sdi_if.h

Go to the documentation of this file.

```
1Â /********************************************************************************** 2Â * Product: ADK Secure Data Interface (SDI) 3Â * Company: Verifone 4Â * Author: GSS R&D Germany 5Â * Content: Client (structure) interface - non-EMV part 6Â **********************************************************************************/ 7Â #ifndef CLIENT_SDI_IF_H_ 8Â #define CLIENT_SDI_IF_H_ 9Â  10Â #ifndef DOXYGEN 11Â #ifdef __cplusplus 12Â #endif 13Â #include <vector> 14Â #include <string> 15Â #include <stdint.h> 16Â #include <emv/EMV_Common_Interface.h> // for EMV_ADK_INFO 17Â #include <sysinfo/sysinfo.h> // for property enumerations 18Â #include <html/prt.h> // for printer property enumerations 19Â  20Â namespace libsdi { 21Â  25Â enum SDI_SW12 26Â { 27Â  SDI_SW12_NONE = 0, 28Â  SDI_SW12_SUCCESS = 0x9000, 29Â  30Â  SDI_SW12_TAG_ERROR = 0x6200, 31Â  SDI_SW12_TAG_LENGTH_ERROR = 0x6300, 32Â  SDI_SW12_EXEC_ERROR = 0x6400, 33Â  SDI_SW12_CANCELED_BY_USER = 0x6405, 34Â  SDI_SW12_BUSY = 0x640A, 35Â  SDI_SW12_TIMEOUT_PIN_ENTRY = 0x640C, 36Â  SDI_SW12_TIMEOUT_NO_MSR_DATA = 0x64F6, 37Â  SDI_SW12_TIMEOUT_CARD_REMOVAL = 0x64F7, 38Â  SDI_SW12_INTERCHAR_PIN_ENTRY = 0x64F8, 39Â  SDI_SW12_COMMAND_NOT_ALLOWED = 0x64F9, 40Â  SDI_SW12_MAIN_CONNECTION_USED = 0x64FA, 41Â  SDI_SW12_INVALID_FILE_CONTENT = 0x64FB, 42Â  SDI_SW12_FILE_ACCESS_ERROR = 0x64FC, 43Â  SDI_SW12_LOGIC_ERROR = 0x64FD, 44Â  SDI_SW12_SDI_PARAMETER_ERROR = 0x64FE, 45Â  SDI_SW12_LUHN_CHECK_FAILED = 0x64FF, 46Â  SDI_SW12_EXECUTION_ABORTED = 0x6500, 47Â  SDI_SW12_EXECUTION_TIMEOUT = 0x6600, 48Â  SDI_SW12_MESSAGE_LENGTH_ERROR = 0x6700, 49Â  50Â  SDI_SW12_NO_SDI_PLUGIN_AVAILABLE = 0x6800, 51Â  SDI_SW12_UNKNOWN_PLUGIN_ID = 0x6801, 52Â  SDI_SW12_UNKNOWN_PLUGING_ID = 0x6801, 53Â  SDI_SW12_INVALID_PLUGIN_RESPONSE = 0x6802, 54Â  55Â  SDI_SW12_EPP_CONNECTION_ERROR = 0x6900, 56Â  57Â  SDI_SW12_UNKNOWN_INS_BYTE = 0x6D00, 58Â  SDI_SW12_UNKNOWN_CLA_BYTE = 0x6E00, 59Â  60Â  SDI_SW12_CMAC_ERROR = 0x6FB0, 61Â  SDI_SW12_CMAC_LENGTH_ERROR = 0x6FB1, 62Â  SDI_SW12_CMAC_MISSING_ERROR = 0x6FB2, 63Â  SDI_SW12_ENCRYPTION_ERROR = 0x6FB4, 64Â  SDI_SW12_ENCRYPTION_LENGTH_ERROR = 0x6FB5, 65Â  SDI_SW12_ENCRYPTION_MISSING_ERROR = 0x6FB6, 66Â  SDI_SW12_DECRYPTION_ERROR = 0x6FB8, 67Â  SDI_SW12_DECRYPTION_LENGTH_ERROR = 0x6FB9, 68Â  SDI_SW12_DECRYPTION_MISSING_ERROR = 0x6FBA, 69Â  70Â  SDI_SW12_EXCESSIVE_PIN_REQUESTS = 0x6FC0, 71Â  SDI_SW12_LOW_BATTERY = 0x6FD0, 72Â  SDI_SW12_NO_DUKPT_KEYS_LOADED = 0x6FE0, 73Â  SDI_SW12_UNIT_TAMPERED = 0x6FF0, 74Â  SDI_SW12_RECOVERY_MODE = 0x6FF1, 75Â  76Â  SDI_SW12_PIN_BYPASSED = 0x9070, 77Â  SDI_SW12_NO_MACTCH_FOR_CARD_VALIDATION = 0x9071, 78Â  SDI_SW12_SMART_CARD_REMOVED = 0x9401, 79Â  SDI_SW12_SMART_CARD_ERROR_TRM = 0x9402, 80Â  SDI_SW12_SMART_CARD_ERROR = 0x9403, 81Â  SDI_SW12_TWO_CARDS = 0x9404, 82Â  SDI_SW12_SMART_CARD_ERR_INIT = 0x9405, 83Â  SDI_SW12_SMART_CARD_ERR_PARAM = 0x9406, 84Â  SDI_SW12_EMV_TLV_ERROR = 0x94F3, // error In TLV data object (only in relation with EMV commands) 85Â  86Â  // There are from previous header version and kept for backward compatibility 87Â  SDI_SW12_ERROR = 0x6400, 88Â  SDI_SW12_TIMEOUT = 0x6600, 89Â  SDI_SW12_NOT_ALLOWED = 0x64FD, 90Â  SDI_SW12_PARAMETER_ERROR = 0x90E6 91Â }; 92Â  100Â enum SDICLIENT_ERROR 101Â { 102Â  SDICLIENT_ERROR_NONE = 0, 103Â  SDICLIENT_ERROR_COMMUNICATION = -1, 104Â  SDICLIENT_ERROR_CONCURRENT_USE = -2, 105Â  SDICLIENT_ERROR_CONNECT = -3, 106Â  SDICLIENT_ERROR_OVERFLOW = -4, 107Â  SDICLIENT_ERROR_PARAM = -5, 108Â  SDICLIENT_ERROR_OTHER = -6, 109Â  SDICLIENT_ERROR_NO_RECEIVE = -7, 110Â  SDICLIENT_ERROR_NOT_SUPPORTED = -10, 112Â  SDICLIENT_ERROR_NOT_ALLOWED = -11 113Â }; 114Â  115Â #define MSR_CLIENT_ERROR_OFFSET 100 116Â  117Â #define VALIDATION_CHECK_OPTION_RETURN_ALL_MATCHING_RANGES 0x01 //Return all matching card ranges. per default only the best match is returned 118Â  122Â enum SDICLIENT_ERROR getNfcClientError(); 123Â  127Â enum SDI_SW12 getNfcSW12(); 128Â  129Â struct MatchingRecord 130Â { 131Â  std::string json; 132Â  unsigned char expirycheck_result; 133Â  unsigned char luhncheck_result; 134Â  unsigned char activationcheck_result; 135Â  MatchingRecord(const std::string &_json): json(_json), expirycheck_result(0xFF), luhncheck_result(0xFF), activationcheck_result(0xFF){}; 136Â }; 137Â  139Â enum SYSUploadType { 140Â  SYS_UPLOAD_SOFTWARE_UPDATE, 141Â  SYS_UPLOAD_CONFIG_WHITELIST, 142Â  SYS_UPLOAD_CONFIG_SENSITIVE_TAGS, 143Â  SYS_UPLOAD_CONFIG_CARD_RANGES, 144Â  SYS_UPLOAD_INSTALL_CP_PACKAGE = 11, 145Â  SYS_UPLOAD_EMV_CONFIGURATION 146Â }; 147Â  148Â  152Â class SdiBase 153Â { 154Â protected: 155Â  unsigned short sw12; 156Â  int additionalResultValue; 157Â  SDICLIENT_ERROR clientErr; 158Â  void setSdiSw12(enum SDI_SW12 s) {sw12=s;} 159Â  166Â  void setClientError(int libsdiprotocol_result); 167Â  168Â public: 169Â  SdiBase() : sw12(0), additionalResultValue(0), clientErr(SDICLIENT_ERROR_NONE){} 170Â  enum SDI_SW12 getSdiSw12(); 171Â  176Â  int getAdditionalResultValue(){return additionalResultValue;} 177Â  184Â  SDICLIENT_ERROR getClientError(){return clientErr;} 185Â  196Â  enum SDI_SW12 receiveSW12(); 197Â  201Â  void clear() 202Â  { 203Â  sw12 = SDI_SW12_NONE; 204Â  additionalResultValue = 0; 205Â  clientErr = SDICLIENT_ERROR_NONE; 206Â  }; 207Â  211Â  void importResults(const SdiBase& intermediate) 212Â  { 213Â  sw12 = intermediate.sw12; 214Â  additionalResultValue = intermediate.additionalResultValue; 215Â  clientErr = intermediate.clientErr; 216Â  } 217Â  218Â  struct PluginResult 219Â  { 220Â  int32_t pluginId; 221Â  int32_t responseCode; 224Â  std::vector<unsigned char> responseData; 226Â  }; 227Â  228Â private: 229Â  // make SdiBase non-copyable because some subclasses hold pointers and should not be copied 230Â  SdiBase(const SdiBase&); 231Â  SdiBase& operator=(const SdiBase&); 232Â  // since C++11 we have to avoid the move, too 233Â #if __cplusplus >= 201103L 234Â  SdiBase(const SdiBase&&); 235Â  SdiBase& operator=(SdiBase&&); 236Â #endif 237Â }; 238Â  239Â  240Â  266Â class SdiCmd: public SdiBase 267Â { 268Â protected: 269Â  void* dataIn; 270Â  void* dataOut; 271Â  272Â public: 273Â  SdiCmd(); 274Â  virtual ~SdiCmd(); 275Â  293Â  enum SDI_SW12 sendReceive(unsigned char cla, unsigned char ins, 294Â  unsigned char p1 = 0, unsigned char p2 = 0, 295Â  unsigned maxResponseSize = 2048); 296Â  309Â  int send(unsigned char cla, unsigned char ins, 310Â  unsigned char p1 = 0, unsigned char p2 = 0); 311Â  320Â  enum SDI_SW12 receive(unsigned maxResponseSize = 2048); 321Â  322Â  // generic setters modify #dataIn before command execution 335Â  void set(const char* path, int value, unsigned fixedLength = 0); 336Â  349Â  void set(const char* path, uint32_t value, unsigned fixedLength = 0); 350Â  361Â  void set(const char* path, const unsigned char* data, unsigned dataLen); 362Â  372Â  inline void set(const char* path, unsigned char byteValue) 373Â  { 374Â  set(path, &byteValue, 1); 375Â  } 376Â  386Â  void set(const char* path, const std::vector<unsigned char>&
