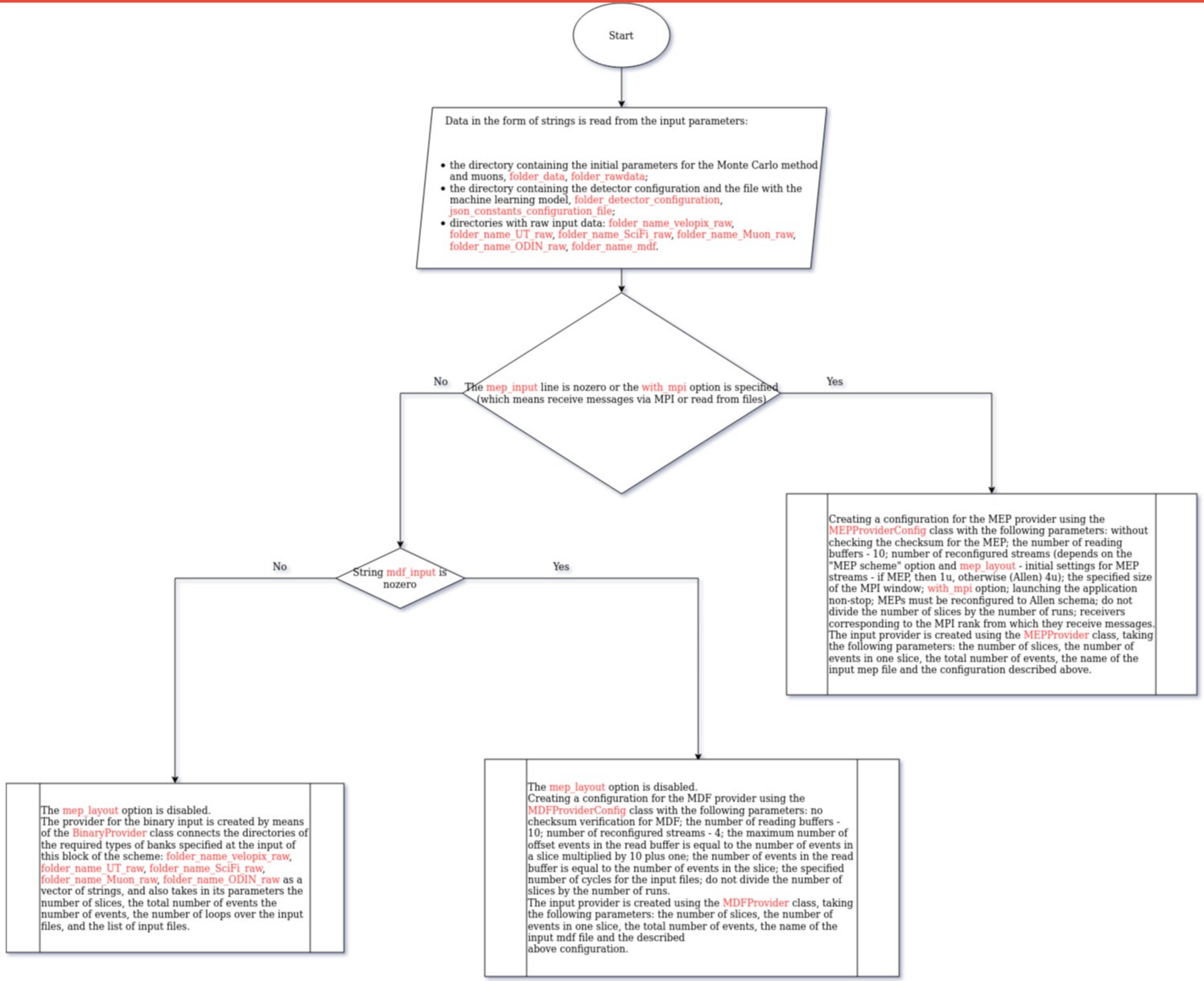
Some Allen algorithms related to the I / O data lifecycle

Anna Belova, june 2021

Initialization of the input provider depending on the type of data input



Start

Data in the form of strings is read from the input parameters:

- the directory containing the initial parameters for the Monte Carlo method and muons, folder_data, folder_rawdata;
- the directory containing the detector configuration and the file with the machine learning model, folder_detector_configuration, json_constants_configuration_file;
- directories with raw input data: folder_name_velopix_raw, folder_name_UT_raw, folder_name_SciFi_raw, folder_name_Muon_raw, folder_name_ODIN_raw, folder_name_mdf.

