Jpp I/O

M. de Jong

29/08/2019

Interfaces (1/6)

- Access to a device (e.g. file)

```
class JLANG::JAccessible {
    virtual bool is_open() = 0;
    virtual void open(const char* file_name) = 0;
    virtual void close() = 0;
};
```

Interfaces (2/6)

- Rewinding of device

```
template<class T>
class JLANG::JRewindable {

    virtual void rewind() = 0;

};
```

Interfaces (3/6)

- Input of objects¶

```
template<class T>
class JLANG::JObjectIterator {

    typedef JPointer<T>  pointer_type;

    virtual bool hasNext() = 0;
    virtual pointer_type next() = 0;

    JObjectIterator& operator>>(JObjectOutput<T>&);
};
```

return type

input iteration

copy all data types in input

¶ Template parameter may be a type list.

Interfaces (4/6)

- Extensions

```
template<class T>
class JAccessibleObjectIterator :
    public virtual JObjectIterator<T>,
    public virtual JAccessible
{};
```

```
template<class T>
class JRewindableObjectIterator :
    public virtual JObjectIterator<T>,
    public virtual JRewindable    <T>
{};
```

Interfaces (5/6)

- Output of objects¶

```
template<class T>
class JLANG::JObjectOutput {
    virtual bool put(const T& object) = 0;

    JObjectOutput& operator<<(JObjectIterator<T>&);
};
```

} output

} copy all data types in output

¶ Template parameter may be a type list.

Interfaces (6/6)

- Extensions

```
template<class T>
class JAccessibleObjectOutput :
    public virtual JObjectOutput<T>,
    public virtual JAccessible
{};
```

File formats (1/2)

file name extension	format	data types
.root	ROOT	Monte Carlo DAQ trigger parameters meta information
.detx	ASCII	detector calibration
.datx	binary	
.det	ASCII	
.evt	ASCII	Monte Carlo
.gz	gzip	Monte Carlo detector calibration
.txt	ASCII	trigger parameters PMT parameters
.dat	binary	DAQ

File formats (2/2)

- ROOT I/O
  i. TTree events, time slices, etc.
  ii. TObject meta data, trigger parameters, etc.
- TTree configured using data structure JTreeParameters
  - TTree/Branch name, split level, compression level, etc.
  - availability of method
    JTreeParameters getTreeParameters(JType<data type>);
    instructs Jpp I/O to use corresponding ROOT TTree
- All ROOT TTree's for KM3NeT (and Antares) data types are defined in include file JSupport.hh
  - preserves modularity of auxiliary classes

Implementations (1/12)

- File reading

```
template<class T>
class JSUPPORT::JFileScanner :
    public JAccessibleObjectReader<T>          // implements JObjectIterator<T> via pointer
{
    virtual void open(const char* file_name);  // sets pointer to JObjectIterator<T>
                                               // based on file name extension
};
```

Implementations (2/12)

- Multiple file reading

```
template<class T>
class JSUPPORT::JMultipleFileScanner :
    public JRewindableObjectIterator<T>   // implements
{
    virtual bool hasNext();               // re-implemented to open new file if necessary

private:
    JFileScanner<T> scanner;              // worker object
};
```

Implementations (3/12)

- Parallel file reading

```
template<type list>                                  // list of data types read one-to-one in parallel
class JSUPPORT::JParallelFileScanner :
    public JRewindableObjectIterator<type list>   // implements
{
    // consistent covariant return type with each JObjectIterator<>
    typedef JMultiPointer<type list>  multi_pointer_type;

    virtual multi_pointer_type next();               // linked pointers to objects according type list
};
```

Implementations (4/12)

- Monte Carlo file reading (e.g. output of JTriggerEfficiency)

```
template<>                                          // optional template arguments
class JSUPPORT::JTriggeredFileScanner :
    public JParallelFileScanner<JDAQEvent, ...>
{
    typedef JMultiPointer<JDAQEvent, Evt, ...>  multi_pointer_type;

    virtual multi_pointer_type next();          // linked pointers to same Monte Carlo and DAQ
    {}                                          // event and optionally also other events
};
```

Implementations (5/12)

- ROOT TTree reading¶

```
template<class T>
class JSUPPORT::JTreeScanner :
    public JRewindableObjectIterator<T>        // implements
{
    virtual pointer_type next();               // unordered iteration

    [const_]iterator begin();
    [const_]iterator end();
    [const_]iterator rbegin();
    [const_]iterator rend();
};
```

¶ TTree parameters are obtained using method getTreeParameters().