```

```
data); 387Â  397Â  void set(const char* path, const std::string& value); 398Â  407Â  void
setCommandDestination(bool epp, bool force = false); 408Â  412Â  virtual void clear(); 413Â  420Â  virtual void
clear(const char* path); 421Â  425Â  virtual void clearResults(); 426Â  434Â  void importResults(const SdiCmd&
intermediate); 435Â  443Â  void importResults(const unsigned char* sw12, const unsigned char* tlvData, unsigned
tlvSize); 444Â  445Â  // generic getter to read from dataOut after command execution 453Â  bool get(const char*
path, int& value); 454Â  462Â  bool get(const char* path, uint32_t& value); 463Â  472Â  int get(const char*
path, unsigned char* buffer, unsigned bufferSize); 473Â  481Â  inline bool get(const char* path, unsigned char&
value) 482Â  { 483Â  return get(path, &value, 1) == 1; 484Â  } 485Â  493Â  bool get(const char* path,
std::vector<unsigned char>& buffer); 494Â  502Â  bool get(const char* path, std::string& value); 503Â  510Â
std::string getString(const char* path); 511Â }; 512Â  513Â class SDI: public SdiBase 514Â { 515Â public: 516Â
526Â  bool getDateTime(unsigned char* buffer); 527Â  537Â  bool setDateTime(const unsigned char* dateTime);
538Â  546Â  unsigned char getLanguage(); 547Â  555Â  bool setLanguage(unsigned char languageId); 556Â  569Â
unsigned char getCardDataEntryDeactivation(); 570Â  583Â  bool setCardDataEntryDeactivation(unsigned char
value); 584Â  594Â  unsigned char getCardDataEntryMode(); 595Â  605Â  bool setCardDataEntryMode(unsigned char
value); 606Â  614Â  bool setIdleText(std::string text, bool epp = true); 615Â  624Â  bool showMacDesktop(bool
epp = true); 625Â  636Â  int getProperty(enum vfisysinfo::SYSPropertyInt p, bool epp = false); // 20 1A 00 00
637Â  648Â  enum vfisysinfo::sysError getProperty(enum vfisysinfo::SYSPropertyInt p, int& value, bool epp =
false); // 20 1A 00 00 649Â  660Â  enum vfisysinfo::sysError setProperty(enum vfisysinfo::SYSPropertyInt p, int
value, bool epp = false); // 20 19 00 00 661Â  672Â  std::string getProperty(enum vfisysinfo::SYSPropertyString
p, bool epp = false); // 20 1A 00 01 673Â  684Â  enum vfisysinfo::sysError getProperty(enum
vfisysinfo::SYSPropertyString p, std::string& value, bool epp = false); 685Â  696Â  enum vfisysinfo::sysError
setProperty(enum vfisysinfo::SYSPropertyString p, const std::string& value, bool epp = false); // 20 19 00 01
697Â  704Â  std::string getVersionInfo(bool epp = false); 705Â  712Â  bool checkForUpdate(unsigned char
updateKind); 713Â  720Â  bool waitCardRemoval(unsigned seconds); 721Â  730Â  int sendWaitCardRemoval(unsigned
seconds); 731Â  732Â  // Data Interface (without crypto handle) 733Â  737Â  void clearDataStore(); 738Â  749Â
bool vclRegistartSRED(std::vector<unsigned char>& track1, 750Â  std::vector<unsigned char>& track2, 751Â
std::vector<unsigned char>& track3); 752Â  760Â  bool vclStatusRequest(std::vector<unsigned char>& vclStatus);
761Â  772Â  bool vclAdvanceDDK(std::vector<unsigned char>& track1, 773Â  std::vector<unsigned char>& track2,
774Â  std::vector<unsigned char>& track3); 775Â  787Â  bool vclRequestEParms(std::vector <unsigned char>&
eParms); 788Â  799Â  std::string vclGetDiagnosticData(unsigned char format = 1, 800Â  unsigned char pageNumber =