The mep_input line is nozero or the with_mpi option is specified (which means receive messages via MPI or read from files)

Yes

Creating a configuration for the MEP provider using the MEPProviderConfig class with the following parameters: without checking the checksum for the MEP; the number of reading buffers - 10; number of reconfigured streams (depends on the "MEP scheme" option and mep_layout - initial settings for MEP streams - if MEP, then 1u, otherwise (Allen) 4u); the specified size of the MPI window; with_mpi option; launching the application non-stop; MEPs must be reconfigured to Allen schema; do not divide the number of slices by the number of runs; receivers corresponding to the MPI rank from which they receive messages.
The input provider is created using the MEPProvider class, taking the following parameters: the number of slices, the number of events in one slice, the total number of events, the name of the input mep file and the configuration described above.

No

String mdf_input is nozero

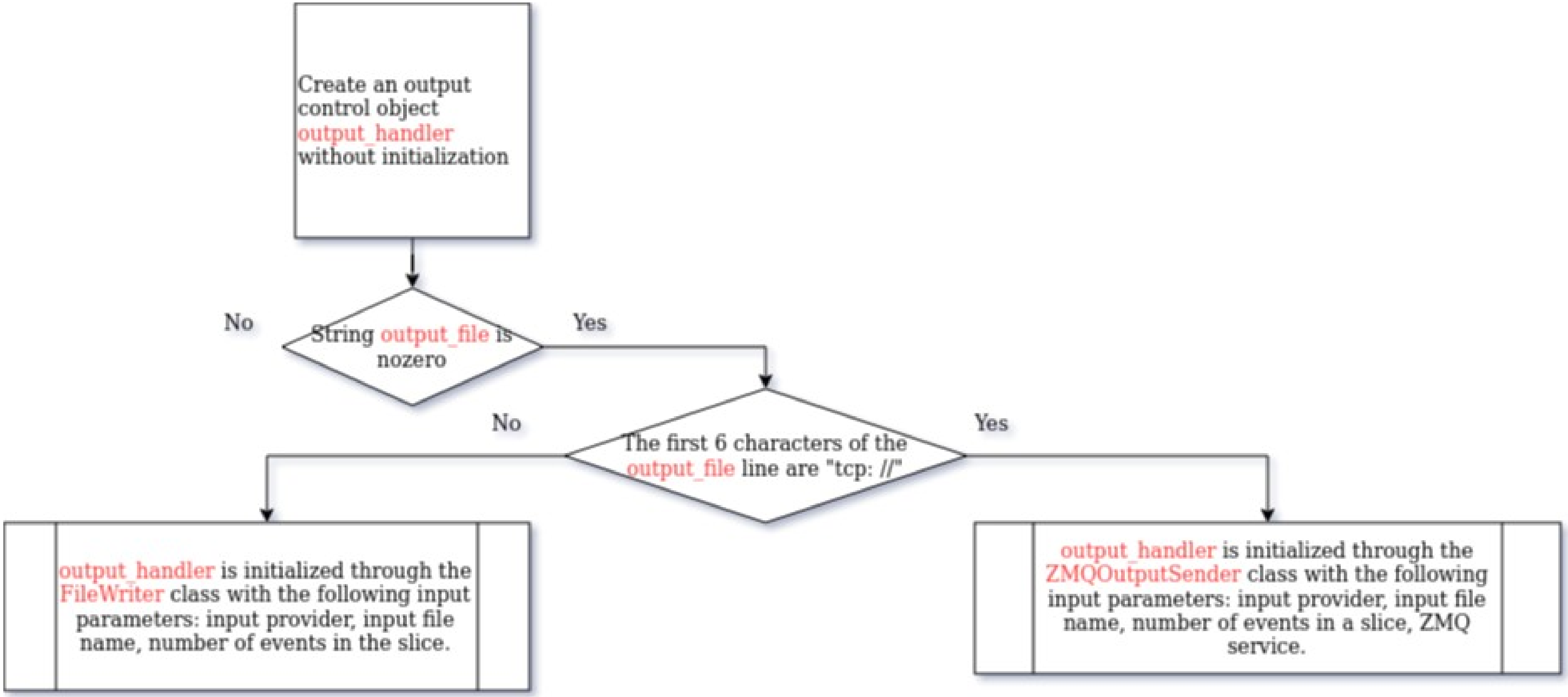
Yes

The mep_layout option is disabled.
Creating a configuration for the MDF provider using the MDFProviderConfig class with the following parameters: no checksum verification for MDF; the number of reading buffers - 10; number of reconfigured streams - 4; the maximum number of offset events in the read buffer is equal to the number of events in a slice multiplied by 10 plus one; the number of events in the read buffer is equal to the number of events in the slice; the specified number of cycles for the input files; do not divide the number of slices by the number of runs.
The input provider is created using the MDFProvider class, taking the following parameters: the number of slices, the number of events in one slice, the total number of events, the name of the input mdf file and the described
above configuration.

