

ADQ Disk Streaming

User Guide

Author(s): Teledyne SP Devices
Document ID: 22-2761
Classification: Public
Revision: A
Print date: 2022-06-10

Contents

1	Introduction	3
1.1	Definitions and Abbreviations	3
2	Overview	4
2.1	ADQAPI	4
2.2	ADNVDS	4
2.3	Installation	4
2.3.1	Windows	5
2.3.2	Linux	5
2.3.3	Firmware	5
2.4	Data Flow	6
2.5	Hardware support	6
3	Data acquisition	7
3.1	Monitoring channels	7
3.2	Disk streaming data and metadata	8
3.3	Disk sequencing	8
3.4	Disk interleaving	9
3.5	Overflow conditions	9
3.6	Parameter differences relative to standard firmware	9
3.6.1	Data acquisition parameters	9
3.6.2	Data transfer parameters	10
3.6.3	Data readout parameters	10
4	Disk streaming example	10
5	Disk reading	12
5.1	Disk read example	12
5.2	Metadata parsing	12
A	ADQAPI Reference	15
A.1	Structures	16
A.2	FWDSU Functions	26
A.3	Metadata Parsing Functions	29

Document History

Revision	Date	Section	Description	Author
A	2022-06-13	-	Initial revision	TSPD

1 Introduction

ADQ Disk Streaming is a high-speed data recording solution which enables writing data directly from a digitizer to NVMe drives over PCI Express without high load on host CPU or RAM. ADQ Disk Streaming is available for both Windows and Linux operating systems. For supported versions see [1]. With ADQ Disk Streaming, data from a digitizer can be continuously written to multiple disks and multiple digitizers can simultaneously write to disks in the same system as long as the disks are not shared.

! Important

This document is only valid for the following digitizer models:

- ADQ7 (requires FWDSU firmware package)

1.1 Definitions and Abbreviations

Table 1 lists the definitions and abbreviations used in this document.

Table 1: Definitions and abbreviations used in this document.

Item	Description
NVMe	Non-Volatile Memory Express
ADNVDS	ADQ Non-Volatile Data Storage
FWDSU	Disk storage firmware package for digitizers
BAR	Base Address Register

2 Overview

To use ADQ Disk Streaming on a system the disks and digitizers must be connected to a common PCI Express bus. The digitizers need to run the FWDSU firmware package with a valid license installed. A complete specification for the FWDSU firmware package can be found in the FWDSU datasheet [2] Disk streaming is intended to be integrated in existing applications by using documented API calls. Examples are provided as a starting point for development. As illustrated in Fig. 1 the digitizer is controlled via the ADQAPI library and the disk by the ADNVDs library respectively.

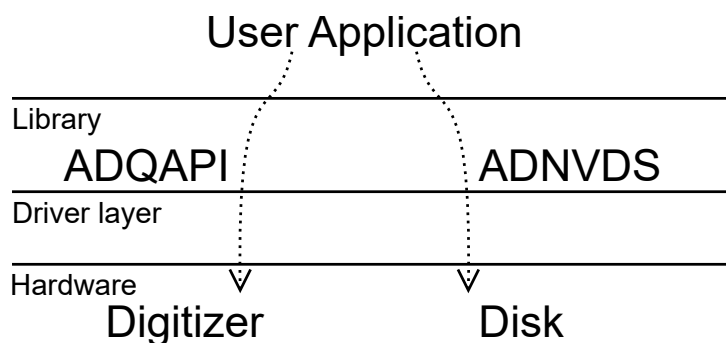


Figure 1: ADQ Disk Streaming user applications use two different libraries.

2.1 ADQAPI

ADQAPI is the standard library for controlling the digitizers. For details see *ADQAPI Reference Guide* [3]

2.2 ADNVDs

ADNVDS is the software library that enables disks to act as a storage endpoints for ADQ digitizers. It consists of the following parts:

- Library initialization - enabling usage of the library.
- Device management - retrieving device parameters and binding disks to ADNVDs.
- Data storage - handles the writing of data from digitizer to the disk.
- Data readout - copying data from the disk to the host RAM.

A disk that is used as storage endpoint has no file system. To retrieve data from disks ADNVDs support reads of data both as stored, and through a *dataset* abstraction. For a detailed description of the ADNVDs API, please see the ADNVDs library reference manual [4].

2.3 Installation

All the necessary software and firmware components for using disk streaming are provided in the FWDSU release archive.

2.3.1 Windows

For Microsoft Windows the ADQAPI SDK is installed by running

```
TSPD-SDK-installer_rXXXXX.exe
```

and following the instructions. The XXXXX part of the file name is the version number. The archive contains installation files, example code and documentation.

The ADNVDs library and NVMe driver is installed by running

```
ADQNVMeInstaller.exe
```

The disks that are intended to be used for streaming must have their driver updated manually by providing the serial numbers to NVMeBind.exe found in C:\Program Files\SP Devices ADQNVMe\driver. Note that this command must be executed as administrator and that it is only possible to install for device listed in Table 2.

2.3.2 Linux

The ADQAPI SDK is supported for several Linux distributions and versions. The complete list can be found in the document listing operating system support [1]. The installation files are included in

```
adqguitools_linux_rXXXXX.tar.gz
```

where XXXXX is the version number. The archive contains installation files, example code and documentation. The README file, located in the root directory of the archive, describes the installation procedure in detail for the different distributions.

