



# **IDTech Universal SDK Guide for C / C++ / Java Developers**

**#80144504-001**

**Rev. A**

## Revision History

Revision	Description and Reason for Change	Date
A	Initial Release - Manual;User;C/C++/JAVA;SDK	02/14/2017

# Contents

<b>1</b>	<b>ID TECH Universal SDK Reference Guide for Linux/Windows/Mac (C++/Java)</b>	<b>1</b>
1.1	Demo Apps . . . . .	1
1.2	Purpose . . . . .	2
<b>2</b>	<b>Important Security Notice</b>	<b>3</b>
2.1	Applicability . . . . .	3
2.2	What Does PA-DSS Mean to You? . . . . .	3
2.3	Third Party Applications . . . . .	4
2.4	PA-DSS Guidelines . . . . .	4
2.5	More Information . . . . .	9
<b>3</b>	<b>Main Transaction Commands</b>	<b>11</b>
3.1	EMV Methods . . . . .	11
3.2	MSR Methods . . . . .	12
<b>4</b>	<b>Core Implementation: C/C++</b>	<b>13</b>
4.1	Integrating with IDTechSDK . . . . .	13
4.2	Import the Necessary Libraries . . . . .	13
4.3	Add Include Statements to Use Libraries . . . . .	14
4.4	Implement the Callback Function . . . . .	14
4.5	Initialize SDK and Set the Target Device . . . . .	14
<b>5</b>	<b>Core Implementation: Java</b>	<b>15</b>
5.1	Integrating with IDTechSDK . . . . .	15
5.2	Add the Necessary Libraries . . . . .	15
5.3	Add Import Statements to Use Libraries . . . . .	15
5.4	Implement the Callback Function . . . . .	16
5.5	Initialize SDK . . . . .	16
<b>6</b>	<b>LCD Foreign Language Mapping Table</b>	<b>18</b>
<b>7</b>	<b>Error Code Reference</b>	<b>21</b>
<b>8</b>	<b>Enumeration Reference</b>	<b>26</b>

<b>9</b>	<b>EMV Callback</b>	<b>32</b>
<b>10</b>	<b>EMV Tag Reference</b>	<b>33</b>
<b>11</b>	<b>File Index</b>	<b>47</b>
11.1	File List . . . . .	47
<b>12</b>	<b>File Documentation</b>	<b>48</b>
12.1	Source_C/libIDT_Augusta.h File Reference . . . . .	48
12.1.1	Detailed Description . . . . .	51
12.1.2	Macro Definition Documentation . . . . .	51
12.1.2.1	IN . . . . .	51
12.1.2.2	IN_OUT . . . . .	51
12.1.2.3	OUT . . . . .	51
12.1.3	Typedef Documentation . . . . .	51
12.1.3.1	ftpComm_callBack . . . . .	51
12.1.3.2	httpComm_callBack . . . . .	51
12.1.3.3	pCMR_callBack . . . . .	51
12.1.3.4	pCSFS_callBack . . . . .	51
12.1.3.5	pEMV_callBack . . . . .	51
12.1.3.6	pFW_callBack . . . . .	52
12.1.3.7	pMessageHotplug . . . . .	52
12.1.3.8	pMSR_callBack . . . . .	52
12.1.3.9	pMSR_callBackp . . . . .	52
12.1.3.10	pPIN_callBack . . . . .	52
12.1.3.11	pReadDataLog . . . . .	52
12.1.3.12	pSendDataLog . . . . .	52
12.1.3.13	v4Comm_callBack . . . . .	52
12.1.4	Function Documentation . . . . .	52
12.1.4.1	config_getBeeperController . . . . .	52
12.1.4.2	config_getEncryptionControl . . . . .	53
12.1.4.3	config_getLEDController . . . . .	53
12.1.4.4	config_getModelNumber . . . . .	54
12.1.4.5	config_getModelNumber_Len . . . . .	54
12.1.4.6	config_getSerialNumber . . . . .	54
12.1.4.7	config_getSerialNumber_Len . . . . .	54
12.1.4.8	config_setBeeperController . . . . .	55
12.1.4.9	config_setEncryptionControl . . . . .	55
12.1.4.10	config_setLEDController . . . . .	55
12.1.4.11	device_close . . . . .	56
12.1.4.12	device_controlBeep . . . . .	56

12.1.4.13 device_controlLED . . . . .	57
12.1.4.14 device_controlLED_ICC . . . . .	57
12.1.4.15 device_controlLED_MSR . . . . .	57
12.1.4.16 device_getCurrentDeviceType . . . . .	58
12.1.4.17 device_getDRS . . . . .	58
12.1.4.18 device_getFirmwareVersion . . . . .	59
12.1.4.19 device_getFirmwareVersion_Len . . . . .	60
12.1.4.20 device_getKeyStatus . . . . .	60
12.1.4.21 device_getResponseCodeString . . . . .	61
12.1.4.22 device_getSDKWaitTime . . . . .	71
12.1.4.23 device_getThreadStackSize . . . . .	71
12.1.4.24 device_init . . . . .	71
12.1.4.25 device_isAttached . . . . .	71
12.1.4.26 device_isConnected . . . . .	71
12.1.4.27 device_rebootDevice . . . . .	72
12.1.4.28 device_registerCameraCallbk . . . . .	72
12.1.4.29 device_registerCardStatusFrontSwitchCallbk . . . . .	72
12.1.4.30 device_registerFWCallbk . . . . .	72
12.1.4.31 device_SendDataCommand . . . . .	72
12.1.4.32 device_setCurrentDevice . . . . .	72
12.1.4.33 device_setSDKWaitTime . . . . .	73
12.1.4.34 device_setThreadStackSize . . . . .	73
12.1.4.35 device_updateFirmware . . . . .	73
12.1.4.36 emv_activateTransaction . . . . .	74
12.1.4.37 emv_allowFallback . . . . .	75
12.1.4.38 emv_authenticateTransaction . . . . .	76
12.1.4.39 emv_authenticateTransactionWithTimeout . . . . .	76
12.1.4.40 emv_callbackResponseLCD . . . . .	77
12.1.4.41 emv_callbackResponseMSR . . . . .	77
12.1.4.42 emv_cancelTransaction . . . . .	77
12.1.4.43 emv_completeTransaction . . . . .	77
12.1.4.44 emv_getAutoAuthenticateTransaction . . . . .	78
12.1.4.45 emv_getAutoCompleteTransaction . . . . .	78
12.1.4.46 emv_getEMVConfigurationCheckValue . . . . .	78
12.1.4.47 emv_getEMVKernelCheckValue . . . . .	78
12.1.4.48 emv_getEMVKernelVersion . . . . .	79
12.1.4.49 emv_getEMVKernelVersion_Len . . . . .	79
12.1.4.50 emv_registerCallbk . . . . .	79
12.1.4.51 emv_removeAllApplicationData . . . . .	79
12.1.4.52 emv_removeAllCAPK . . . . .	79

12.1.4.53 emv_removeAllCRL . . . . .	80
12.1.4.54 emv_removeApplicationData . . . . .	80
12.1.4.55 emv_removeCAPK . . . . .	80
12.1.4.56 emv_removeCRL . . . . .	80
12.1.4.57 emv_removeTerminalData . . . . .	81
12.1.4.58 emv_retrieveAIDList . . . . .	81
12.1.4.59 emv_retrieveApplicationData . . . . .	81
12.1.4.60 emv_retrieveCAPK . . . . .	82
12.1.4.61 emv_retrieveCAPKList . . . . .	82
12.1.4.62 emv_retrieveCRL . . . . .	83
12.1.4.63 emv_retrieveTerminalData . . . . .	84
12.1.4.64 emv_retrieveTerminalID . . . . .	84
12.1.4.65 emv_retrieveTerminalID_Len . . . . .	84
12.1.4.66 emv_retrieveTransactionResult . . . . .	84
12.1.4.67 emv_setApplicationData . . . . .	85
12.1.4.68 emv_setAutoAuthenticateTransaction . . . . .	85
12.1.4.69 emv_setAutoCompleteTransaction . . . . .	85
12.1.4.70 emv_setCAPK . . . . .	85
12.1.4.71 emv_setCRL . . . . .	86
12.1.4.72 emv_setTerminalData . . . . .	86
12.1.4.73 emv_setTerminalID . . . . .	87
12.1.4.74 emv_startTransaction . . . . .	87
12.1.4.75 icc_disable . . . . .	87
12.1.4.76 icc_enable . . . . .	88
12.1.4.77 icc_exchangeAPDU . . . . .	89
12.1.4.78 icc_exchangeEncryptedAPDU . . . . .	89
12.1.4.79 icc_getAPDU_KSN . . . . .	91
12.1.4.80 icc_getFunctionStatus . . . . .	91
12.1.4.81 icc_getICCReaderStatus . . . . .	91
12.1.4.82 icc_getKeyFormatForICCDUKPT . . . . .	92
12.1.4.83 icc_getKeyTypeForICCDUKPT . . . . .	93
12.1.4.84 icc_powerOffICC . . . . .	93
12.1.4.85 icc_powerOnICC . . . . .	93
12.1.4.86 msr_cancelMSRSwipe . . . . .	94
12.1.4.87 msr_captureMode . . . . .	94
12.1.4.88 msr_disable . . . . .	94
12.1.4.89 msr_getClearPANID . . . . .	94
12.1.4.90 msr_getExpirationMask . . . . .	94
12.1.4.91 msr_getKeyFormatForICCDUKPT . . . . .	95
12.1.4.92 msr_getKeyTypeForICCDUKPT . . . . .	95

12.1.4.93	msr_getMSRData	95
12.1.4.94	msr_getSetting	96
12.1.4.95	msr_getSwipeForcedEncryptionOption	96
12.1.4.96	msr_getSwipeMaskOption	96
12.1.4.97	msr_registerCallBk	96
12.1.4.98	msr_registerCallBkp	97
12.1.4.99	msr_setClearPANID	97
12.1.4.100	msr_setExpirationMask	97
12.1.4.101	msr_setKeyFormatForICCDUKPT	97
12.1.4.102	msr_setKeyTypeForICCDUKPT	97
12.1.4.103	msr_setSetting	98
12.1.4.104	msr_setSwipeForcedEncryptionOption	98
12.1.4.105	msr_setSwipeMaskOption	98
12.1.4.106	msr_startMSRSwipe	99
12.1.4.107	parseMSRData	100
12.1.4.108	pin_cancelPINEntry	100
12.1.4.109	pin_registerCallBk	100
12.1.4.110	registerHotplugCallBk	100
12.1.4.111	registerLogCallBk	100
12.1.4.112	SDK_Version	100
12.1.4.113	setAbsoluteLibraryPath	100
12.2	Source_C/libIDT_Device.h File Reference	101
12.2.1	Detailed Description	108
12.2.2	Macro Definition Documentation	109
12.2.2.1	IN	109
12.2.2.2	IN_OUT	109
12.2.2.3	OUT	109
12.2.3	Typedef Documentation	109
12.2.3.1	ftpComm_callBack	109
12.2.3.2	httpComm_callBack	109
12.2.3.3	pCMR_callBack	109
12.2.3.4	pCSFS_callBack	109
12.2.3.5	pEMV_callBack	109
12.2.3.6	pFW_callBack	109
12.2.3.7	pLCD_callBack	110
12.2.3.8	pLog_callback	110
12.2.3.9	pMessageHotplug	110
12.2.3.10	pMSR_callBack	110
12.2.3.11	pMSR_callBackp	110
12.2.3.12	pPIN_callBack	110