0); 801Â  808Â  int vclGetKeyDerivationMode(); 809Â  820Â  bool vclOverrideMessageQuery(std::vector<unsigned
char>& track1, 821Â  std::vector<unsigned char>& track2, 822Â  std::vector<unsigned char>& track3); 823Â  832Â
bool vclKsnRequest(std::string& ksn); 833Â  841Â  bool vclKmailinRequest(); 842Â  853Â  libsdi::SDI_SW12
pluginCommand(const unsigned char plugin, const unsigned char cmd, 854Â  const std::vector<unsigned char>& data,
855Â  std::vector<unsigned char>& response); 856Â  866Â  bool setManualPAN(const std::vector<unsigned char>&
pan); 867Â  877Â  bool setManualPAN(const std::string& pan); 878Â  892Â  libsdi::SDI_SW12
performValidationChecks(std::vector<MatchingRecord>& records, 893Â  const std::vector<unsigned char>&
currentDate, 894Â  const unsigned char options = 0, 895Â  const std::vector<unsigned char> IIN = std::vector<
unsigned char>()); 896Â  904Â  libsdi::SDI_SW12 getValidationInfo(std::string& info); 905Â  915Â
libsdi::SDI_SW12 sysUploadStart(const std::string& filename, 916Â  SYSUploadType type, 917Â  bool epp = false);
918Â  927Â  libsdi::SDI_SW12 sysUploadTransfer(const std::vector<unsigned char>& packet, 928Â  unsigned
packet_no, 929Â  bool epp = false); 930Â  942Â  libsdi::SDI_SW12 sysUploadFinalize(const std::vector<unsigned
char>* md5 = NULL, 943Â  const std::vector<unsigned char>* mac = NULL, 944Â  bool epp = false); 945Â  952Â  bool
sysShutdown(bool epp = false); 953Â  960Â  bool sysReboot(bool epp = false); 961Â  969Â  bool sysSleep(bool epp
= false); 970Â  978Â  bool sysHybernate(bool epp = false); 979Â  985Â  bool installSponsorCert(const std::vector
<unsigned char>& cert); 986Â  996Â  bool getLastInstallError(std::string& json, std::string& bundle,
std::string& package, bool epp = false); 997Â  998Â  1003Â  void externalButton(void); 1004Â  1017Â  bool
enableEpp(int& status); 1018Â  1031Â  bool disableEpp(int& status); 1032Â  1037Â  bool factoryReset(); 1038Â
1045Â  bool readCertificate(const std::string& name, std::vector<unsigned char>& certificate); 1046Â  1053Â
bool setUseCurrencyAbbreviation(const unsigned char* currency, bool useAbbreviation); 1054Â  1061Â  bool
setUseCurrencyAbbreviation(unsigned currency, bool useAbbreviation); 1062Â  1072Â  bool msrSwitchLeds(int led1,
int led2, int led3, int duration); 1073Â  1080Â  bool msrSetSensitivty(unsigned char level, bool epp = false);
1081Â  1082Â  1092Â  enum vfiprt::PrtError setPrinterProperty(enum vfiprt::PrtPropertyInt p, int value); // 25
00 00 00 1093Â  1103Â  enum vfiprt::PrtError setPrinterProperty(enum vfiprt::PrtPropertyString p, const
std::string& value); // 25 00 00 01 1104Â  1114Â  enum vfiprt::PrtError getPrinterProperty(enum
vfiprt::PrtPropertyInt p, int& value); // 25 01 00 00 1115Â  1125Â  enum vfiprt::PrtError
getPrinterProperty(enum vfiprt::PrtPropertyString p, std::string& value); // 25 01 00 01 1126Â  1134Â  enum
vfiprt::PrtError printHtml(const std::string& html, bool landscape = false); 1135Â  1144Â  enum vfiprt::PrtError
printBitmap(const int width, const int height, const std::vector<unsigned char>& data); 1145Â  1146Â private:
1147Â  bool sysPower(unsigned char p2, bool epp); 1148Â  bool enableEpp(int& status, unsigned char p2); 1149Â
1150Â }; // class SDI 1151Â  1152Â  1171Â class CardDetection: public SdiCmd 1172Â { 1173Â public: 1174Â  const
static unsigned char SDI_TEC_CHIP = 1; // ISO 7816 1175Â  const static unsigned char SDI_TEC_MAGN = 2; // ISO
7810 1176Â  const static unsigned char SDI_TEC_CTLS = 4; // ISO 14443 1177Â  const static unsigned char
