Project Name:	Valiquad
Document Title:	Protocol
Autor(s):	Manuel Stübler
Created on:	05.11.10
Last modified:	20.12.10
Version:	v2.0
Status:	| | in progress | | presented |x| closed
Located at:	/docs/se/protocol.odt

Revision History:

#	Date	Version	Section	Description	Author	Status
1	05.11.10	v1.0	all	First creation	Manuel Stübler	closed
2	14.11.10	v1.1	3	Updated description	Manuel Stübler	closed
3	14.11.10	v1.2	3	Updated protocol	Manuel Stübler	closed
4	15.11.10	v1.3	3	Added frame picture	Manuel Stübler	closed
5	15.11.10	v1.4	3	Added frame description	Manuel Stübler	closed
6	19.11.10	v1.5	3	Changed ParID structure	Manuel Stübler	closed
7	26.11.10	v1.6	3	Changed MsgIDs	Manuel Stübler	closed
8	30.11.10	v1.7	3	Added declaration	Manuel Stübler	closed
9	03.12.10	v1.8	3	Explained “why”	Manuel Stübler	closed
10	14.12.10	v1.9	3	Changed type IDs	Manuel Stübler	closed
11	20.12.10	v2.0	3	Changed Message IDs	Manuel Stübler	closed

Table of Contents

List of Figures

List of Tables

1 Introduction

This document describes the protocol used for the communication between the helicopter (Quadrocopter) and the analyzing software (Valiquad). Implementation details are part of the hardware implementation document (impl-hw.odt) and are not specified any further here.

This protocol is independent from the implementation and the sample-rates the system is using. This approach is useful to separate the protocol from the system constraints. Of course there will be a matching of the protocol to the given system, but separating both leads to a higher portability of the protocol. This idea is also part of the ISO/OSI-Model.

The protocol described in this document acts on the layers 6 and 7 (Application and Presentation Layer) of the ISO/OSI-Model and is independent from the sub-layers it is working on. One possibility for those sub-layers might be the ZigBee protocol working upon an XBee module.

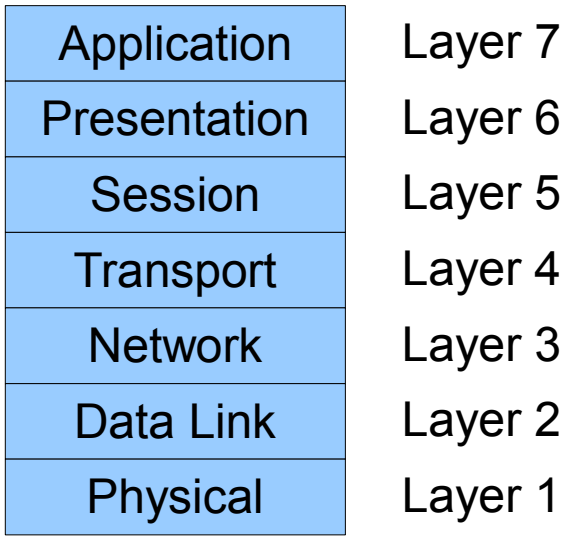

Figure 1: ISO/OSI-Model

2 Protocol in Detail

2.1 Packet structure

The protocol used for the data transmission is based upon packets that are sent from one instance to another. Such a packet consists of a message identification (MsgID) that describes the structure of a packet, the size of the payload data (length), the payload itself and a check-sum that ensures the integrity of a packet.

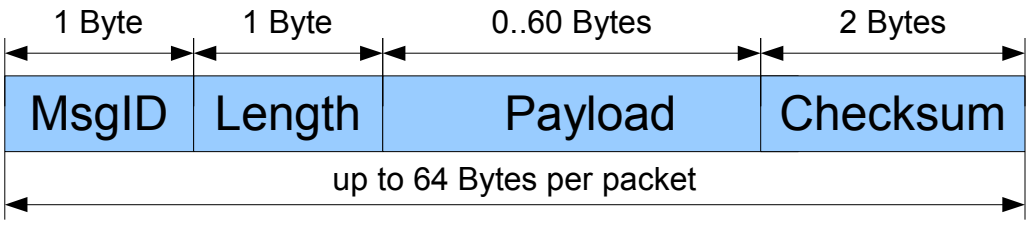

Figure 2: Packet structure

Currently such a packet has a maximum length of 64 Bytes, this is used to ensure that the protocol works with hardware layers that can only transmit packets with a maximum size of 64 Bytes. The XBee modules for example allow the transmission of packets with a size of 70 to 100 Bytes, dependent on the version of the module.

