FlukaIO: A library for particle exchange

David Siñuela Pastor `<david.sinuela.pastor@cern.ch>`

Table of Contents

Abstract

This document describes the FlukaIO protocol and discusses the implementation of the FlukaIO library. FlukaIO is a message passing protocol on top of TCP/IP, it was developed to enable the on-line communication of the Fluka Monte Carlo particle tracker and single particle trackers as SixTrack or Icosim.

1. Introduction

Two families of highly specialized simulators can be used to study the effects of a beam in an accelerator. One family is used to study the effects of the interaction of the beam with static matter, simulating individual particles and their interactions as they go though the matter. This kind of simulations are based on the Monte Carlo technique, after thousands of particles simulated the averaged results are very similar to those observed in reality. The second family, the single particle trackers, simulate the particles as they orbit throughout the accelerator ring. These kind of trackers simulate individual particles flying in the ring by applying them the optical functions of the ring elements.

For certain studies it is interesting to evaluate the long-term effects of the beam-matter interactions on the beam dynamics and vice versa. Therefore each kind of simulator plays an important role in these studies.

The current coupling infrastructure was designed to support a simulation where the beam circulates along the beam during thousands of turns. The trackig of the particles is performed by a single particle tracker as SixTrack or Icosim, and the Monte Carlo tracker Fluka is used to simulate one or more areas of the ring where the beam-matter interaction effects are important.

The test case simulation comprised a section of the accelerator with a moving scraper, a scraper is a piece of matter that moves into the beam pipe producing a reduction of the beam halo. With the infrastructure here described we were able to simulate the long term effects of the scraper on the beam stability. Two kind of outputs were obtained by these simulations: the pattern of lost particles along the ring and the energy deposition on the scraper geometry.

FlukaIO is the protocol and the library that provides the API for managing the communication between the trackers. The library implements a message passing protocol on top of TCP/IP and can be easily linked to other codes.

Both trackers must run at the same time and they will transport the particles in the portion of the accelerator that they are in charge to simulate. One of the two processes acts as the server and the other acts as the client. The disctinction only means that one process starts listening on one network port and the other starts the connection to that process. After the connection is established there is no disctinction in the way the server or the client work.

In the current implementation Fluka is set up to be the server and the single particle tracker acts as the client. By doing so we can easily change the single particle tracker keeping the server code (which is slightly more complicated) intact.

1.1. Getting the library

The library is distributed via the main CERN’s subversion repository [flukaiosvn] as a source code package that must be compiled by the user. There exist several tags on the repository marking stable releases, please try to always use one of these. Compatible versions (API and protocol) are kept under the same major version number. That is, applications using release version 2.0 can safely migrate to 2.1 and are encouraged to do so.

1.2. Compiling and Linking

The detailed instructions on how to compile FlukaIO can be found in the README file. The resulting library can be linked to a variety of languages, please refer to the documentation of your language to learn how to interoperate with C libraries. We have successfully linked the library to software written in Fortran 77, Fortran 90, C, C++ and Matlab.

For C/C++ code the header files `FlukaIO.h` and `FlukaIOServer.h` must be included. In the case of Fortran the interface is defined in `FortranFlukaIO.h` and `FortranFlukaIOServer.h`.

Four sample programs are provided in the `samples` folder, one client [../samples/ClientTest.c] and server [../samples/ServerTest.c] in C and one client [../samples/fclient.f] and server [../samples/fserver.f] in Fortran. These can be used as a starting point for developing new servers or clients.

2. User Guide

2.1. Execution flow

In our setup both processes start executing aproximately at the same time. Fluka takes the role of the server and starts listening on a network port and the single particle tracker connects to the Fluka server.

A handshake takes place when the communication is started and both processes tell the other the version of the protocol they implement. If the major version of the protocol differs they end the connection and an error is reported.

Once the two processes have succesfully established the connection the simulation starts on the single particle tracker.