SDI_TEC_MANU = 8; // ITU-T E.161 1178Â  1183Â  enum DetectionMode { 1184Â  DETECTION_MODE_BLOCKING, 1185Â
DETECTION_MODE_POLLING, 1186Â  DETECTION_MODE_CALLBACK 1187Â  }; 1188Â  1194Â  void setDetectionMode(const enum
DetectionMode mode); 1195Â  1201Â  void setTecStartOptions(const std::vector<unsigned char>& opts); // see
cts_StartSelection 1202Â  1208Â  void setTecConfig(const std::vector<unsigned char>& opts); // see
cts_SetOptions 1209Â  1215Â  void setCancelButton(bool enable); 1216Â  1222Â  void
```

```
setCardEntryValueDeativation(unsigned char b); 1223Â  1229Â  void setAlternativeInputFormat(const char* f);
1230Â  1241Â  void setCardRemovalTimeout(unsigned timeout_ms); 1242Â  1249Â  void setCallback(void
(*cb)(unsigned char technology, void* context), void* ctx); 1250Â  1251Â  1258Â  void
cardDetectedCallback(unsigned char *dataIn, unsigned short sizeIn); 1259Â  1260Â  1291Â  int
startSelection(unsigned char supportedTechnologies, unsigned seconds); 1292Â  1302Â  enum libsdi::SDI_SW12
stopSelection(); 1303Â  1341Â  unsigned char receiveTechnology(); 1342Â  1351Â  bool cardReadAtEpp(); 1352Â
1361Â  unsigned char pollTechnology(); 1362Â  1370Â  enum libsdi::SDI_SW12 addTechnology(unsigned char
technology, const std::vector<unsigned char>& opts); 1371Â  1378Â  enum libsdi::SDI_SW12
removeTechnology(unsigned char technology); 1379Â  1384Â  std::string getPan(); 1385Â  1390Â  std::string
getTrack2(); 1391Â  1397Â  int getTrack2Bin(std::vector<unsigned char>& data); 1398Â  1403Â  std::string
getCardholderName(); 1404Â  1409Â  std::string getServiceCode(); 1410Â  1415Â  std::string getTrack1(); 1416Â
1425Â  int getPluginResponseData(std::vector<unsigned char>& data); 1426Â  1440Â  int
getPluginResponseData(unsigned index, 1441Â  int32_t& pluginId, 1442Â  int32_t& pluginResponseCode, 1443Â
std::vector<unsigned char>& data); 1444Â  1451Â  bool getPluginResponseData(std::vector<PluginResult>& results);
1452Â  1459Â  std::string getString(unsigned CTS_DATA_TAG); 1460Â  1476Â  unsigned char getValue(unsigned
CTS_DATA_TAG, unsigned char defaultValue); 1477Â  1485Â  unsigned char getValue(unsigned CTS_DATA_TAG, int n,
unsigned char defaultValue); 1486Â  1492Â  int getData(std::vector<unsigned char>& data); 1493Â  1501Â  int
getData(unsigned CTS_DATA_TAG, std::vector<unsigned char>& data); 1502Â  1512Â  int getData(unsigned
CTS_DATA_TAG, int n, std::vector<unsigned char>& data); 1513Â  1522Â  int getTecselData(std::vector<unsigned
char>& data); 1523Â  1538Â  int startMsrRead(unsigned timeout_sec); 1539Â  1551Â  int msrSetOptions(const
std::vector<unsigned char>& opts); 1552Â  1561Â  int msrGetTrackStatus(int track); 1562Â  1569Â  int
msrGetCardSpecificToken(std::vector<unsigned char>& token); 1570Â  1571Â  CardDetection(); 1572Â
~CardDetection(); 1573Â  1581Â  virtual void clear(); 1582Â  1583Â private: 1584Â  void (*callback)(unsigned
char technology, void* context); 1585Â  void* context; 1586Â  enum DetectionMode mode; 1587Â }; 1588Â
1592Â class PED: public SdiCmd 1593Â { 1594Â public: 1598Â  enum NavigatorMode 1599Â  { 1600Â
NAVIGATOR_MODE_OFF, 1601Â  NAVIGATOR_MODE_DOUBLE_TAB, 1602Â  NAVIGATOR_MODE_TACTILE_BUTTON 1603Â  }; 1604Â
1613Â  void setTimeout(unsigned seconds); 1614Â  1622Â  bool setDefaultTimeout(unsigned seconds); 1623Â  1639Â
void setTouchCoordinates(const unsigned char* array, unsigned size); 1640Â  1645Â  void setNavigatorMode(enum
