```
#ifndef CPP_TOOLS_FILE_LOADER_H
#define CPP_TOOLS_FILE_LOADER_H

#include <memory>
#include <string>
#include <exception>
#include <queue>
#include <set>
#include <boost/algorithm/string.hpp>
#include <boost/property_tree/ptree.hpp>

#include <boost/program_options.hpp>
#include "CPPTools/Tools.h"
#include "CPPTools/Exception.h"


namespace cpp_tools
{
    /*Element root.config_files.file is special
     * It is assumed to be a path to another config file which is going to be
loaded.
     * One should avoid circular references.
     <?xml version="1.0" encoding="utf-8" ?>
    <main>
      <config_files>
            <file>
                /export/home/ihersht/main/US/EVD/Far/PsAdapter/Config/common.xml
            </file>
            <file>

/export/home/ihersht/main/US/EVD/Far/PsAdapter/Config/market_data.xml
            </file>
            <file>

/export/home/ihersht/main/US/EVD/Far/PsAdapter/Config/pricing_paramets.xml
            </file>
            <file>

/export/home/ihersht/main/US/EVD/Far/PsAdapter/Config/pricing_data_input.xml
            </file>
       </config_files>
  <dump_xml>/export/home/ihersht/main/US/EVD/Far/PsAdapter/Config/out.xml</dump_xml>

</main>
     */

    class ConfigFileLoader
    {
        using CmdsT = std::map<std::string, std::string> ;
        using TreeT = boost::property_tree::ptree;
    public:
        ConfigFileLoader(const ConfigFileLoader&) = delete;
        ConfigFileLoader& operator = (const ConfigFileLoader&) = delete;
        ConfigFileLoader() {}
        using ErrorsT =  std::vector<std::exception_ptr >;
        enum MergePolicy{ CMD_PREFERRED };
        enum SavePolicy{ FILE_SAVE };
        enum FileType{UNKNOWN_FILE_TYPE = -1,  FILE_XML, FILE_CSV, FILE_INI};

       void Load(int argc, char* argv[] );
       std::exception_ptr LoadFromOneFileNoexcept(const std::string &fileName)
noexcept;
       //save config to a file where
```

```
        std::exception_ptr Save(const std::string &where) noexcept;
        ////save config to a file from the node main.dump_xml
        std::exception_ptr Save() noexcept;
        const ErrorsT& GetErrors() const noexcept{
            return m_errors;
        }

         template<typename T> std::pair<T, std::exception_ptr>  GetValue(const
std::string &key) const noexcept
         {
               try{
                    T value = m_configTree.get<T>(key);
                    return {value, nullptr};
               }catch(...){
                 return {T(), std::current_exception()} ;
               }
         }

          template<typename MapT> std::pair<MapT, std::exception_ptr>  MakeMap()
const noexcept
         {
               try{
                    MapT map;
                    TreeT::path_type path;
                    AddToMap(m_configTree.begin(), m_configTree.end(), map, path);
                    return {map, nullptr};
               }catch(...){
                 return {MapT(), std::current_exception()} ;
               }
         }

    private:
         static std::string GetMainConfigFileName(const CmdsT  &cmds)  noexcept;
         std::exception_ptr LoadCmdNoexcept(int argc, char* argv[], CmdsT  &cmds)
noexcept;
         std::exception_ptr LoadConfigFileNoexcept(const std::string &fileName, TreeT
&out ) noexcept;
         void LoadConfigFile(const std::string &fileName, TreeT &out ) ;
         void LoadCmd(int argc, const char* const argv[], CmdsT  &cmds);
         std::exception_ptr MergeCmdAndConfig(const CmdsT  &cmds, TreeT &out)
noexcept;
         static  FileType GetFileType(const std::string &fileName);
         static std::string GetRootElement(const std::string &fileName)noexcept;
         static std::string AppendPath(const std::string &begin, const std::string
&path )  noexcept
         {
             return begin + "." + path;
         }
         std::exception_ptr CollectIncludedFileNames(const TreeT &tree, const
std::string &root, std::set<std::string> &names) noexcept;
         void clear() noexcept
         {
             ErrorsT errors;
             m_errors.swap(errors);
             m_configTree.clear();
         }