The installation files for the ADNVDs library are included in

```
dsu7_sdk_linux_rXXXXX.tar.gz
```

To allow ADNVDs access to the drives, a shell script named

```
setup_adnvd.sh
```

is also provided. This shell script must be run once after every boot of the system, before any disk streaming acquisition takes place. The default behavior of the shell script is to look for a file named

```
$(hostname)_SPD*.txt
```

in the same directory as the script, containing a space-separated list of disk serial numbers to associate with the ADNVDs driver. This file must be created by the user and filled with the correct serial numbers for the system before the shell script can be executed.

2.3.3 Firmware

FWDSU firmware images can be found in the firmware folder of the release archive. Please see the ADQUpdater User Guide [5] for instructions on how to upload the firmware images to the digitizer. The

FWDSU firmware requires a valid FWDSU firmware license to be present on the digitizer.

2.4 Data Flow

For a complete definition of how NVMe disks operate, see the NVMe specification [6]. In short, the concept is that instructions are sent to the disk which define how data should be transferred. The actual data transfer is then executed by the disk itself. Normally these operations are data reads by the disk from RAM to be stored on the disk, or data writes from the disk to RAM. When using ADNVDs the disk is instead instructed to fetch data directly from the digitizer. This happens without placing data in host RAM first. Fig. 2 shows how the user application drives the data transfer by adding commands in the command queue. The status of completed commands are read from a response queue.

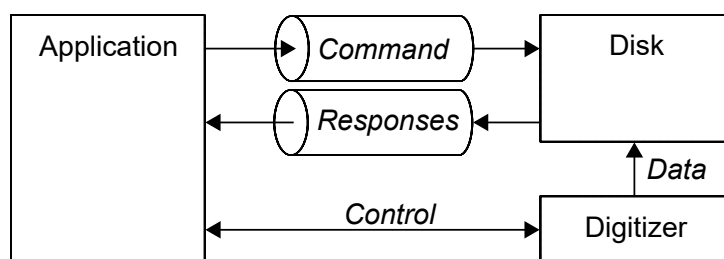


Figure 2: Data is read directly from the digitizer to the disk when using ADNVDs.

⚠ Warning

In the event of a crash of the user application the disks will continue their operation and will fetch commands from the now invalid RAM, and an accidental overwrite of previously recorded data may occur. Therefore, disks used by ADNVDs should not be seen as permanent storage and captured data that is critical should be read out to permanent storage before recording new data. Terminating the data recording in a non-graceful way risks causing a crash.

2.5 Hardware support

Only disks recommended by Teledyne SP Devices are guaranteed to work with ADNVDs. All disks used by ADNVDs in a system should be of the same model and size for reliable operation. See Table 2 for a list of recommended disks. It is recommended to contact us for guidance. Note that write speed may be limited when writing to less than 8 drives simultaneously.

❗ Important

When using Windows it is not possible to use other devices than listed in Table 2. For Linux it is possible but not recommended.

In PCI Express systems, there are many third party alternatives for disk carrier boards. See Table 3 for a list of carrier boards that are verified with ADNVDs. It is recommended to contact us for guidance.

Table 2: Supported NVMe drives.

Manufacturer	Product	Comment
Samsung	980 PRO 1 TB	
Samsung	980 PRO 2 TB	
Samsung	970 PRO 1 TB	
Sabrent	Rocket 4 TB	
Sabrent	Rocket 4+ 4 TB	Heat sink is required

Table 3: Supported PCIe carrier boards.

Manufacturer	Product	Comment
Asus	Hyper M2 x16	Motherboard x4x4x4x4 bifurcation support required
Asus	Hyper M2 x16 gen4	Motherboard x4x4x4x4 bifurcation support required
Supermicro	AOC-SHG3-4M2P-O	Limited to 5GB/s

In PXI Express systems, there are fewer disk carrier alternatives. For this reason, Teledyne SP Devices provides the ADQDSU carrier board, complete with disks. See the ADQDSU datasheet [7] for more information.

3 Data acquisition

Much of the initial setup of a disk streaming capable digitizer works in the same way as with the standard FWDAQ firmware. The ADQGen3 Streaming User Guide [8] should be reviewed first. The information presented here is intended to describe differences and new features in the configuration of the FWDSU firmware package relative to the standard firmware, not to give a complete description. Note also that the `gen3_streaming` code example is supported by the FWDSU firmware and can be used as a first step to test acquisition settings without needing to stream to disk.

3.1 Monitoring channels

The FWDSU firmware package adds duplicates of the standard channels to the firmware data path. The duplicate channels, referred to as monitoring channels, will have the sample skip or decimation setting as the standard channel, but can have their trigger source and acquisition parameters such as record length configured independently from the standard channels. The typical use case for the monitoring channels is to periodically acquire records to the host RAM via regular streaming during a disk streaming acquisition, and continuously analyze it to ensure that the measurement is working as intended.

Typically, the data rate for the standard channels which are streamed to the disks is high and cannot be monitored continuously by the host. The extra monitoring channels provide a way to reduce the data

rate, by for example using a shorter record length compared to the standard channels, or using a trigger source with a lower trigger rate than the one use for the standard channels.