NavigatorMode mode); 1646Â  1657Â  void setPinDigitCountMinMax(unsigned char min, unsigned char max); 1658Â
1665Â  void setLanguage(unsigned char lang); 1666Â  1673Â  void setAmount(const unsigned char* amount); 1674Â
1681Â  void setCurrency(const unsigned char* currency); 1682Â  1690Â  void setAppLabel(const std::string&
appLabel); 1691Â  1692Â  1719Â  int startPinInput(); 1720Â  1731Â  int startPinInput(bool enablePinBypass);
1732Â  1752Â  bool startPinEntry(unsigned pinBypassKey = 0); 1753Â  1760Â  bool pollPinEntry(std::vector<
unsigned char>& status); 1761Â  1767Â  bool stopPinEntry(); 1768Â  1777Â  bool setPinInputClearKeyMode(bool
clearAllDigits); 1778Â  1785Â  void setClearKeyMode(bool clearAllDigits); 1786Â  1793Â  void
setPinBypassKeyAndMode(unsigned char value); 1794Â  1801Â  void setAutoConfirmation(bool enable); 1802Â  1809Â
void setDialogOptions(uint32_t options); 1810Â  1817Â  void setIntercharTimeout(uint32_t millis); 1818Â  1827Â
void setHeaderLabel(const std::string& label); 1828Â  1835Â  void setEchoChar(uint32_t unicode); 1836Â  1855Â
bool sendPinInputParameters(bool epp = true); 1856Â  1863Â  enum SDI_SW12 receiveGetPinResult(); 1864Â  1873Â
bool getPinBypassKey(unsigned char& value); 1874Â }; 1875Â  1884Â class SdiCrypt: public SdiBase 1885Â {
1886Â public: 1887Â  // SDI Crypto Interface 1888Â  SdiCrypt(); 1889Â  ~SdiCrypt(); 1890Â  1899Â  bool
open(const char* hostName); 1900Â  1907Â  bool close(); 1908Â  1914Â  bool isOpen() const; 1915Â  1922Â
uint32_t getCryptoHandle(); 1923Â  1935Â  bool encrypt(const std::vector<unsigned char>& data, 1936Â
std::vector<unsigned char>& encrypted); 1937Â  1949Â  bool decrypt(const std::vector<unsigned char>& encrypted,
1950Â  std::vector<unsigned char>& decrypted); 1951Â  1963Â  bool sign(const std::vector<unsigned char>& data,
1964Â  std::vector<unsigned char>& signature); 1965Â  1977Â  bool verify(const std::vector<unsigned char>& data,
1978Â  const std::vector<unsigned char>& signature); 1979Â  1994Â  bool updateKey(unsigned char keyType, const
std::vector<unsigned char>& keyData, 1995Â  std::vector<unsigned char>* proprietaryData = NULL, 1996Â  const
unsigned char AS2805 = 0, 1997Â  std::vector<unsigned char>* KCV = NULL); 1998Â  2010Â  bool
setKeySetId(uint32_t ksid, uint32_t mksid = 0, bool asAttribute = false); 2011Â  2020Â  bool
getEncryptedPin(unsigned char pinBlockFormat, 2021Â  std::vector<unsigned char>& pinBlock, 2022Â  bool
requestZeroPinBlock = false); 2023Â  2029Â  std::string getKeyInventory(); 2030Â  2039Â  bool
getKeyData(unsigned char keyType, std::vector<unsigned char>& keyData, unsigned char kekFlag = 0); 2040Â  2047Â
std::string getStatus(); 2048Â  2058Â  std::string getStatus(std::string hostName); 2059Â  2060Â  2067Â  static
std::string getVersions(int& additionalResult); 2068Â  2079Â  void setInitialVector(const std::vector<unsigned
char>& iv) 2080Â  { 2081Â  initialVector = iv; 2082Â  } 2083Â  2089Â  void getInitialVector(std::vector<unsigned
char>& iv) const 2090Â  { 2091Â  iv = initialVector; 2092Â  } 2093Â  2094Â  2100Â  void
getKeySerialNumber(std::vector<unsigned char>& ksn) const 2101Â  { 2102Â  ksn = keySerialNumber; 2103Â  } 2104Â
2108Â  struct Placeholder 2109Â  { 2110Â  std::vector<unsigned char> tagList; 2111Â  std::vector<unsigned char>
applicationData; 2112Â  std::vector<unsigned char> dataOptions; 2113Â  }; 2114Â  2129Â  bool getEncData(const
Placeholder& descriptor, 2130Â  std::vector<unsigned char>& encrypted, 2131Â  bool useStoredData = false, 2132Â
