```
#ifndef CPP_TOOLS_DATETIME_H
#define CPP_TOOLS_DATETIME_H

#include <string>
#include <utility>
#include <memory>
#include "boost/date_time/local_time/local_time.hpp"
#include "CPPTools/Tools.h"
/*
 * As of boost_1_54_0. If -std=c++11 used then
 -DBOOST_NO_CXX11_EXPLICIT_CONVERSION_OPERATORS should be set (explicit set in conversion
 to
 * bool in std::shared_ptr
 */
namespace cpp_tools
{
namespace tz_db
{

class  DbWrapper{
    DbWrapper(const DbWrapper&) = delete;
    DbWrapper& operator = (const DbWrapper&) = delete;
 public:
     DbWrapper(){}
    StatusT loadFromFile(const std::string &dbFile)
    {
        if(!m_dbFile.empty())// merging dbs from 2 files could be implemented, but for
        now we assume it is an error
        {
            return StatusT(std::string("Time zone DB has alredy been loaded with " +
            m_dbFile), false);
        }
        try{
              m_tzDb.load_from_file(dbFile);
              m_dbFile = dbFile;
              return StatusT(std::string("Time zone data file " + dbFile + " has been
              successfully loaded."), true);
          }catch(boost::local_time::data_not_accessible dna) {
             return StatusT(std::string("Error with time zone data file " + dbFile + ". "
             + dna.what() +
                     ". Default time zone will be used."), false);
          }catch(boost::local_time::bad_field_count bfc) {
              return StatusT(std::string("Error with time zone data file " + dbFile + ".
              " + bfc.what() +
                      ". Default time zone will be used."), false);
          }catch(...){
             return StatusT(std::string("An error with time zone data file " + dbFile +
             ". Default time zone will be used."), false);
          }
    }
    boost::local_time::time_zone_ptr TimeZoneFromRegion(const std::string &tzKey) const
    {
```

```
        return m_tzDb.time_zone_from_region(tzKey);
    }
    private:
    boost::local_time::tz_database          m_tzDb;
    std::string                             m_dbFile;
};

}// namexpace tz_db
    /*
     *  SetTimeZoneDb and SetDefaultTimeZone are write operation => are not thead save
     * local_time() are  read operations => can be called from different threads (if no
     write operation at the same time)
     */
  template<typename time_type, template<typename> class clock_type> class Clock
  {
      Clock(const Clock&) = delete;
      Clock& operator = (const Clock&) = delete;
      typedef std::unique_ptr<tz_db::DbWrapper> DbPtr;

   public:
      Clock(){}
      static StatusT SetTimeZoneDb(const std::string &dbFile)
      {
          std::unique_ptr<tz_db::DbWrapper>
          ptr(cpp_tools::make_unique<tz_db::DbWrapper>());
          const StatusT status = ptr->loadFromFile(dbFile);
          if(status.second)
          {
             Instance().m_dbPtr = std::move(ptr) ;
          }
          return status;
       }

      static StatusT SetDefaultTimeZone(const std::string &tzKey){
          Clock&  cl= Instance();
          if(!cl.m_dbPtr)
          {
              return StatusT(std::string("Time zone BD has not been set."), false);
          }
          cl.m_defaultTz= cl.m_dbPtr->TimeZoneFromRegion(tzKey);
          if(cl.m_defaultTz){
               return StatusT(std::string("Default time zone has been set with key ") +
               tzKey, true);
          }else{
             return StatusT(std::string("Time zone key ") +  tzKey + " cannot be found.",
             false);
          }
       }

      static time_type local_time()
      {
          auto tz_ptr = Instance().m_defaultTz;
          if(!tz_ptr){
```

```
            return clock_type<boost::posix_time::ptime>::local_time();
        }else{
             return local_time(tz_ptr);
        }
    }

    static time_type local_time(const std::string &tzKey)
    {
        Clock&  cl= Instance();
        if(!cl.m_dbPtr)
        {
            return InvalidTime();;
        }
        auto tz_ptr = cl.m_dbPtr->TimeZoneFromRegion(tzKey);
        if(tz_ptr){
            return local_time(tz_ptr);
        }else{
            return InvalidTime();
        }
    }
    static time_type local_time(boost::local_time::time_zone_ptr tz_ptr)
    {
        if(!tz_ptr)
        {
            return InvalidTime();
        }
        try{
           return
           clock_type<boost::local_time::local_date_time>::local_time(tz_ptr).local_time(
           );
        }catch(std::exception &)
        {
            return InvalidTime();
        }
    }
    static bool IsTimeValid(const time_type &time){
        return !time.is_special();
    }
private:
     static time_type& InvalidTime()
     {
         static time_type time;
         return time;
     }
     static Clock& Instance()
    {
       static  Clock cl;
       return cl;
    }

private:
    DbPtr                                   m_dbPtr;
    boost::local_time::time_zone_ptr        m_defaultTz;
```

```
    };


 typedef Clock<boost::posix_time::ptime, boost::date_time::microsec_clock > microsec_clock;
}
#endif  /* CPP_TOOLS_DATETIME_H */

/*
 *
 *
#include<iostream>
#include "CPPTools/DateTime.h"


int main(int argc, char *argv[]) {

    typedef cpp_tools::microsec_clock microsec_clock;

    boost::posix_time::ptime pd1= microsec_clock::local_time();
    std::cout <<"default time: " << "is valid=" <<
    microsec_clock::IsTimeValid(pd1)<<":"<<boost::posix_time::to_simple_string(pd1 )
    <<std::endl;

    std::cout << microsec_clock::SetTimeZoneDb("xxxxx").first <<std::endl;
    std::cout <<
    microsec_clock::SetTimeZoneDb("/export/home/ihersht/main/US/EVD/Far/CPPTools/data/date_
    time_zonespec.csv").first <<std::endl;
    std::cout << microsec_clock::SetDefaultTimeZone("xxxx").first <<std::endl;

    boost::posix_time::ptime pd2= microsec_clock::local_time();
    std::cout <<"default time: " << "is valid=" <<
    microsec_clock::IsTimeValid(pd2)<<":"<<boost::posix_time::to_simple_string(pd2 )
    <<std::endl;

    std::cout << microsec_clock::SetDefaultTimeZone("America/Sao_Paulo").first <<std::endl;
    boost::posix_time::ptime pd3= microsec_clock::local_time();
    std::cout <<"default time: " << "is valid=" <<
    microsec_clock::IsTimeValid(pd3)<<":"<<boost::posix_time::to_simple_string(pd3 )
    <<std::endl;

    boost::posix_time::ptime p4= microsec_clock::local_time("Africa/Dar_es_Salaam");
    std::cout <<"time: " << "is valid=" <<
    microsec_clock::IsTimeValid(p4)<<":"<<boost::posix_time::to_simple_string(p4 )
    <<std::endl;

    //
    return 0;
}
 */
```