Important

There is a maximum limit for the *combined* record length for all the active monitoring channels of 32768 samples. One active monitoring channel can have up to 32768 samples of record length, two active monitoring channels can have record lengths up to 16384 samples each, and so on.

To support a different trigger source for the monitoring channels compared to the regular disk streaming channels, the monitoring channels all use the auxiliary trigger, which is set via the ADQAPI command `SetAuxTriggerMode`. Refer to Table 4 and Table 5 for the channel indexes, trigger sources and disk streaming capabilities of the channels in the available FWDSU versions.

Table 4: Channel configuration on ADQ7 with two-channel FWDSU firmware.

Index	Analog input	Trigger source	Supports disk streaming
0	Channel A	Channel A trigger source	X
1	Channel B	Channel B trigger source	X
2	Channel A	Auxiliary trigger	
3	Channel B	Auxiliary trigger	

Table 5: Channel configuration on ADQ7 with one-channel FWDSU firmware.

Index	Analog input	Trigger source	Supports disk streaming
0	Channel X	Channel X trigger source	X
1	Channel X	Auxiliary trigger	

3.2 Disk streaming data and metadata

During a disk streaming acquisition, both data and metadata will be written to the disks. Some examples of metadata stored for each record are the timestamp of the trigger event, the record length, and status information that can help detect if data has been lost due to data rate limitations in the system. The disks therefore need to be partitioned according to the expected ratio between data and metadata for the given acquisition parameters. The partitioning is specified via the `metadata_start_lba` parameter when registering an ADNVDs transfer via `adnvds_wr_register_transfer`.

3.3 Disk sequencing

For long acquisitions which will generate large amounts of data, multiple disks can be put together in sequence for a digitizer channel. In this case, the streaming will fill up one disk, and then move on to the next and continue writing there. The disk switching is seamless and will not cause any gaps

in the acquired data. Disk sequencing is enabled by setting the `num_endpoints` parameter to a value greater than one, and providing a list of disk serial numbers of corresponding length via `ns_handles` when registering an ADNVDs transfer through `adnvd_w_r_register_transfer`.

3.4 Disk interleaving

For acquisitions which will produce a high continuous data rate, the write speed of a single disk may not be enough to keep up. To handle this, multiple disks can be interleaved for each digitizer channel. The streaming of data will then continuously switch between the interleaved disks, writing a burst of data to one disk at a time. Disk interleaving is enabled by sequencing the disks as described in Section 3.3, followed by setting an interleaving factor to a value greater than 1 via the ADNVDs function `adnvd_w_r_set_group_interleave`. Refer to Table 6 for supported disk interleaving factors.

Table 6: Supported disk interleaving factors per channel.

Product	Firmware type	Supported interleaving factors per channel
ADQ7	Two-channel	1, 2, 4
ADQ7	One-channel	1, 2, 4, 8

3.5 Overflow conditions

The digitizer will use its on-board DRAM as a buffer between the data from the digitizer channels, and the streaming of the data to disk. In the event that the disk cannot keep up with the data from the digitizer, the DRAM buffer will fill up, and data will start to be discarded at the DRAM write side. This can be detected by reviewing the record metadata. Both the `DSU7MetadataRaw` structure and the fully parsed `DSU7RecordHeader` contain counters for `LostRecords` and `LostCycles`. The standard overflow detection via the ADQAPI function `GetStreamOverflow` will also signal an overflow when this occurs.

3.6 Parameter differences relative to standard firmware

This section will describe new features in the parameter interface that are specific to disk streaming. Please see the ADQGen3 Streaming User Guide [8] for documentation of the rest of the parameter structures.

3.6.1 Data acquisition parameters

The `dsu_forced_metadata_interval` parameter, when set to a value lower than the record length, will force the firmware to periodically generate record metadata throughout the record. This means that a record will have multiple corresponding record headers stored on the disk. This can be seen in the header by checking bit 0 in the `RecordStatus` field, which will only be set to 1 in the final metadata generated by the record.

In the FWDSU firmware, record metadata is not written to disk until at the end of a record. This means that during, for example, a data acquisition with the record length is set to `ADQ_INFINITE_RECORD_LENGTH`, no metadata will be generated, unless `dsu_forced_metadata_interval` is set.

Setting a fixed metadata interval also allows for more granularity when determining where data was discarded during an overflow condition, as the `LostRecords` and `LostCycles` will cover a lower amount of data.

This can also be useful as a safety precaution in the event of fatal events such as power outages and system crashes. The metadata is written to disk in bursts of length `ADNVDS_COMMAND_SIZE` multiplied with the interleaving factor, which means that the metadata for a large number of records can be lost at a sudden stop.

3.6.2 Data transfer parameters

The `dsu_doorbell_value_mask` and `dsu_operation_size` parameters must be set with values from the ADNVDS library for disk streaming to function correctly. Please see the parameter descriptions for further information.

To enable disk streaming for a given channel, the `dsu_record_enabled` and `dsu_metadata_enabled` parameters must be set. It is not possible to enable disk streaming on the channel indexes corresponding to the monitoring channels.

