skope

SKOPE-FM / SKOPE-FX
REMOTE ACQUISITION CONTROL AND DATA STREAMING
INTERFACE SPECIFICATION

Revision 2017a

TABLE OF CONTENTS

1. OVERVIEW

The skope-fm and skope-fx applications allow for streaming acquired data to external applications running on the same or separate machines, as well as remote access to all relevant acquisition parameters in real-time via custom clients. The communication with skope-fm/skope-fx is performed over TCP and is based on a client-server architecture with a simple protocol for synchronization. Each data block and command message sent or received via TCP is preceded by a block header of predefined size. The header provides, among other things, information about the size and type of the appended data as shown in Table 1. The protocol is open and a detailed description is given in Chapter 2 of this document. LabVIEW® and Matlab® client examples are also provided.

The following document specifies the interface to the skope-fm/skope-fx Data Streaming and Remote Acquisition Control modules, detailing the protocol to allow users to develop custom clients in a development environment of choice suited to the needs of the application. This allows extending the functionality provided by skope-fm/skope-fx. Figure 1 displays two examples highlighting the differences between the unilateral communication (server → client) as applied for data streaming and the bidirectional (server ↔ client) application control.

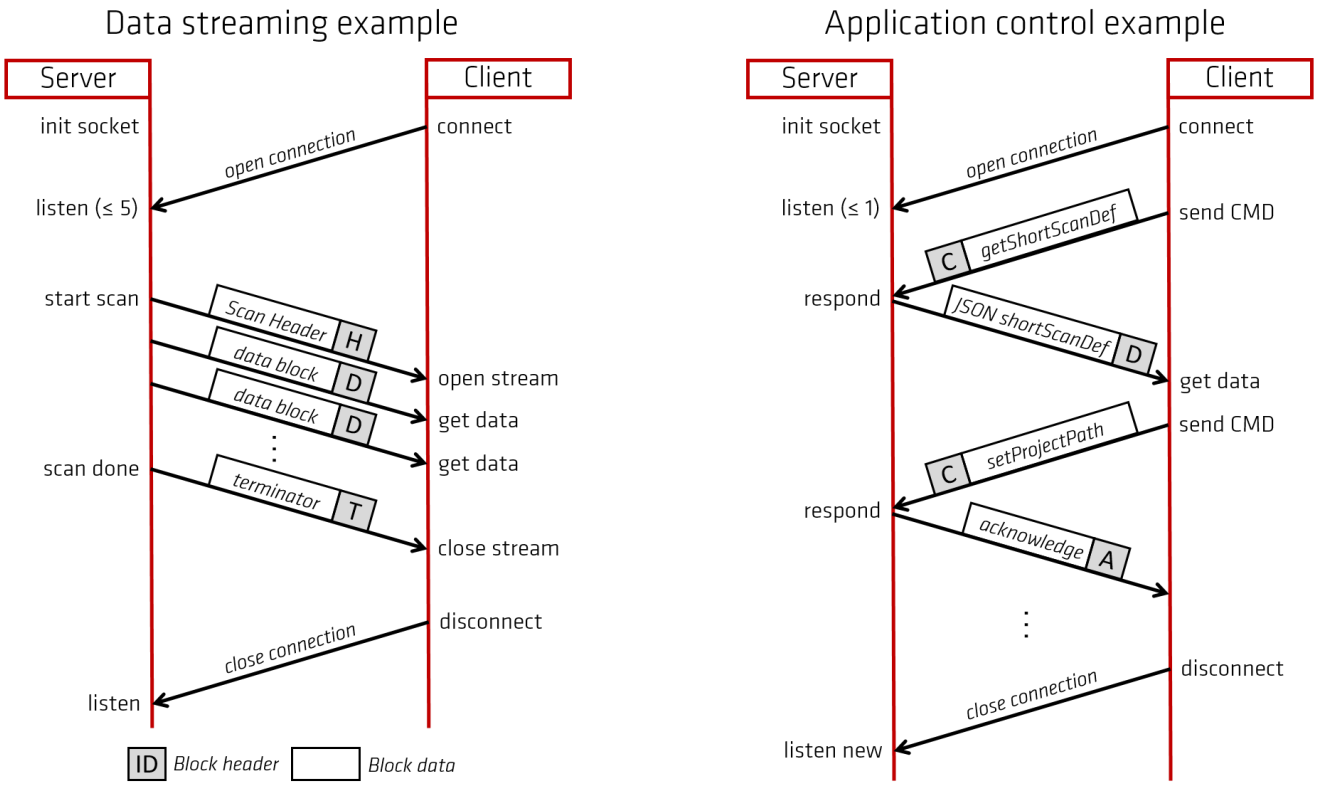

Figure 1: Client-server communication using the Data Streaming and Remote Acquisition Control modules.