12.2.3.13 pReadDataLog . . . . .	110
12.2.3.14 pRKI_callBack . . . . .	110
12.2.3.15 pSendDataLog . . . . .	110
12.2.3.16 v4Comm_callBack . . . . .	111
12.2.4 Function Documentation . . . . .	111
12.2.4.1 config_getBeeperController . . . . .	111
12.2.4.2 config_getEncryptionControl . . . . .	111
12.2.4.3 config_getLEDController . . . . .	111
12.2.4.4 config_getModelNumber . . . . .	112
12.2.4.5 config_getModelNumber_Len . . . . .	112
12.2.4.6 config_getSerialNumber . . . . .	112
12.2.4.7 config_getSerialNumber_Len . . . . .	113
12.2.4.8 config_setBeeperController . . . . .	114
12.2.4.9 config_setCmdTimeOutDuration . . . . .	114
12.2.4.10 config_setEncryptionControl . . . . .	114
12.2.4.11 config_setLEDController . . . . .	115
12.2.4.12 ctls_activateTransaction . . . . .	115
12.2.4.13 ctls_cancelTransaction . . . . .	116
12.2.4.14 ctls_displayOnlineAuthResult . . . . .	117
12.2.4.15 ctls_getAllConfigurationGroups . . . . .	118
12.2.4.16 ctls_getConfigurationGroup . . . . .	118
12.2.4.17 ctls_removeAllApplicationData . . . . .	118
12.2.4.18 ctls_removeAllCAPK . . . . .	118
12.2.4.19 ctls_removeApplicationData . . . . .	119
12.2.4.20 ctls_removeCAPK . . . . .	119
12.2.4.21 ctls_removeConfigurationGroup . . . . .	119
12.2.4.22 ctls_retrieveAIDList . . . . .	119
12.2.4.23 ctls_retrieveApplicationData . . . . .	120
12.2.4.24 ctls_retrieveCAPK . . . . .	120
12.2.4.25 ctls_retrieveCAPKList . . . . .	121
12.2.4.26 ctls_retrieveTerminalData . . . . .	121
12.2.4.27 ctls_setApplicationData . . . . .	121
12.2.4.28 ctls_setCAPK . . . . .	122
12.2.4.29 ctls_setConfigurationGroup . . . . .	123
12.2.4.30 ctls_setTerminalData . . . . .	123
12.2.4.31 ctls_startTransaction . . . . .	124
12.2.4.32 device_activateTransaction . . . . .	125
12.2.4.33 device_buzzerOnOff . . . . .	126
12.2.4.34 device_calibrateParameters . . . . .	126
12.2.4.35 device_cancelTransaction . . . . .	127



12.2.4.36 device_cancelTransactionSilent . . . . .	127
12.2.4.37 device_close . . . . .	127
12.2.4.38 device_configureButtons . . . . .	127
12.2.4.39 device_controlBeep . . . . .	127
12.2.4.40 device_controlIndicator . . . . .	128
12.2.4.41 device_controlLED . . . . .	128
12.2.4.42 device_controlLED_ICC . . . . .	129
12.2.4.43 device_controlLED_MSR . . . . .	129
12.2.4.44 device_controlUserInterface . . . . .	130
12.2.4.45 device_createDirectory . . . . .	132
12.2.4.46 device_deleteDirectory . . . . .	133
12.2.4.47 device_deleteFile . . . . .	133
12.2.4.48 device_disableBlueLED . . . . .	133
12.2.4.49 device_enableBlueLED . . . . .	133
12.2.4.50 device_enableExternalLCDMessages . . . . .	134
12.2.4.51 device_enableL100PassThrough . . . . .	134
12.2.4.52 device_enablePassThrough . . . . .	134
12.2.4.53 device_enableRFAntenna . . . . .	136
12.2.4.54 device_enhancedPassthrough . . . . .	136
12.2.4.55 device_enterStopMode . . . . .	136
12.2.4.56 device_getButtonConfiguration . . . . .	136
12.2.4.57 device_getCurrentDeviceType . . . . .	137
12.2.4.58 device_getDateTime . . . . .	137
12.2.4.59 device_getDateTime_Len . . . . .	137
12.2.4.60 device_getDeviceMemoryUsageInfo . . . . .	137
12.2.4.61 device_getDriveFreeSpace . . . . .	138
12.2.4.62 device_getDRS . . . . .	138
12.2.4.63 device_getFirmwareVersion . . . . .	138
12.2.4.64 device_getFirmwareVersion_Len . . . . .	139
12.2.4.65 device_getIDGStatusCodeString . . . . .	139
12.2.4.66 device_getKeyStatus . . . . .	140
12.2.4.67 device_getL100PassThroughMode . . . . .	141
12.2.4.68 device_getMerchantRecord . . . . .	141
12.2.4.69 device_getMerchantRecord_Len . . . . .	142
12.2.4.70 device_getResponseCodeString . . . . .	142
12.2.4.71 device_getRTCDatetime . . . . .	152
12.2.4.72 device_getSDKWaitTime . . . . .	152
12.2.4.73 device_getSpectrumProKSN . . . . .	153
12.2.4.74 device_getSpectrumProKSN_Len . . . . .	153
12.2.4.75 device_getThreadStackSize . . . . .	154

12.2.4.76 device_init	154
12.2.4.77 device_isAttached	154
12.2.4.78 device_isConnected	154
12.2.4.79 device_lcdDisplayClear	154
12.2.4.80 device_lcdDisplayLine1Message	155
12.2.4.81 device_lcdDisplayLine2Message	156
12.2.4.82 device_listDirectory	156
12.2.4.83 device_pingDevice	156
12.2.4.84 device_pollCardReader	157
12.2.4.85 device_pollCardReader_Len	159
12.2.4.86 device_pollForToken	161
12.2.4.87 device_queryFile	161
12.2.4.88 device_rebootDevice	161
12.2.4.89 device_registerCameraCallBk	162
12.2.4.90 device_registerCardStatusFrontSwitchCallBk	162
12.2.4.91 device_registerFWCallBk	162
12.2.4.92 device_registerRKICallBk	162
12.2.4.93 device_selfCheck	162
12.2.4.94 device_SendDataCommand	162
12.2.4.95 device_SendDataCommandITP	162
12.2.4.96 device_SendDataCommandNEO	163
12.2.4.97 device_setBurstMode	163
12.2.4.98 device_setCancelTransactionMode	163
12.2.4.99 device_setConfigPath	164
12.2.4.100 device_setCurrentDevice	164
12.2.4.101 device_setMerchantRecord	165
12.2.4.102 device_setNEO2DevicesConfigs	165
12.2.4.103 device_setPollMode	165
12.2.4.104 device_setRTCDateTime	165
12.2.4.105 device_setSDKWaitTime	166
12.2.4.106 device_setSleepModeTime	166
12.2.4.107 device_setSystemLanguage	166
12.2.4.108 device_setThreadStackSize	166
12.2.4.109 device_setTransactionExponent	166
12.2.4.110 device_startListenNotifications	167
12.2.4.111 device_startQRCodeScan	167
12.2.4.112 device_startRKI	167
12.2.4.113 device_startTakingPhoto	167
12.2.4.114 device_startTransaction	168
12.2.4.115 device_stopListenNotifications	169

12.2.4.116device_stopQRCodeScan . . . . .	169
12.2.4.117device_stopTakingPhoto . . . . .	169
12.2.4.118device_toSDCard . . . . .	169
12.2.4.119device_transferFile . . . . .	170
12.2.4.120device_turnOffYellowLED . . . . .	170
12.2.4.121device_turnOnYellowLED . . . . .	170
12.2.4.122device_updateFirmware . . . . .	171
12.2.4.123device_verifyBackdoorKey . . . . .	171
12.2.4.124emv_activateTransaction . . . . .	172
12.2.4.125emv_allowFallback . . . . .	173
12.2.4.126emv_authenticateTransaction . . . . .	173
12.2.4.127emv_authenticateTransactionWithTimeout . . . . .	173
12.2.4.128emv_callbackResponseLCD . . . . .	175
12.2.4.129emv_callbackResponseMSR . . . . .	175
12.2.4.130emv_cancelTransaction . . . . .	176
12.2.4.131emv_completeTransaction . . . . .	176
12.2.4.132emv_getAutoAuthenticateTransaction . . . . .	176
12.2.4.133emv_getAutoCompleteTransaction . . . . .	176
12.2.4.134emv_getEMVConfigurationCheckValue . . . . .	176
12.2.4.135emv_getEMVKernelCheckValue . . . . .	177
12.2.4.136emv_getEMVKernelVersion . . . . .	177
12.2.4.137emv_getEMVKernelVersion_Len . . . . .	177
12.2.4.138emv_registerCallBk . . . . .	177
12.2.4.139emv_removeAllApplicationData . . . . .	178
12.2.4.140emv_removeAllCAPK . . . . .	178
12.2.4.141emv_removeAllCRL . . . . .	178
12.2.4.142emv_removeApplicationData . . . . .	178
12.2.4.143emv_removeCAPK . . . . .	178
12.2.4.144emv_removeCRL . . . . .	179
12.2.4.145emv_removeTerminalData . . . . .	179
12.2.4.146emv_retrieveAIDList . . . . .	179
12.2.4.147emv_retrieveApplicationData . . . . .	179
12.2.4.148emv_retrieveCAPK . . . . .	180
12.2.4.149emv_retrieveCAPKList . . . . .	181
12.2.4.150emv_retrieveCRL . . . . .	181
12.2.4.151emv_retrieveTerminalData . . . . .	182
12.2.4.152emv_retrieveTerminalID . . . . .	182
12.2.4.153emv_retrieveTerminalID_Len . . . . .	182
12.2.4.154emv_retrieveTransactionResult . . . . .	182
12.2.4.155emv_setApplicationData . . . . .	183