         void LoadFromOneFile(const std::string &fileName, TreeT &tree);
         void DumpErrors();
         template<typename MapT>
         void  AddToMap(TreeT::const_iterator begin, TreeT::const_iterator end, MapT
&map,  TreeT::path_type &path ) const
         {
             for(TreeT::const_iterator pos  = begin; pos != end; ++pos)
```

```
            {
                std::string key = pos->first;
                TreeT::path_type newPath = path;
                newPath /=  TreeT::path_type(key);
                std::string value = pos->second.data(); boost::trim(value);
                if((!key.empty()) && ( !value.empty()) ){
                    const std::string mapKey = newPath.dump();
                    map.insert(typename MapT::value_type(mapKey,value ));
                }
                AddToMap(pos->second.begin(), pos->second.end(), map, newPath);
            }

        }
      private:
        MergePolicy                     m_mergePolicy {CMD_PREFERRED};
        SavePolicy                      m_savePolicy {FILE_SAVE};
        TreeT                           m_configTree;
        ErrorsT                         m_errors;
        const static TreeT              EMPTY_TREE;


    };

    using ConfigPtr = std::shared_ptr<cpp_tools::ConfigFileLoader>;
}
#endif  /* CPP_TOOLS_FILE_LOADER_H*/


#include <boost/property_tree/xml_parser.hpp>
#include "CPPTools/LoggerInterface.h"
#include "CPPTools/ConfigFileLoader.h"
namespace cpp_tools
{

const  ConfigFileLoader::TreeT   ConfigFileLoader::EMPTY_TREE;

void ConfigFileLoader::Load(int argc, char* argv[] )
{
  CmdsT                     cmds;
  std::exception_ptr error = LoadCmdNoexcept (argc,argv, cmds);
  if(error) {
     m_errors.push_back(error);
  }
  TreeT out;
  error = LoadConfigFileNoexcept(GetMainConfigFileName(cmds) , out);
  if(error) {
     m_errors.push_back(error);
  }
  error = MergeCmdAndConfig(cmds, out);
  if(error) {
     m_errors.push_back(error);
  }
  m_configTree.swap(out);
  error = Save();
  if(error) {
     m_errors.push_back(error);
  }
 // m_configTree.sort();
  DumpErrors();
}
 std::exception_ptr ConfigFileLoader::LoadFromOneFileNoexcept(const std::string
&fileName) noexcept
 {
```

```
     clear();
     try{
        LoadFromOneFile(fileName, m_configTree);
        return nullptr;
    }catch(...){
      clear();
     return std::current_exception();
   }
 }

 void ConfigFileLoader::LoadFromOneFile(const std::string &fileName, TreeT &tree)
 {
        if( fileName.empty())
    {
        std::string msg = SRC_LOCATION + " Cannot find configuration file";
        throw ConfigError(msg);
    }
    TreeT temp;
    FileType type = GetFileType(fileName);
    if(type == FILE_XML )
    {
      // boost::property_tree::read_xml(fileName, temp,
boost::property_tree::xml_parser::trim_whitespace);
        boost::property_tree::read_xml(fileName, temp);
        tree.insert(tree.end(), temp.front() );
       //out.insert(out.end(), EMPTY_TREE.front() );
       //
boost::property_tree::write_xml("/export/home/ihersht/main/US/EVD/Far/PsAdapter/Conf
ig/out.xml", out);
    }else{
        std::string msg = SRC_LOCATION + " Cannot find File Type";
        throw ConfigError(msg);
    }

 }
void ConfigFileLoader::LoadCmd(int argc, const char* const argv[], CmdsT  &cmds)
{
    try
    {
       boost::program_options::options_description desc;
       boost::program_options::command_line_parser parser(argc, argv );
       parser.options(desc);
       parser.allow_unregistered();
       auto ops =parser.run();
        for(auto pr : ops.options)
        {
            std::string key = pr.string_key;
            if(key.empty() ) {
                continue;
            }

            auto vec = pr.value;
            if(vec.empty() ) {
                continue;
            }

            std::string value = vec.at(0);

            if(value.empty() ) {
                 continue;
             }
            cmds[key] = value;
       }
```

```
    }catch (std::exception& ex)
    {
        cmds.clear();
        std::string msg = SRC_LOCATION + " Cannot parse command line " + ex.what();
        throw ConfigError(msg);
    } catch (...)
