KM3NeT

Jpp Event-Weighting Framework

A consistent event-weighting scheme for MC-combinations

Bouke Jung, Maarten de Jong

Universiteit van Amsterdam

Nikhef

In a nutshell

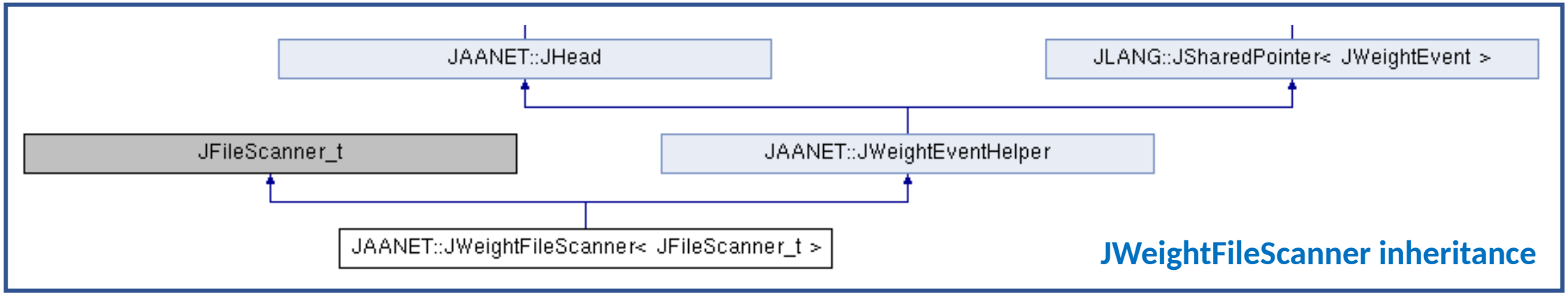
- Running a program over many different (types) of MC files is possible...
  But not always practical or intuitive at the moment:
  - Requires run-time classification and combination of (many different) file-headers
  - Requires run-time assignment of correct (combined) weights to each event

- A general Jpp-framework has been made available to accomplish the above and avoid inconsistensies
  - Inspired by Maarten's JHeadSet application and Alba and Alfonso's prior work

Software Structure

- Top datastructure called "JWeightFileScannerSet"
  - Ordered set of JWeightFileScanner objects
  - Ordering guaranteed by JHead::less operator

- A JWeightFileScanner object consists of:
  1. A filescanner which can be used to loop over all events corresponding to one MC-header type
  2. An event-weight helper, which does the bookkeeping on the combined header info and assigns the correct weight to each event correspondingly