12.2.4.156	emv_setApplicationDataTLV	183
12.2.4.157	emv_setAutoAuthenticateTransaction	183
12.2.4.158	emv_setAutoCompleteTransaction	183
12.2.4.159	emv_setCAPK	183
12.2.4.160	emv_setCRL	184
12.2.4.161	emv_setTerminalData	184
12.2.4.162	emv_setTerminalID	185
12.2.4.163	emv_setTerminalMajorConfiguration	185
12.2.4.164	emv_setTransactionParameters	185
12.2.4.165	emv_startTransaction	186
12.2.4.166	felica_authentication	186
12.2.4.167	felica_poll	187
12.2.4.168	felica_read	187
12.2.4.169	felica_readWithMac	187
12.2.4.170	felica_requestService	188
12.2.4.171	felica_SendCommand	188
12.2.4.172	felica_write	188
12.2.4.173	felica_writeWithMac	189
12.2.4.174	icc_disable	189
12.2.4.175	icc_enable	189
12.2.4.176	icc_exchangeAPDU	189
12.2.4.177	icc_exchangeEncryptedAPDU	190
12.2.4.178	icc_getAPDU_KSN	190
12.2.4.179	icc_getFunctionStatus	191
12.2.4.180	icc_getICCReaderStatus	192
12.2.4.181	icc_getKeyFormatForICCDUKPT	192
12.2.4.182	icc_getKeyTypeForICCDUKPT	192
12.2.4.183	icc_powerOffICC	193
12.2.4.184	icc_powerOnICC	193
12.2.4.185	icc_setKeyFormatForICCDUKPT	193
12.2.4.186	icc_setKeyTypeForICCDUKPT	194
12.2.4.187	iso8583_deserializeFromXML	194
12.2.4.188	iso8583_displayMessage	194
12.2.4.189	iso8583_freeMessage	194
12.2.4.190	iso8583_get1987Handler	195
12.2.4.191	iso8583_get1993Handler	195
12.2.4.192	iso8583_get2003Handler	195
12.2.4.193	iso8583_getField	195
12.2.4.194	iso8583_getMessageField	196
12.2.4.195	iso8583_initializeMessage	197

12.2.4.196so8583_packMessage . . . . .	197
12.2.4.197so8583_removeMessageField . . . . .	197
12.2.4.198so8583_serializeToXML . . . . .	197
12.2.4.199so8583_setMessageField . . . . .	198
12.2.4.200so8583_unpackMessage . . . . .	198
12.2.4.201lcd_addButton . . . . .	198
12.2.4.202cd_addEthernet . . . . .	199
12.2.4.203cd_addImage . . . . .	200
12.2.4.204cd_addItemToList . . . . .	201
12.2.4.205cd_addLED . . . . .	201
12.2.4.206cd_addText . . . . .	203
12.2.4.207cd_addVideo . . . . .	206
12.2.4.208cd_cancelSlideShow . . . . .	206
12.2.4.209cd_captureSignature . . . . .	207
12.2.4.210cd_clearDisplay . . . . .	207
12.2.4.211lcd_clearEventQueue . . . . .	207
12.2.4.212cd_clearScreenInfo . . . . .	207
12.2.4.213cd_cloneScreen . . . . .	208
12.2.4.214cd_createInputField . . . . .	209
12.2.4.215cd_createInputField_Len . . . . .	210
12.2.4.216cd_createList . . . . .	211
12.2.4.217cd_createList_Len . . . . .	212
12.2.4.218cd_createScreen . . . . .	213
12.2.4.219cd_customDisplayMode . . . . .	214
12.2.4.220cd_destroyScreen . . . . .	215
12.2.4.221lcd_displayButton . . . . .	215
12.2.4.222cd_displayButton_Len . . . . .	217
12.2.4.223cd_displayMessage . . . . .	218
12.2.4.224cd_displayParagraph . . . . .	218
12.2.4.225cd_displayPrompt . . . . .	219
12.2.4.226cd_displayText . . . . .	219
12.2.4.227cd_displayText_Len . . . . .	221
12.2.4.228cd_enableBacklight . . . . .	222
12.2.4.229cd_getActiveScreen . . . . .	222
12.2.4.230cd_getAllObjects . . . . .	223
12.2.4.231lcd_getAllScreens . . . . .	223
12.2.4.232cd_getBacklightStatus . . . . .	223
12.2.4.233cd_getButtonEvent . . . . .	224
12.2.4.234cd_getInputEvent . . . . .	224
12.2.4.235cd_getInputEvent_Len . . . . .	226

12.2.4.236	cd_getInputFieldValue	228
12.2.4.237	cd_getSelectedItem	228
12.2.4.238	cd_getSelectedItem_Len	228
12.2.4.239	cd_loadScreenInfo	228
12.2.4.240	cd_queryObjectbyID	229
12.2.4.241	cd_queryObjectbyName	229
12.2.4.242	cd_queryScreenbyID	229
12.2.4.243	cd_queryScreenbyName	229
12.2.4.244	cd_registerCallBk	230
12.2.4.245	cd_removeItem	230
12.2.4.246	cd_resetInitialState	230
12.2.4.247	cd_savePrompt	230
12.2.4.248	cd_setBackgroundImage	231
12.2.4.249	cd_setBacklight	232
12.2.4.250	cd_setDisplayImage	232
12.2.4.251	cd_setForeBackColor	233
12.2.4.252	cd_showScreen	233
12.2.4.253	cd_startSlideShow	233
12.2.4.254	cd_storeScreenInfo	234
12.2.4.255	cd_updateColor	234
12.2.4.256	cd_updateLabel	235
12.2.4.257	cd_updatePosition	235
12.2.4.258	loyalty_cancelTransaction	236
12.2.4.259	loyalty_cancelTransactionSilent	236
12.2.4.260	loyalty_registerCallBk	236
12.2.4.261	loyalty_startTransaction	236
12.2.4.262	msr_cancelMSRSwipe	238
12.2.4.263	msr_captureMode	238
12.2.4.264	msr_clearMSRData	238
12.2.4.265	msr_disable	239
12.2.4.266	msr_flushTrackData	239
12.2.4.267	msr_getClearPANID	239
12.2.4.268	msr_getExpirationMask	239
12.2.4.269	msr_getFunctionStatus	239
12.2.4.270	msr_getKeyFormatForICCDUKPT	240
12.2.4.271	msr_getKeyTypeForICCDUKPT	240
12.2.4.272	msr_getMSRData	240
12.2.4.273	msr_getSwipeForcedEncryptionOption	241
12.2.4.274	msr_getSwipeMaskOption	241
12.2.4.275	msr_registerCallBk	241

12.2.4.276msr_registerCallBkp . . . . .	241
12.2.4.277msr_setClearPANID . . . . .	241
12.2.4.278msr_setExpirationMask . . . . .	242
12.2.4.279msr_setKeyFormatForICCDUKPT . . . . .	242
12.2.4.280msr_setKeyTypeForICCDUKPT . . . . .	242
12.2.4.281msr_setSwipeForcedEncryptionOption . . . . .	242
12.2.4.282msr_setSwipeMaskOption . . . . .	243
12.2.4.283msr_startMSRSwipe . . . . .	243
12.2.4.284parseMSRData . . . . .	243
12.2.4.285parsePINBlockData . . . . .	243
12.2.4.286parsePINData . . . . .	244
12.2.4.287pin_cancelPINEntry . . . . .	244
12.2.4.288pin_capturePin . . . . .	244
12.2.4.289pin_capturePinExt . . . . .	245
12.2.4.290pin_getEncryptedOnlinePIN . . . . .	245
12.2.4.291pin_getEncryptedPIN . . . . .	246
12.2.4.292pin_getFunctionKey . . . . .	246
12.2.4.293pin_getPAN . . . . .	247
12.2.4.294pin_getPanEntry . . . . .	248
12.2.4.295pin_getPIN . . . . .	248
12.2.4.296pin_inputFromPrompt . . . . .	249
12.2.4.297pin_promptCreditDebit . . . . .	249
12.2.4.298pin_promptForAmount . . . . .	250
12.2.4.299pin_promptForAmountInput . . . . .	250
12.2.4.300pin_promptForKeyInput . . . . .	254
12.2.4.301pin_promptForNumericKey . . . . .	257
12.2.4.302pin_promptForNumericKeyWithSwipe . . . . .	258
12.2.4.303pin_registerCallBk . . . . .	258
12.2.4.304pin_sendBeep . . . . .	258
12.2.4.305pin_setKeyValues . . . . .	259
12.2.4.306registerHotplugCallBk . . . . .	259
12.2.4.307registerLogCallBk . . . . .	259
12.2.4.308rs232_device_init . . . . .	259
12.2.4.309SDK_Version . . . . .	260
12.2.4.310setAbsoluteLibraryPath . . . . .	260
12.2.4.311ws_deleteSSLCert . . . . .	260
12.2.4.312ws_getCertChainType . . . . .	260
12.2.4.313ws_loadSSLCert . . . . .	260
12.2.4.314ws_requestCSR . . . . .	261
12.2.4.315ws_revokeSSLCert . . . . .	261

12.2.4.31	ws_updateRootCertificate	261
12.3	Source_C/libIDT_KioskIII.h File Reference	262
12.3.1	Detailed Description	263
12.3.2	Macro Definition Documentation	263
12.3.2.1	IN	263
12.3.2.2	IN_OUT	263
12.3.2.3	OUT	263
12.3.3	Typedef Documentation	264
12.3.3.1	ftpComm_callBack	264
12.3.3.2	httpComm_callBack	264
12.3.3.3	pCMR_callBack	264
12.3.3.4	pCSFS_callBack	264
12.3.3.5	pEMV_callBack	264
12.3.3.6	pMessageHotplug	264
12.3.3.7	pMSR_callBack	264
12.3.3.8	pMSR_callBackp	264
12.3.3.9	pPIN_callBack	265
12.3.3.10	pReadDataLog	265
12.3.3.11	pSendDataLog	265
12.3.3.12	v4Comm_callBack	265
12.3.4	Function Documentation	265
12.3.4.1	config_getSerialNumber	265
12.3.4.2	config_getSerialNumber_Len	265
12.3.4.3	ctls_activateTransaction	266
12.3.4.4	ctls_cancelTransaction	268
12.3.4.5	ctls_getAllConfigurationGroups	268
12.3.4.6	ctls_getConfigurationGroup	268
12.3.4.7	ctls_registerCallBk	268
12.3.4.8	ctls_registerCallBkp	268
12.3.4.9	ctls_removeAllApplicationData	268
12.3.4.10	ctls_removeAllCAPK	269
12.3.4.11	ctls_removeApplicationData	269
12.3.4.12	ctls_removeCAPK	269
12.3.4.13	ctls_removeConfigurationGroup	269
12.3.4.14	ctls_retrieveAIDList	269
12.3.4.15	ctls_retrieveApplicationData	270
12.3.4.16	ctls_retrieveCAPK	270
12.3.4.17	ctls_retrieveCAPKList	271
12.3.4.18	ctls_retrieveTerminalData	271
12.3.4.19	ctls_setApplicationData	271



