Pulse Streamer-API migration document

Pulse Streamer 8/2, V0.9 -> V1.0.3

Backwards Compatibility:

The current Pulse Streamer 8/2 client software (available in Python, Matlab and LabView) is not backwards compatible with firmware V0.9 and the firmware versions 1.0.x are not working with the client software related to devices shipped before November 2018. All Pulse Streamer 8/2 delivered before November 2018 were shipped with firmware V0.9. We recommend to update all devices shipped before August 2019 to our latest firmware version V1.0.3.

Modified functionality (with regard to V0.9):

Default and recommended communication protocol between the PC and the Pulse Streamer is now JSON-RPC instead of Google-RPC.

The Pulse Streamer is automatically rearmed after a sequence with a finite number of `n_runs` has finished. That means if another trigger occurs after the sequence has finished, the sequence is running again.
To disable the auto rearm use the method `setTrigger(...)`. If automatic rearm functionality is disabled, you can manually rearm the Pulse Streamer by using the method `rearm()`. After that, you can retrigger a successfully finished sequence exactly one time by the trigger mode you selected with the `start` argument.

Underflows do not occur anymore, even at the highest possible data rate, which is streaming a new pattern every 1ns. Therefore, the `getUnderflow()` method always returns false.

The recommended way to stop the Pulse Streamer streaming is to set its output to a constant value via the method `constant()`. However, if you want to stop a running sequence and force it to the dedicated final state, you can do this by calling the method `forceFinal()`. If no final state was declared in the current sequence, the output of Pulse Streamer will change to (or stay in) the last known constant state. Furthermore, if upload-performance is crucial to your application, you can call this function directly before streaming the next sequence. This will increase the upload-performance by about 20 percent.

In order to communicate with the Pulse Streamer, you need to know its IP. By default, the Pulse Streamer will attempt to acquire an IP address via DHCP. The former fallback IP192.168.1.100 does no longer exist. Instead, there is a permanent second static IP-address 169.254.8.2/16. Via this address you can connect by directly plugging the Pulse Streamer to your computer. Maybe you will have to reboot Windows to detect the Pulse Streamer, if there was a connection via DHCP before. Furthermore, you can disable DHCP and

configure a static IP instead. We provide tools for doing so on our website. For help please contact support@swabianinstruments.com.

Using the revised Pulse Streamer 8/2 client software:

For the Pulse Streamer 8/2 we provide software clients to allow easy and convenient access to the device. The latest release of the clients harmonizes the terminology and functionality of different programming languages. It is available in Python, Matlab and LabView. In general, the clients consist of a PulseStreamer and a Sequence module. The class PulseStreamer is a wrapper for the RPC interface provided by the Pulse Streamer hardware. It handles the connection to the hardware and exposes all available methods. The Sequence contains information about the patterns and channel assignments. It allows you to create sophisticated sequences for your Pulse Streamer application.

The latest client software version makes several changes to or replaces methods of the PulseStreamer class in relation to the version shipped with firmware V0.9. To adapt code to the latest client software interfaces, the following lines shall give a small overview to the changes of the rpc-calls. For a more detailed description, please have a look at our profound documentation:

https://www.swabianinstruments.com/static/documentation/pulse-streamer/v1.1/index.html

Modified methods:

```
stream(…)
```

Removed methods:

```
isRunning()
```

New methods:

```
reset()
createSequence()
forceFinal()
isStreaming()
hasFinished()
setTrigger(…)
rearm()
selectClock(…)
getFirmwareVersion()
getSerial()
```

If you need assistance with the transcription of your code,
please contact support@swabianinstruments.com.

Using the low-level RPC interface:

Although we recommend to use our client software to handle the Pulse Streamer 8/2, this section describes the changes of the low-level RPC interface of the Pulse Streamer with regard to firmware version V0.9. You can use this information for a deeper understanding of how to control Pulse Streamer hardware or to develop/adapt your own communication programs.