Implementations (6/12)

- ROOT TTree reading

```
template<class T, class JEvaluator_t>          // optional template argument for sorting
class JSUPPORT::JTreeScanner :
    public JRewindableObjectIterator<T>        // implements
{
    virtual pointer_type next();               // ordered iteration

    Long64_t find(..) const;                   // find nearest entry

    [const_]iterator begin();                  // first  entry according evaluator
    [const_]iterator end();
    [const_]iterator rbegin();                 // last   entry according evaluator
    [const_]iterator rend();
};
```

Implementations (7/12)

- ControlHost reading as client[¶]

```
template<class T>
class JNET::JControlHostObjectIterator :
    public JObjectIterator<T>                    // implements
{
    virtual bool hasNext();                      // checks for new data within timeout
};
```

[¶] ControlHost tag is obtained using method getTag().

Implementations (8/12)

- ControlHost reading as server¶

```
template<class T>
class JNET::JLigierObjectIterator :
    public JObjectIterator<T>                    // implements
{
    JLigierObjectIterator(int port);             // server port

    virtual bool hasNext();                      // checks for new data from new connection
};
```

Implementations (9/12)

- Printing

```
template<class T>
class JLANG::JStreamObjectOutput :
    public JObjectOutput<T>                                  // implements
{
    JStreamObjectOutput(std::ostream&       out,             // output stream
                        const std::string& sep);             // text between consecutive objects
};
```

Implementations (10/12)

- File writing

```
template<class T>
class JSUPPORT::JFileRecorder :
    public JAccessibleObjectWriter<T>,          // implements JObjectOutput<T> via pointer
{
    virtual void open(const char* file_name);   // sets pointer to JObjectOutput<T>
                                                // based on file name extension
};
```

Implementations (11/12)

- ControlHost writing[¶]

```
template<class T>
class JNET::JControlHostObjectOutput :
    public JObjectOutput<T>                    // implements
{
    virtual bool put(const T& object);         // sends data
};
```

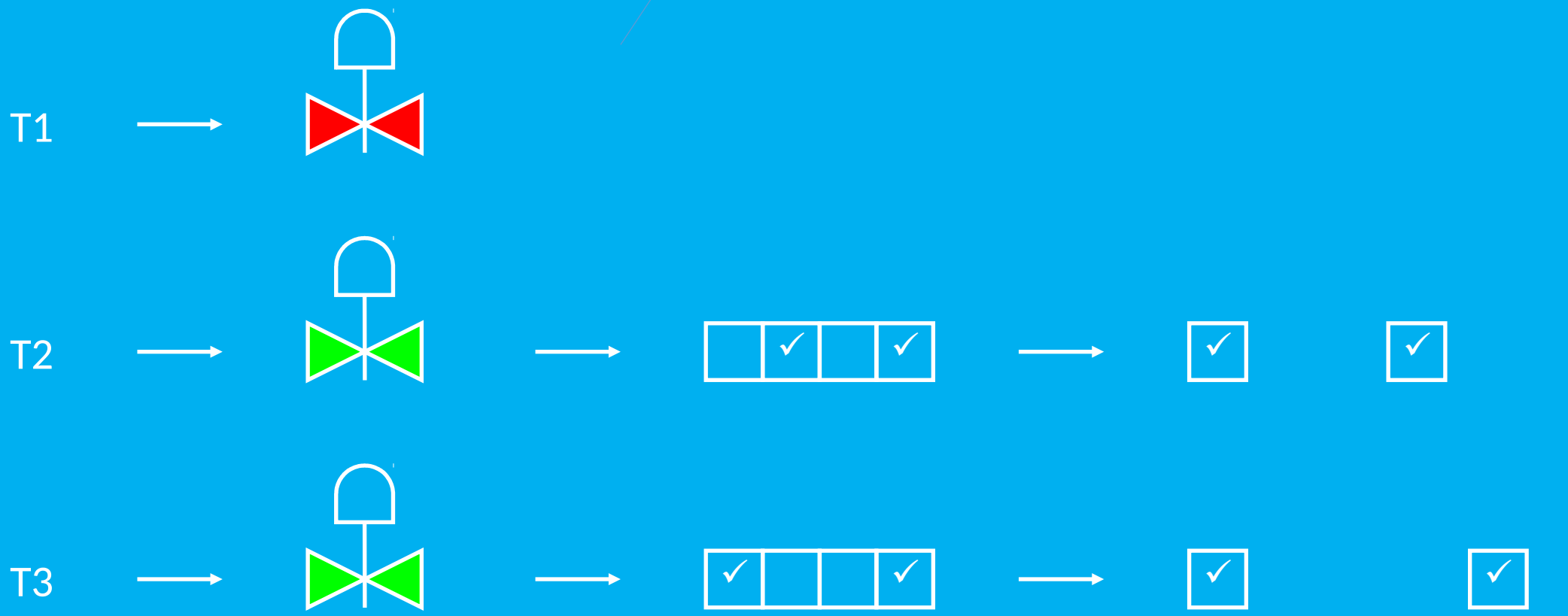
[¶] ControlHost tag is obtained using method getTag().