12.3.4.20	<a href="#">ctls_setCAPK</a>	271
12.3.4.21	<a href="#">ctls_setConfigurationGroup</a>	272
12.3.4.22	<a href="#">ctls_setTerminalData</a>	272
12.3.4.23	<a href="#">ctls_startTransaction</a>	273
12.3.4.24	<a href="#">device_close</a>	274
12.3.4.25	<a href="#">device_controlUserInterface</a>	274
12.3.4.26	<a href="#">device_enablePassThrough</a>	275
12.3.4.27	<a href="#">device_getCurrentDeviceType</a>	275
12.3.4.28	<a href="#">device_getFirmwareVersion</a>	275
12.3.4.29	<a href="#">device_getFirmwareVersion_Len</a>	275
12.3.4.30	<a href="#">device_getIDGStatusCodeString</a>	276
12.3.4.31	<a href="#">device_getMerchantRecord</a>	277
12.3.4.32	<a href="#">device_getMerchantRecord_Len</a>	277
12.3.4.33	<a href="#">device_getSDKWaitTime</a>	278
12.3.4.34	<a href="#">device_getThreadStackSize</a>	278
12.3.4.35	<a href="#">device_getTransactionResults</a>	278
12.3.4.36	<a href="#">device_init</a>	279
12.3.4.37	<a href="#">device_isAttached</a>	279
12.3.4.38	<a href="#">device_isConnected</a>	279
12.3.4.39	<a href="#">device_pingDevice</a>	279
12.3.4.40	<a href="#">device_registerCameraCallBk</a>	279
12.3.4.41	<a href="#">device_registerCardStatusFrontSwitchCallBk</a>	279
12.3.4.42	<a href="#">device_SendDataCommandNEO</a>	279
12.3.4.43	<a href="#">device_setBurstMode</a>	280
12.3.4.44	<a href="#">device_setCurrentDevice</a>	280
12.3.4.45	<a href="#">device_setMerchantRecord</a>	281
12.3.4.46	<a href="#">device_setPollMode</a>	281
12.3.4.47	<a href="#">device_setSDKWaitTime</a>	282
12.3.4.48	<a href="#">device_setThreadStackSize</a>	282
12.3.4.49	<a href="#">emv_registerCallBk</a>	282
12.3.4.50	<a href="#">parseMSRData</a>	282
12.3.4.51	<a href="#">pin_registerCallBk</a>	282
12.3.4.52	<a href="#">registerHotplugCallBk</a>	282
12.3.4.53	<a href="#">registerLogCallBk</a>	282
12.3.4.54	<a href="#">rs232_device_init</a>	283
12.3.4.55	<a href="#">SDK_Version</a>	283
12.3.4.56	<a href="#">setAbsoluteLibraryPath</a>	283
12.4	<a href="#">Source_C/libIDT_L100.h File Reference</a>	283
12.4.1	<a href="#">Detailed Description</a>	285
12.4.2	<a href="#">Macro Definition Documentation</a>	285

12.4.2.1	IN	285
12.4.2.2	IN_OUT	285
12.4.2.3	OUT	285
12.4.3	Typedef Documentation	285
12.4.3.1	ftpComm_callBack	285
12.4.3.2	httpComm_callBack	285
12.4.3.3	pCMR_callBack	286
12.4.3.4	pCSFS_callBack	286
12.4.3.5	pEMV_callBack	286
12.4.3.6	pFW_callBack	286
12.4.3.7	pMessageHotplug	286
12.4.3.8	pMSR_callBack	286
12.4.3.9	pMSR_callBackp	286
12.4.3.10	pPIN_callBack	286
12.4.3.11	pReadDataLog	286
12.4.3.12	pSendDataLog	287
12.4.3.13	v4Comm_callBack	287
12.4.4	Function Documentation	287
12.4.4.1	config_getModelNumber	287
12.4.4.2	config_getModelNumber_Len	287
12.4.4.3	config_getSerialNumber	287
12.4.4.4	config_getSerialNumber_Len	287
12.4.4.5	device_close	288
12.4.4.6	device_enterStopMode	288
12.4.4.7	device_getCurrentDeviceType	288
12.4.4.8	device_getDateTime	288
12.4.4.9	device_getDateTime_Len	288
12.4.4.10	device_getFirmwareVersion	289
12.4.4.11	device_getFirmwareVersion_Len	289
12.4.4.12	device_getKeyStatus	289
12.4.4.13	device_getResponseCodeString	290
12.4.4.14	device_init	300
12.4.4.15	device_isAttached	300
12.4.4.16	device_isConnected	300
12.4.4.17	device_rebootDevice	300
12.4.4.18	device_registerCameraCallBk	301
12.4.4.19	device_registerCardStatusFrontSwitchCallBk	301
12.4.4.20	device_registerFWCallBk	301
12.4.4.21	device_SendDataCommand	301
12.4.4.22	device_setCurrentDevice	301

12.4.4.23 device_setSleepModeTime . . . . .	302
12.4.4.24 device_updateFirmware . . . . .	302
12.4.4.25 emv_registerCallBk . . . . .	303
12.4.4.26 lcd_displayMessage . . . . .	303
12.4.4.27 lcd_displayPrompt . . . . .	303
12.4.4.28 lcd_enableBacklight . . . . .	304
12.4.4.29 lcd_getBacklightStatus . . . . .	304
12.4.4.30 lcd_savePrompt . . . . .	304
12.4.4.31 msr_registerCallBk . . . . .	304
12.4.4.32 msr_registerCallBkp . . . . .	305
12.4.4.33 pin_getEncryptedPIN . . . . .	305
12.4.4.34 pin_getFunctionKey . . . . .	305
12.4.4.35 pin_promptForAmountInput . . . . .	306
12.4.4.36 pin_promptForKeyInput . . . . .	309
12.4.4.37 pin_registerCallBk . . . . .	312
12.4.4.38 pin_sendBeep . . . . .	312
12.4.4.39 pin_setKeyValues . . . . .	313
12.4.4.40 registerHotplugCallBk . . . . .	313
12.4.4.41 registerLogCallBk . . . . .	313
12.4.4.42 SDK_Version . . . . .	313
12.4.4.43 setAbsoluteLibraryPath . . . . .	313
12.5 Source_C/libIDT_MiniSmartII.h File Reference . . . . .	314
12.5.1 Detailed Description . . . . .	316
12.5.2 Macro Definition Documentation . . . . .	316
12.5.2.1 IN . . . . .	316
12.5.2.2 IN_OUT . . . . .	316
12.5.2.3 OUT . . . . .	316
12.5.3 Typedef Documentation . . . . .	316
12.5.3.1 ftpComm_callBack . . . . .	316
12.5.3.2 httpComm_callBack . . . . .	316
12.5.3.3 pCMR_callBack . . . . .	317
12.5.3.4 pCSFS_callBack . . . . .	317
12.5.3.5 pEMV_callBack . . . . .	317
12.5.3.6 pMessageHotplug . . . . .	317
12.5.3.7 pMSR_callBack . . . . .	317
12.5.3.8 pMSR_callBackp . . . . .	317
12.5.3.9 pPIN_callBack . . . . .	317
12.5.3.10 pReadDataLog . . . . .	317
12.5.3.11 pSendDataLog . . . . .	317
12.5.3.12 v4Comm_callBack . . . . .	318

12.5.4	Function Documentation	318
12.5.4.1	comm_registerHTTPCallback	318
12.5.4.2	comm_registerV4Callback	318
12.5.4.3	config_getBeeperController	318
12.5.4.4	config_getEncryptionControl	318
12.5.4.5	config_getLEDController	319
12.5.4.6	config_getModelNumber	319
12.5.4.7	config_getModelNumber_Len	319
12.5.4.8	config_getSerialNumber	320
12.5.4.9	config_getSerialNumber_Len	320
12.5.4.10	config_setBeeperController	320
12.5.4.11	config_setEncryptionControl	320
12.5.4.12	config_setLEDController	322
12.5.4.13	device_close	322
12.5.4.14	device_controlBeep	322
12.5.4.15	device_controlLED	323
12.5.4.16	device_controlLED_ICC	323
12.5.4.17	device_controlLED_MSR	323
12.5.4.18	device_getCurrentDeviceType	324
12.5.4.19	device_getFirmwareVersion	324
12.5.4.20	device_getFirmwareVersion_Len	324
12.5.4.21	device_getKeyStatus	325
12.5.4.22	device_getResponseCodeString	325
12.5.4.23	device_getSDKWaitTime	335
12.5.4.24	device_getThreadStackSize	336
12.5.4.25	device_init	336
12.5.4.26	device_isAttached	336
12.5.4.27	device_isConnected	336
12.5.4.28	device_rebootDevice	336
12.5.4.29	device_registerCameraCallBk	337
12.5.4.30	device_registerCardStatusFrontSwitchCallBk	337
12.5.4.31	device_SendDataCommand	337
12.5.4.32	device_setCurrentDevice	337
12.5.4.33	device_setSDKWaitTime	338
12.5.4.34	device_setThreadStackSize	338
12.5.4.35	device_updateFirmware	338
12.5.4.36	emv_activateTransaction	339
12.5.4.37	emv_allowFallback	340
12.5.4.38	emv_authenticateTransaction	341
12.5.4.39	emv_authenticateTransactionWithTimeout	341

12.5.4.40	emv_callbackResponseLCD	342
12.5.4.41	emv_callbackResponseMSR	342
12.5.4.42	emv_cancelTransaction	342
12.5.4.43	emv_completeTransaction	343
12.5.4.44	emv_getAutoAuthenticateTransaction	343
12.5.4.45	emv_getAutoCompleteTransaction	343
12.5.4.46	emv_getEMVConfigurationCheckValue	343
12.5.4.47	emv_getEMVKernelCheckValue	343
12.5.4.48	emv_getEMVKernelVersion	344
12.5.4.49	emv_getEMVKernelVersion_Len	344
12.5.4.50	emv_registerCallBk	344
12.5.4.51	emv_removeAllApplicationData	344
12.5.4.52	emv_removeAllCAPK	344
12.5.4.53	emv_removeAllCRL	345
12.5.4.54	emv_removeApplicationData	345
12.5.4.55	emv_removeCAPK	345
12.5.4.56	emv_removeCRL	345
12.5.4.57	emv_removeTerminalData	346
12.5.4.58	emv_retrieveAIDList	346
12.5.4.59	emv_retrieveApplicationData	346
12.5.4.60	emv_retrieveCAPK	347
12.5.4.61	emv_retrieveCAPKList	347
12.5.4.62	emv_retrieveCRL	348
12.5.4.63	emv_retrieveTerminalData	349
12.5.4.64	emv_retrieveTerminalID	349
12.5.4.65	emv_retrieveTerminalID_Len	349
12.5.4.66	emv_retrieveTransactionResult	349
12.5.4.67	emv_setApplicationData	350
12.5.4.68	emv_setAutoAuthenticateTransaction	350
12.5.4.69	emv_setAutoCompleteTransaction	350
12.5.4.70	emv_setCAPK	350
12.5.4.71	emv_setCRL	351
12.5.4.72	emv_setTerminalData	351
12.5.4.73	emv_setTerminalID	352
12.5.4.74	emv_startTransaction	352
12.5.4.75	icc_disable	352
12.5.4.76	icc_enable	353
12.5.4.77	icc_exchangeAPDU	354
12.5.4.78	icc_exchangeEncryptedAPDU	354
12.5.4.79	icc_getAPDU_KSN	356