Any disk streaming enabled channels must have both `dsu_record_enabled_endpoints_mask` and `dsu_metadata_enabled_endpoints_mask` set with values from the ADNVDS function `adnvs_wr_get_ADQ_DSU_setup_params`.

To enable streaming of records to the host RAM, the `record_enabled` and `metadata_enabled` parameters must be set. This is typically only done for the monitoring channels, but can be enabled on any channel if desired.

3.6.3 Data readout parameters

The data readout parameter interface is only used for readout of monitoring channel data to the host system. For the streaming of data to disk, the data readout parameters are not relevant. There are no changes to the data readout parameters relative to the standard firmware.

4 Disk streaming example

The example source code for setting up a disk streaming acquisition can be found in the SDK installation directory, under

```
<Path to installation directory>/examples/disk_streaming
```

Note

Before compiling and executing the example, the `settings.h` file should be reviewed and updated. At a minimum, the list of disk serial numbers must be updated to match the disks in present in the system.

The outline of the program flow in the `disk_streaming` example is as follows:

1. Initialize ADQAPI and connect to the digitizer
2. Configure data acquisition, readout and transfer parameters according to the given use case. By default, the example will configure an infinite number of records, triggered by the periodic event source.
3. Initialize ADNVDs and attach the disks to the ADNVDs instance
4. Configure disk streaming for each digitizer channel and corresponding disk group, including
 - (a) Partition the disks into data and metadata sections according to the ratio of metadata to data that will be generated by the given data acquisition parameters
 - (b) Retrieve the PCI Express BAR address information from the digitizer
 - (c) Set up and start an ADNVDs disk transfer from the BAR address to the disks
 - (d) Set disk streaming specific data transfer parameters for the digitizer
5. Start the data acquisition, and then loop
 - (a) Call the ADNVDs function `adnvd_poll`, which will update the queues of NVMe I/O commands to keep the transfers going
 - (b) Check the ADNVDs transfer status for each digitizer channel to see how much new data has been stored
 - (c) If no new bytes have been stored for the default time-out of two seconds, or if we've reached the data limit from the settings, `FlushDMA()` will be issued to get any remaining data transferred. When a time-out occurs again after flushing, the loop will exit.
 - (d) Receive any new record buffers from the monitoring channels. By default the example discards the records without processing them in any way.
6. Let ADNVDs write the final metadata to the disks with the finish command
7. Stop the data acquisition
8. Stop and unregister all ADNVDs transfers
9. Shut down the ADQAPI and ADNVDs instances, and then exit

Warning

It is important that the `adnvd_poll` function is continuously called at a high rate during the acquisition to keep the disk streaming going. Adding, for example, monitoring channel data analysis to the main loop of the example can reduce the polling rate and cause the disk streaming to terminate. Keep any calculations that take significant time in a separate thread from the main loop.

Important

The `FlushDMA()` function should always be called at the end of a disk streaming acquisition, to ensure that no record metadata is lost.

5 Disk reading

5.1 Disk read example

The example source code for reading data and metadata from a disk can be found in the SDK installation directory, under

```
<Path to installation directory>/examples/disk_streaming/disk_read
```

Note

Before compiling and executing the example, the settings.h file should be reviewed and updated. At a minimum, the list of disk serial numbers must be updated to match the disks in present in the system.

The example is by default limited to only read a small amount of data and only the first few metadata blocks. The program flow is as follows:

1. Initialize ADNVDS and attach the disks to the ADNVDS instance
2. Read the `adnvds_rd_data_set` struct using `adnvds_rd_init`.
3. For each dataset:
 - (a) Print the dataset information found in the struct
 - (b) Read a small number of `DSU7MetadataRaw` headers using `adnvds_rd_get_metadata`.
 - (c) Generate complete `DSU7RecordHeader`s using the method in Section 5.2.
 - (d) Print the information found in the record headers.
 - (e) Read a small amount of sample data and store it to a file
4. Free the datasets
5. Shut down ADNVDS, and exit

5.2 Metadata parsing

The disk streaming acquisition stores metadata for each record on the disk in the form of `DSU7MetadataRaw` headers. It also stores general information about the data acquisition setup at the start of the disk, which can be read out using the ADNVDS function `adnvds_rd_init`. By combining the acquisition information from the ADNVDS dataset and the raw metadata headers, complete `DSU7RecordHeader` metadata can be generated. The process is as follows:

1. Read the `adnvds_rd_data_set` struct using `adnvds_rd_init`.
2. Read the `DSU7MetadataRaw` headers using `adnvds_rd_get_metadata`.
3. Create a parser object using `ADQData_Create()`.
4. Initialize the parser object with `ADQData_InitPacketStream()`, by passing `adnvds_rd_data_set->adqdata_device`.

5. Allocate memory for storing the final `DSU7RecordHeaders`
6. Parse the metadata into `DSU7RecordHeaders` using `ADQData_ParseDiskStreamHeaders()`.