Implementations (12/12)

- Fast access to summary data (singles rates, PMT status, etc.)

```
class JSUPPORT::JSummaryFileRouter
{
    JSummaryFileRouter(const std::string& file_name,      // file name
                       const double       rate_Hz);       // default rate

    void update(const JDAQHeader& header);                // update internal buffer(s)
                                                          // for given event

    const JDAQSummaryFrame& getSummaryFrame(const JDAQModuleIdentifier& module);
};
```

Pipes (1/4)


data type
type valve
object selector
throughput regulator
T1
T2
T3
time

Pipes (2/4)

- Basic pipe

```
template<class T>
class JLANG::JPipe :
    public JObjectIterator<T>              // implements
{
    virtual bool hasNext();                // re-implements

    JValve<T>                valve;        // open/close
    JObjectSelector<T>       selector;     // object selection
    JRegulator               regulator;    // regulate throughput
};
```

Pipes (3/4)

- Auxiliary class for multiple sequential pipes

```
template<class T, int N>
class JLANG::JMultiPipe :
    public JPipe<T>
{
    static JSinglePointer< JMultiPipe<T> > pipe;     // common usage
};
```

Pipes (4/4)

- Possible syntax

```
JObjectIterator<T>()   | JValve<T>()              | JObjectOutput<T>();  // select data type

JObjectIterator<T>()   | JObjectSelector<T>()     | JObjectOutput<T>();  // select objects

JObjectIterator<T>()   | JRegulator()             | JObjectOutput<T>();  // regulate throughput
```

- or any combinations hereof

JPrint

- Implementation

```
JMultipleFileScanner   <JAllTypes_t> inputFile;
JStreamObjectOutput <JAllTypes_t> out(cout);

inputFile | JValve<>(selection) | out;
```

- Syntax

  JPrint -f <file name> -C -JDAQ\.*
  will print everything but DAQ data

JConvert

- Implementation

```
JMultipleFileScanner <JAllTypes_t> inputFile;
JFileRecorder        <JAllTypes_t> outputFile;

inputFile | JValve<>(selection) | outputFile;
```

- Syntax

  JConvert -f <file name> -C -\.* -C +JDAQEvent
  will only write DAQ events to output file (e.g. to save disk space)

JRegurgitate

- Implementation

```
JMultipleFileScanner          <JAllTypes_t> inputFile;
JControlHostObjectOutput<JAllTypes_t> outputFile;

inputFile | JValve<>(selection) | regulator | outputFile;
```

