GenericByteProtocolHandler

The idea behind the GenericByteProtocolHandler is that you often have a binary protocol, which you don't need to be translated in detail in a data structure. You often only need a specific part of the protocol.

With the GenericByteProtocolHandler it is possible to map bits to simple data types in a schema. See the example below.

Parameters

- options: There must be an options key for each schema entry and they must be in the order as they occur as bits in the input protocol. The value is the amount of bits, e.g. ['eth_mac_destination', 48]. For the last entry, the amount of bits can be set to -1 to indicate that all remaining bits shall be used.

Example

In the following example, a pcap file is read and the resulting packet data (byte array) is processed by a converter that uses the GenericByteProtocolHandler.

```
#PARSER PQL

#ADDQUERY
/// 1. read recorded network traffic from pcap file
pcap_input = ACCESS({
                  transport = 'file',
                  protocol = 'pcap',
                  wrapper = 'GenericPull',
                  source = 'Pcap',
                  datahandler = 'keyvalueobject',
                  options = [
                    ['filename', '${PCAP_FILE}']

]


}

              )
/// 2. relevant block type is EnhancedPacketHeader
enhanced_packet_header = SELECT({
                              predicate = 'blocktype ==
"EnhancedPacketHeader"'

                            },
                            Pcap_input
                          )
/// 3. filter packet data and transform to tuple for further processing
packet_data_tuple = TOTUPLE({
                       schema = [['packetData', 'packetData',
'List_Byte']]

                     },
                     enhanced_packet_header
                   )
/// 4. Transform to list_byte
packet_data_string = MAP({
                         expressions = [
                                                                ['toByteList(packetData)', 'packetData']
                           ]
                       },
                       packet_data_tuple
                     )
/// 5. convert to ethernet objects
packet_data_itemized = CONVERTER({
                           protocol = 'GenericByteProtocol',
                           outputdatahandler = 'tuple',
                           inputdatahandler = 'tuple',
                           options = [
                             ['charset', '${ENCODING}'],
                             ['newline', 'false'],
```

```
                            ['eth_mac_destination', 48],
                            ['eth_mac_source', 48],
                            ['eth_ether_type', 16],
                            ['ip_version', 4],
                            ['ip_ihl', 4],
                            ['ip_tos', 8],
                            ['ip_total_length', 16],
                            ['ip_identification', 16],
                            ['ip_flags', 3],
                            ['ip_fragment_offset', 13],
                            ['ip_ttl', 8],
                            ['ip_protocol', 8],
                            ['ip_header_checksum', 16],
                            ['ip_source_ip_address', 32],
                            ['ip_destination_ip_address', 32],
                            ['tcp_source_port', 16],
                            ['tcp_destination_port', 16],
                            ['tcp_sequence_number', 32],
                            ['tcp_ack_number', 32],
                            ['tcp_data_offset', 4],
                            ['tcp_reserved', 6],
                            ['tcp_flags', 6],
                            ['tcp_window_size', 16],
                            ['tcp_checksum', 16],
                            ['tcp_urgentPointer', 16],
                            ['tcp_payload', -1]
                          ],
                          schema = [
                            ['eth_mac_destination', 'List_Byte'],
                            ['eth_mac_source', 'List_Byte'],
                            ['eth_ether_type', 'Short'],
                            ['ip_version', 'Byte'],
                            ['ip_ihl', 'Byte'],
                            ['ip_tos', 'Byte'],
                            ['ip_total_length', 'Short'],
                            ['ip_identification', 'Short'],
                            ['ip_flags', 'Byte'],
                            ['ip_fragment_offset', 'List_Byte'],
                            ['ip_ttl', 'Byte'],
                            ['ip_protocol', 'Byte'],
                            ['ip_header_checksum', 'Short'],
                            ['ip_source_ip_address', 'List_Byte'],
                            ['ip_destination_ip_address', 'List_Byte'],
                            ['tcp_source_port', 'Short'],
                            ['tcp_destination_port', 'Short'],
                            ['tcp_sequence_number', 'Integer'],
                            ['tcp_ack_number', 'Integer'],
                            ['tcp_data_offset', 'Byte'],
                            ['tcp_reserved', 'Byte'],
                            ['tcp_flags', 'Byte'],
                            ['tcp_window_size', 'Short'],
                            ['tcp_checksum', 'Short'],
                            ['tcp_urgentPointer', 'Short'],
                            ['tcp_payload', 'List_Byte']

]

                          },
                          packet_data_string
                        )
```