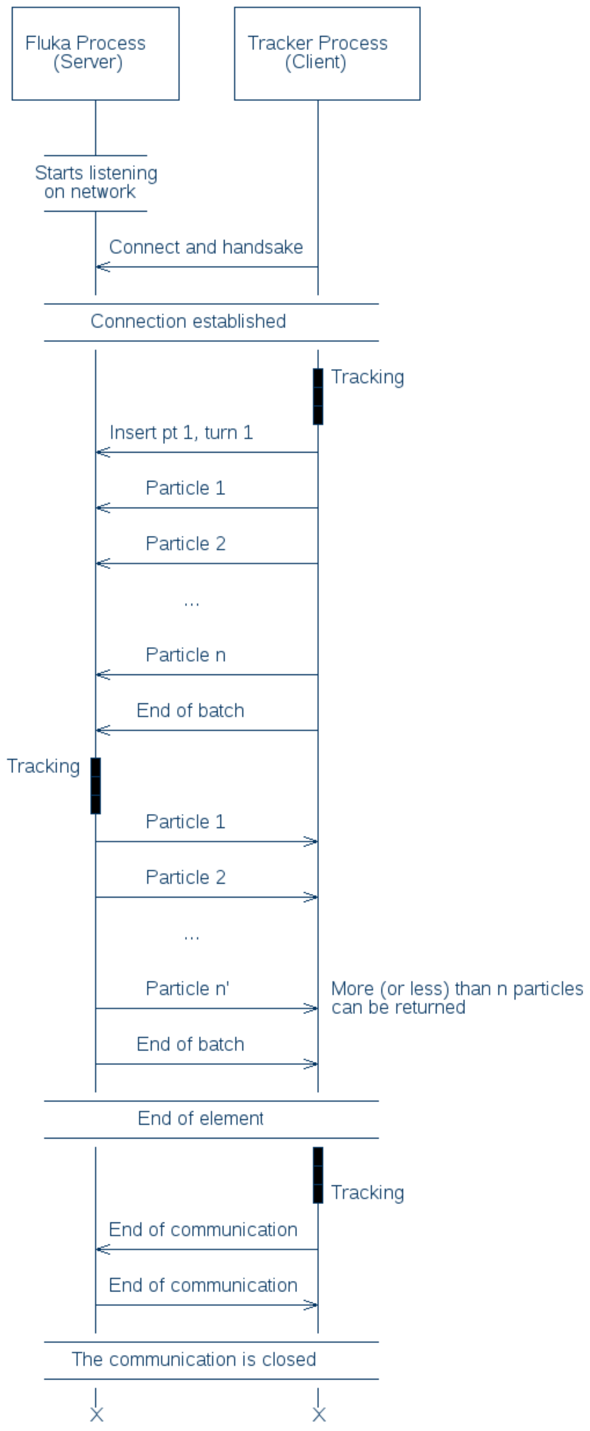
The particles are inserted and tracked along the lattice. The simulation continues as normal until it reaches an element in the lattice labelled as a Fluka element. At this point the simulation has to continue in Fluka, therefore all the particles have to be sent to the Fluka process.

Before sending the particles the single particle tracker must tell Fluka where in space and time the particles have to be inserted. For that purpose the SPT sends an Insertion PoinT message (IPT) which the lattice element number and current turn. There must be a convention for Fluka to know which element number corresponds to which position in the Fluka space and the turn number can be used to compute the time shift in cases where it is important for the simulation.

The particles are then sent to the Fluka process in individual messages followed an End Of Bunch message (EOB). The SPT then remains waiting for incoming network messages. Fluka tracks the particles one by one and sends them back to the SPT if they reach the exit region of the geometry. Once all the particles received by Fluka are sent or lost Fluka sends the end of bunch message to the SPT. Once the SPT receives the EOB message knows all the remaining particles have been received. The simulation continues in the single particle tracker from the next element.

The simulation will finish when the number of turns requested is exhausted or because all the particles were lost. The SPT will send an End of Communication message (EOC) and Fluka will reply with and EOC message to notify that the message was received and the process is going to finish.

