madk_pp_protocol.h

Go to the documentation of this file.

```
1Â /********************************************************************************** 2Â  * Company: Verifone 3Â  *
Author: GSS R&D Germany 4Â  * Content: SDI-Server 5Â
*********************************************************************************/ 6Â #ifndef _MADK_PP_PROTOCOL_H_ 7
Â #define _MADK_PP_PROTOCOL_H_ 8Â  9Â #include <vector> 10Â #include <string> 11Â #include <pthread.h> 12Â  14
Â struct mADK_PP_Prot_Loader; 16Â struct ProtStatus; 17Â struct ConnectInfo; 18Â struct TcpInfo; 19Â struct
UdsInfo; 20Â  23Â class mADK_PP_Prot 24Â { 25Â  public: 26Â  27Â  enum ProtocolState 28Â  { 29Â  PS_Stopped = 0,
30Â  PS_Running = 1, 31Â  PS_Aborted = 2 32Â  }; 33Â  34Â  enum AbortState 35Â  { 36Â  AS_NotAborted = 0, 37Â
AS_AbortIdle = 1, 38Â  AS_ConnectionLost = 2 41Â  }; 42Â  43Â  private: 44Â  pthread_t proto_thread; 45Â
pthread_mutex_t mutex; 46Â  pthread_cond_t cond; 47Â  std::vector<unsigned char> cmdbuf; 48Â  char prot_type; 50
Â  unsigned msg_id; 51Â  unsigned msg_id_payment; 52Â  unsigned msg_id_nested; 53Â  enum ProtocolState
prot_state; 54Â  bool cmd_active; 55Â  bool nested_cmd; 56Â  enum AbortState aborted; 57Â  unsigned cmd_counter;
59Â  struct mADK_PP_Prot_Loader *lib; 63Â  enum CommandType 64Â  { 65Â  Cmd_Any = 0, 66Â  Cmd_Abort = 1, 67Â
Cmd_LockPayment = 2, 69Â  Cmd_ResetLink = 3, 72Â  Cmd_Check4Update = 4, 73Â  Cmd_ExtButton = 5, 74Â
Cmd_EmvCbResponse = 6, 75Â  Cmd_CtrlCbResponse = 7, 76Â  Cmd_TrgrCbResponse = 8, 77Â  Cmd_PaymentNotificationCb
= 9 78Â  }; 80Â  enum RequestType 81Â  { 82Â  Request_None = 0, 83Â  Request_EMV = 1, 84Â  Request_Ctrl = 2, 85
Â  Request_Trgr = 3 86Â  }; 87Â  RequestType req_type; 88Â  bool recovery_mode; 89Â  bool busy; 90Â  bool
cancelled; 91Â  bool unlock_payment_pending; 94Â  void *main_handle; 101Â  void *busy_handle; 104Â  static
mADK_PP_Prot *obj; // singleton 105Â  106Â  // disable calling copy constructor and assignment operator 107Â
mADK_PP_Prot(const mADK_PP_Prot &); 108Â  void operator=(const mADK_PP_Prot &); 109Â #if __cplusplus >= 201103L
// since C++11 also forbid usage of move 110Â  mADK_PP_Prot(const mADK_PP_Prot&&); 111Â  mADK_PP_Prot&
operator=(mADK_PP_Prot&&); 112Â #endif 113Â  114Â  // disable constructor and force caller to use getInstance()
115Â  mADK_PP_Prot(); 118Â  virtual ~mADK_PP_Prot(); 119Â  122Â  static void *run(void *data); 123Â  130Â
static enum CommandType getCommandType(const unsigned char *cmd, unsigned size); 131Â  136Â  static bool
isCallbackResponse(const enum CommandType cmd); 137Â  145Â  bool _poll(int timeout_msec); 146Â  155Â  bool
_receive(std::vector<unsigned char> &cmd, int timeout_msec); 156Â  163Â  bool _send(void *handle, const unsigned
char *cmd, unsigned size, char protType, unsigned msgId) const; 164Â  171Â  int _get_connection_type(void
*handle); 172Â  public: 173Â  176Â  static mADK_PP_Prot *getInstance(); 177Â  181Â  inline struct
mADK_PP_Prot_Loader *loader() 182Â  { 183Â  return lib; 184Â  } 185Â  192Â  bool poll(int timeout_msec = -1);
193Â  201Â  bool receive(std::vector<unsigned char> &cmd, int timeout_msec = -1); 202Â  211Â  bool abort();