- Syntax

  JRegurgitate -f <file name> -H <host name> -C JDAQEvent -R <rate Hz>

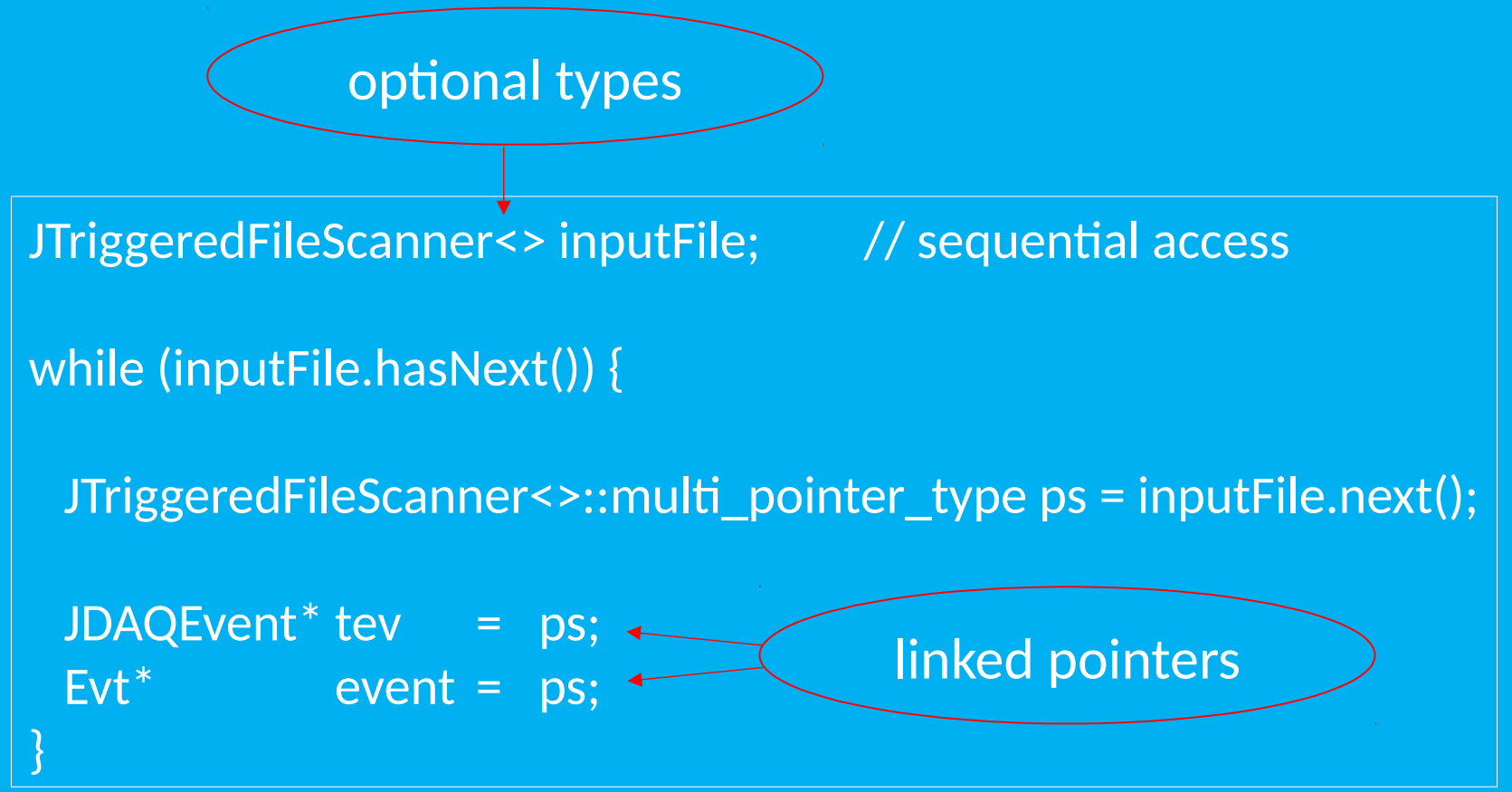
  will read DAQ events from the file with given name and sends them to the JLigier on given host at specified rate

Example 1

data type to read

```
JMultipleFileScanner<JDAQEvent> inputFile;      // sequential access

JTreeScanner<Evt> in(inputFile);                // direct access to Monte Carlo TTree

while (inputFile.hasNext()) {

  JDAQEvent* tev   =  inputFile.next();
  Evt*       event =  in.getEntry(tev->getCounter());

  const JTimeConverter converter(*event, *tev);

  Hit::t ≈ converter.getTime(JDAQHit);

  Hit::t = converter.getTime(JDAQHit, JCalibration);
}
```

Example 2


optional types
JTriggeredFileScanner<> inputFile; // sequential access
while (inputFile.hasNext()) {
JTriggeredFileScanner<>::multi_pointer_type ps = inputFile.next();
JDAQEvent* tev = ps;
Evt* event = ps;
}
linked pointers

Example 3


```
JMultipleFileScanner<> inputFile;      // simple file list

JTreeScanner<JDAQEvent, JDAQEvaluator> in(inputFile);

while (in.hasNext()) {

   JDAQEvent* tev = in.next();          // time ordered
}
```

Example 4

```
struct A {
   A() {}

   ClassDef(A,1);       // needed by ROOT I/O
};

// Existence of following method makes ROOT-Jpp
// use a TTree for I/O of A

inline JTreeParameters getTreeParameters(JType<A>)
{
   return JTreeParameters(“A", “This is A::a", “a", 0);
}
```

Example 5

```
JMultipleFileScanner<A>   inputFile;
JFileRecorder<typelist>   outputFile;      // type list includes A

while (inputFile.hasNext()) {

   A* a = inputFile.next();

   // possibly modify a

    outputFile.put(*a);
}

JMultipleFileScanner<JRemove<typelist, A>::typelist> io(inputFile);

io >> outputFile;                          // copy everything but A
```

Example 6

```
JMultipleFileScanner<JDAQEvent> inputFile;      // sequential access

JTreeScanner<JDAQSummaryslice, JDAQEvaluator> in(inputFile);

while (inputFile.hasNext()) {

   JDAQEvent*           tev   =   inputFile.next();

   JDAQSummaryslice*  slice =   in.find(*tev);      // same frame index
}
```