The `DSU7RecordHeader` is relatively similar to the `ADQRecordHeader` generated during regular streaming. There are a few key differences:

1. The `RecordStatus` field has a different bit definition
2. The `RecordLength` is 64-bit instead of 32-bit to handle long acquisitions
3. The `LostRecords` member has been added
4. The `LostCycles` member has been added

References

- [1] Teledyne Signal Processing Devices Sweden AB, *15-1494 Digitizer OS Support*. Technical Specification.
- [2] Teledyne Signal Processing Devices Sweden AB, *21-2578 ADQ7 FWDSU datasheet*. Technical Specification.
- [3] Teledyne Signal Processing Devices Sweden AB, *14-1351 ADQAPI Reference Guide*. Technical Manual.
- [4] Teledyne Signal Processing Devices Sweden AB, *ADNVDS Library Reference Guide*. Technical Manual.
- [5] Teledyne Signal Processing Devices Sweden AB, *18-2059 ADQUpdater User Guide*. Technical Manual.
- [6] NVM Express, Inc., *NVM Express Base Specification, revision 2.0b*, January 2022. Technical Specification.
- [7] Teledyne Signal Processing Devices Sweden AB, *21-2575 ADQDSU datasheet*. Technical Specification.
- [8] Teledyne Signal Processing Devices Sweden AB, *20-2465 ADQGen3 Streaming User Guide*. Technical Manual.

A ADQAPI Reference

This appendix contains the documentation of the structures and functions that are used in disk streaming and that differ relative to the documentation in the ADQGen3 Streaming User Guide [8]. These descriptions are intended to complement the ADQAPI reference guide [3].

Important

All objects described in the following sections are defined in the `ADQAPI.h` header file. Please refrain from redefining constants and structures.

A.1 Structures

ADQDataAcquisitionParameters	16
ADQDataAcquisitionParametersCommon	16
ADQDataAcquisitionParametersChannel	17
ADQDataTransferParameters	18
ADQDataTransferParametersCommon	18
ADQDataTransferParametersChannel	20

```

struct ADQDataAcquisitionParameters {
    enum ADQParameterId          id;
    int32_t                      reserved;
    struct ADQDataAcquisitionParametersCommon common;
    struct ADQDataAcquisitionParametersChannel channel[ADQ_MAX_NOF_CHANNELS];
    uint64_t                     magic;
}
  
```

Description

This struct defines the parameters for the data acquisition process.

Members

`id` (`enum ADQParameterId`)

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`reserved` (`int32_t`)

Reserved

`common` (`struct ADQDataAcquisitionParametersCommon`)

A `ADQDataAcquisitionParametersCommon` struct holding parameters that apply to all channels.

`channel[ADQ_MAX_NOF_CHANNELS]` (`struct ADQDataAcquisitionParametersChannel`)

An array of `ADQDataAcquisitionParametersChannel` structs where each element represents the data acquisition parameters for a channel. The struct at index 0 targets the first channel.

`magic` (`uint64_t`)

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

```

struct ADQDataAcquisitionParametersCommon {
    int64_t reserved;
}
  
```

Description

This struct is a member of `ADQDataAcquisitionParameters` and defines data acquisition parameters that apply to all channels.

Members

reserved ([int64_t](#))

Reserved

```
struct ADQDataAcquisitionParametersChannel {  
    int64_t          horizontal_offset;  
    int64_t          record_length;  
    int64_t          nof_records;  
    int64_t          dsu_forced_metadata_interval;  
    enum ADQEventSource trigger_source;  
    enum ADQEdge      trigger_edge;  
    enum ADQFunction   trigger_blocking_source;  
}
```

Description

This struct is a member of [ADQDataAcquisitionParameters](#) and defines data acquisition parameters for a channel.

Members

horizontal_offset ([int64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

record_length ([int64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

nof_records ([int64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

dsu_forced_metadata_interval ([int64_t](#))

This parameter only applies to channels that are being streamed to disk. For long (or infinite length) records, this parameter can be used to transmit metadata more often than just once per record, by setting it to a value lower than the record length. The value is set in units of samples.

Important

In the current FWDSU release, using different values for `dsu_forced_metadata_interval` per channel is not supported. The value for channel index 0 will be used for all channels.

trigger_source ([enum](#) ADQEventSource)

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

trigger_edge ([enum](#) ADQEdge)

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

trigger_blocking_source ([enum ADQFunction](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

```
struct ADQDataTransferParameters {
    enum ADQParameterId          id;
    int32_t                      reserved;
    struct ADQDataTransferParametersCommon common;
    struct ADQDataTransferParametersChannel channel[ADQ_MAX_NOF_CHANNELS];
    uint64_t                     magic;
}
```

Description

This struct defines the parameters for the data transfer process.

Members

id ([enum ADQParameterId](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

reserved ([int32_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

common ([struct ADQDataTransferParametersCommon](#))

A [ADQDataTransferParametersCommon](#) struct holding data transfer parameters that apply to all channels.

channel[ADQ_MAX_NOF_CHANNELS] ([struct ADQDataTransferParametersChannel](#))

An array of [ADQDataTransferParametersChannel](#) structs where each element represents the data transfer parameters for a channel. The struct at index 0 targets the first channel.

magic ([uint64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

```
struct ADQDataTransferParametersCommon {
    int64_t          record_buffer_packed_size;
    int64_t          metadata_buffer_packed_size;
    enum ADQMarkerMode marker_mode;
    int32_t          write_lock_enabled;
    int32_t          transfer_records_to_host_enabled;
    int32_t          packed_buffers_enabled;
    uint32_t         dsu_doorbell_value_mask;
    int32_t          dsu_operation_size;
}
```

Description

This struct is a member of [ADQDataTransferParameters](#) and defines data transfer parameters that apply

to all channels.