The following diagram shows the data flow between the two processes:



Notice that the number of particles is always variable, not only particles can be lost in the accelerator tracker or in the Fluka section but also new particles can be generated in Fluka. Moreover, this processes can alter the order of the particles sent back to the original process.

2.2. Types of Messages

Particle Message

A message containing the data of a particle:

- id: particle id

- gen: particle generation
- weigth: statistical weigth
- x: position in cm
- y: position in cm
- z: position in cm
- tx: director cosine along the x axis
- ty: director cosine along the y axis
- tz: director cosine along the z axis
- aa: mass number
- zz: ion charge
- m: rest mass in GeV/c^2
- p: momentum in GeV/c
- t: time shift with respect to reference particle

End of batch

A message specifying that all the particles of the current turn/batch have been processed and sent.

End of computation

Tells the other side that is ready to finish the simulation, the other end should then reply with an end of computation message and finish the process.

Insertion point

Usually sent from the tracker to the Fluka process, it contains the turn number and the elemenet number, instructing Fluka where the upcoming batch of particles has to be inserted.

As explained before, certain convention must exist among Fluka and the SPT, Fluka must be able to know which position in the space corresponds to each of the element identifiers.

2.3. API

The API is splitted in two major sections: the client API and the server API. The client API is used by both the client and the server, while the server API contains only those functions needed to start the server. Two equivalent APIs are provided, one in C and the other in Fortran. Since Fluka is implemented in Fortran the Fortran interface is needed to use FlukaIO.

Note. All the API functions return -1 in case of error, the return value of every call should always be checked.

Client API

Create the connection object

C.

```
flukaio_connection_t *flukaio_connect(const char *host, int port);
```

Fortran.

```
integer function ntconnect(host, port)
character(len=*) :: host
integer :: port
```

Connect to a FlukaIO server through host and port

This call connects the current process to a FlukaIO server and returns the connection information. The connection should be opened at the initialization phase of the program and closed before finishing the process.

C.

```
flukaio_connection_t *flukaio_connect(flukaio_connection_t *conn, const char *host, int
```

It returns a pointer to a new flukaio_connection_t structure that must be passed to all subsequent calls to the functions of the API.

As it is usually the first call to flukaio in the client code it is recommended to use it like follows:

C.

```
flukaio_connection_t *conn = flukaio_connect(flukaio_conn(), "host", 1234);
```

Fortran.

```
integer function ntconnect(host, port)
character(len=*) :: host
integer :: port
```

In the case of Fortran the function returns an integer with the connection identifier that should be passed to all the following calls to the API. If the identifier is smaller than 0 an error occurred.

Disconnect

Closes the connection to the server and releases its resources. The connection memory is freed inside the call.

C.

```
void flukaio_disconnect(flukaio_connection_t *conn);
```

Fortran.

```
integer function ntend(cid)
```

```
        integer :: cid
```

Configure connection read timeout

Two functions are provided to read incoming particles: read_message and wait_message, the latter blocks the process until a message is received or an error occurred. It can happen that the other end of the communication never sends any message and therefore the process would keep waiting forever. This timeout specifies the maximum amount of time a wait_message call should wait.

C.

```
int connection_set_read_timeout(flukaio_connection_t *conn, long timeout);
```

Fortran.

```
integer function ntrtimeout(cid, seconds)
        integer :: cid
        integer :: seconds
```

Configure connection write timeout

C.

```
int connection_set_write_timeout(flukaio_connection_t *conn, long timeout);
```

Fortran.

```
integer function ntwtimeout(cid, seconds)
        integer :: cid
        integer :: seconds
```

Send a particle

This call creates a particle message with the particle information provided and saves it in the outgoing buffer. The particle information can be safely modified immediately after calling this function.

In the case of C a particle_info_t pointer with the particle data must be passed and in the case of Fortran all the particle properties must be passed as arguments.