    {
        cmds.clear();
        std::string msg = SRC_LOCATION + " Cannot parse command line " ;
        throw ConfigError(msg);
    }

}

 std::string ConfigFileLoader::GetMainConfigFileName(const CmdsT  &cmds)  noexcept
{
    auto pos = cmds.find("c");
    if(pos != cmds.end() ){
        std::string name = pos->second;
        boost::trim(name);
        return name;
    }else{
        return "";
    }
}

std::exception_ptr ConfigFileLoader::LoadCmdNoexcept(int argc, char* argv[], CmdsT
&cmds) noexcept
{
    try{
          LoadCmd(argc, argv, cmds );
          return nullptr;
      }catch(...)
     {
        return std::current_exception();
     }

}

std::exception_ptr ConfigFileLoader::LoadConfigFileNoexcept(const std::string
&fileName, TreeT &out ) noexcept
{
    TreeT temp = out;
    try{
           LoadConfigFile(fileName, temp);
           out.swap(temp);
           return nullptr;
       }catch(...)
      {
         return std::current_exception();
      }

}

 ConfigFileLoader::FileType ConfigFileLoader::GetFileType(const std::string
&fileName)
{

     std::string ex= GetFileExtension(fileName);
     boost::to_lower(ex);
     boost::trim(ex);
    if(ex == ".xml") {return FILE_XML;}
    if(ex == ".csv") {return FILE_CSV;}
```

```
    if(ex == ".ini") {return FILE_INI;}
    return UNKNOWN_FILE_TYPE;
}

std::exception_ptr ConfigFileLoader::MergeCmdAndConfig(const CmdsT  &cmds, TreeT
&out) noexcept
{

    try
    {
        if (m_mergePolicy != CMD_PREFERRED)
        {
            std::string msg = SRC_LOCATION + " Unknown merge policy." ;
            throw ConfigError(msg);
        }
        TreeT temp = out;
        for(auto pos : cmds)
        {
           const std::string key =   pos.first;
           const std::string value = pos.second;
           if(key.empty() && value.empty()){
               continue; //should not be here
           }
           temp.put(key, value);
           out.swap(temp);
        }
         return nullptr;
    }catch(...)
    {
       return std::current_exception();
    }

}

/*TODO
 * Good (not just by string) check on "include" files circular references.
 */
void ConfigFileLoader::LoadConfigFile(const std::string &fileName, TreeT &out)
{

    LoadFromOneFile(fileName, out);

    const std::string root = GetRootElement(fileName);
    std::set<std::string> includedFileNames;
    std::exception_ptr err = CollectIncludedFileNames(out, root, includedFileNames);
    if(err)
    {
        m_errors.push_back(err);
    }

    for(auto name : includedFileNames)
    {
        std::exception_ptr err = LoadConfigFileNoexcept(name, out);
        if(err)
        {
            m_errors.push_back(err);
        }
    }

}

std::string ConfigFileLoader::GetRootElement(const std::string &fileName) noexcept{
```

```
    try{
        boost::filesystem::path p(fileName);
        const std::string stem =  p.stem().string();
        return stem;
    }catch(...)
    {
        return "";
    }
}

std::exception_ptr ConfigFileLoader::CollectIncludedFileNames(const TreeT &tree,
const std::string &root, std::set<std::string> &names) noexcept
{
     std::string fromPath =  AppendPath(root, "config_files");

    try{
        for(auto v : tree.get_child(fromPath, EMPTY_TREE) )
        {
           std::string name = v.second.data();
           boost::trim(name);
           names.insert(name);
        }
        return nullptr ;
    }catch(...)
