```
#ifndef CPP_TOOLS_INTBITSTATE_H
#define CPP_TOOLS_INTBITSTATE_H

#include          <cstdint>
#include          <atomic>
#include          <type_traits>
#include          <memory>
#include          <map>

namespace cpp_tools
{

 template <typename T > struct UIntState{

  using value_type = T;
  static_assert(std::is_integral<value_type>::value &&
  std::is_unsigned<value_type>::value, "This class defined only for unsigned integral
  types.");

   UIntState(value_type value): m_value(value){}

   value_type Get()const noexcept{
       return m_value.load();
    }
  void Set(value_type state ) noexcept{
        m_value.store(state);
    }
   bool IsSet(value_type state)const noexcept{
       auto temp = (Get() & state);
       return (temp == state);
    }
   void Append(value_type state ) noexcept{
        m_value.fetch_or(state);
    }
   void Clear(value_type state ) noexcept{
       m_value.fetch_and(~state);
    }
   void Clear( ) noexcept{
       Set( value_type{} );
    }
 private:
    std::atomic<value_type>  m_value;
};

using BitState = UIntState<std::uint64_t>;

class SystemState{
    using key_type = long;
    using value_type = std::shared_ptr<BitState>;
    using map_type = std::map<key_type, value_type>;
    SystemState() = default;
public:
      static SystemState& Instance(){
```

```
        static SystemState state;
        return state;
      }
      bool Insert(key_type key, value_type value){
        auto res = m_map.insert({ key, value});
        return res.second;
    }

    template<typename StateType>  StateType&  Get() noexcept {
        auto iter = m_map.find(StateType::Id() );
        if(iter == m_map.end() ){
            static StateType state;
            return state;
        }
        return (StateType&)(*(iter->second) );
    }
private:
    map_type m_map;
};

}//namespace cpp_tools

#ifndef BUSINESS_TOOLS_SYSTEMSTATE_H
#define BUSINESS_TOOLS_SYSTEMSTATE_H
#include "CPPTools/IntBitState.h"


namespace business_tools {

    using StateValueType = cpp_tools::BitState::value_type;
    enum struct CommonStateTypeId {Config =0, Log, ExternalConfig, PriceLoader, MAX_NUM};
    template <StateValueType> struct StateCode;

  template <> struct StateCode<(StateValueType)CommonStateTypeId::Config> : public
  cpp_tools::BitState {
        static constexpr StateValueType Id(){return
        (StateValueType)CommonStateTypeId::Config ;}
        enum  StateId : StateValueType {OK = 0, ERROR = 0x1, NOT_INIT = 0x2 };
        StateCode(): cpp_tools::BitState( (StateValueType)NOT_INIT){}
 };
 using  ConfigStateCode = StateCode<(StateValueType)CommonStateTypeId::Config>;

 template <> struct StateCode<(StateValueType)CommonStateTypeId::Log> : public
 cpp_tools::BitState {
        static constexpr StateValueType Id(){return
        (StateValueType)CommonStateTypeId::Log ;}
        enum  StateId : StateValueType {OK = 0, ERROR = 0x1, NOT_INIT = 0x2 };
        StateCode(): cpp_tools::BitState( (StateValueType)NOT_INIT){}
 };
  using  LogStateCode = StateCode<(StateValueType)CommonStateTypeId::Log>;

 template <> struct StateCode<(StateValueType)CommonStateTypeId::ExternalConfig> : public
 cpp_tools::BitState {
```

```
        static constexpr StateValueType Id(){return
        (StateValueType)CommonStateTypeId::ExternalConfig ;}
        enum  StateId : StateValueType {OK = 0, ERROR = 0x1, NOT_INIT = 0x2 };
        StateCode(): cpp_tools::BitState( (StateValueType)NOT_INIT){}
 };
 using  ExternalConfigCode = StateCode<(StateValueType)CommonStateTypeId::ExternalConfig>;

template <> struct StateCode<(StateValueType)CommonStateTypeId::PriceLoader> : public
cpp_tools::BitState {
   static constexpr StateValueType Id(){return
   (StateValueType)CommonStateTypeId::PriceLoader ;}
   enum  StateId : StateValueType {OK = 0, ERROR = 0x1, NOT_INIT = 0x2 };
    StateCode(): cpp_tools::BitState( (StateValueType)NOT_INIT){}
};
using PriceLoaderCode = StateCode<(StateValueType)CommonStateTypeId::PriceLoader>;

}//namespace business_tools
#endif  /* BUSINESS_TOOLS_SYSTEMSTATE_H */

#include <atomic>
#include "BusinessTools/SystemState.h"
#include "CPPTools/Tools.h"


/*// how to add state on this level
 namespace business_tools {
  template <> struct StateCode<(StateValueType)CommonStateTypeId::PriceLoader> : public
  cpp_tools::BitState {
        static constexpr StateValueType Id(){return
        (StateValueType)CommonStateTypeId::PriceLoader ;}
        enum  StateId : StateValueType {OK = 0, ERROR = 0x1, NOT_INIT = 0x2 };
        StateCode(): cpp_tools::BitState( (StateValueType)NOT_INIT){}
    };
    using PriceLoaderCode = StateCode<(StateValueType)CommonStateTypeId::PriceLoader>;
 }
*/


namespace ps_adapter
{
    using StateValueType = business_tools::StateValueType;

    using  ConfigStateCode =  business_tools::ConfigStateCode;
    using  LogStateCode =  business_tools::LogStateCode;
    using  ExternalConfigCode =  business_tools::ExternalConfigCode;
    using  PriceLoaderCode =  business_tools::PriceLoaderCode;

    inline void InitSystemState(){
        cpp_tools::SystemState &state = cpp_tools::SystemState::Instance();

        state.Insert(ConfigStateCode::Id(),          std::make_shared<ConfigStateCode>());
        state.Insert(LogStateCode::Id(),             std::make_shared<LogStateCode>());
        state.Insert(ExternalConfigCode::Id(),
```

```
        std::make_shared<ExternalConfigCode>());
        state.Insert(PriceLoaderCode::Id(),          std::make_shared<PriceLoaderCode>());
    }

 }
#endif  /* ADAPTER_STATE_H */


#endif  /* CPP_TOOLS_INTBITSTATE_H */
```