C.

```
ssize_t
flukaio_send_particle(flukaio_connection_t *conn, const particle_info_t *part);
```

Fortran.

```
integer function ntsendp(cid, id, gen, wgt, x, y, z, tx, ty, tz, aa, zz, m, p, t)
        integer :: cid
        integer(kind=4) :: id
        integer(kind=4) :: gen
        real(kind=8) :: wgt
        real(kind=8) :: x
        real(kind=8) :: y
        real(kind=8) :: z
        real(kind=8) :: tx
        real(kind=8) :: ty
        real(kind=8) :: tz
        integer(kind=2) :: aa
        integer(kind=2) :: zz
        real(kind=8) :: m
        real(kind=8) :: p
        real(kind=8) :: t
```

Send End Of Batch message

End of turn message tells the other end that all the particles in the current batch were sent.

C.

```
ssize_t flukaio_send_eob(flukaio_connection_t *conn);
```

Fortran.

```
integer function ntsendeob(cid)
        integer :: cid
```

Send Insertion Point

Tells the other end where the following particles should be inserted. It sends an integer with the insertion point identifier and an integer with the turn number.

C.

```
ssize_t flukaio_send_ipt(flukaio_connection_t *conn, uint32_t turn, uint32_t ipt);
```

Fortran.

```
integer function ntsendipt(cid, ipt, turn)
        integer :: cid
        integer(kind=4) :: ipt
        integer(kind=4) :: turn
```

Send End Of Computation message

Informs the other end that the simulation has finished.

C.

```
ssize_t flukaio_send_eoc(flukaio_connection_t *conn);
```

Fortran.

```
integer function ntsendeoc(cid)
        integer :: cid
```

Read a message (non-blocking)

Reads messages from the incoming buffer, if nothing was read from the network returns -1, otherwise it returns the size of the read message.

C.

```
ssize_t
flukaio_receive_message(flukaio_connection_t *conn, flukaio_message_t *msg);
```

Stores the incoming message in msg. The type of the message can be inspected by looking at the msg.type field.

Fortran.

```
integer function ntrecv(cid, mtype, id, gen, wgt, x, y, z, tx, ty, tz, aa, zz, m, p, t)
        integer :: cid
        integer :: mtype
        integer(kind=4) :: id
        integer(kind=4) :: gen
        real(kind=8) :: wgt
        real(kind=8) :: x
        real(kind=8) :: y
        real(kind=8) :: z
        real(kind=8) :: tx
        real(kind=8) :: ty
        real(kind=8) :: tz
        integer(kind=2) :: aa
        integer(kind=2) :: zz
        real(kind=8) :: m
        real(kind=8) :: p
        real(kind=8) :: t
```

Stores the incoming message in the arguments, the type of the message is stored in mtype.

Read a message (blocking)

If no message was received it blocks the application until a message arrives. Returns -1 if an error occurred (the other end disconnected or the read operation timed out).

C.

```
ssize_t
flukaio_wait_message(flukaio_connection_t *conn, flukaio_message_t *msg);
```

Stores the incoming message in msg. The type of the message can be inspected by looking at the msg.type field.

Fortran.

```
integer function ntwait(cid, mtype, id, gen, wgt, x, y, z, tx, ty, tz, aa, zz, m, p, t)
        integer :: cid
        integer :: mtype
        integer(kind=4) :: id
        integer(kind=4) :: gen
        real(kind=8) :: wgt
        real(kind=8) :: x
        real(kind=8) :: y
        real(kind=8) :: z
        real(kind=8) :: tx
        real(kind=8) :: ty
        real(kind=8) :: tz
        integer(kind=2) :: aa
        integer(kind=2) :: zz
        real(kind=8) :: m
        real(kind=8) :: p
        real(kind=8) :: t
```

Stores the incoming message in the arguments, the type of the message is stored in mtype.

Server API

Create server object.

C.

```
flukaio_server_t* flukaio_server_create();
```

This creates a structure holding all server attributes. This structure has to be used to call all other server operations.

Fortran.

```
integer function ntserver()
```

Returns the server identified to be used to all subsequetn calls.