The central point of this protocol is the reading and setting of parameters and the cyclic transmission of sensor values, called observables. Each parameter is described by a unique parameter identification number (ParID). Using this generic approach it is possible to make the software and the protocol as dynamic as possible. If a new parameter is added to the system, the analyzing tool does not need adaptation for the new purposes. By defining a new parameter with its according ID and its data type, the system is able to fulfill all its purposes immediately without knowing any further things about this parameter, than its parameter ID, which contains all relevant data like the data type, the group membership and a sequence number that is unique within group.

As the helicopter contains several sensors that shall be read out periodically, it is more efficient to introduce a mechanism that tells the helicopter to send special parameters at a given sample rate without polling for every single value over and over again, which would cause a dramatic raise of payload to be transmitted. The software can tell the helicopter which parameters shall be sent within a given period of time, those parameters are also called observables.

2.2 Frames

To reduce the complexity of the protocol and still guarantee the possibility to configure the system individually, four different frames are created: A, B, C and D. Every frame is linked to a periodically sent packet that can be configured by the application. Those frames are sent one after another periodically at a given sample rate. Every frame has a payload length of up to 60 Bytes.

For every frame a list of parameters can be defined, that are sent within this frame from the helicopter to the analyzing tool. Those frames can be configured individually by adding / deleting parameters to / from the frame list.

To sample a value at the highest possible sample rate it has to be put into all frames (A, B, C and D). If a value is sent within the frames A and C or B and D, its sampling rate is half of the highest possible one. By putting a value only in one of the four frames, it is sampled at a forth of the original rate.

The following figure illustrates the frame-transmission over time:

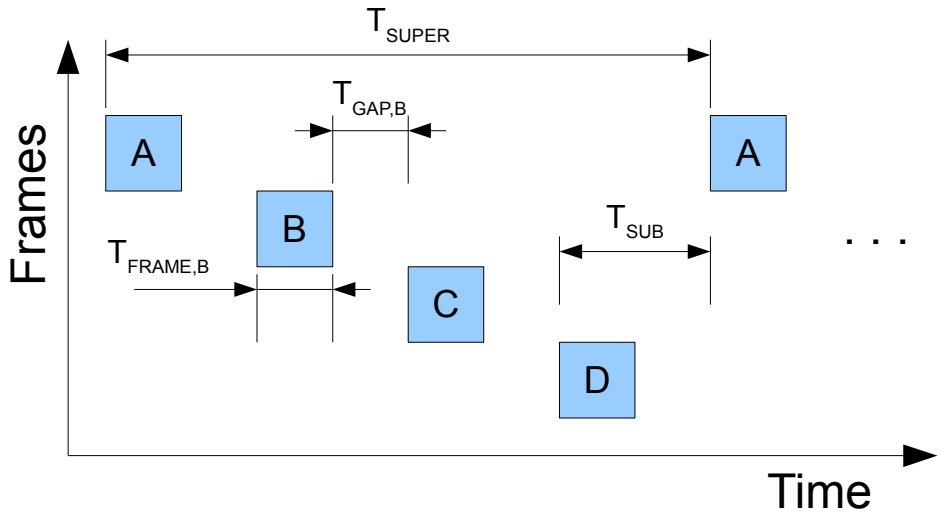

Figure 3: Frames over time.

T_{SUPER} is the super-period and T_{SUB} is the sub-period of the system, they are the same for every frame. $T_{FRAME,B}$ is the time for transmitting frame B and $T_{GAP,B}$ is the gap between the end of the transmission of frame B and the beginning of the next sub-period. The relations between those values are:

- $T_{SUPER} = 4 \cdot T_{SUB}$
- $T_{SUB} = T_{GAP,i} + T_{FRAME,i}$ *with* $i \in \{A, B, C, D\}$

To ensure that the processor has still enough time for doing other things than just transmitting frames, the following relation must be fulfilled for every single frame:

- $T_{GAP,i} \approx T_{FRAME,i}$ *with* $i \in \{A, B, C, D\}$

By using four frames it is possible to create 3 different sample rates for the observables. Those are one, two or four times the super-period T_{SUPER}. Having only two frames would restrict the configuration of the sample-rate too much, having more than 4 frames would cause an enormous raise of complexity. Using three frames is not a good choice, as this uneven number would result in the impossibility to sample a value at twice the super-period, because the placing of the values within the frames is fixed and cannot be changed from one super-period to another.