No

The mep_layout option is disabled.
The provider for the binary input is created by means of the BinaryProvider class connects the directories of the required types of banks specified at the input of this block of the scheme: folder_name_velopix_raw, folder_name_UT_raw, folder_name_SciFi_raw, folder_name_Muon_raw, folder_name_ODIN_raw as a vector of strings, and also takes in its parameters the number of slices, the total number of events the number of events, the number of loops over the input files, and the list of input files.

Initialization of the output handler and input provider starting



Create an output control object output_handler without initialization

String output_file is nozero

No

Yes

The first 6 characters of the output_file line are "tcp: //"

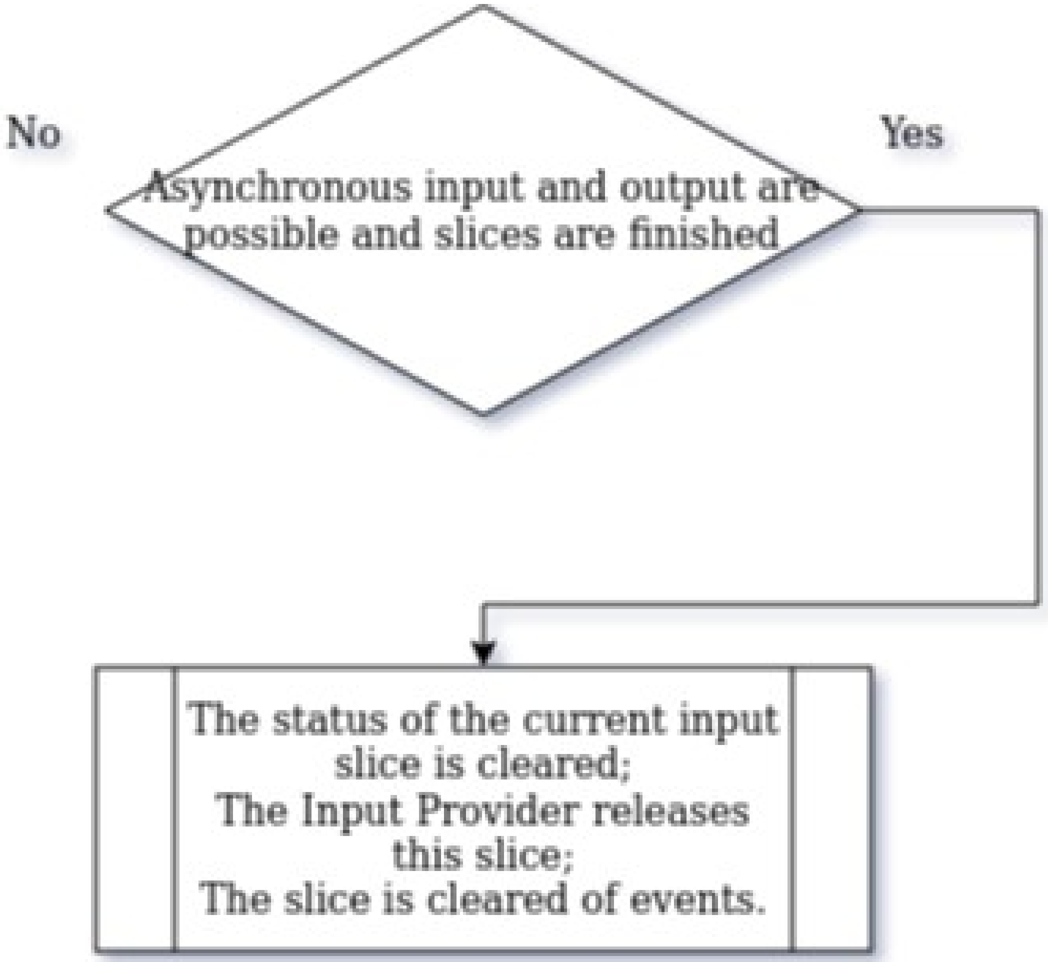
No

Yes