Setup a server listening on port

Returns the assigned port or -1 if error. The desired port can be specified in the port argument, if 0 is provided the port will be randomly assigned by the operating system.

C.

```
int flukaio_server_start(flukaio_server_t* server, unsigned int port);
```

Fortran.

```
integer function ntstart(sid, port)
        integer :: sid
        integer :: port
```

The first variable is the serverid obtained with ntserver().

Accept connection

Waits for a client connection and returns a structure with the connection information.

C.

```
flukaio_connection_t *flukaio_server_accept(flukaio_server_t* server);
```

This returns the incoming connection object that can be used with the client API.

Fortran.

```
integer function ntaccept(sid)
        integer :: sid
```

Returns a connection identifier if there was a connection, otherwise -1.

Shutdown server

Frees all server resources, stops listening for connections.

C.

```
int flukaio_server_shutdown(flukaio_server_t* server);
```

Fortran.

```
integer function ntshwnd()
        integer :: sid
```

Get server port

Returns the port number on which the server is listening.

C.

```
int flukaio_server_get_port(flukaio_server_t* server);
```

Fortran.

```
integer function ntgetport(sid)
        integer :: sid
```

2.4. Troubleshooting

The source code is distributed with a complete test suite under the folder “tests”. In order to run the tests the CppUTest [1] test harness must be correctly installed. If CppUTest is installed the test suite is executed and a report is generated each time the code is compiled using make.

If something is failing please run the test suite and check that everything is working as expected. If that is the case try to sniff the network connection with `tcpdump`.

3. Technical documentation

3.1. Main FlukaIO features

Although FlukaIO is meant to link Fluka to single particle tracking codes it has been designed to be extensible, a small list of features follows:

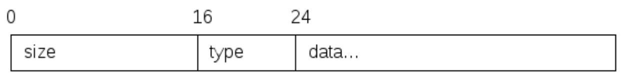
- Two equivalent APIs, the native C API and a compatibility layer for Fortran codes.
- TCP/IP communication: enabling client/server architectures

- Variable size messages to use minimum bandwidth
- Extensible list of message types, implemented now: particle, end of turn, end of computation and insertion point
- Buffered input and output to improve performance
- Modular Object-Oriented architecture
- Complete test-suite

3.2. Protocol Description

Message format

The messages have this common header format:



The header of the message contains the total size of the message in bytes (unsigned integer: 2 bytes) and the type of message (unsigned integer: 1 byte), 3 bytes in total.

Types of Messages

The type field in a message can have one of these values:

- **0 = ERR**: Error (used internally)
- **1 = PART**: A message containing a particle
- **2 = TURN**: End of turn message
- **3 = END**: End of computation
- **4 = CONF**: Configuration, reserved
- **5 = IPT**: Insertion point

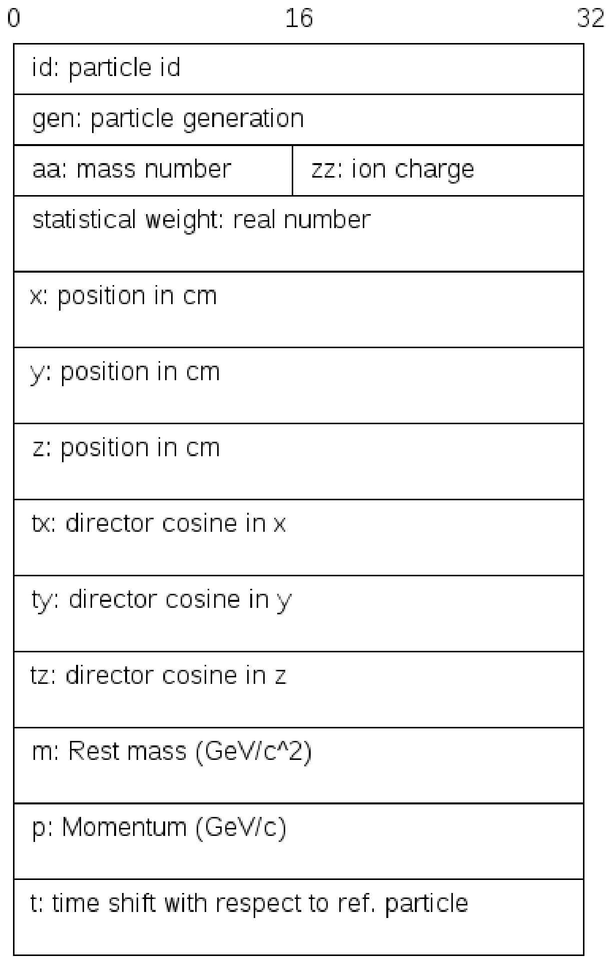
The message types and their meanings are not strictly defined, interpretation depends on the application. Following definitions are just a guideline.