bool incrementKSN = false); 2133Â  2149Â  bool getEncMsgData(const std::vector<unsigned char>& messageTemplate,
2150Â  const std::vector<Placeholder>& placeholder, 2151Â  std::vector<unsigned char>& encrypted, 2152Â  bool
useStoredData = false, 2153Â  bool incrementKSN = false); 2154Â  2170Â  bool getMsgSignature(const std::vector<
unsigned char>& messageTemplate, 2171Â  const std::vector<Placeholder>& placeholder, 2172Â  std::vector<unsigned
char>& signature, 2173Â  bool useStoredData = false, 2174Â  bool incrementKSN = false); 2175Â  2183Â  bool
getEncTrxData(const std::vector<unsigned long> tags, std::vector<unsigned char>& data); 2184Â  2191Â  bool
setEncTrxData(const std::vector<unsigned char> data); 2192Â  2198Â  bool endEncTrxData(); 2199Â  2200Â private:
```

```
2201Â  unsigned char cryptoHandle[4]; 2202Â  std::vector<unsigned char> initialVector; 2203Â  std::vector<
unsigned char> keySerialNumber; 2204Â  uint32_t keySetId; 2205Â  uint32_t masterKeySetId; 2206Â }; 2207Â
2217Â class ManualEntry: public SdiCmd 2218Â { 2219Â public: 2220Â  ManualEntry(); 2221Â  ~ManualEntry(); 2222Â
2229Â  void setLanguage(unsigned char language); 2230Â  2231Â  2238Â  int setTimeout(unsigned seconds); 2239Â
2253Â  int setCvvEntryDeactivation(unsigned char cvvEntry); 2254Â  2255Â  2264Â  int
setCardDataEntryMode(unsigned char mode); 2265Â  2266Â  2273Â  int setCvvInputString(const std::string
&inputString); 2274Â  2290Â  int setTouchCoordinates(const unsigned char* array, unsigned size); 2291Â  2301Â
int setMimimumDigits(unsigned char minimumDigits); 2302Â  2311Â  int setDoubleConfirmationMode(unsigned char
mode); 2312Â  2333Â  int start(); 2334Â  2351Â  int receiveResult(std::string& obfuscatedPAN, std::vector<
unsigned char>& token); 2352Â  2361Â  int getPluginResponseData(std::vector<unsigned char>& data); 2362Â  2376Â
int getPluginResponseData(unsigned index, 2377Â  int32_t& pluginId, 2378Â  int32_t& pluginResponseCode, 2379Â
std::vector<unsigned char>& data); 2380Â  2387Â  bool getPluginResponseData(std::vector<PluginResult>& results);
2388Â }; 2389Â  2435Â class Dialog: public SdiCmd 2436Â { 2437Â public: 2438Â  2445Â  2451Â  const static int
DIALOG_SUCCESS = 0; 2452Â  2458Â  const static int DIALOG_CANCEL_PRESSED = -1; 2459Â  2465Â  const static int
DIALOG_TIMEOUT = -3; 2466Â  2473Â  const static int DIALOG_SYS_ABORT = -8; 2474Â  2480Â  const static int
DIALOG_CLIENT_ERROR = -14; 2481Â  2487Â  const static int DIALOG_SDI_SW12 = -15; 2488Â  2493Â  const static int
DIALOG_NO_ASYNC_DIALOG = 1; 2494Â  2499Â  const static int DIALOG_IN_PROGRESS = 2; 2500Â  2502Â  2509Â  const
static unsigned DLG_DisplayOnly = 0x00000000; 2510Â  const static unsigned DLG_CancelKey = 0x00000002; 2511Â
const static unsigned DLG_ClearKey = 0x00000004; 2512Â  const static unsigned DLG_EnterKey = 0x00000008; 2513Â
const static unsigned DLG_NoLEDs = 0x00000010; 2514Â  const static unsigned DLG_CtlsLogo = 0x00000020; 2515Â
const static unsigned DLG_QuestionLogo = 0x00000040; 2516Â  const static unsigned DLG_WarningLogo = 0x00000080;
2517Â  const static unsigned DLG_ErrorLogo = 0x00000100; 2518Â  const static unsigned DLG_SuccessLogo =
0x00000200; 2519Â  const static unsigned DLG_WaitLogo = 0x00000400; 2520Â  const static unsigned DLG_Async =
0x00000800; 2521Â  const static unsigned DLG_StoreAsyncResult = 0x00001000; 2522Â  const static unsigned
