = xiaStopRun(0);
CHECK_ERROR(status);
```

NOTE

For historical reasons, Handel and the DPP2 RS-232 command for starting a run have a different idea of how to interpret the clear MCA value. The RS-232 command uses 0 to mean “resume run” and 1 to mean “clear the MCA”. **Handel uses 0 to mean “clear the MCA” and 1 to mean “resume run”**.

8.2 Reading out the MCA Spectrum

Assuming that we are still running the PARSET 0 configuration from the previous section, we know that our MCA spectrum length is 4096. In case you want to reduce the number of bytes that have to be sent to the host PC, you can request either 1, 2, or 3 bytes per bin. The default setting in Handel is 3 bytes per bin, which is the same as the raw value stored in the DSP’s memory. If you want to use 3 bytes per bin then you do not have to change anything. If you

want to only return a single byte per bin, then use the following code:

```
double bytesPerBin = 1.0;
status = xiaSetAcquisitionValues(0, "bytes_per_bin", (void *)&bytesPerBin);
CHECK_ERROR(status);
```

IMPORTANT

If the number of counts in a bin exceeds the requested bytes per bin, the DPP2 does not return an error. For example, if there are 0xADCDEF counts in a bin and you read out the MCA spectrum with bytes per bin set to 1, that bin will return 0xEF!

With the bytes per bin configured correctly, we are now ready to read out the MCA spectrum.

```
unsigned long mca[4096];
status = xiaGetRunData(0, "mca", (void *)mca);
CHECK_ERROR(status);
```

8.3 Preset Length Runs

The DPP2 supports preset runs, which allow you to specify that a run stop automatically after a certain amount of time has passed or other criteria have been met. The four types of preset runs are fixed livetime, fixed realtime, fixed output counts and fixed input counts. A fixed livetime run will execute until the specified amount of livetime has elapsed. Similarly, a fixed realtime run will execute until the specified amount of realtime has elapsed. The fixed input and output count runs continue until the requested number of counts have occurred. The following is an example of setting a fixed realtime run for 5 seconds, including how to poll the device waiting for the run to complete:

```
int status;
double realtime = 5.0;
double realtimeType = XIA_PRESET_FIXED_REALTIME;
double presetData[2];
unsigned short clearMCA = 0;
unsigned short runActive;

presetData[0] = realtimeType;
presetData[1] = realtime;

status = xiaBoardOperation(0, "set_preset", (void *)presetData);
CHECK_ERROR(status);
status = xiaStartRun(0, clearMCA);
CHECK_ERROR(status);

do {
      Sleep(1);
      status = xiaGetRunData(0, "run_active", (void *)&runActive);
      CHECK_ERROR(status);
   } while (runActive);

/* Once the run is no longer active, we know that the preset run has completed
and that it is safe to stop the run. */
```

```
status = xiaStopRun(0);
CHECK_ERROR(status);
/* Read out the spectrum, etc. */
```

NOTE

Please note that the livetime value corresponds to the fast filter and not the energy filter. Further the livetime preset run would stop as soon as the livetime of the fast filter has reached the selected value.
It is not possible to set a livetime preset of the energy filter.

9 Cleaning Up

Before exiting Handel, call xiaExit() to safely shutdown the communication channel:

```
int status;
status = xiaExit();
CHECK_ERROR(status);
```

10 Example code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "handel.h"
#include "handel_errors.h"
#include "md_generic.h"
#include "handel_constants.h"
/* For Sleep */
#include <windows.h>

#define MAX_PTS 24
static void CHECK_ERROR(int status);
static void CHECK_MEM(void *mem);
static void print_array(int n, unsigned long *vec);
static void save_spectrum_to_textfile(int n, unsigned long *array);
static void log_message(char *message, void *message_value);

int main(int argc, char *argv[])
{
   int status;
   int i;
   unsigned short ignored = 0;

      /* Acquisition Values */
   double nMCA     = 4096.0;
   double thresh   = 25.0;
   double gain     = 5.700;
   double currentGENSET;
   double currentPARSET;

   double setGENSET = 0;
   double setPARSET = 3;


   /* After setting the GENSET and PARSET, it is recommended to save the co-
   niguration with the same value */
   unsigned short saveGENSET = (unsigned short) setGENSET;
   unsigned short savePARSET = (unsigned short) setPARSET;

   unsigned short numberPeakingTimes = 0;

   unsigned long mcaLen = 0;

   unsigned long *mca = NULL;

   double *peakingTimes = NULL;
```

```
/* Fixed Realtime Variables */
double realtime = 10;          //in seconds
double realtimeType = XIA_PRESET_FIXED_REAL;
double presetData[2];
unsigned short clearMCA = 0;
unsigned short runActive;

presetData[0] = realtimeType;
presetData[1] = realtime;

/* RunData */
double runtime = 0;
unsigned long events_in_run = 0;
double all_statistics [6];

/* Setup logging here */
printf("Configuring the Handel log file.\n");
xiaSetLogLevel(MD_WARNING);
xiaSetLogOutput("handel.log");

/* Load ini file */
printf("Loading the .ini file.\n");
status = xiaInit("DPP2.ini");
CHECK_ERROR(status);

xiaSetIOPriority(MD_IO_PRI_HIGH);

status = xiaStartSystem();
CHECK_ERROR(status);

/* Set PARSET and GENSET */
status = xiaSetAcquisitionValues(0, "parset", (void *)&setPARSET);
CHECK_ERROR(status);

status = xiaSetAcquisitionValues(0, "genset", (void *)&setGENSET);
CHECK_ERROR(status);

/* Modify some acquisition values */
status = xiaSetAcquisitionValues(0, "number_mca_channels", &nMCA);
CHECK_ERROR(status);
status = xiaSetAcquisitionValues(0, "trigger_threshold", &thresh);
CHECK_ERROR(status);
status = xiaSetAcquisitionValues(0, "gain", &gain);
CHECK_ERROR(status);

/* Apply changes to parameters */
status = xiaBoardOperation(0, "apply", (void *)&ignored);

/* Save the settings to the current GENSET and PARSET */
status = xiaGetAcquisitionValues(0, "genset", &currentGENSET);
CHECK_ERROR(status);
```

```
/* Check if current settings are equal to the chosen ones */
printf("Current GENSET = %lf\n", currentGENSET);
status = xiaGetAcquisitionValues(0, "parset", &currentPARSET);
CHECK_ERROR(status);
printf("Current PARSET= %lf\n", currentPARSET);

if ( currentPARSET == setPARSET && currentGENSET == setGENSET )
{
   status = xiaBoardOperation(0, "save_genset", &saveGENSET);
   CHECK_ERROR(status);
   status = xiaBoardOperation(0, "save_parset", &savePARSET);
   CHECK_ERROR(status);
}

/* Print out the current peaking times */
status = xiaBoardOperation(0, "get_number_pt_per_fippi", &numberPeaking-
Times);
CHECK_ERROR(status);
peakingTimes = (double *)malloc(numberPeakingTimes * sizeof(double));
CHECK_MEM(peakingTimes);
status = xiaBoardOperation(0, "get_current_peaking_times", peakingTimes);
CHECK_ERROR(status);

for (i = 0; i < MAX_PTS; i++)
{
   printf("peaking time %d = %lf\n", i, peakingTimes[i]);
}

/* Fixed Realtime */
status = xiaBoardOperation(0, "set_preset", (void *)presetData);
CHECK_ERROR(status);
status = xiaStartRun(0, clearMCA);
printf("Started run. Sleeping...\n");
CHECK_ERROR(status);
do
{
   Sleep(1);
   status = xiaGetRunData(0, "run_active", (void *)&runActive);
   CHECK_ERROR(status);
} while (runActive);

/* Once the run is no longer active, we know that the preset run has com-
pleted and that it is safe to stop the run. */
status = xiaStopRun(0);
CHECK_ERROR(status);

/* Prepare to read out MCA spectrum */
status = xiaGetRunData(0, "mca_length", &mcaLen);
CHECK_ERROR(status);
```

```
    if (mcaLen > 0)
    {
       printf("Got run data\n");
    }

    /* If you don't want to dynamically allocate memory here,
       then be sure to declare mca as an array of length 8192,
       since that is the maximum length of the spectrum. */
    mca = (unsigned long *)malloc(mcaLen * sizeof(unsigned long));
    CHECK_MEM(mca);
    status = xiaGetRunData(0, "mca", (void *)mca);
    CHECK_ERROR(status);

    /* Additional RunData */
    status = xiaGetRunData(0, "runtime", (void *)&runtime);
    CHECK_ERROR(status);
    printf("Runtime = %lf\n", runtime);
    log_message("Runtime = ",(void *) &runtime);

    status = xiaGetRunData(0, "events_in_run", (void *)&events_in_run);
    CHECK_ERROR(status);
    printf("Events in Run = %lu\n", events_in_run);
    status = xiaGetRunData(0, "all_statistics", (void *)all_statistics);

    for(i=0; i<6; i++)
    {
       printf("StatisticArray[%d] = %lf\n", i, all_statistics[i]);
       log_message("StatisticArray RunData = ", (void *) &all_statistics[i]);
    }

    /* Print the Spectrum and save it to a textfile */
    print_array(mcaLen, mca);
    save_spectrum_to_textfile(mcaLen, mca);

    xiaSetIOPriority(MD_IO_PRI_NORMAL);
    free(mca);
    free(peakingTimes);

    status = xiaExit();
    CHECK_ERROR(status);
    xiaCloseLog();
    return 0;
}
```

```
static void CHECK_ERROR(int status)
{
   /* XIA_SUCCESS is defined in handel_errors.h */
   if (status != XIA_SUCCESS) {
      fprintf(stderr, "Error encountered! Status = %d\n", status);
      exit(status);
   }
}

static void CHECK_MEM(void *mem)
{
   if (mem == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
}

static void print_array(int n, unsigned long *lu_array)
{
   int i;
   for (i = 0; i < n; i++) {
      printf("%lu\n", lu_array[i]);
   }
   return;
}

static void log_message(char *message, void *message_value)
{
   FILE *fptr;
   fptr = fopen("spectrum.txt", "a");
   fprintf(fptr, "%s", message);
   fprintf(fptr, "%lf\n", *(double *)message_value);
   fclose(fptr);
   return;
}

static void save_spectrum_to_textfile(int n, unsigned long *lu_array)
{
   FILE *fptr;
   int i = 0;
   fptr=fopen("spectrum.txt","a");
   for (i=0; i < n; i++)
   {
      fprintf(fptr, "%lu", lu_array[i]);
      fprintf(fptr,"\n");
   }
   printf("Successfully saved to file 'spectrum.txt'\n");
   fclose(fptr);
   return;
}
```