The implementation of the protocol can have various interfaces for the user to configure the frames. One possibility is to configure every frame separately, this a very generic way that offers a lot of individual settings for the system. The more user friendly way is to tell the system which parameter shall be sampled at which rate and the system itself handles the matching of those parameters to the frames. Which interfaces are offered is part of the responsibility of the implementation and further described in the implementation document of the hardware (impl-hw.odt).

If the observables are automatically placed to their corresponding frames, the system must guarantee an adequate balancing mechanism. The mapping of the observables must be done equally, this means that a new observable that shall be sampled at a given rate shall be put into the smallest frames. The smallest frames are those, that have the highest amount of free bytes. This helps to utilize the frames in a good manner and can prevent the system from running into a one-way-street. If the system came into such a one-way-street street, no more new parameters can be added to the frames, even if there is theoretically still enough space for the placement of those observable parameters.

For example if frame A is full with observables that are sampled within the lowest possible sample-rate, which means that they were only put into this frame A, no new observable with the highest possible sample-rate can be added to the system, because nothing can be placed into frame A as this frame is already full. If those observables would have been mapped equally to the frames A to D, none of those frames would be completely full, but all frames would have about a quarter of their maximum possible payload filled with those observable parameters. Now the same observables are sent at the same sample-rate, but this time there is still enough space for placing new observables to the system, even when using the highest possible sample-rate, which would need the placement of this observable into every single frame, namely the frames A, B, C and D.

2.3 Parameter IDs (ParIDs)

Parameter IDs (ParIDs) consist of 2 Bytes, and identify a parameter within the system uniquely. There are two different types of parameters, the observables and the normal ones. Observables are values a sensor delivers periodically within the cyclic frames (as defined in Chapter 2.2). Normal parameters cannot be sent within those frames, they are set and read out by polling the system.

The composition of those 2 Bytes is described in the following figure:

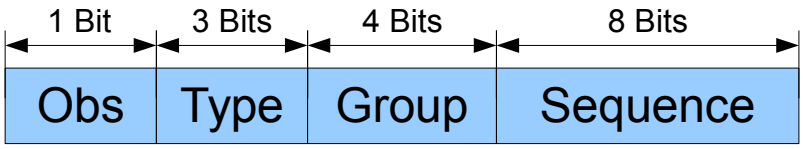

Figure 4: ParID Composition

The particular elements of a Parameter ID (ParID) are described in the next table:

Name	Range	Description
Obs	0x0..0x1	Parameter is an observable (Obs = 1) or not (Obs = 0).
Type	0x0..0x7	Defines the data type of the parameter.
Group	0x0..0xF	Assigns a group ID to the parameter.
Sequence	0x00..0xFF	The sequence number must be unique within a group.

Table 1: Parameter ID (ParID) Composition

The group ID and the sequence number form a unique ID for every parameter, therefore it has to be enforced that no sequence number is used several times for parameters in the same group. Those groups are used to arrange and structure parameters. Because only the group ID and the sequence number are used to describe a unique parameter, the type and the observable state can be changed easily without affecting the group ID and sequence number. But changing the group of a parameter may result in the need of also changing the sequence number of it, as there might be another parameter in the new group with the same sequence number.

Those group IDs were introduced to group several parameters logically together. Those groups can be used to arrange parameters together, for example within the graphical user interface.

The following table assigns the existing data types to the available type IDs. Such a type ID is encoded in the Parameter ID (ParID) to calculate the size of a configured frame and check for the integrity of this configuration (for example the configured frame does not exceed in size (max. 60 Bytes payload)). It is also necessary to interpret a parameter correctly on the host.