    {
         names.clear();
         return std::current_exception();
    }
}

std::exception_ptr ConfigFileLoader::Save(const std::string &where) noexcept
{
    // TODO pretty printing
    try{
        boost::property_tree::write_xml(where, m_configTree);
        return nullptr;
    }catch(...){
         return std::current_exception();
    }
}

std::exception_ptr ConfigFileLoader:: Save() noexcept
{
    try{
         std::string fileName = m_configTree.get<std::string>("common.dump_xml");
         boost::trim(fileName);
        std::exception_ptr err = Save(fileName);
        return err;
    }catch(...){
         return std::current_exception();
    }
}
void ConfigFileLoader::DumpErrors()
{
    for(auto err: m_errors)
    {
        std::string msg = SRC_LOCATION + GetExceptionMsg(err);
        LogWrapper::Log(msg);
    }
}
}

#ifndef CPP_TOOLS_CONFIGINFO_H
```

```
#define CPP_TOOLS_CONFIGINFO_H
#include <boost/container/flat_map.hpp>
#include <string>
#include <boost/lexical_cast.hpp>
#include <boost/algorithm/string.hpp>
#include <vector>
#include <iterator>
namespace cpp_tools
{

template <typename MapT>struct ConfigInfoBase
{
    using map_type = MapT;
    using const_iterator =  typename map_type:: const_iterator;
    using element_type =    typename map_type::value_type;
    using key_type =        typename element_type::first_type;
    using value_type =      typename element_type::second_type;
    enum Status{FOUND, NOT_UNIQUE_FOUND, NOT_FOUND, CAST_ERROR };
    ConfigInfoBase(){}
    ConfigInfoBase(const MapT & map): m_map{map}{
        m_map.shrink_to_fit();
     }
    ConfigInfoBase(MapT && map): m_map{ std::move(map) }{
        m_map.shrink_to_fit();
      }

    const_iterator begin()  const noexcept { return m_map.begin() ; }
    const_iterator end()  const noexcept { return m_map.end() ; }

    std::pair<const_iterator, const_iterator> equal_range(key_type key) const
noexcept{
        return m_map.equal_range(key);
    }

    template<typename T> std::vector<std::pair<T, Status> > Get(key_type key,T
defaultValue) const noexcept
    {
        auto range = m_map.equal_range(key);
        std::vector<std::pair<T, Status> > vec;
        for(auto pos = range.first, end = range.second; pos != end; ++pos)
        {
            std::pair<T, Status> element{defaultValue, CAST_ERROR};
            try{
              element.first = boost::lexical_cast<T>(pos->second);
              element.second = FOUND;
           }catch(...){
           }
           vec.push_back(element);
        }
       return vec;
    }

    template<typename T> std::pair<T, Status> GetUnique(key_type key, T
defaultValue) const noexcept
    {
       auto range = m_map.equal_range(key);
       std::size_t dist = std::distance(range.first, range.second);
       std::pair<T, Status> out{defaultValue, NOT_FOUND};
       if(dist == 0 )
       {
           return out;
       }
       try{
```

```
            out.first =  boost::lexical_cast<T>(range.first->second);
        }catch(...){
            return {defaultValue, CAST_ERROR};
        }
        if(dist == 1){
           out.second  = FOUND;
        }else{
            out.second  = NOT_UNIQUE_FOUND;
        }
        return out;
    }

    std::pair<bool, Status> GetUnique(key_type key, bool defaultValue) const noexcept
    {
        auto range = m_map.equal_range(key);
        std::size_t dist = std::distance(range.first, range.second);
        std::pair<bool, Status> out ={defaultValue, NOT_FOUND };
        if(dist == 0 )
        {
            return out;
        }

        std::string outStr = range.first->second;
        boost::to_lower(outStr);
        if(outStr == "true"){
            out.first = true;
        }else if(outStr == "false"){
            out.first = false;
        }else{
           return {defaultValue, CAST_ERROR};
        }

        if(dist == 1){
            out.second = FOUND;
        }else{
             out.second = NOT_UNIQUE_FOUND;
        }
        return out;
    }

    private:
        MapT m_map;
};


using ConfigInfo = ConfigInfoBase< boost::container::flat_multimap< std::string,
std::string> > ;
using ConfigInfoPtr = std::shared_ptr< ConfigInfo >;
}
#endif  /* CPP_TOOLS_CONFIGINFO_H */
```