12.5.4.80	icc_getFunctionStatus	356
12.5.4.81	icc_getICCReaderStatus	356
12.5.4.82	icc_getKeyFormatForICCDUKPT	357
12.5.4.83	icc_getKeyTypeForICCDUKPT	358
12.5.4.84	icc_powerOffICC	358
12.5.4.85	icc_powerOnICC	358
12.5.4.86	msr_registerCallBk	359
12.5.4.87	msr_registerCallBkp	359
12.5.4.88	pin_registerCallBk	359
12.5.4.89	registerHotplugCallBk	359
12.5.4.90	registerLogCallBk	359
12.5.4.91	rs232_device_init	359
12.5.4.92	SDK_Version	360
12.5.4.93	setAbsoluteLibraryPath	360
12.6	Source_C/libIDT_NEO2.h File Reference	360
12.6.1	Detailed Description	365
12.6.2	Macro Definition Documentation	365
12.6.2.1	IN	365
12.6.2.2	IN_OUT	365
12.6.2.3	OUT	365
12.6.3	Typedef Documentation	365
12.6.3.1	ftpComm_callBack	365
12.6.3.2	httpComm_callBack	365
12.6.3.3	pCMR_callBack	366
12.6.3.4	pCSFS_callBack	366
12.6.3.5	pEMV_callBack	366
12.6.3.6	pFW_callBack	366
12.6.3.7	pLCD_callBack	366
12.6.3.8	pMessageHotplug	366
12.6.3.9	pMSR_callBack	366
12.6.3.10	pMSR_callBackp	366
12.6.3.11	pPIN_callBack	366
12.6.3.12	pReadDataLog	367
12.6.3.13	pSendDataLog	367
12.6.3.14	v4Comm_callBack	367
12.6.4	Function Documentation	367
12.6.4.1	comm_registerHTTPCallback	367
12.6.4.2	comm_registerV4Callback	367
12.6.4.3	config_getModelNumber	367
12.6.4.4	config_getModelNumber_Len	367

12.6.4.5	<a href="#">config_getSerialNumber</a>	368
12.6.4.6	<a href="#">config_getSerialNumber_Len</a>	368
12.6.4.7	<a href="#">config_setCmdTimeOutDuration</a>	368
12.6.4.8	<a href="#">ctls_activateTransaction</a>	368
12.6.4.9	<a href="#">ctls_cancelTransaction</a>	370
12.6.4.10	<a href="#">ctls_displayOnlineAuthResult</a>	370
12.6.4.11	<a href="#">ctls_getAllConfigurationGroups</a>	370
12.6.4.12	<a href="#">ctls_getConfigurationGroup</a>	370
12.6.4.13	<a href="#">ctls_registerCallBk</a>	371
12.6.4.14	<a href="#">ctls_registerCallBkp</a>	371
12.6.4.15	<a href="#">ctls_removeAllApplicationData</a>	371
12.6.4.16	<a href="#">ctls_removeAllCAPK</a>	371
12.6.4.17	<a href="#">ctls_removeApplicationData</a>	371
12.6.4.18	<a href="#">ctls_removeCAPK</a>	371
12.6.4.19	<a href="#">ctls_removeConfigurationGroup</a>	372
12.6.4.20	<a href="#">ctls_retrieveAIDList</a>	372
12.6.4.21	<a href="#">ctls_retrieveApplicationData</a>	372
12.6.4.22	<a href="#">ctls_retrieveCAPK</a>	372
12.6.4.23	<a href="#">ctls_retrieveCAPKList</a>	373
12.6.4.24	<a href="#">ctls_retrieveTerminalData</a>	373
12.6.4.25	<a href="#">ctls_setApplicationData</a>	374
12.6.4.26	<a href="#">ctls_setCAPK</a>	374
12.6.4.27	<a href="#">ctls_setConfigurationGroup</a>	375
12.6.4.28	<a href="#">ctls_setTerminalData</a>	375
12.6.4.29	<a href="#">ctls_startTransaction</a>	375
12.6.4.30	<a href="#">device_activateTransaction</a>	377
12.6.4.31	<a href="#">device_buzzerOnOff</a>	378
12.6.4.32	<a href="#">device_cancelTransaction</a>	378
12.6.4.33	<a href="#">device_cancelTransactionSilent</a>	378
12.6.4.34	<a href="#">device_close</a>	378
12.6.4.35	<a href="#">device_configureButtons</a>	379
12.6.4.36	<a href="#">device_controlUserInterface</a>	380
12.6.4.37	<a href="#">device_deleteDirectory</a>	382
12.6.4.38	<a href="#">device_deleteFile</a>	383
12.6.4.39	<a href="#">device_disableBlueLED</a>	383
12.6.4.40	<a href="#">device_enableBlueLED</a>	383
12.6.4.41	<a href="#">device_enableExternalLCDMessages</a>	384
12.6.4.42	<a href="#">device_enableL100PassThrough</a>	385
12.6.4.43	<a href="#">device_enablePassThrough</a>	385
12.6.4.44	<a href="#">device_enableRFAntenna</a>	385

12.6.4.45 device_getButtonConfiguration . . . . .	386
12.6.4.46 device_getCurrentDeviceType . . . . .	387
12.6.4.47 device_getDeviceMemoryUsageInfo . . . . .	387
12.6.4.48 device_getFirmwareVersion . . . . .	387
12.6.4.49 device_getFirmwareVersion_Len . . . . .	387
12.6.4.50 device_getIDGStatusCodeString . . . . .	388
12.6.4.51 device_getKeyStatus . . . . .	389
12.6.4.52 device_getL100PassThroughMode . . . . .	390
12.6.4.53 device_getMerchantRecord . . . . .	390
12.6.4.54 device_getMerchantRecord_Len . . . . .	390
12.6.4.55 device_getResponseCodeString . . . . .	391
12.6.4.56 device_getSDKWaitTime . . . . .	401
12.6.4.57 device_getThreadStackSize . . . . .	401
12.6.4.58 device_getTransactionResults . . . . .	401
12.6.4.59 device_init . . . . .	401
12.6.4.60 device_isAttached . . . . .	401
12.6.4.61 device_isConnected . . . . .	402
12.6.4.62 device_lcdDisplayClear . . . . .	402
12.6.4.63 device_lcdDisplayLine1Message . . . . .	402
12.6.4.64 device_lcdDisplayLine2Message . . . . .	402
12.6.4.65 device_listDirectory . . . . .	403
12.6.4.66 device_pingDevice . . . . .	404
12.6.4.67 device_pollForToken . . . . .	404
12.6.4.68 device_queryFile . . . . .	404
12.6.4.69 device_registerCameraCallBk . . . . .	405
12.6.4.70 device_registerCardStatusFrontSwitchCallBk . . . . .	405
12.6.4.71 device_registerFWCallBk . . . . .	405
12.6.4.72 device_SendDataCommandNEO . . . . .	405
12.6.4.73 device_setBurstMode . . . . .	405
12.6.4.74 device_setCancelTransactionMode . . . . .	406
12.6.4.75 device_setConfigPath . . . . .	406
12.6.4.76 device_setCurrentDevice . . . . .	406
12.6.4.77 device_setMerchantRecord . . . . .	407
12.6.4.78 device_setNEO2DevicesConfigs . . . . .	407
12.6.4.79 device_setPollMode . . . . .	408
12.6.4.80 device_setSDKWaitTime . . . . .	408
12.6.4.81 device_setThreadStackSize . . . . .	408
12.6.4.82 device_setTransactionExponent . . . . .	408
12.6.4.83 device_startListenNotifications . . . . .	408
12.6.4.84 device_startQRCodeScan . . . . .	408



12.6.4.85 device_startTakingPhoto . . . . .	409
12.6.4.86 device_startTransaction . . . . .	409
12.6.4.87 device_stopListenNotifications . . . . .	410
12.6.4.88 device_stopQRCodeScan . . . . .	411
12.6.4.89 device_stopTakingPhoto . . . . .	411
12.6.4.90 device_toSDCard . . . . .	411
12.6.4.91 device_transferFile . . . . .	411
12.6.4.92 device_turnOffYellowLED . . . . .	411
12.6.4.93 device_turnOnYellowLED . . . . .	412
12.6.4.94 device_updateFirmware . . . . .	412
12.6.4.95 emv_activateTransaction . . . . .	412
12.6.4.96 emv_allowFallback . . . . .	413
12.6.4.97 emv_authenticateTransaction . . . . .	413
12.6.4.98 emv_authenticateTransactionWithTimeout . . . . .	413
12.6.4.99 emv_cancelTransaction . . . . .	415
12.6.4.100emv_completeTransaction . . . . .	415
12.6.4.101emv_getAutoAuthenticateTransaction . . . . .	416
12.6.4.102emv_getAutoCompleteTransaction . . . . .	416
12.6.4.103emv_getEMVConfigurationCheckValue . . . . .	416
12.6.4.104emv_getEMVKernelCheckValue . . . . .	416
12.6.4.105emv_getEMVKernelVersion . . . . .	416
12.6.4.106emv_getEMVKernelVersion_Len . . . . .	416
12.6.4.107emv_registerCallBk . . . . .	417
12.6.4.108emv_removeAllApplicationData . . . . .	417
12.6.4.109emv_removeAllCAPK . . . . .	417
12.6.4.110emv_removeAllCRL . . . . .	417
12.6.4.111emv_removeApplicationData . . . . .	417
12.6.4.112emv_removeCAPK . . . . .	418
12.6.4.113emv_removeCRL . . . . .	418
12.6.4.114emv_retrieveAIDList . . . . .	418
12.6.4.115emv_retrieveApplicationData . . . . .	418
12.6.4.116emv_retrieveCAPK . . . . .	419
12.6.4.117emv_retrieveCAPKList . . . . .	420
12.6.4.118emv_retrieveCRL . . . . .	420
12.6.4.119emv_retrieveTerminalData . . . . .	420
12.6.4.120emv_retrieveTransactionResult . . . . .	420
12.6.4.121emv_setApplicationData . . . . .	421
12.6.4.122emv_setApplicationDataTLV . . . . .	421
12.6.4.123emv_setAutoAuthenticateTransaction . . . . .	421
12.6.4.124emv_setAutoCompleteTransaction . . . . .	422

12.6.4.125	emv_setCAPK	422
12.6.4.126	emv_setCRL	423
12.6.4.127	emv_setTerminalData	423
12.6.4.128	emv_setTerminalMajorConfiguration	423
12.6.4.129	emv_setTransactionParameters	424
12.6.4.130	emv_startTransaction	424
12.6.4.131	felica_authentication	425
12.6.4.132	felica_poll	426
12.6.4.133	felica_read	426
12.6.4.134	felica_readWithMac	426
12.6.4.135	felica_requestService	427
12.6.4.136	felica_SendCommand	427
12.6.4.137	felica_write	427
12.6.4.138	felica_writeWithMac	428
12.6.4.139	icc_exchangeAPDU	428
12.6.4.140	icc_getICCRReaderStatus	428
12.6.4.141	icc_powerOffICC	429
12.6.4.142	icc_powerOnICC	429
12.6.4.143	cd_addButton	429
12.6.4.144	cd_addEthernet	430
12.6.4.145	cd_addImage	431
12.6.4.146	cd_addLED	432
12.6.4.147	cd_addText	433
12.6.4.148	cd_addVideo	436
12.6.4.149	cd_clearScreenInfo	436
12.6.4.150	cd_cloneScreen	437
12.6.4.151	cd_createScreen	437
12.6.4.152	cd_destroyScreen	437
12.6.4.153	cd_getActiveScreen	437
12.6.4.154	cd_getAllObjects	438
12.6.4.155	cd_getAllScreens	438
12.6.4.156	cd_getButtonEvent	438
12.6.4.157	cd_loadScreenInfo	439
12.6.4.158	cd_queryObjectbyID	439
12.6.4.159	cd_queryObjectbyName	439
12.6.4.160	cd_queryScreenbyID	440
12.6.4.161	cd_queryScreenbyName	440
12.6.4.162	cd_registerCallBk	440
12.6.4.163	cd_removeItem	440
12.6.4.164	cd_setBacklight	441