11 Acquisition Values List

Below is a list of all of the supported acquisition values for the DPP2. All of the acquisition values are of type *double*.

- **parset**: The current PARSET.
- **genset**: The current GENSET.
- **energy_gap_time**: The gap time of the energy filter, specified in µs.
- **trigger_peak_time**: The peaking time of the trigger filter, specified in µs.
- **trigger_gap_time**: The gap time of the trigger filter, specified in µs.
- **trigger_threshold**: Trigger filter threshold in arbitrary units.
- **baseline_threshold**: Baseline filter threshold in arbitrary units.
- **number_mca_channels**: The number of bins in the MCA spectrum, defined in bins.
- **mca_bin_width**: Width of an individual bin in the MCA, using the "custom" width specified in the RS-232 Command Reference.
- **bytes_per_bin**: The number of bytes returned per bin when reading out the MCA spectrum. Can be either 1, 2 or 3 bytes.
- **adc_trace_wait**: When acquiring an ADC trace for readout, the amount of time to wait between ADC samples, specified in µs.
- **gain**: The base gain in arbitrary units.
- **preamp_value**: The reset interval for reset-type preamplifiers detectors. The reset interval is specified in µs.
- **gain_trim**: Adjusts the base gain per PARSET, specified in arbitrary units.

For experts only:

- **baseline_length**: The number of samples averaged together for the baseline filter.
- **energy_threshold**: Energy filter threshold in arbitrary units.
- **peak_interval**: The value of PEAKINT, specified in µs.
- **peak_interval_offset**: The peak interval specified as an offset from the peaking time and gap time, specified in µs. Effectively sets PEAKINT = SLOWLEN + SLOWGAP + peak_interval_offset.
- **peak_sample**: The value of PEAKSAM, specified in µs.
- **peak_sample_offset**: Energy filter sampling time measured backward from the peaking time and gap time, specified in µs. Effectively sets PEAKSAM = SLOWLEN + SLOWGAP - peak_sample_offset.
- **max_width**: The value of MAXWIDTH, specified in µs.

12 Run Data List

These are the different types of run data that can be read using xiaGetRunData(). The C type of the run data is printed in *italics* after the name.

- **mca_length** (*unsigned long*): The number of bins in the MCA spectrum.
- **mca** (*unsigned long* *): The MCA spectrum.
- **livetime** (*double*): The calculated fast filter livetime, reported in seconds.
- **runtime** (*double*): The runtime, reported in seconds.
- **input_count_rate** (*double*): The measured input count rate, reported as counts / second.
- **output_count_rate** (*double*): The output count rate, reported as counts / second.

- **events_in_run** (*unsigned long*): The total number of events in the current run.
- **triggers** (*unsigned long*): The number of input triggers in the current run.
- **baseline_length** (*unsigned long*): The current size of the baseline histogram buffer.
- **baseline** (*unsigned long* *): The baseline histogram.
- **run_active** (*unsigned short*): The current state of the processor: a 1 means that a run is currently active, a 0 means that no run is active.
- **all_statistics** (*double[6]*) Returns an array of the six statistics available for the DPP2: livetime, runtime, triggers, events in run, input count rate and output count rate.