212Â  224Â  bool send(const unsigned char *cmd, unsigned size); 225Â  227Â  inline bool send(const
std::vector<unsigned char> &cmd) 228Â  { 229Â  return send(cmd.size() > 0 ? &cmd[0] : 0, cmd.size()); 230Â  }
231Â  243Â  bool command(const unsigned char *cmd, unsigned size); 244Â  261Â  bool request(const unsigned char
*cmd, unsigned size); 262Â  264Â  inline bool request(const std::vector<unsigned char> &cmd) 265Â  { 266Â
return request(cmd.size() > 0 ? &cmd[0] : 0, cmd.size()); 267Â  } 268Â  281Â  bool
recv_response(std::vector<unsigned char> &response, int timeout_msec = -1); 282Â  306Â  bool set_busy(bool flag,
bool cmd_ctx = true); 307Â  311Â  bool loadAcl(); 312Â  314Â  enum CMDActiveType 315Â  { 316Â  CMD_None = 0,
317Â  CMD_Active = 1, 318Â  CMD_Nested = 2 319Â  }; 323Â  inline enum CMDActiveType command_active() const 324Â
{ 325Â  if (cmd_active) 326Â  { 327Â  if (nested_cmd) 328Â  { 329Â  return CMD_Nested; 330Â  } 331Â  332Â
return CMD_Active; 333Â  } 334Â  335Â  return CMD_None; 336Â  } 337Â  341Â  bool start(); 342Â  346Â  void
stop(); 347Â  350Â  void setRecoveryMode(bool on_off); 351Â  354Â  inline enum ProtocolState
get_protocol_state() const 355Â  { 356Â  return prot_state; 357Â  } 358Â  364Â  inline bool connected() const
365Â  { 366Â  return !!get_handle(); 367Â  } 368Â  373Â  inline bool check_abort() const 374Â  { 375Â  return
cancelled; 376Â  } 377Â  386Â  void set_command(void *handle, char *cmd, int size, char protType, unsigned
msgId); 387Â  389Â  bool get_status(struct ProtStatus *status) const; 390Â  392Â  bool
get_connection_info(struct ConnectInfo *info) const; 393Â  395Â  void free_connection_info(struct ConnectInfo
*info) const; 396Â  398Â  void set_com_profile(const char *file) const; 399Â  404Â  bool trusted_connection();
405Â  412Â  bool get_tcp_info(struct TcpInfo *info); 413Â  420Â  bool get_uds_info(struct UdsInfo *info); 421Â
426Â  void *get_handle() const; 427Â  430Â  char get_protocol_type() const; 431Â  446Â  bool
isConnectionThread() const; 447Â  455Â  bool isMainConnectionThread() const; 456Â  463Â  int select(int
comInterfaces, char **ComFileName); 464Â  467Â  void reset(); 468Â }; 469Â  472Â class mADK_PP_ProtBusyLock
473Â { 474Â  mADK_PP_Prot *m_proto; 475Â  bool m_locked; 476Â  477Â  // disable copy constructor and assign
operator 478Â  mADK_PP_ProtBusyLock(const mADK_PP_ProtBusyLock&); 479Â  mADK_PP_ProtBusyLock& operator=(const
mADK_PP_ProtBusyLock&); 480Â #if __cplusplus >= 201103L // since C++11 also forbid usage of move 481Â
mADK_PP_ProtBusyLock(const mADK_PP_ProtBusyLock&&); 482Â  mADK_PP_ProtBusyLock&
operator=(mADK_PP_ProtBusyLock&&); 483Â #endif 484Â  485Â  public: 490Â  mADK_PP_ProtBusyLock(bool cmd_ctx =
true) 491Â  { 492Â  m_proto = mADK_PP_Prot::getInstance(); 493Â  m_locked = m_proto->set_busy(true, cmd_ctx);
494Â  } 496Â  virtual ~mADK_PP_ProtBusyLock() 497Â  { 498Â  unlock(); 499Â  } 502Â  void unlock() 503Â  { 504Â
if (m_locked) 505Â  { 506Â  m_proto->set_busy(false); 507Â  m_locked = false; 508Â  } 509Â  } 512Â  bool
locked() const 513Â  { 514Â  return m_locked; 515Â  } 516Â }; 517Â  518Â #endif
```