2. GENERAL PROTOCOL DESCRIPTION

For synchronization, any communication from the server to the client and vice-versa is preceded by a so-called block header that has a fixed size of 42 bytes and provides all necessary information to receive and interpret the following data block. The data blocks can contain acquired data, control commands, scan information, status messages or error information and are thus of variable size. The size of the block is provided in the block header, which is necessary to read the correct amount of data from the TCP socket. The type of client and streamed data depends on the port on which the connection is initially established. An overview of the different ports is given in Table 1.

Basic parameters of the communication are set in the System Configuration file (.conf). The following options are set here: port base (default: 6400), maximum number of connections per data stream (default: 5), maximum number of timeouts during streaming before the connection is terminated (default: 1) and timeout [ms] (default: 100).

BLOCK HEADER FORMAT

The block header consists of the following fields: version number, data ID, 3 timestamps (send time, acquisition time and processing latency), the number of channels and the number of samples (or block size in bytes if it is not acquired data). All numbers are in big-endian format and are converted byte-wise to char prior to transmission.

The Data ID indicates the type of information contained in the subsequent block. For streaming applications, the Data ID can have the values **H**eader, **D**ata or **T**erminate, whereas for application control the values **C**ommand (from client to server), as well as **D**ata, **S**tatus, **E**rror and **A**cknowledge (from server to client) are permitted.

Version	Data ID	Send time	Acquisition time	Proc latency	Number	Size
'2017.0.0000'	**H**eader	[s] since 1.1.1904 00:00:00	unused	unused	unused	Block size (bytes)
	Data (streaming)		[s]	[s]	#Channels	#Samples
	Terminate		unused	unused	unused	unused
	Command					Block size (bytes)
	Data (control)					Block size (bytes)
	Status					Block size (bytes)
	Error					Block size (bytes)
	Acknowledge					unused
char array	char	double	double	double	U16 integer	U32 integer
11 bytes	1 byte	8 bytes	8 bytes	8 bytes	2 bytes	4 bytes
42 bytes						

Table 1: Block header structure. Note that the numbers are represented in big-endian format and converted byte-wise to char.

3. DATA STREAMING

The Data Streaming module enables users to stream all of the acquired data to external clients running on the same network. The type of data that is streamed is determined by the port on which the client connects, i.e. its position in relation to the port base as defined in the System Configuration file (.conf). The data types that can be streamed are summarized in the table below. It must be ensured that client and server are in the same network and that a firewall on the client side is not blocking the communication. When connected, the server sends data to all connected clients for each acquisition during the scan.

Data type	Bandwidth	Numeric Representation	Unit	Port
Command	-	-	-	portBase+0
Phase	1 MHz	Real double	Radian	portBase+1
Raw	1 MHz	complex integer (2 x 32-bit)	(relative value)	portBase+2
Trajectory (k)	1 MHz	Real double	(corresponding to basis function)	portBase+3
B fit	1/aq TR	Real double	Tesla (B field offset)	portBase+4
G fit	1/aq TR	Real double	(corresponding to basis function)	portBase+5
Log	-	-	-	portBase+6

Table 2: Overview of data types with their bandwidth, numerical representation and port. All numerical values are represented in big-endian format.

At the beginning of each measurement a scan header is sent to all connected data clients. The scan header is preceded by the predefined block header described above where the Data ID is set to "**H**". The scan header itself is sent in JSON format as name/value pairs and contains basic information about the performed measurement. Thereafter, data blocks (each preceded by a block header with Data ID value "**D**") are streamed for each interleave to the connected clients. Real-valued data (Phase, k, Bfit and Gfit) is streamed as 64 bit floating point numbers, whereas complex raw data is streamed as 32 bit integers. All numeric values are represented in big endian format and arranged as shown below.

Complex-valued data

Sample 1								...
Ch 1		Ch 2		...		Ch K		
Re	Im	Re	Im	...	...	Re	Im	

Real-valued data

Sample 1				...
Ch 1	Ch 2	...	Ch K	

Table 3: Data structure for complex- and real-valued data.