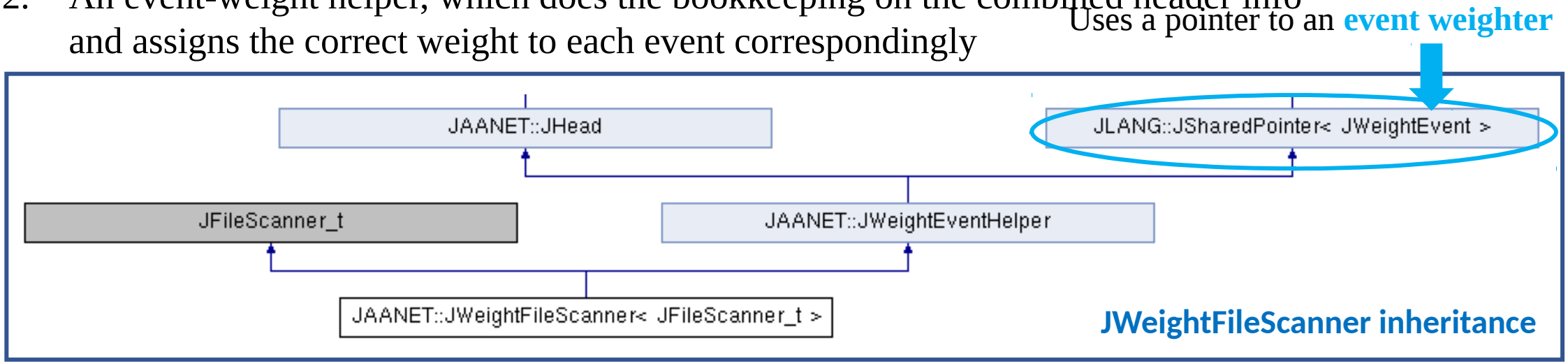


JWeightFileScanner inheritance

Software Structure

- Top datastructure called "JWeightFileScannerSet"
  - Ordered set of JWeightFileScanner objects
  - Ordering guaranteed by JHead::less operator

- A JWeightFileScanner object consists of:
  1. A filescanner which can be used to loop over all events corresponding to one MC-header type
  2. An event-weight helper, which does the bookkeeping on the combined header info and assigns the correct weight to each event correspondingly



JWeightEvent

- Simple interface containing three methods
  - A configure-function, which sets the global event weight given a (combined) header
  - A check-method to evaluate consistency of a header with this event weighter
  - A getWeight-function which returns the correct weight (in Hz), given a MC-event
- So far, three implementations
  1. Mupage weighter
  2. GSeaGen weighter
  3. KM3BUU weighter

```
struct JWeightEvent :
  public JClonable<JWeightEvent>
{
  /**
   * Virtual destructor.
   */
  virtual ~JWeightEvent()
  {}


  /**
   * Configuration.
   *
   * \param  header            header
   */
  virtual void configure(const JHead& header) = 0;


  /**
   * Check whether header is consistent with this event weighter.
   */
  virtual bool check(const JHead& head) const = 0;


  /**
   * Get weight of given event.
   *
   * \param  evt               event
   * \return                   weight       [Hz]
   */
  virtual double getWeight(const Evt& evt) const = 0;


  /**
   * Get weight of given event.
   *
   * \param  evt               event
   * \param  flux              neutrino flux [m^-2 s^-1 sr^-1 GeV^-1]
   * \return                   weight       [Hz]
   */
  virtual double getWeight(const Evt&   evt,
                           const double flux) const
  {
    return getWeight(evt);
  }
};
```

JWeightEvent

- Simple interface containing three methods
  - A configure-function, which sets the global event weight given a (combined) header
  - A check-method to evaluate consistency of a header with this event weighter
  - A getWeight-function which returns the correct weight (in Hz), given a MC-event

```
struct JWeightEvent :
  public JClonable<JWeightEvent>
{
  /**
   * Virtual destructor.
   */
  virtual ~JWeightEvent()
  {}


  /**
   * Configuration.
   *
   * \param  header            header
   */
  virtual void configure(const JHead& header) = 0;


  /**
   * Check whether header is consistent with this event weighter.
   */
  virtual bool check(const JHead& head) const = 0;


  /**
   * Get weight of given event.
```

- So far, three implementations
  1. Mupage weighter
  2. GSeaGen weighter
  3. KM3BUU weighter

Function-overloads for user-specifiable flux

```
/**
 * Get weight of given event.
 *
 * \param  evt              event
 * \param  flux             neutrino flux [m^-2 s^-1 sr^-1 GeV^-1]
 * \return                  weight       [Hz]
 */
virtual double getWeight(const Evt&   evt,
                         const double flux) const override
{
  if (evt.w.size() > 1 && !(evt.w[1] < 0.0 || flux < 0.0)) {

    return W * evt.w[1] * flux;

  } else {

    if (evt.w.size() < 2) {
      THROW(JIndexOutOfRange, "JWeightKM3BUU::getWeight(): w2-weight is empty.");
    } else {
      THROW(JValueOutOfRange, "JWeightKM3BUU::getWeight(): w2-weight or flux is negative.");
    }
  }
}
```