Members

`record_buffer_packed_size (int64_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`metadata_buffer_packed_size (int64_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`marker_mode (enum ADQMarkerMode)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`write_lock_enabled (int32_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`transfer_records_to_host_enabled (int32_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`packed_buffers_enabled (int32_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`dsu_doorbell_value_mask (uint32_t)`

This parameter must be set to `doorbell_wrap_depth - 1`, retrieved via the function `adnvds_wr_get_ADQ_DSU_setup_params`, to let the digitizer know the depth of the NVMe I/O command queue.

`dsu_operation_size (int32_t)`

The size of an NVMe I/O command in bytes. Must be set to the `ADNVDS_COMMAND_SIZE` define from the ADNVDS library.

```
struct ADQDataTransferParametersChannel {
    uint64_t      record_buffer_bus_address;
    uint64_t      metadata_buffer_bus_address;
    uint64_t      marker_buffer_bus_address;
    int64_t       nof_buffers;
    int64_t       record_size;
    int64_t       record_buffer_size;
    int64_t       metadata_buffer_size;
    int64_t       record_buffer_packed_offset;
    int64_t       metadata_buffer_packed_offset;
    volatile void * record_buffer;
    volatile void * metadata_buffer;
    volatile void * marker_buffer;
    int32_t       record_length_infinite_enabled;
    int32_t       record_enabled;
    int32_t       metadata_enabled;
    int32_t       dsu_record_enabled;
    int32_t       dsu_metadata_enabled;
    uint32_t      dsu_record_enabled_endpoints_mask;
    uint32_t      dsu_metadata_enabled_endpoints_mask;
}
```

Description

This struct is a member of [ADQDataTransferParameters](#) and defines the data transfer parameters for a channel.

Members

`record_buffer_bus_address` ([uint64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`metadata_buffer_bus_address` ([uint64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`marker_buffer_bus_address` ([uint64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`nof_buffers` ([int64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`record_size` ([int64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`record_buffer_size` ([int64_t](#))

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`metadata_buffer_size (int64_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`record_buffer_packed_offset (int64_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`metadata_buffer_packed_offset (int64_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`record_buffer (volatile void *)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`metadata_buffer (volatile void *)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`marker_buffer (volatile void *)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`record_length_infinite_enabled (int32_t)`

Please see the ADQGen3 Streaming User Guide [8] for a complete description.

`record_enabled (int32_t)`

When set to 1, enables the transfer of record data via the regular streaming interface to the host RAM.

`metadata_enabled (int32_t)`

When set to 1, enables record headers in the transfer of records via the regular streaming interface to the host RAM.

`dsu_record_enabled (int32_t)`

When set to 1, enables the transfer of record data to disk via disk streaming.

`dsu_metadata_enabled (int32_t)`

When set to 1, enables the transfer of record metadata to disk via disk streaming.

`dsu_record_enabled_endpoints_mask (uint32_t)`

When `dsu_record_enabled` is set to 1, this parameter is used to set the endpoint indexes that are active for streaming of data from this channel. The indexing is handled by ADNVDs and should not be set manually, and must instead be retrieved via the function `adnvds_wr_get_ADQ_DSU_setup_params`.

`dsu_metadata_enabled_endpoints_mask (uint32_t)`

When `dsu_metadata_enabled` is set to 1, this parameter is used to set the endpoint indexes that are active for streaming of metadata from this channel. The indexing is handled by ADNVDs and should not be set manually, and must instead be retrieved via the function `adnvds_wr_get_ADQ_DSU_setup_params`.

```
struct DSU7MetadataRaw {
    uint8_t    pad;
    uint8_t    status;
    uint8_t    trigger;
    uint8_t    user_id;
    uint16_t   general_purpose_start;
    uint16_t   trigger_extended_precision;
    uint64_t   timestamp;
    uint16_t   lost_records;
    uint16_t   lost_cycles;
    uint32_t   record_number;
    uint64_t   record_length;
}
```

The raw record metadata stored on disk during a disk streaming acquisition, which can be parsed into a complete [DSU7RecordHeader](#).

Members

pad ([uint8_t](#))

Reserved.

status ([uint8_t](#))

This member has the same definition as [RecordStatus](#) in the complete header.

trigger ([uint8_t](#))

This member contains trigger information used internally by the ADQAPI during generation of the final [DSU7RecordHeader](#). It should not be used directly.

user_id ([uint8_t](#))

This member has the same definition as [UserID](#) in the complete header.

general_purpose_start ([uint16_t](#))

This member contains data path state information used internally by the ADQAPI during generation of the final [DSU7RecordHeader](#). It should not be used directly.

trigger_extended_precision ([uint16_t](#))

This member contains trigger information used internally by the ADQAPI during generation of the final [DSU7RecordHeader](#). It should not be used directly.

timestamp ([uint64_t](#))

This member contains timestamp information used internally by the ADQAPI during generation of the final [DSU7RecordHeader](#). It should not be used directly.