13 Board Operations List

The allowed board operations for the DPP2, accessed via. xiaBoardOperation(). If no C type is specified after the operation, a dummy, non-NULL value must be passed into the value argument.

- **get_serial_number** (*char[16]*): Get the DPP2 board's serial number.
- **get_current_peaking_times** (*double[5]*): Get the current peaking times for the selected FiPPI, where the peaking time at index i in the returned list corresponds to PARSET i for the selected FiPPI.
- **get_temperature** (*double*): Returns the current temperature of the board, accurate to 1/16th of a degree of Celsius.
- **apply**: Applies the current DSP parameter settings to the hardware. This should be done after modifying any acquisition values.
- **save_parset** (*unsigned short*): Saves the current DSP parameter settings to the specified PARSET.
- **save_genset** (*unsigned short*): Saves the current DSP parameter settings to the specified GENSET.
- **set_preset** (*double[2]*): Configure a preset run by passing in the preset type and value. The allowed types, defined in handel_constants.h are XIA_PRESET_FIXED_REALTIME, XIA_PRESET_FIXED_LIVETIME, XIA_PRESET_FIXED_TRIGGERS, XIA_PRESET_FIXED_EVENTS. The values are defined as time in seconds, for the time based runs and counts for the other types.
- **get_board_info** (*unsigned char[26]*): Returns the array of board information listed in command 0x49 of the RS-232 Command Reference.
- **get_hardware_status** (*unsigned char[5]*): Returns the array of status information listed in command 0x4B of the RS-232 Command Reference.
- **passthrough** (*void*[4]*): Tunnels data to and from an additional microcontroller, which enables the so-called Advanced Functions (available for selected KETEK products only). For more information refer to the dedicated product manuals.

14 PARSET/Acquisition Value Mapping

Shows the mapping from PARSET parameter to Handel acquisition value. Note that not all PARSET parameters can or need to be changed from Handel and therefore do not have a corresponding acquisition value. Some PARSET values are sensitive to the currently loaded GENSET and store one value for each GENSET. As a Handel user, you only need to worry about setting the value; the DPP2 manages the per-GENSET values automatically.

Tab. 1 PARSET/ Acquisition Value Mapping

PARSET name	Acquisition value name	Description
SLOWGAP	energy_gap_time	The size of the "slow" or energy filter gap.
PEAKINT	peak_interval	FiPPI setting
FASTLEN	trigger_peak_time	The length of the "fast" or trigger filter.
FASTGAP	trigger_gap_time	The size of the "fast" or trigger filter gap.
THRESHOLD	trigger_threshold	The threshold for the "fast" or trigger filter.
MAXWIDTH	max_width	FiPPI setting (experts only)
SLOWTHRESH	energy_threshold	The threshold for the "slow" or energy filter.
BASETHRESH	baseline_threshold	The threshold for the baseline filter.
GAINTWEAK0-4	gain_trim	Adjusts the gain for this specific combination of PARSET and GENSET.
THRESHOLD0-4	trigger_threshold	The trigger threshold for this specific combination of PARSET and GENSET.
BASETHRESH0-4	baseline_threshold	The baseline threshold for this specific combination of PARSET and GENSET.

15 GENSET/Acquisition Value Mapping

Tab. 2 GENSET/ Acquisition Value Mapping

GENSET name	Acquisition value name	Description
MCALEN	number_mca_channels	The number of channels in the MCA spectrum.
BINGRANULAR	mca_bin_width	MCA bin width granularity.
GAINBASE	gain	The base gain setting.

16 GLOBSET/Acquisition Value Mapping

Tab. 3 GLOBESET/ Acquisition Value Mapping

GLOBESET name	Acquisition value name	Description
RESETINT	preamp_value	Reset interval for reset-type preamplifiers.

17 Available Peaking Times

Tab. 4 Available Peaking Times

PARSET no.	Peaking time [µs]
0	0.1
1	0.15
2	0.2
3	0.25
4	0.3
5	0.4
6	0.5
7	0.6
8	0.8
9	1
10	1.2
11	1.6
12	2
13	2.4
14	3.2
15	4
16	4.8
17	6.4
18	8
19	9.6
20	12.8
21	16
22	19.2
23	24

18 Contact

If you have any questions regarding this or any of our products please do not hesitate to contact us via e-mail or phone.

Address of our KETEK Headquarters Sales Office:

KETEK GmbH
Hofer Str. 3
81737 Munich
Germany

E-mail info@ketek.net
Phone +49 (0) 89 673467 70
Fax +49 (0) 89 673467 77
Homepage www.ketek.net