Modified methods:

void stream(std::vector<Pulse> sequence,
int64_t n_runs=INFINITE,
Pulse final=CONSTANT_ZERO)

Running a pulse sequence corresponds to a single function call where you pass your pulse sequence as the `sequence` argument.
You can repeat a pulse sequence indefinitely or an integer number of times which is controlled via the parameter `n_runs`.
A sequence run will start from the current constant output state. After the sequence has been repeated the given `n_runs`, the `final` output state will be reached.
By default, the sequence is started immediately. Alternatively, you can tell the system to wait for a later software start command or for an external hardware trigger applied via `setTrigger(...)`.
The sequence is repeated infinitely if n_runs < 0 and a finite number of repetitions otherwise. `INFINITE` is a symbolic constant with the value -1. `final` represent the constant output after the sequence is finished (the tick values are ignored).
`CONSTANT_ZERO` is a symbolic constant for a pulse with the value {0,0,0,0}. A pulse has the data structure:

```
struct Pulse {uint32_t  ticks, // duration in ns
      uint8_t digi, // digital channel bit mask
      int16_t ao0, // analog channel 0
      int16_t ao1, // analog channel 1
};
```

All parameters except `sequence` have default values and can be omitted.
The parameter `start mode` has been moved to the method *setTrigger(start_t star, ...)*
The `underflow` state does not exist any more, because undeflows cannot occur anymore even when every ns a different output pattern is defined.
The `initial` state can be set via *setConstant(...)*.

Removed methods:

bool isRunning()

The method *isRunning()* in the old firmware version did not return whether the pulse streamer was currently streaming output. That is why it is removed. New methods to check the current state of the Pulse Streamer are: *isStreaming()*, *hasFinished()*, and *hasSequence()*

New methods:

void reset()

All outputs are set to 0V and all functional configurations are set to default. The automatic rearm functionality is enabled, the clock source is the internal clock of the device. No specific trigger functionality is enabled, which means that each sequence is streamed immediately when its upload is completed.

std::string getSerial(serial_t serial=MAC)

The method returns a hexadecimal string containing either the serial number/MAC-address or the ID-number of the FPGA depending of the value of the argument serial. `Serial` is an enum with the mapping {ID:0; MAC:1}.

std::string getFirmwareVersion()

The method returns the version number of the current firmware.

void selectClock(clocking_t clocksource=INTERNAL)

The Pulse Streamer can be fed in with three different clock sources. By default, the clock source is the internal clock of the device. It is also possible to feed in the system by an external clock of 125MHz (sampling clock) or an external 10MHz reference clock. `clock_source` is an enum with {INTERNAL:0, EXT_125MHZ:1, EXT_10MHZ:2}.

bool isStreaming()

This method replaces its predecessor `isRunning()`. In contrast to `isRunning()`, this method only returns `true` if the Pulse Streamer is streaming the current sequence. When the sequence is finished and the device remains in the final state, this method returns `false` again.

bool hasFinished()

This method returns `true` if the Pulse Streamer remains in the final state after having finished the sequence.

void setTrigger(start_t start, trigger_mode_t mode=NORMAL)

This method configures the trigger functionality, which is no longer set in the stream-method.
`start` is an enum with the mapping {IMMEDIATE:0, SOFTWARE:1, HARDWARE_RISING:2, HARDWARE_FALLING:3, HARDWARE_RISING_AND_FALLING:4} specifying how the stream should be started. If you have passed start=SOFTWARE, you can start the sequence using the method `startNow()`. If you want to trigger the Pulse Streamer by using the external trigger input of the device you have to pass HARDWARE_RISING (rising edge is the active trigger flank), HARDWARE_FALLING (falling edge is the active trigger flank) or HARDWARE_RISING_AND_FALLING (both edges are active) to the `start` argument.

`mode` is an enum with the mapping{NORMAL:0, SINGLE:1}. If automatic rearm functionality is enabled (mode=NORMAL) you can retrigger a successfully finished sequence, by the trigger mode you selected with the `start` argument. You can disable the automatic rearm by passing SINGLE to the `mode` argument.

Unchanged methods:

void constant(Pulse pulse)

This method sets all outputs to 0V. If you set the device to a constant output an eventually currently streamed sequence is stopped. It is not possible to retrigger the last streamed sequence after setting the Pulse Streamer constant.

void startNow()

By using this method, you can start a sequence, if you have passed SOFTWARE to the `start` argument via `setTrigger(…)`.

bool hasSequence()

The method returns `true` if a uploaded sequence is ready within the memory of the Pulse Streamer.