12.6.4.165	cd_showScreen	441
12.6.4.166	cd_storeScreenInfo	441
12.6.4.167	cd_updateColor	441
12.6.4.168	cd_updateLabel	442
12.6.4.169	cd_updatePosition	442
12.6.4.170	loyalty_cancelTransaction	443
12.6.4.171	loyalty_cancelTransactionSilent	443
12.6.4.172	loyalty_registerCallBk	443
12.6.4.173	loyalty_startTransaction	443
12.6.4.174	msr_cancelMSRSwipe	446
12.6.4.175	msr_registerCallBk	446
12.6.4.176	msr_registerCallBkp	446
12.6.4.177	msr_startMSRSwipe	446
12.6.4.178	parseMSRData	446
12.6.4.179	pin_cancelPINEntry	447
12.6.4.180	pin_capturePin	447
12.6.4.181	pin_capturePinExt	447
12.6.4.182	pin_getPanEntry	448
12.6.4.183	pin_inputFromPrompt	449
12.6.4.184	pin_promptForNumericKey	450
12.6.4.185	pin_promptForNumericKeyWithSwipe	450
12.6.4.186	pin_registerCallBk	451
12.6.4.187	pin_setKeyValues	451
12.6.4.188	registerHotplugCallBk	451
12.6.4.189	registerLogCallBk	452
12.6.4.190	s232_device_init	452
12.6.4.191	SDK_Version	452
12.6.4.192	setAbsoluteLibraryPath	452
12.7	Source_C/libIDT_PipReader.h File Reference	453
12.7.1	Detailed Description	454
12.7.2	Macro Definition Documentation	454
12.7.2.1	IN	454
12.7.2.2	IN_OUT	454
12.7.2.3	OUT	454
12.7.3	Typedef Documentation	455
12.7.3.1	ftpComm_callBack	455
12.7.3.2	httpComm_callBack	455
12.7.3.3	pCMR_callBack	455
12.7.3.4	pCSFS_callBack	455
12.7.3.5	pEMV_callBack	455

12.7.3.6	pMessageHotplug	455
12.7.3.7	pMSR_callBack	455
12.7.3.8	pMSR_callBackp	455
12.7.3.9	pPIN_callBack	456
12.7.3.10	pReadDataLog	456
12.7.3.11	pSendDataLog	456
12.7.3.12	v4Comm_callBack	456
12.7.4	Function Documentation	456
12.7.4.1	config_getSerialNumber	456
12.7.4.2	config_getSerialNumber_Len	456
12.7.4.3	ctls_activateTransaction	457
12.7.4.4	ctls_cancelTransaction	459
12.7.4.5	ctls_getAllConfigurationGroups	459
12.7.4.6	ctls_getConfigurationGroup	459
12.7.4.7	ctls_registerCallBk	459
12.7.4.8	ctls_registerCallBkp	459
12.7.4.9	ctls_removeAllApplicationData	459
12.7.4.10	ctls_removeAllCAPK	460
12.7.4.11	ctls_removeApplicationData	460
12.7.4.12	ctls_removeCAPK	460
12.7.4.13	ctls_removeConfigurationGroup	460
12.7.4.14	ctls_retrieveAIDList	460
12.7.4.15	ctls_retrieveApplicationData	461
12.7.4.16	ctls_retrieveCAPK	461
12.7.4.17	ctls_retrieveCAPKList	462
12.7.4.18	ctls_retrieveTerminalData	462
12.7.4.19	ctls_setApplicationData	462
12.7.4.20	ctls_setCAPK	462
12.7.4.21	ctls_setConfigurationGroup	463
12.7.4.22	ctls_setTerminalData	463
12.7.4.23	ctls_startTransaction	464
12.7.4.24	device_close	465
12.7.4.25	device_controlUserInterface	465
12.7.4.26	device_enablePassThrough	466
12.7.4.27	device_getCurrentDeviceType	466
12.7.4.28	device_getFirmwareVersion	466
12.7.4.29	device_getFirmwareVersion_Len	466
12.7.4.30	device_getIDGStatusCodeString	467
12.7.4.31	device_getMerchantRecord	468
12.7.4.32	device_getMerchantRecord_Len	468

12.7.4.33 device_getSDKWaitTime . . . . .	469
12.7.4.34 device_getTransactionResults . . . . .	469
12.7.4.35 device_init . . . . .	469
12.7.4.36 device_isAttached . . . . .	469
12.7.4.37 device_isConnected . . . . .	470
12.7.4.38 device_pingDevice . . . . .	470
12.7.4.39 device_registerCameraCallBk . . . . .	470
12.7.4.40 device_registerCardStatusFrontSwitchCallBk . . . . .	470
12.7.4.41 device_SendDataCommandNEO . . . . .	470
12.7.4.42 device_setBurstMode . . . . .	471
12.7.4.43 device_setCurrentDevice . . . . .	472
12.7.4.44 device_setMerchantRecord . . . . .	473
12.7.4.45 device_setPollMode . . . . .	473
12.7.4.46 device_setSDKWaitTime . . . . .	473
12.7.4.47 emv_registerCallBk . . . . .	473
12.7.4.48 parseMSRData . . . . .	473
12.7.4.49 pin_registerCallBk . . . . .	474
12.7.4.50 registerHotplugCallBk . . . . .	474
12.7.4.51 registerLogCallBk . . . . .	474
12.7.4.52 rs232_device_init . . . . .	474
12.7.4.53 SDK_Version . . . . .	474
12.7.4.54 setAbsoluteLibraryPath . . . . .	475
12.8 Source_C/libIDT_SpectrumPro.h File Reference . . . . .	475
12.8.1 Detailed Description . . . . .	477
12.8.2 Macro Definition Documentation . . . . .	477
12.8.2.1 IN . . . . .	477
12.8.2.2 IN_OUT . . . . .	477
12.8.2.3 OUT . . . . .	477
12.8.3 Typedef Documentation . . . . .	478
12.8.3.1 ftpComm_callBack . . . . .	478
12.8.3.2 httpComm_callBack . . . . .	478
12.8.3.3 pCMR_callBack . . . . .	478
12.8.3.4 pCSFS_callBack . . . . .	478
12.8.3.5 pEMV_callBack . . . . .	478
12.8.3.6 pMessageHotplug . . . . .	478
12.8.3.7 pMSR_callBack . . . . .	478
12.8.3.8 pMSR_callBackp . . . . .	478
12.8.3.9 pPIN_callBack . . . . .	479
12.8.3.10 pReadDataLog . . . . .	479
12.8.3.11 pSendDataLog . . . . .	479

12.8.3.12 v4Comm_callBack . . . . .	479
12.8.4 Function Documentation . . . . .	479
12.8.4.1 config_getModelNumber . . . . .	479
12.8.4.2 config_getModelNumber_Len . . . . .	479
12.8.4.3 config_getSerialNumber . . . . .	479
12.8.4.4 config_getSerialNumber_Len . . . . .	480
12.8.4.5 device_close . . . . .	480
12.8.4.6 device_getCurrentDeviceType . . . . .	480
12.8.4.7 device_getFirmwareVersion . . . . .	480
12.8.4.8 device_getFirmwareVersion_Len . . . . .	480
12.8.4.9 device_getResponseCodeString . . . . .	481
12.8.4.10 device_getSDKWaitTime . . . . .	491
12.8.4.11 device_getSpectrumProKSN . . . . .	491
12.8.4.12 device_getSpectrumProKSN_Len . . . . .	491
12.8.4.13 device_getThreadStackSize . . . . .	492
12.8.4.14 device_init . . . . .	492
12.8.4.15 device_isAttached . . . . .	492
12.8.4.16 device_isConnected . . . . .	492
12.8.4.17 device_pollCardReader . . . . .	492
12.8.4.18 device_pollCardReader_Len . . . . .	494
12.8.4.19 device_rebootDevice . . . . .	496
12.8.4.20 device_registerCameraCallBk . . . . .	496
12.8.4.21 device_registerCardStatusFrontSwitchCallBk . . . . .	496
12.8.4.22 device_SendDataCommand . . . . .	496
12.8.4.23 device_setCurrentDevice . . . . .	496
12.8.4.24 device_setSDKWaitTime . . . . .	497
12.8.4.25 device_setThreadStackSize . . . . .	497
12.8.4.26 device_updateFirmware . . . . .	497
12.8.4.27 emv_activateTransaction . . . . .	498
12.8.4.28 emv_allowFallback . . . . .	499
12.8.4.29 emv_authenticateTransaction . . . . .	499
12.8.4.30 emv_authenticateTransactionWithTimeout . . . . .	499
12.8.4.31 emv_callbackResponseLCD . . . . .	500
12.8.4.32 emv_callbackResponseMSR . . . . .	500
12.8.4.33 emv_cancelTransaction . . . . .	500
12.8.4.34 emv_completeTransaction . . . . .	500
12.8.4.35 emv_getAutoAuthenticateTransaction . . . . .	502
12.8.4.36 emv_getAutoCompleteTransaction . . . . .	502
12.8.4.37 emv_getEMVConfigurationCheckValue . . . . .	502
12.8.4.38 emv_getEMVKernelCheckValue . . . . .	502

12.8.4.39 emv_getEMVKernelVersion . . . . .	503
12.8.4.40 emv_getEMVKernelVersion_Len . . . . .	503
12.8.4.41 emv_registerCallBk . . . . .	503
12.8.4.42 emv_removeAllApplicationData . . . . .	503
12.8.4.43 emv_removeAllCAPK . . . . .	503
12.8.4.44 emv_removeAllCRL . . . . .	504
12.8.4.45 emv_removeApplicationData . . . . .	504
12.8.4.46 emv_removeCAPK . . . . .	504
12.8.4.47 emv_removeCRL . . . . .	504
12.8.4.48 emv_removeTerminalData . . . . .	505
12.8.4.49 emv_retrieveAIDList . . . . .	505
12.8.4.50 emv_retrieveApplicationData . . . . .	505
12.8.4.51 emv_retrieveCAPK . . . . .	506
12.8.4.52 emv_retrieveCAPKList . . . . .	506
12.8.4.53 emv_retrieveCRL . . . . .	507
12.8.4.54 emv_retrieveTerminalData . . . . .	508
12.8.4.55 emv_retrieveTerminalID . . . . .	508
12.8.4.56 emv_retrieveTerminalID_Len . . . . .	508
12.8.4.57 emv_retrieveTransactionResult . . . . .	508
12.8.4.58 emv_setApplicationData . . . . .	509
12.8.4.59 emv_setAutoAuthenticateTransaction . . . . .	509
12.8.4.60 emv_setAutoCompleteTransaction . . . . .	509
12.8.4.61 emv_setCAPK . . . . .	509
12.8.4.62 emv_setCRL . . . . .	510
12.8.4.63 emv_setTerminalData . . . . .	510
12.8.4.64 emv_setTerminalID . . . . .	511
12.8.4.65 emv_startTransaction . . . . .	511
12.8.4.66 icc_getICCReaderStatus . . . . .	511
12.8.4.67 icc_powerOffICC . . . . .	512
12.8.4.68 icc_powerOnICC . . . . .	512
12.8.4.69 msr_cancelMSRSwipe . . . . .	512
12.8.4.70 msr_clearMSRData . . . . .	512
12.8.4.71 msr_getMSRData . . . . .	512
12.8.4.72 msr_registerCallBk . . . . .	513
12.8.4.73 msr_registerCallBkp . . . . .	513
12.8.4.74 msr_startMSRSwipe . . . . .	513
12.8.4.75 parseMSRData . . . . .	513
12.8.4.76 parsePINBlockData . . . . .	513
12.8.4.77 parsePINData . . . . .	513
12.8.4.78 pin_cancelPINEntry . . . . .	514