JWeightEventHelper

- Manager of (combined) header information and event weighter

- Provides configuration-, getWeight- and check-functions,
  which call the underlying methods of JWeightEvent

- Provides an addition-function, which combines a given header
  with the current header info (if check OK)
  and updates the global event weight

```
/**
 * Add header.
 *
 * \param  head                header
 */
void add(const JHead& head)
{
  if (check(head)) {

    JHead::add(head);
    get()->configure(JHead::getHeader());

  } else {

    THROW(JNullPointerException, "JWeightEvent
  }
}
```

Header bookkeeping

- Header ordering enabled by JHead::less
  - Goes through all header fields
  - Calls the associated less method, e.g.:

```
/**
 * Normalisation of MUPAGE events.
 */
struct livetime {
  /**
   * Default constructor.
   */
  livetime() :
    numberOfSeconds(0),
    errorOfSeconds(0)
  {}

  /**
   * Comparison.
   *
   * \param  object              livetime
   * \return                     true if this livetime is less than given livetime; else false
   */
  inline bool less(const livetime& object) const
  {
    return numberOfSeconds < object.numberOfSeconds;
  }
```

```
    /**
     * Comparison of headers.
     *
     * \param  header          header
     * \return                 true if this header less than given header; else false
     */
    inline bool less(const JHead& header) const
    {
#define RETURN_IF_DIFFERENT(A, B)       \
      if (less(A,B)) { return true;  }  \
      if (less(B,A)) { return false; }

      // compare physics

      RETURN_IF_DIFFERENT(this->physics,     header.physics);

      // compare simulation

      RETURN_IF_DIFFERENT(this->simul,       header.simul);

      // compare generation data

      RETURN_IF_DIFFERENT(this->primary,     header.primary);
      RETURN_IF_DIFFERENT(this->spectrum,    header.spectrum);
      RETURN_IF_DIFFERENT(this->cut_primary, header.cut_primary);
      RETURN_IF_DIFFERENT(this->cut_seamuon, header.cut_seamuon);
      RETURN_IF_DIFFERENT(this->cut_in,      header.cut_in);
      RETURN_IF_DIFFERENT(this->cut_nu,      header.cut_nu);
      RETURN_IF_DIFFERENT(this->genvol,      header.genvol);

      // compare compatibility

      if (is_valid(this->livetime) == is_valid(header.livetime)  &&
          is_valid(this->DAQ)      == is_valid(header.DAQ))      {
        return false;
      }

      THROW(JException, "JHead::less() headers do not compare.");

#undef RETURN_IF_DIFFERENT
    }
```

Header bookkeeping

- Header ordering enabled by JHead::less
  - Goes through all header fields
  - Calls the associated less method
  - Returns true if any field in given header smaller than the associated field of this header
- Header matching enabled by JHead::match
  - Goes through all header fields
  - [illegible]

```
/**
 * Test match.
 *
 * \param  object               DAQ
 * \return                      true if matches; else false
 */
inline bool match(const DAQ& object) const
{
  return ((livetime_s == 0.0 && object.livetime_s == 0.0) ||
          (livetime_s >  0.0 && object.livetime_s >  0.0));
}
```

```
/**
 * Test match of headers.
 *
 * Note that if option is set to <tt>false</tt>,
 * the match applies only to data which have a corresponding entry in the underlying map of the given header.
 *
 * \param  header          second header
 * \param  option          option
 * \return                 true if matches; else false
 */
inline bool match(const JHead& header, const bool option = true) const
{
  return (match(*this, header, option, &JHead::cut_primary)       &&
          match(*this, header, option, &JHead::cut_seamuon)       &&
          match(*this, header, option, &JHead::cut_in)            &&
          match(*this, header, option, &JHead::cut_nu)            &&
          match(*this, header, option, &JHead::physics)           &&
          match(*this, header, option, &JHead::simul)             &&
          match(*this, header, option, &JHead::spectrum)          &&
          match(*this, header, option, &JHead::can)               &&
          match(*this, header, option, &JHead::fixedcan)          &&
          match(*this, header, option, &JHead::genvol)            &&
          match(*this, header, option, &JHead::coord_origin)      &&
          match(*this, header, option, &JHead::norma)             &&
          match(*this, header, option, &JHead::livetime)          &&
          match(*this, header, option, &JHead::seabottom)         &&
          match(*this, header, option, &JHead::primary)           &&
          match(*this, header, option, &JHead::DAQ));
}
```