The end of the measurement is communicated to the clients by sending a block header with Data ID set to "**T**". The data clients can be disconnected from the server by simply closing the connection on the client side, or they can remain connected for further acquisitions. In case of timeouts caused by long delays or TCP buffer overflows, the clients are disconnected by the server to prevent inconsistent streaming.

The log stream is a special data stream that duplicates the log in skope-fm/skope-fx, which is also written to the log file, and can contain relevant information for building applications. The log messages are sent in order of occurrence as ASCII text and are preceded by a block header with Data ID value "**D**".

4. REMOTE ACQUISITION CONTROL

If the Remote Acquisition Control module is part of the software package provided by Skope, then the skope-fm/skope-fx application (server) awaits connections on the port base (default 6400). If a command client connects, the controls on the graphical user interface of skope-fm/skope-fx are disabled and no further connections are accepted until the client disconnects or the connection is lost. After the connection is closed the controls are re-enabled and a new connection can be made.

The following commands are available:

- *disconnect*
- *startScan*
- *stopScan*
- *setProjectPath*
- *getProjectPath*
- *resetScanDef*
- *setShortScanDef*
- *getShortScanDef*
- *quit*

After connection, the commands must be sent as JSON text to the server after a block header with Data ID value "**C**" has been transmitted. Commands are processed sequentially. The string sent to the server comprises a command name and, if necessary, an additional value (for the *set* commands):

{"command":"*command name*", "value":*desired value*}

Physical values are all returned and interpreted in SI units. If no problems occur, the server responds to the commands by sending a single **D**ata (for *get* commands) or **A**cknowledge (for all other commands) block back to the client. If information such as warnings or errors need to be communicated a **S**tatus (string) or **E**rror (JSON string) block is sent. Additional information for each command can be found in the documentation of the corresponding VI accessible in LabVIEW®. The *startScan* command may require up to 120 seconds to return a reply and always returns a **S**tatus block, which in the first line contains a HW initialization status and in the following lines any warnings and/or potential conflicts pertaining to the scan.

5. TCP CLIENT TOOLKIT

To facilitate the use of the Data Streaming and Remote Acquisition Control modules, the TCP Client Toolkit provides an implementation of the specified communication protocol for Matlab® and LabVIEW®. The toolkit methods are distributed for reuse and several example clients are proposed as a starting point for the development of more sophisticated user applications.

GETTING STARTED IN Matlab®

The toolkit is provided as a set of methods (.m files) which can be utilized in user implementations. Individual data streaming and remote acquisition control examples are provided as Matlab® scripts (.m files). Version R2016b or newer is required to run the provided examples. Note that the *Instrument Control Toolbox* is necessary to build any clients using capabilities of the Data Streaming and Remote Acquisition Control modules in Matlab®.

GETTING STARTED IN LabVIEW®

The LabVIEW® toolkit is provided as a packed library (.lvlibp) and requires LabVIEW® 2018 to run. To start using the TCP Client Toolkit extract the .zip file to the desired location in the project folder. After opening the project, simply drag and drop the packed library file (.lvlibp) into the project overview. To enable easy access to the provided methods the provided palette file (.mnu) can be added to the standard function palette set, as seen in **Figure 2**. This is performed by selecting *Tools → Advanced → Edit Palette Set* from the menu bar, then right-clicking at the desired free location and selecting *Insert → Subpalette(s) → Link to an existing palette file (.mnu)*[1]

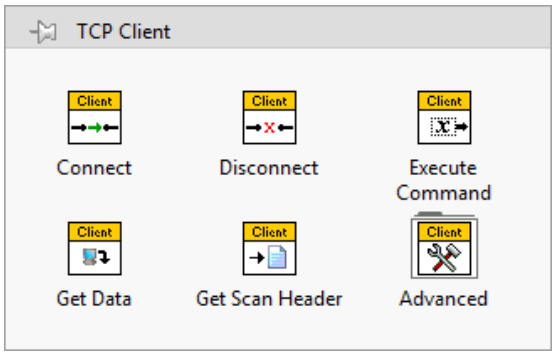

Figure 2: View of the TCP Toolkit palette set in LabVIEW® showing the provided methods.

The VIs are explained and additionally documented via standard LabVIEW® VI documentation, accessible though the context help (Ctrl+H).

[1] zone.ni.com/reference/en-XX/help/371361N-01/lvhowto/creating_and_editing_a_pal