lost_records ([uint16_t](#))

This member has the same definition as [LostRecords](#) in the complete header.

lost_cycles (uint16_t)

This member has the same definition as [LostCycles](#) in the complete header.

record_number (uint32_t)

This member has the same definition as [RecordNumber](#) in the complete header.

record_length (uint64_t)

The record length as a 64-bit value. Here, this is expressed in units of *datapath clock cycles*, while the [RecordLength](#) field in the complete header counts in *samples*.

```
struct DSU7RecordHeader {
    uint8_t    RecordStatus;
    uint8_t    UserID;
    uint8_t    Channel;
    uint8_t    DataFormat;
    uint32_t    SerialNumber;
    uint32_t    RecordNumber;
    int32_t     SamplePeriod;
    uint64_t    Timestamp;
    int64_t     RecordStart;
    uint64_t    RecordLength;
    uint16_t    GeneralPurpose0;
    uint16_t    GeneralPurpose1;
    uint16_t    LostRecords;
    uint16_t    LostCycles;
}
```

The complete record header structure, after parsing the metadata from a disk streaming acquisition.

Members

RecordStatus (uint8_t)

This member is a 8-bit wide bit field holding status information about the record.

Bit 0

If this bit is set, this header corresponds to data up to and including the end of the record. When [dsu_forced_metadata_interval](#) is set to a value lower than the record length, multiple headers will be generated and only the last header will have this bit set.

Bit 1

If this bit is set, an overrange / clipping condition occurred.

Bit 2

If this bit is set, the trigger occurred on a rising trigger source edge, otherwise on a falling edge. Not relevant for all trigger sources.

Bit 4-7
Reserved

UserID ([uint8_t](#))

An 8-bit value that may be set from the development kit.

Channel ([uint8_t](#))

The originating channel, zero based.

DataFormat ([uint8_t](#))

The binary representation used for the record data:

- 0: 16-bit, 2's complement representation.
- 1: 32-bit, 2's complement representation.

SerialNumber ([uint32_t](#))

The numeric part of the digitizer's serial number. For example, this field would be set to 9999 for records acquired by a digitizer with serial number "SPD-09999".

RecordNumber ([uint32_t](#))

The record number as a 32-bit unsigned value. The first record acquired after starting the data acquisition will have this field set to zero.

Important

The record number wraps to zero at the maximum value.

SamplePeriod ([int32_t](#))

The time between two samples, expressed in units of 25 ps on ADQ8 and ADQ7, and 125 ps on ADQ14.

Timestamp ([uint64_t](#))

The timestamp of the trigger event, expressed in units of 25 ps on ADQ8 and ADQ7, and 125 ps on ADQ14.

RecordStart ([int64_t](#))

The time between the trigger event and the first sample in the record, expressed in units of 25 ps on ADQ8 and ADQ7, and 125 ps on ADQ14. This means that the timestamp of the first sample in the record is *the sum* of the values of [Timestamp](#) and [RecordStart](#).

- A value less than zero implies that the first sample in the record was acquired *before* the trigger event occurred (pretrigger).
- A value equal to zero implies that the first sample in the record was acquired *precisely* when the trigger event occurred.
- A value greater than zero implies that the first sample in the record was acquired *after* the trigger event occurred (trigger delay).

RecordLength ([uint64_t](#))

The length of the record as a 64-bit unsigned value, expressed in units of *samples*. If [dsu_forced_metadata_interval](#) is set to a value lower than the full record length, the value in this field will only correspond to the data between the generated metadata.

GeneralPurpose0 ([uint16_t](#))

Contains the state of the FWDSU data path general purpose bits at the start of the record, which can be controlled via the firmware development kit.

GeneralPurpose1 ([uint16_t](#))

Unused, defaults to zero.

LostRecords ([uint16_t](#))

This value shows the state of a 16-bit counter that, during overflow conditions, will increment by one every time the end of a record is seen and discarded.

LostCycles ([uint16_t](#))

This value shows the state of a 16-bit counter that, during overflow conditions, will increment by one every time a datapath clock cycle of data is discarded.

A.2 FWDSU Functions

GetDSUParameters	26
DSUUpdateDoorbellAddress	27
FlushDMA	27

```
int GetDSUParameters(
    uint64_t      * BAR_addr,
    unsigned int  * BAR_size_MiB,
    unsigned int  * read_size_min,
    unsigned int  * read_size_max,
    unsigned int  * nof_endpoints_max,
    unsigned int  * nof_dsu_ch
)
```

Retrieve FWDSU-specific information from the digitizer.

Return value

If the operation is successful, 1 is returned. Otherwise, 0 is returned.

Description

This function is used to retrieve information from the FWDSU-enable digitizer which is required to configure ADNVDs and the disks correctly, such as the PCI Express base address where the disks can access the data from the digitizer.

Parameters

BAR_addr ([uint64_t](#) *)

The BAR_addr parameter is a pointer to a 64-bit unsigned integer where the function will store the PCI Express base address where disk streaming data is made available by the digitizer.