Header bookkeeping

- Header ordering enabled by JHead::less
  - Goes through all header fields
  - Calls the associated less method
  - Returns true if any field in given header smaller than the associated field of this header
- Header matching enabled by JHead::match
  - Goes through all header fields
  - Calls the associated match method
  - True if all header fields match
- Addition via JHead::add
  - Loops over **relevant fields**
  - Adds associated variables, e.g.:

```
/**
 * Addition of headers.
 *
 * \param  header          header
 * \return                 this header
 */
inline JHead& add(const JHead& header)
{
  if (match(header)) {

    genvol   .add(header.genvol);
    norma    .add(header.norma);
    livetime.add(header.livetime);
    DAQ      .add(header.DAQ);

  } else {

    THROW(JException, "JHead::add() headers do not match.");
  }

  return *this;
}
```

```
/**
 * Addition.
 *
 * \param  object              live time
 * \return                     this live time
 */
inline livetime& add(const livetime& object)
{
  numberOfSeconds += object.numberOfSeconds;
  errorOfSeconds   = sqrt(errorOfSeconds        * errorOfSeconds  +
                          object.errorOfSeconds * object.errorOfSeconds);

  return *this;
}
```

JEventWeighterMap

- A mapping between the default MC-generator headers and event weighters is provided in `JWeightEventToolkit.hh`

  - Default headers implemented in `JHeadToolkit.hh`
  - Two operators to retrieve the default weighter corresponding to a given header and vice versa

```
/**
 * Look-up table for event weighters.
 */
struct JEventWeighterMap :
  public std::map<JHead, JSharedPointer<JWeightEvent>>
{
  /*
   * Constructor
   */
  JEventWeighterMap()
  {
    this->insert(make_pair(getDAQHeader(),      new JWeightDAQ()));
    this->insert(make_pair(getMUPAGEHeader(),   new JWeightMupage()));
    this->insert(make_pair(getGSeaGenHeader(), new JWeightGSeaGen()));
    this->insert(make_pair(getKM3BUUHeader(),   new JWeightKM3BUU()));
  }
```

```
/**
 * Match header for MUPAGE.
 */
struct getMUPAGEHeader :
  public JHead
{
  /**
   * Default constructor.
   */
  getMUPAGEHeader()
  {
    this->physics.resize(1);
    this->physics[0].program = JHead::HEMAS;
    this->push(&JHead::physics);

    this->simul.resize(1);
    this->simul[0].program = JHead::MUPAGE;
    this->push(&JHead::simul);

    this->livetime.numberOfSeconds = 1.0;
    this->push(&JHead::livetime);
  }
};
```

JEventWeighterMap

- A mapping between the default MC-generator headers and event weighters is provided in `JWeightEventToolkit.hh`
  - Default headers implemented in `JHeadToolkit.hh`
  - Two operators to retrieve the default weighter corresponding to a given header and vice versa
- Look-up operators can be called directly using two auxiliary function objects:

```
extern JEventWeighterMap  getEventWeighter;              //!< Function object for mapping header to event weighter.
static JEventWeighterMap& getHeader = getEventWeighter;  //!< Function object for mapping event weighter to default header.
```