12.8.4.79	pin_getPIN . . . . .	514
12.8.4.80	pin_registerCallBk . . . . .	514
12.8.4.81	registerHotplugCallBk . . . . .	515
12.8.4.82	registerLogCallBk . . . . .	515
12.8.4.83	rs232_device_init . . . . .	515
12.8.4.84	SDK_Version . . . . .	515
12.8.4.85	setAbsoluteLibraryPath . . . . .	515
12.9	Source_C/libIDT_SREDKey2.h File Reference . . . . .	516
12.9.1	Detailed Description . . . . .	517
12.9.2	Macro Definition Documentation . . . . .	517
12.9.2.1	IN . . . . .	517
12.9.2.2	IN_OUT . . . . .	518
12.9.2.3	OUT . . . . .	518
12.9.3	Typedef Documentation . . . . .	518
12.9.3.1	ftpComm_callBack . . . . .	518
12.9.3.2	httpComm_callBack . . . . .	518
12.9.3.3	pCMR_callBack . . . . .	518
12.9.3.4	pCSFS_callBack . . . . .	518
12.9.3.5	pEMV_callBack . . . . .	518
12.9.3.6	pFW_callBack . . . . .	518
12.9.3.7	pLCD_callBack . . . . .	518
12.9.3.8	pMessageHotplug . . . . .	519
12.9.3.9	pMSR_callBack . . . . .	519
12.9.3.10	pMSR_callBackp . . . . .	519
12.9.3.11	pPIN_callBack . . . . .	519
12.9.3.12	pReadDataLog . . . . .	519
12.9.3.13	pSendDataLog . . . . .	519
12.9.3.14	v4Comm_callBack . . . . .	519
12.9.4	Function Documentation . . . . .	519
12.9.4.1	comm_registerHTTPCallback . . . . .	519
12.9.4.2	comm_registerV4Callback . . . . .	520
12.9.4.3	config_getModelNumber . . . . .	521
12.9.4.4	config_getModelNumber_Len . . . . .	521
12.9.4.5	config_getSerialNumber . . . . .	521
12.9.4.6	config_getSerialNumber_Len . . . . .	521
12.9.4.7	ctls_registerCallBk . . . . .	522
12.9.4.8	ctls_registerCallBkp . . . . .	522
12.9.4.9	device_close . . . . .	522
12.9.4.10	device_getCurrentDeviceType . . . . .	522
12.9.4.11	device_getFirmwareVersion . . . . .	522



12.9.4.12 device_getFirmwareVersion_Len . . . . .	522
12.9.4.13 device_getIDGStatusCodeString . . . . .	522
12.9.4.14 device_getKeyStatus . . . . .	524
12.9.4.15 device_init . . . . .	524
12.9.4.16 device_isAttached . . . . .	525
12.9.4.17 device_isConnected . . . . .	525
12.9.4.18 device_pingDevice . . . . .	525
12.9.4.19 device_rebootDevice . . . . .	525
12.9.4.20 device_registerCameraCallBk . . . . .	525
12.9.4.21 device_registerCardStatusFrontSwitchCallBk . . . . .	525
12.9.4.22 device_registerFWCallBk . . . . .	526
12.9.4.23 device_SendDataCommand . . . . .	526
12.9.4.24 device_SendDataCommandITP . . . . .	526
12.9.4.25 device_SendDataCommandNEO . . . . .	526
12.9.4.26 device_setConfigPath . . . . .	527
12.9.4.27 device_setCurrentDevice . . . . .	527
12.9.4.28 device_setNEO2DevicesConfigs . . . . .	528
12.9.4.29 device_setSystemLanguage . . . . .	529
12.9.4.30 device_setTransactionExponent . . . . .	529
12.9.4.31 device_updateFirmware . . . . .	529
12.9.4.32 emv_registerCallBk . . . . .	530
12.9.4.33 lcd_registerCallBk . . . . .	530
12.9.4.34 msr_disable . . . . .	530
12.9.4.35 msr_getClearPANID . . . . .	530
12.9.4.36 msr_getExpirationMask . . . . .	530
12.9.4.37 msr_getFunctionStatus . . . . .	531
12.9.4.38 msr_getSwipeForcedEncryptionOption . . . . .	531
12.9.4.39 msr_getSwipeMaskOption . . . . .	531
12.9.4.40 msr_registerCallBk . . . . .	531
12.9.4.41 msr_registerCallBkp . . . . .	532
12.9.4.42 msr_setClearPANID . . . . .	532
12.9.4.43 msr_setExpirationMask . . . . .	532
12.9.4.44 msr_setSwipeForcedEncryptionOption . . . . .	532
12.9.4.45 msr_setSwipeMaskOption . . . . .	532
12.9.4.46 pin_registerCallBk . . . . .	533
12.9.4.47 registerHotplugCallBk . . . . .	533
12.9.4.48 registerLogCallBk . . . . .	533
12.9.4.49 rs232_device_init . . . . .	533
12.9.4.50 SDK_Version . . . . .	533
12.9.4.51 setAbsoluteLibraryPath . . . . .	534

12.10Source_C/libIDT_UniPayI_V.h File Reference . . . . .	535
12.10.1 Detailed Description . . . . .	537
12.10.2 Macro Definition Documentation . . . . .	537
12.10.2.1 IN . . . . .	537
12.10.2.2 IN_OUT . . . . .	537
12.10.2.3 OUT . . . . .	537
12.10.3 Typedef Documentation . . . . .	537
12.10.3.1 ftpComm_callBack . . . . .	537
12.10.3.2 httpComm_callBack . . . . .	537
12.10.3.3 pCMR_callBack . . . . .	537
12.10.3.4 pCSFS_callBack . . . . .	537
12.10.3.5 pEMV_callBack . . . . .	537
12.10.3.6 pMessageHotplug . . . . .	538
12.10.3.7 pMSR_callBack . . . . .	538
12.10.3.8 pMSR_callBackp . . . . .	538
12.10.3.9 pPIN_callBack . . . . .	538
12.10.3.10pReadDataLog . . . . .	538
12.10.3.11pSendDataLog . . . . .	538
12.10.3.12v4Comm_callBack . . . . .	538
12.10.4 Function Documentation . . . . .	538
12.10.4.1 comm_registerHTTPCallback . . . . .	538
12.10.4.2 comm_registerV4Callback . . . . .	539
12.10.4.3 config_getSerialNumber . . . . .	540
12.10.4.4 config_getSerialNumber_Len . . . . .	540
12.10.4.5 device_close . . . . .	540
12.10.4.6 device_enablePassThrough . . . . .	540
12.10.4.7 device_getCurrentDeviceType . . . . .	541
12.10.4.8 device_getFirmwareVersion . . . . .	541
12.10.4.9 device_getFirmwareVersion_Len . . . . .	541
12.10.4.10device_getIDGStatusCodeString . . . . .	541
12.10.4.11device_getMerchantRecord . . . . .	542
12.10.4.12device_getMerchantRecord_Len . . . . .	543
12.10.4.13device_getSDKWaitTime . . . . .	543
12.10.4.14device_getThreadStackSize . . . . .	543
12.10.4.15device_init . . . . .	544
12.10.4.16device_isAttached . . . . .	544
12.10.4.17device_isConnected . . . . .	544
12.10.4.18device_pingDevice . . . . .	544
12.10.4.19device_registerCameraCallBk . . . . .	544
12.10.4.20device_registerCardStatusFrontSwitchCallBk . . . . .	544

12.10.4.21	device_SendDataCommandNEO	544
12.10.4.22	device_setCurrentDevice	545
12.10.4.23	device_setMerchantRecord	546
12.10.4.24	device_setSDKWaitTime	546
12.10.4.25	device_setThreadStackSize	547
12.10.4.26	emv_activateTransaction	547
12.10.4.27	emv_allowFallback	547
12.10.4.28	emv_authenticateTransaction	547
12.10.4.29	emv_authenticateTransactionWithTimeout	548
12.10.4.30	emv_cancelTransaction	548
12.10.4.31	emv_completeTransaction	549
12.10.4.32	emv_getAutoAuthenticateTransaction	549
12.10.4.33	emv_getAutoCompleteTransaction	549
12.10.4.34	emv_registerCallBk	549
12.10.4.35	emv_removeAllApplicationData	549
12.10.4.36	emv_removeAllCAPK	550
12.10.4.37	emv_removeAllCRL	550
12.10.4.38	emv_removeApplicationData	550
12.10.4.39	emv_removeCAPK	550
12.10.4.40	emv_removeCRL	550
12.10.4.41	emv_retrieveAIDList	551
12.10.4.42	emv_retrieveApplicationData	551
12.10.4.43	emv_retrieveCAPK	552
12.10.4.44	emv_retrieveCAPKList	552
12.10.4.45	emv_retrieveCRL	553
12.10.4.46	emv_retrieveTerminalData	554
12.10.4.47	emv_setApplicationData	554
12.10.4.48	emv_setApplicationDataTLV	554
12.10.4.49	emv_setAutoAuthenticateTransaction	555
12.10.4.50	emv_setAutoCompleteTransaction	555
12.10.4.51	emv_setCAPK	555
12.10.4.52	emv_setCRL	556
12.10.4.53	emv_setTerminalData	556
12.10.4.54	emv_setTerminalMajorConfiguration	556
12.10.4.55	emv_startTransaction	557
12.10.4.56	icc_exchangeAPDU	557
12.10.4.57	icc_getICCRReaderStatus	557
12.10.4.58	icc_powerOffICC	558
12.10.4.59	icc_powerOnICC	558
12.10.4.60	msr_cancelMSRSwipe	558