Error message

When the reads a message of type ERR means that the incoming message was not recognised. It is used internally, but it is also the type of message that the functions read_message and wait_message return when there is an error (in addition to returning -1).

Particle message

Contains a particle in the data section, for more details read the header file `include/ParticleInfo.h`:



End Of Batch message

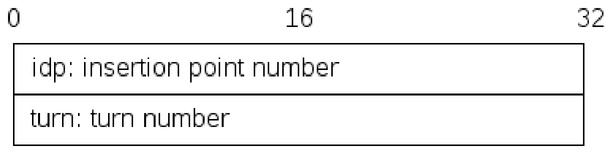
Empty message communicating that all the particles were sent.

End Of Computation message

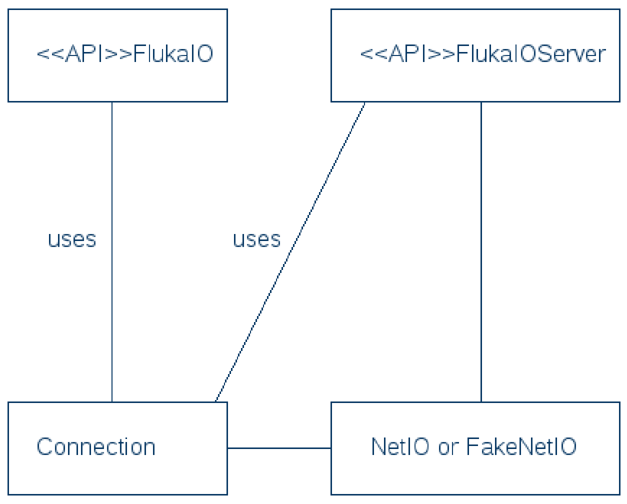
Informs the other end that computation was finished and must disconnect. Doesn't contain any data.

Insertion point

Contains a 4 bytes unsigned integer with the insertion point number, and a 4 bytes unsigned integer with the turn number.



3.3. Logical Structure



The library offers two APIs: FlukaIO and FlukaIOServer, a client application would normally use only FlukaIO while a server application would use both.

When a connection is established the FlukaIO library returns a Connection object (or connection id in the Fortran interface) which holds all the data needed during a communication session.