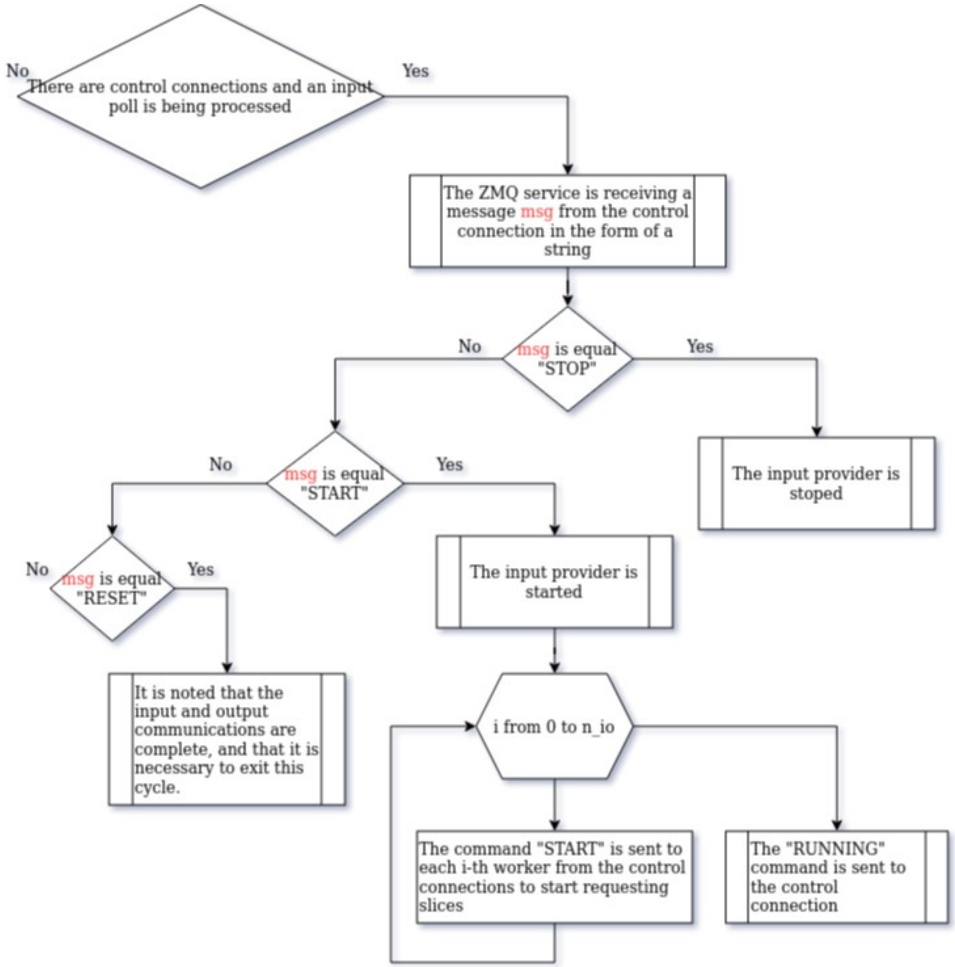
output_handler is initialized through the FileWriter class with the following input parameters: input provider, input file name, number of events in the slice.

output_handler is initialized through the ZMQOutputSender class with the following input parameters: input provider, input file name, number of events in a slice, ZMQ service.

Async checking


No
Yes
Asynchronous input and output are possible and slices are finished
The status of the current input slice is cleared;
The Input Provider releases this slice;
The slice is cleared of events.

ZMQ-service communication



There are control connections and an input poll is being processed

The ZMQ service is receiving a message msg from the control connection in the form of a string

msg is equal "STOP"

The input provider is stoped

msg is equal "START"