Type ID	Type Description
0x0	float32 (4 Byte)
0x1	float64 (8 Byte)
0x2	uint8 (1 Byte)
0x3	int8 (1 Byte)
0x4	uint16 (2 Byte)
0x5	int16 (2 Byte)
0x6	uint32 (4 Byte)
0x7	int32 (4 Byte)

Table 2: Type IDs

2.4 Message IDs (MsgIDs)

In order to fulfill the task of getting / setting / sampling parameters, a few methods have to be defined. Those methods are uniquely described through a so-called Message ID (MsgID):

Message ID (MsgID)	Explanation
0x01	Set sub-period. (T_{SUB}) If this is set to 0, no frames are sent and the frame transmission is stopped. This message is only acknowledged if the transmission is stopped, otherwise the acknowledgment is indirectly done with the first frame packet received. Setting the sub-period to another value requires to stop it first and restart the transmission with the new value.
0x02	Set parameter(s).
0x03	Get parameter(s).

Table 3: Message IDs (MsgID) Set / Get Parameter

As the parameter ID (ParID) of a value has 2 Bytes, the smallest types (uint8 / int8) are encoded in just 1 Byte and the size of the packets is fixed, every frame is divided into another two sub-frames (A1, A2 … D1, D2). Every sub-frame (first and second half of a normal frame) is configured separately, but the values that are sent from the helicopter to the analyzing software are put together into one single frame. If just values are used that are encoded in 2 or more Bytes ((u)int16 and (u)int32), configuration of a frame only requires the first configuration packet (A1 / B1 / C1 / D1). This mechanism is just some kind of a fragmentation of packets, because the configuration packets can be twice as long as the frames themselves and both packet types have the same maximum transmission size.

To be even more precise, the first configuration packet configures up to 30 parameters for a frame, if there is still space for more parameters in this frame, they are configured with the second configuration packet, which itself can also hold up to 30 parameters. In return the answer of the helicopter when requesting the current frame configuration is structured the same way, the analyzing software requests the first half and the second half with two separate request messages.

The packet configuration and calculation is done at the host, because there is more processing power available, than on the helicopter.

Configure the frames A to D, by splitting them into two sub-frames that are configured separately but transmitted at once. To configure those frames, the frame transmission has to be stopped (setting the sub-period (T_{SUB}) to 0). Those messages are sent from the analyzing software to the helicopter:

Message ID (MsgID)	Explanation
0x10	Configure parameter(s) for frame A, part 1. (A1)
0x11	Configure parameter(s) for frame A, part 2. (A2)
0x12	Configure parameter(s) for frame B, part 1. (B1)
0x13	Configure parameter(s) for frame B, part 2. (B2)
0x14	Configure parameter(s) for frame C, part 1. (C1)
0x15	Configure parameter(s) for frame C, part 2. (C2)
0x16	Configure parameter(s) for frame D, part 1. (D1)
0x17	Configure parameter(s) for frame D, part 2. (D2)

Table 4: Message IDs (MsgID) Frame Configuration Messages

Response messages to a request have the same ID. Because the participants of the protocol are asymmetric, response messages can be matched to an according request without using a separate Message ID for those responses. If no error occurred, an empty response packet is sent, if an error occurred, an error code is sent back within the response packet.

Those are the frame messages that are send periodically from the helicopter to the analyzing software at the defined sample-rate. Only observables can be sent within those frames, retrieving the values of normal parameters is done with the GetParameter method as defined previously.

Message ID (MsgID)	Explanation
0x30	Frame A with the according sampling values.
0x31	Frame B with the according sampling values.
0x32	Frame C with the according sampling values.
0x33	Frame D with the according sampling values.

Table 5: Message IDs (MsgID) Frame-Transmission

Message IDs below 0x80 are the well-known IDs and do not change for this protocol. Message IDs above 0x80 are reserved for user-defined purposes and may change over time.

2.5 Frame merging

If transmitting a frame at a given sample-rate is not possible, because this would cause the calculation of the control-loop parameters to stumble, several frames shall be put together in one super-frame and sent at a lower sample-rate. For the sampled values it is no problem, if they arrive a bit later.

If frames are merged and sent at a lower rate than the sampling rate of the containing values, the sensor data has to be buffered within the helicopter. The possible merging of packets is part of the implementation and is not described any further here.