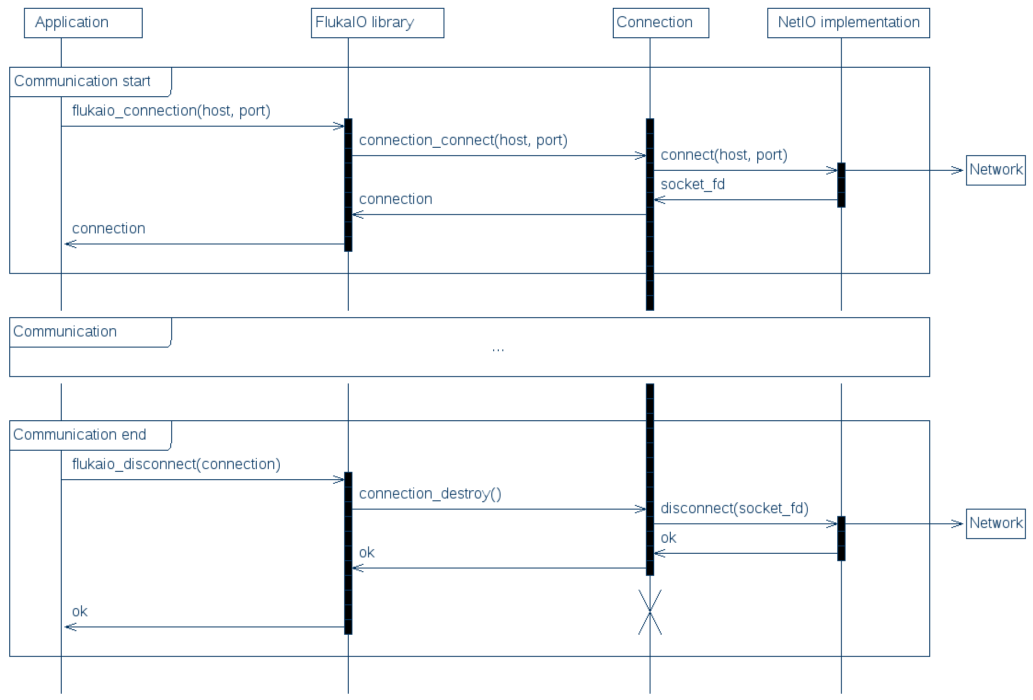
All the incoming and outgoing messages are sent through a Connection object, which stores them in the appropriate buffer if needed. When enough data is stored in the output buffer or a flush operation is requested the connection object sends all the contents of the output buffer and erases them from memory.

All network operations are defined as function pointers in the Connection struct. This is used to replace all operating system calls during tests avoiding hitting the network and providing reliable and replicable test results.