The input provider is started

i from 0 to n_io

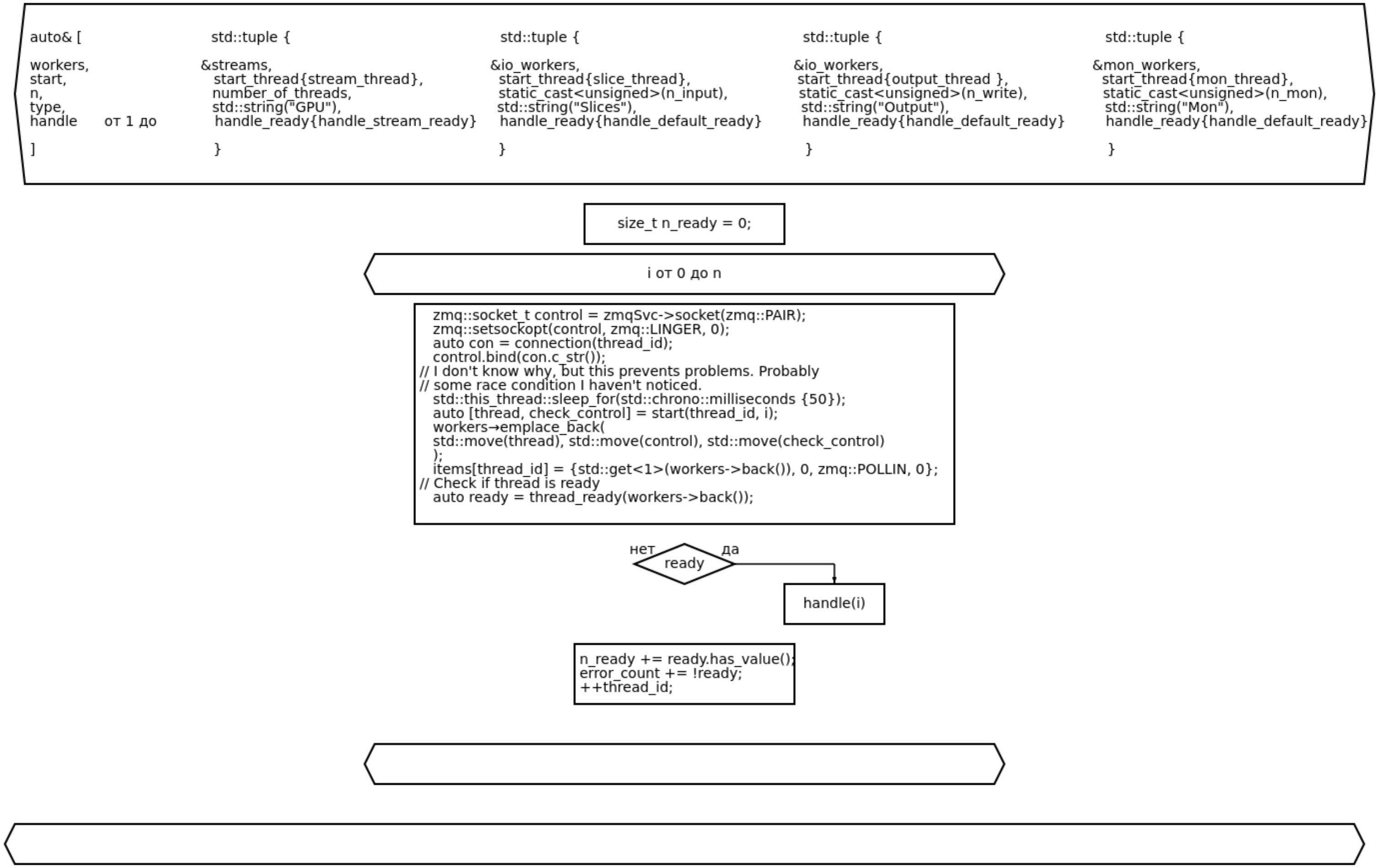
The command "START" is sent to each i-th worker from the control connections to start requesting slices

The "RUNNING" command is sent to the control connection

msg is equal "RESET"

It is noted that the input and output communications are complete, and that it is necessary to exit this cycle.

Thread work



```
auto& [                  std::tuple {                          std::tuple {                        std::tuple {                          std::tuple {

workers,                &streams,                             &io_workers,                        &io_workers,                          &mon_workers,
start,                    start_thread{stream_thread},          start_thread{slice_thread},         start_thread{output_thread },          start_thread{mon_thread},
n,                        number_of_threads,                    static_cast<unsigned>(n_input),     static_cast<unsigned>(n_write),        static_cast<unsigned>(n_mon),
type,                     std::string("GPU"),                   std::string("Slices"),              std::string("Output"),                 std::string("Mon"),
handle      от 1 до       handle_ready{handle_stream_ready}     handle_ready{handle_default_ready}  handle_ready{handle_default_ready}      handle_ready{handle_default_ready}

]                         }                                     }                                   }                                      }
```

```
size_t n_ready = 0;
```

```
   zmq::socket_t control = zmqSvc->socket(zmq::PAIR);
   zmq::setsockopt(control, zmq::LINGER, 0);
   auto con = connection(thread_id);
   control.bind(con.c_str());
// I don't know why, but this prevents problems. Probably
// some race condition I haven't noticed.
   std::this_thread::sleep_for(std::chrono::milliseconds {50});
   auto [thread, check_control] = start(thread_id, i);
   workers→emplace_back(
   std::move(thread), std::move(control), std::move(check_control)
   );
   items[thread_id] = {std::get<1>(workers->back()), 0, zmq::POLLIN, 0};
// Check if thread is ready
   auto ready = thread_ready(workers->back());
```

```
n_ready += ready.has_value();
error_count += !ready;
++thread_id;
```

Conclusions

- Allen has three types of reading data: mep input (or with mpi), mdf and binary input;
- Output data can be written to a file or sent via ZMQ;
- Branching conditions are often incomplete, the algorithm for an exhaustive number of cases is not explicitly specified.