```
#ifndef _BuildTrajectories_h
#define _BuildTrajectories_h

/*
 * Copyright (c) 2015, Sandia Corporation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */


/* Assemble points to make trajectories
 *
 * This process is the prerequisite for most of the analysis and
 * rendering that Tracktable can do.  By reading pre-assembled
 * trajectories we can save a lot of time when working with a data set
 * more than once.
 */

#include <tracktable/Core/TracktableCommon.h>
#include <tracktable/Core/CommonTypes.h>
#include <tracktable/Core/Timestamp.h>

#include <tracktable/Analysis/AssembleTrajectories.h>

#include <tracktable/Domain/Cartesian2D.h>
#include <tracktable/Domain/Terrestrial.h>

#include <tracktable/IO/TrajectoryWriter.h>
#include <tracktable/IO/PointReader.h>

//#include <boost/program_options.hpp>

#include <algorithm>

#include "BuildTrajectories.h"
#include "CommandLineOptions.h"

template<class T>
std::ostream& operator<<(std::ostream& os, const std::vector<T>& v)
{
  std::copy(v.begin(), v.end(), std::ostream_iterator<T>(os, " "));
```

```
  return os;
}

template<typename trajectory_type>
void BuildTrajectories(CommandLineOptions const& options, std::vector<trajectory_type>&
 trajectories)
{
  typedef typename trajectory_type::point_type point_type;
  typedef tracktable::PointReader<point_type> point_reader_type;
  typedef tracktable::AssembleTrajectories<trajectory_type, typename point_reader_type:
:iterator> assembler_type;
  typedef tracktable::TrajectoryWriter trajectory_writer_type;


  point_reader_type point_reader;
  trajectory_writer_type trajectory_writer;
  std::ifstream infile;
  std::ofstream outfile;

  if (options.InputFilename == "-")
    {
    point_reader.set_input(std::cin);
    }
  else
    {
    infile.open(options.InputFilename.c_str());
    if (!infile)
      {
      std::cerr << "ERROR: Cannot open file "
                << options.InputFilename
                << " for input.\n";
      exit(1);
      }

    point_reader.set_input(infile);
    }

  if (options.OutputFilename == "-")
    {
    trajectory_writer.set_output(std::cout);
    }
  else
    {
    outfile.open(options.OutputFilename.c_str());
    if (!outfile)
      {
      std::cerr << "ERROR: Cannot open file "
                << options.OutputFilename
                << " for output.\n";
      exit(1);
      }
    trajectory_writer.set_output(outfile);
    }

  point_reader.set_object_id_column(options.ObjectIdColumn);
  point_reader.set_timestamp_column(options.TimestampColumn);
  point_reader.set_x_column(options.FirstCoordinateColumn);
  point_reader.set_y_column(options.SecondCoordinateColumn);
  point_reader.set_field_delimiter(options.FieldDelimiter);

  for (typename std::vector<field_assignment_type>::const_iterator iter = options.RealF
ields.begin();
       iter != options.RealFields.end();
```

```
       ++iter)
    {
    point_reader.set_real_field_column((*iter).first, (*iter).second);
    }

  for (typename std::vector<field_assignment_type>::const_iterator iter = options.Integ
erFields.begin();
       iter != options.IntegerFields.end();
       ++iter)
    {
    point_reader.set_integer_field_column((*iter).first, (*iter).second);
    }

  for (typename std::vector<field_assignment_type>::const_iterator iter = options.Strin
gFields.begin();
       iter != options.StringFields.end();
       ++iter)
    {
    point_reader.set_string_field_column((*iter).first, (*iter).second);
    }

  for (typename std::vector<field_assignment_type>::const_iterator iter = options.Times
tampFields.begin();
       iter != options.TimestampFields.end();
       ++iter)
    {
    point_reader.set_time_field_column((*iter).first, (*iter).second);
    }

  assembler_type trajectory_assembler(point_reader.begin(), point_reader.end());
  trajectory_assembler.set_separation_distance(options.SeparationDistance);
  trajectory_assembler.set_separation_time(tracktable::seconds(options.SeparationSecond
s));
  trajectory_assembler.set_minimum_trajectory_length(options.MinimumNumPoints);

//  trajectory_writer.write(trajectory_assembler.begin(), trajectory_assembler.end());

  for (typename assembler_type::iterator itr = trajectory_assembler.begin(); itr != tra
jectory_assembler.end(); ++itr) {
    trajectories.push_back(*itr);
  }

}
#endif
```