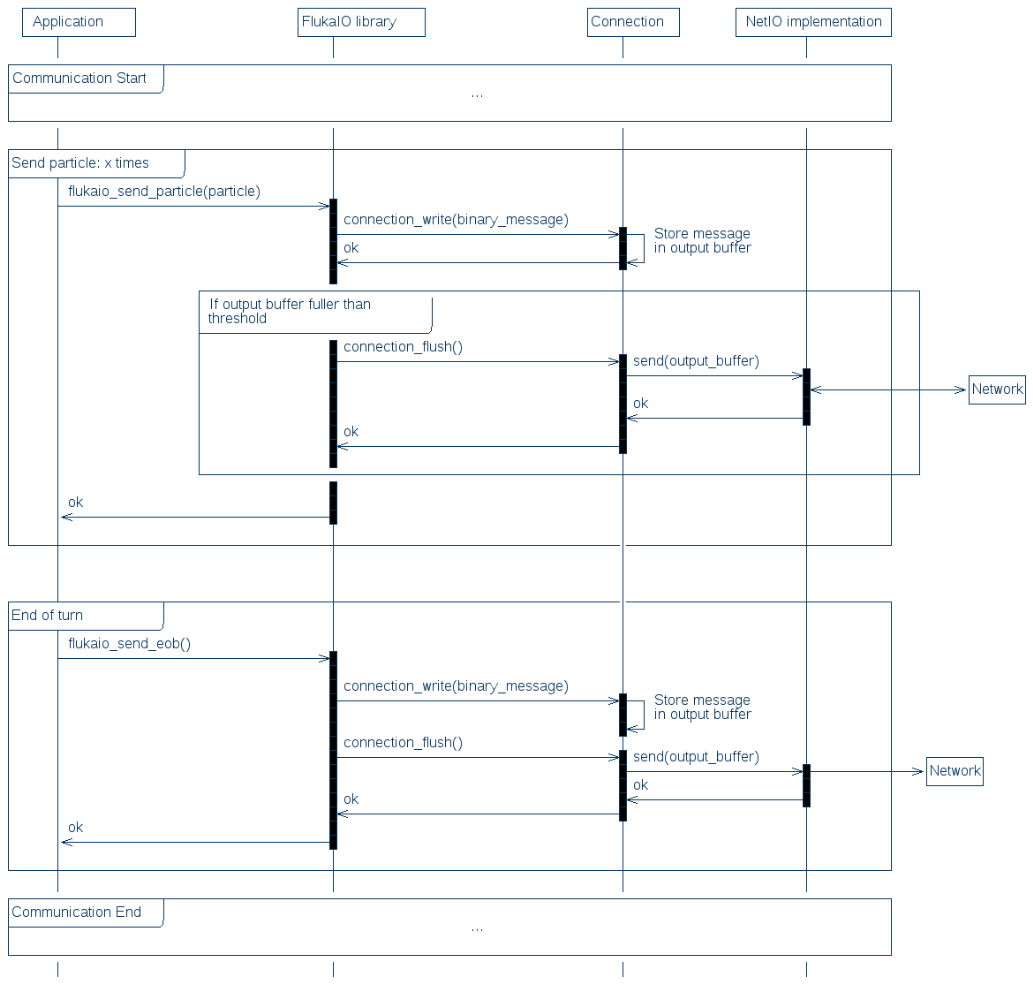
3.4. Data flow diagrams

In this section of the document an overview of the data flow is provided showing the collaboration between the different components of the library.

Starting and ending communication



Sending messages



End Of Computation messages are not shown in the diagram because they follow the same execution flow as the End Of Batch in the sense that they are synchronous as well.

3.5. How to

Add new fields to particles

If the partile structure is to be modified this approach is recommended:

1. Add the field to particle_info_t (`include/ParticleInfo.h`)
2. Update `PARTICLES_EQUAL` helpers in tests (`tests/FlukaIOTest.cpp` and `tests/FortranFlukaIO.cpp`)
3. Run the tests and correct the code if they fail.

The Fortran interface will need more work than the C interface because it exposes each individual field in the function signatures. Because of this the Fortran samples have to be updated to reflect the new interface. The C samples should still be working after the change.

Add new message types

New message types can be introduced by adding the message id to the enum `message_type_t` found in `include/message.h` and modifying the FlukaIO logic. In case more variable size messages have to be implemented the data union field in flukaio_message_t structure can be modified to store the new data.

4. Credits

- David Siñuela Pastor <david.sinuela.pastor@cern.ch [mailto:david.sinuela.pastor@cern.ch]>
- Alessio Mereghetti <Alessio.Mereghetti@cern.ch [mailto:Alessio.Mereghetti@cern.ch]>
- Vasilis Vlachoudis <Vasilis.Vlachoudis@cern.ch [mailto:Vasilis.Vlachoudis@cern.ch]>
- Francesco Cerutti <Francesco.Cerutti@cern.ch [mailto:Francesco.Cerutti@cern.ch]>
- Joel Francois Clivaz <joel.francois.clivaz@cern.ch [mailto:joel.francois.clivaz@cern.ch]>

References

[flukaiosvn] . *FlukaIO subversion repository* . http://svnweb.cern.ch/world/wsvn/FlukaIO

[cpputest] . *CppUTest*. http://sourceforge.net/projects/cpputest/

[flukadesc] . G. Battistoni, S. Muraro, P.R. Sala, F. Cerutti, A. Ferrari, S. Roesler, A. Fasso`, J. Ranft . *The FLUKA code: Description and benchmarking* . Proceedings of the Hadronic Shower Simulation Workshop 2006, Fermilab 6—8 September 2006, M. Albrow, R. Raja eds., AIP Conference Proceeding 896, 31-49, (2007) [flukacode] . A. Fasso`, A. Ferrari, J. Ranft, and P.R. Sala . *FLUKA: a multi-particle transport code* . CERN-2005-10 (2005), INFN/TC_05/11, SLAC-R-773

[sixtrack] . F. Schmidt . *SixTrack, User's Reference Manual* . CERN SL/94-56 (AP), 1994.

[icosim] . Holden, N . *Development of the ICOSIM Program and Application to Magnetised Collimators in the LHC* . CERN-AB-Note-2008-054, 2008