BAR_size_MiB ([unsigned int](#) *)

The BAR_size_MiB parameter is a pointer to an unsigned integer where the function will store the size of its PCI Express address range in mibibytes.

read_size_min ([unsigned int](#) *)

The read_size_min parameter is a pointer to an unsigned integer where the minimum read size will be stored. Disk streaming uses a constant read size of ADNVDs_COMMAND_SIZE which is always supported, so there is no need to check this value.

read_size_max ([unsigned int](#) *)

The read_size_max parameter is a pointer to an unsigned integer where the maximum read size will be stored. Disk streaming uses a constant read size of ADNVDs_COMMAND_SIZE which is always supported, so there is no need to check this value.

`nof_endpoints_max (unsigned int *)`

The `nof_endpoints_max` parameter is a pointer to an unsigned integer where the number of simultaneous active endpoints supported by the digitizer will be stored. This is the total number across all the digitizer channels, and data and metadata for each channel count as separate endpoints.

`nof_dsu_ch (unsigned int *)`

The `nof_dsu_ch` parameter is a pointer to an unsigned integer where the number of disk streaming channels will be stored. Record data and record metadata will here count as separate channels.

```
int DSUUpdateDoorbellAddress(  
    unsigned int  endpoint,  
    unsigned int  STE,  
    uint64_t      doorbell_address  
)
```

Set the doorbell register address.

Return value

If the operation is successful, 1 is returned. Otherwise, 0 is returned.

Description

The doorbell register is used by the FWDSU firmware to let the disks know when more data is available for reading. It needs to be initially set to the first disk (or disks, if interleaving is enabled) in the disk sequence, which is done by the user. It also needs to be updated when switching to the next disk in the sequence, which is handled by ADNVDs automatically.

Parameters

`endpoint (unsigned int)`

The endpoint number. Each channel has two endpoints (data and metadata) in each interleaved disk. The index value is therefore in the range of 0 to `number_of_channels * number_of_interleaved_disks * 2`.

`STE (unsigned int)`

Reserved for internal use by ADNVDs.

`doorbell_address (uint64_t)`

The address to the doorbell register. For initial setup, this can be retrieved from ADNVDs via the function `adnvd_wr_get_ADQ_DSU_setup_params`.

```
int FlushDMA()
```

Force the completion of any partially filled transfer and stop the acquisition.

Return value

If the operation is successful, 1 is returned. Otherwise, 0 is returned.

Description

Calling this function forces any partially filled transfer buffers to fill up and immediately be parsed by the API, and also stops the data acquisition. For disk streaming enabled channels, it will force any incomplete NVME I/O command to be padded with zeroes and transferred to disk. For any record currently in progress it will also force output of the record metadata.

A.3 Metadata Parsing Functions

ADQData_Create	29
ADQData_InitPacketStream	29
ADQData_ParseDiskStreamHeaders	30

```
int ADQData_Create(
    void ** pref
)
```

Create an ADQData parser.

Return value

If the operation is successful, 1 is returned. Otherwise, 0 is returned.

Description

Create an ADQData parser.

Parameters

pref (void **)

A pointer to a void* where the reference to the ADQData parser will be stored.

```
int ADQData_InitPacketStream(
    void      * ref,
    void      * device_struct,
    const char * filename
)
```

Initialize an ADQData parser with information about the data acquisition.

Return value

If the operation is successful, 1 is returned. Otherwise, 0 is returned.

Description

Initialize an ADQData parser with information about the digitizer and the data acquisition. The information is required to successfully generate the complete record headers, and can be found in the dataset structure on the disk.

Parameters

ref (void *)

A reference to an ADQData parser, created via [ADQData_Create\(\)](#).

device_struct (void *)

A device information structure, which can be found in the disk dataset, under `adnvs_rd_data_set->adqdata_devi`

filename (const char *)
 Not used, set to NULL.

```
int ADQData_ParseDiskStreamHeaders(
    void          * ref,
    void          * stored_headers_buffer,
    unsigned int   nof_stored_headers,
    void          * target_headers_buffer,
    unsigned int * headers_added,
    unsigned int   channel_id
)
```

Convert an array of [DSU7MetadataRaw](#) into [DSU7RecordHeader](#).

Return value

If the operation is successful, 1 is returned. Otherwise, 0 is returned.

Description

Convert an array of [DSU7MetadataRaw](#) into [DSU7RecordHeader](#).

Parameters

ref (void *)

A reference to an ADQData parser, created via [ADQData_Create\(\)](#).

stored_headers_buffer (void *)

A pointer to an array of [DSU7MetadataRaw](#).

nof_stored_headers (unsigned int)

The number of [DSU7MetadataRaw](#) entries in the [stored_headers_buffer](#) array.

target_headers_buffer (void *)

A pointer to a memory region where the resulting array of [DSU7RecordHeader](#) will be stored. The memory must be allocated prior to calling this function.

headers_added (unsigned int *)

A pointer to an unsigned int where the number of successfully generated [DSU7RecordHeader](#) entries will be stored.

channel_id (unsigned int)

The index of the channel, zero-based. This can be retrieved from the [adnvds_rd_data_set](#).

Worldwide Sales and Technical Support

spdevices.com

Teledyne SP Devices Corporate Headquarters

Teknikringen 8D

SE-583 30 Linköping

Sweden

Phone: +46 (0)13 645 0600

Fax: +46 (0)13 991 3044

Email: spd_info@teledyne.com