DLG_HideSoftKeys = 0x00002000; 2523Â  const static unsigned DLG_TextAlignLeft = 0x00004000; 2524Â  const static
unsigned DLG_TextAlignRight = 0x00008000; 2525Â  const static unsigned DLG_TextAlignTop = 0x00010000; 2526Â
const static unsigned DLG_TextAlignButton = 0x00020000; 2527Â  const static unsigned DLG_NoHeader = 0x00040000;
2528Â  const static unsigned DLG_ClearOnReturn = 0x00100000; 2529Â  const static unsigned DLG_AbortOnCardRemove
= 0x00200000; 2530Â  const static unsigned DLG_IgnoreExtAbort = 0x00400000; 2532Â  2539Â  const static unsigned
MENU_NoOpts = 0x00000000; 2540Â  const static unsigned MENU_NoLEDs = 0x00000002; 2541Â  const static unsigned
MENU_ClearOnReturn = 0x00000004; 2542Â  const static unsigned MENU_AbortOnCardRemove= 0x00000008; 2543Â  const
static unsigned MENU_IgnoreExtAbort = 0x00000010; 2544Â  const static unsigned MENU_NoHeader = 0x00000020;
2546Â  2550Â  enum REQUEST_CARD_MODE 2551Â  { 2552Â  REQUEST_CARD_STANDARD = 0, 2553Â  REQUEST_CARD_FALLBACK =
1, 2554Â  REQUEST_CARD_RETRY = 2 2555Â  }; 2556Â  2565Â  bool clearScreen(bool epp = true); 2566Â  2590Â  int
display(const std::string& text, bool epp = true); 2591Â  2615Â  int secureInput(bool epp = true); 2616Â  2648Â
int menu(bool epp = true); 2649Â  2667Â  int requestCard(unsigned char technology, 2668Â  enum REQUEST_CARD_MODE
mode = REQUEST_CARD_STANDARD, 2669Â  bool epp = true); 2670Â  2677Â  bool idleScreen(bool epp = true); 2678Â
2694Â  int captureSignature(std::vector<unsigned char>& signature, std::string& format, 2695Â  bool epp = true);
2696Â  2703Â  static void showLedArea(bool show); 2704Â  2726Â  int htmlDialog(const std::string& fname, bool
epp = true); 2727Â  2739Â  int getAsyncResult(bool epp = true); 2740Â  2744Â  static void abort(); 2745Â  2754Â
void setTemplate(unsigned char id); 2755Â  2764Â  void setInputTemplate(unsigned char id); 2765Â  2775Â  void
setLanguage(unsigned char language); 2776Â  2788Â  void addData(const std::string& name, const std::string&
value); 2789Â  2800Â  void setTimeout(int seconds); 2801Â  2814Â  void setOptions(unsigned options); 2815Â
2825Â  void setEnterKeyLabel(const std::string& label); 2826Â  2836Â  void setClearKeyLabel(const std::string&
label); 2837Â  2847Â  void setCancelKeyLabel(const std::string& label); 2848Â  2858Â  void setUpKeyLabel(const
std::string& label); 2859Â  2868Â  void setDownKeyLabel(const std::string& label); 2869Â  2878Â  void
setHeader(const std::string& text); 2879Â  2888Â  void setBeep(bool active = true); 2889Â  2899Â  void
addAction(const std::string& key, const std::string& action); 2900Â  2909Â  void setMaskingCharacter(char c);
2910Â  2919Â  void setAlternativeInputFormat(const char* f); 2920Â  2929Â  void setAmount(const unsigned char*
amount); 2930Â  2941Â  void setCurrency(const unsigned char* currency); 2942Â  2953Â  void
setTransactionType(unsigned char txnType); 2954Â  2963Â  void setMenuText(const std::string& text); 2964Â
2973Â  void addMenuItem(const std::string& item); 2974Â  2985Â  void setPreSelected(unsigned char itemNumber);
2986Â  3001Â  void setAdminMenu(unsigned char adminMenu); 3002Â  3011Â  unsigned char getSelected(); 3012Â
3022Â  std::string get(const std::string& name); 3023Â  3036Â  std::vector<std::string> getInputFieldNames();
3037Â  3038Â private: 3039Â  int startDialog(unsigned char instruction, bool epp); 3040Â }; 3041Â  3042Â } //
namespace sdi 3043Â  3044Â #ifndef DOXYGEN 3045Â #endif // C++ 3046Â #endif 3047Â #endif /* CLIENT_SDI_IF_H_ */
```