JWeightFileScanner

- Under `$JPP_DIR/software/JSupport`
- Configuration method for setting the event weight helper
- Two 'put'-methods for adding single or multiple files to the list and updating the event weighter
- Note:
  - Only files consistent with the set event weighter will be added

```
/**
 * Put list of files.
 *
 * \param  input              file list
 * \return                    number of added files.
 */
size_t put(const JFileScanner_t& input)
{
  size_t N = 0;

  for (typename JFileScanner_t::const_iterator i = input.begin(); i !=

    N += size_t(this->put(*i));
  }

  return N;
}


/**
 * Put file.
 *
 * \param  input              file name
 * \return                    true if successfully added; else false.
 */
bool put(const std::string& input)
{
  using namespace JPP;

  const JHead head = JSUPPORT::getHeader(input);

  if (this->check(head)) {

    if (JFileScanner_t::size() > 0) {
      JWeightEventHelper::add(head);
    } else {
      JWeightEventHelper::setHeader(head);
    }

    JFileScanner_t::push_back(input);

    return true;

  } else {

    return false;
  }
}
```

JWeightFileScannerSet

- = The crown to top it off

- Takes a list (any mixture) of different MC-files and

  1. Combines the files of consistent header-types into individual JWeightFileScanner objects
  2. Orders the file-scanners according to the associated header-information

- Available methods:

  - Two put methods, to insert additional file(s)
  - A get-function to retrieve the JWeightFileScanner object corresponding to a specific header
  - A setLimit-function to limit the number of events read from each scanner (optional)

A simple working example...

- An example program can be found under `$JPP_DIR/examples/JAAnet`
- Allow scanning over any number of gSeaGen, KM3BUU or Mupage files and print out combined weights

```
❯ for i in `seq 601 610`; do FILESTR+="/mnt/nikhef/data/bjung/mc/KM3NeT_00000044/atm_neutrino/generat
or/mcv5.40.gsg_elecCC-CC_1-500GeV.$i.evt "; done
❯ for i in `seq 365 370`; do FILESTR+="/mnt/nikhef/data/bjung/mc/KM3NeT_00000044/atm_muon/generator/m
cv5.40.mupage_10G.$i.evt "; done
❯ $JPP_DIR/examples/JAAnet/JWeightFileScanner -f $FILESTR -n 5 -d 3
Scanning gSeaGen files...
event                weight
1                    0.00000000000000423996
2                    0.00000000000001038336
3                    0.00000000000000077057
4                    0.00000000000000325391
5                    0.00000000000000168056
Scanning MUPAGE files...
event                weight
1                    0.000083139341536415 03
2                    0.000083139341536415 03
3                    0.000083139341536415 03
4                    0.000083139341536415 03
5                    0.000083139341536415 03
 ~/KM3NeT/Jpp  on eventWeighting !7 ?4 ..... ✔  at 15:38:23
```

What's Next

- Try using the framework on gSeaGen and KM3BUU input data
  - W3-weight implementation for KM3BUU under development (**thanks Johannes!**)
  - How to implement getWeight(const Evt& event, const double flux)
    for atmospheric tau-neutrino MC?
- Allow scanning of DAQ-files also
  - When combined with MC-info, may be used for inspection of e.g. trigger rate
- **Feedback and suggestions from the simulations working group** are very relevant!

What's Next

- Try using the framework on gSeaGen and KM3BUU input data
  - W3-weight implementation for KM3BUU under development (**thanks Johannes!**)
  - How to implement getWeight(const Evt& event, const double flux) for atmospheric tau-neutrino MC?
- Allow scanning of DAQ-files also
  - When combined with MC-info, may be used

E.g.:
- gSeaGen v5.0 ORCA 115-string production stores generator name under 'physics' header-field
- GSeaGen v5.40 ORCA 4-string production stores generator name under 'simul' header-field

- **Feedback and suggestions from the simulation working group** are very relevant!
  - Which versions of which generators **require which header fields**?
  - Do specific generators need additional customization of the weight-function?

Summary

- A common software-framework for MC event-weighting has been set up

- **NOT intended as a black-box**, but hopefully can be used as
  a means to (better) **document, harmonize and standardize**
  the MC header formats and event weighting