12.10.4.61	msr_registerCallBk . . . . .	558
12.10.4.62	msr_registerCallBkp . . . . .	558
12.10.4.63	msr_startMSRSwipe . . . . .	558
12.10.4.64	parseMSRData . . . . .	559
12.10.4.65	pin_registerCallBk . . . . .	559
12.10.4.66	registerHotplugCallBk . . . . .	559
12.10.4.67	registerLogCallBk . . . . .	559
12.10.4.68	SDK_Version . . . . .	559
12.10.4.69	setAbsoluteLibraryPath . . . . .	559
12.11	Source_C/libIDT_Vendi.h File Reference . . . . .	560
12.11.1	Detailed Description . . . . .	561
12.11.2	Macro Definition Documentation . . . . .	561
12.11.2.1	IN . . . . .	561
12.11.2.2	IN_OUT . . . . .	561
12.11.2.3	OUT . . . . .	562
12.11.3	Typedef Documentation . . . . .	562
12.11.3.1	ftpComm_callBack . . . . .	562
12.11.3.2	httpComm_callBack . . . . .	562
12.11.3.3	pCMR_callBack . . . . .	562
12.11.3.4	pCSFS_callBack . . . . .	562
12.11.3.5	pEMV_callBack . . . . .	562
12.11.3.6	pMessageHotplug . . . . .	562
12.11.3.7	pMSR_callBack . . . . .	562
12.11.3.8	pMSR_callBackp . . . . .	562
12.11.3.9	pPIN_callBack . . . . .	563
12.11.3.10	pReadDataLog . . . . .	563
12.11.3.11	pSendDataLog . . . . .	563
12.11.3.12	4Comm_callBack . . . . .	563
12.11.4	Function Documentation . . . . .	563
12.11.4.1	comm_registerHTTPCallback . . . . .	563
12.11.4.2	comm_registerV4Callback . . . . .	563
12.11.4.3	config_getSerialNumber . . . . .	563
12.11.4.4	config_getSerialNumber_Len . . . . .	564
12.11.4.5	ctls_activateTransaction . . . . .	564
12.11.4.6	ctls_cancelTransaction . . . . .	565
12.11.4.7	ctls_getAllConfigurationGroups . . . . .	565
12.11.4.8	ctls_getConfigurationGroup . . . . .	565
12.11.4.9	ctls_registerCallBk . . . . .	566
12.11.4.10	ctls_registerCallBkp . . . . .	566
12.11.4.11	ctls_removeAllApplicationData . . . . .	566

12.11.4.12	tls_removeAllCAPK	566
12.11.4.13	tls_removeApplicationData	566
12.11.4.14	tls_removeCAPK	566
12.11.4.15	tls_removeConfigurationGroup	567
12.11.4.16	tls_retrieveAIDList	567
12.11.4.17	tls_retrieveApplicationData	567
12.11.4.18	tls_retrieveCAPK	567
12.11.4.19	tls_retrieveCAPKList	568
12.11.4.20	tls_retrieveTerminalData	568
12.11.4.21	tls_setApplicationData	569
12.11.4.22	tls_setCAPK	569
12.11.4.23	tls_setConfigurationGroup	570
12.11.4.24	tls_setTerminalData	570
12.11.4.25	tls_startTransaction	570
12.11.4.26	device_close	571
12.11.4.27	device_controlUserInterface	572
12.11.4.28	device_enablePassThrough	572
12.11.4.29	device_getCurrentDeviceType	573
12.11.4.30	device_getFirmwareVersion	573
12.11.4.31	device_getFirmwareVersion_Len	573
12.11.4.32	device_getIDGStatusCodeString	573
12.11.4.33	device_getMerchantRecord	574
12.11.4.34	device_getMerchantRecord_Len	575
12.11.4.35	device_getSDKWaitTime	575
12.11.4.36	device_getThreadStackSize	575
12.11.4.37	device_getTransactionResults	575
12.11.4.38	device_init	576
12.11.4.39	device_isAttached	576
12.11.4.40	device_isConnected	576
12.11.4.41	device_pingDevice	576
12.11.4.42	device_registerCameraCallBk	577
12.11.4.43	device_registerCardStatusFrontSwitchCallBk	577
12.11.4.44	device_SendDataCommandNEO	577
12.11.4.45	device_setBurstMode	577
12.11.4.46	device_setCurrentDevice	578
12.11.4.47	device_setMerchantRecord	579
12.11.4.48	device_setPollMode	579
12.11.4.49	device_setSDKWaitTime	579
12.11.4.50	device_setThreadStackSize	579
12.11.4.51	emv_registerCallBk	579

12.11.4.52msr_cancelMSRSwipe . . . . .	580
12.11.4.53msr_registerCallBk . . . . .	580
12.11.4.54msr_registerCallBkp . . . . .	580
12.11.4.55msr_startMSRSwipe . . . . .	580
12.11.4.56parseMSRData . . . . .	580
12.11.4.57pin_registerCallBk . . . . .	580
12.11.4.58registerHotplugCallBk . . . . .	580
12.11.4.59registerLogCallBk . . . . .	580
12.11.4.60SDK_Version . . . . .	581
12.11.4.61setAbsoluteLibraryPath . . . . .	581
12.12Source_C/libIDT_VP3300_AJ.h File Reference . . . . .	581
12.12.1 Detailed Description . . . . .	584
12.12.2 Macro Definition Documentation . . . . .	584
12.12.2.1 IN . . . . .	584
12.12.2.2 IN_OUT . . . . .	584
12.12.2.3 OUT . . . . .	584
12.12.3 Typedef Documentation . . . . .	584
12.12.3.1 ftpComm_callBack . . . . .	584
12.12.3.2 httpComm_callBack . . . . .	584
12.12.3.3 pCMR_callBack . . . . .	584
12.12.3.4 pCSFS_callBack . . . . .	584
12.12.3.5 pEMV_callBack . . . . .	584
12.12.3.6 pMessageHotplug . . . . .	585
12.12.3.7 pMSR_callBack . . . . .	585
12.12.3.8 pMSR_callBackp . . . . .	585
12.12.3.9 pPIN_callBack . . . . .	585
12.12.3.10pReadDataLog . . . . .	585
12.12.3.11pSendDataLog . . . . .	585
12.12.3.12v4Comm_callBack . . . . .	585
12.12.4 Function Documentation . . . . .	585
12.12.4.1 comm_registerHTTPCallback . . . . .	585
12.12.4.2 comm_registerV4Callback . . . . .	586
12.12.4.3 config_getSerialNumber . . . . .	587
12.12.4.4 config_getSerialNumber_Len . . . . .	587
12.12.4.5 ctls_activateTransaction . . . . .	587
12.12.4.6 ctls_cancelTransaction . . . . .	588
12.12.4.7 ctls_getAllConfigurationGroups . . . . .	588
12.12.4.8 ctls_getConfigurationGroup . . . . .	589
12.12.4.9 ctls_registerCallBk . . . . .	589
12.12.4.10ctls_registerCallBkp . . . . .	589

12.12.4.1	ctls_removeAllApplicationData	589
12.12.4.2	ctls_removeAllCAPK	589
12.12.4.3	ctls_removeApplicationData	589
12.12.4.4	ctls_removeCAPK	590
12.12.4.5	ctls_removeConfigurationGroup	590
12.12.4.6	ctls_retrieveAIDList	590
12.12.4.7	ctls_retrieveApplicationData	590
12.12.4.8	ctls_retrieveCAPK	591
12.12.4.9	ctls_retrieveCAPKList	591
12.12.4.20	ctls_retrieveTerminalData	592
12.12.4.21	ctls_setApplicationData	592
12.12.4.22	ctls_setCAPK	592
12.12.4.23	ctls_setConfigurationGroup	593
12.12.4.24	ctls_setTerminalData	593
12.12.4.25	ctls_startTransaction	594
12.12.4.26	device_activateTransaction	595
12.12.4.27	device_cancelTransaction	596
12.12.4.28	device_close	596
12.12.4.29	device_controlUserInterface	596
12.12.4.30	device_enablePassThrough	597
12.12.4.31	device_getCurrentDeviceType	597
12.12.4.32	device_getFirmwareVersion	598
12.12.4.33	device_getFirmwareVersion_Len	598
12.12.4.34	device_getIDGStatusCodeString	598
12.12.4.35	device_getMerchantRecord	599
12.12.4.36	device_getMerchantRecord_Len	600
12.12.4.37	device_getRTCDateTime	600
12.12.4.38	device_getSDKWaitTime	601
12.12.4.39	device_getThreadStackSize	601
12.12.4.40	device_getTransactionResults	601
12.12.4.41	device_init	601
12.12.4.42	device_isAttached	601
12.12.4.43	device_isConnected	602
12.12.4.44	device_pingDevice	602
12.12.4.45	device_registerCameraCallBk	602
12.12.4.46	device_registerCardStatusFrontSwitchCallBk	602
12.12.4.47	device_registerRKICallBk	602
12.12.4.48	device_SendDataCommandNEO	602
12.12.4.49	device_setBurstMode	604
12.12.4.50	device_setCurrentDevice	604

12.12.4.51	device_setMerchantRecord	605
12.12.4.52	device_setPollMode	605
12.12.4.53	device_setRTCDateTime	606
12.12.4.54	device_setSDKWaitTime	606
12.12.4.55	device_setThreadStackSize	606
12.12.4.56	device_setTransactionExponent	606
12.12.4.57	device_startRKI	606
12.12.4.58	device_startTransaction	607
12.12.4.59	emv_activateTransaction	608
12.12.4.60	emv_allowFallback	608
12.12.4.61	emv_authenticateTransaction	609
12.12.4.62	emv_authenticateTransactionWithTimeout	609
12.12.4.63	emv_cancelTransaction	610
12.12.4.64	emv_completeTransaction	610
12.12.4.65	emv_getAutoAuthenticateTransaction	610
12.12.4.66	emv_getAutoCompleteTransaction	610
12.12.4.67	emv_registerCallBk	610
12.12.4.68	emv_removeAllApplicationData	611
12.12.4.69	emv_removeAllCAPK	611
12.12.4.70	emv_removeAllCRL	611
12.12.4.71	emv_removeApplicationData	611
12.12.4.72	emv_removeCAPK	611
12.12.4.73	emv_removeCRL	612
12.12.4.74	emv_retrieveAIDList	612
12.12.4.75	emv_retrieveApplicationData	612
12.12.4.76	emv_retrieveCAPK	612
12.12.4.77	emv_retrieveCAPKList	613
12.12.4.78	emv_retrieveCRL	613
12.12.4.79	emv_retrieveTerminalData	614
12.12.4.80	emv_setApplicationData	614
12.12.4.81	emv_setApplicationDataTLV	614
12.12.4.82	emv_setAutoAuthenticateTransaction	614
12.12.4.83	emv_setAutoCompleteTransaction	615
12.12.4.84	emv_setCAPK	615
12.12.4.85	emv_setCRL	615
12.12.4.86	emv_setTerminalData	616
12.12.4.87	emv_setTerminalMajorConfiguration	616
12.12.4.88	emv_setTransactionParameters	616
12.12.4.89	emv_startTransaction	617
12.12.4.90	cc_exchangeAPDU	617



12.12.4.91	<a href="#">icc_getICCReaderStatus</a>	618
12.12.4.92	<a href="#">cc_powerOffICC</a>	618
12.12.4.93	<a href="#">cc_powerOnICC</a>	618
12.12.4.94	<a href="#">msr_cancelMSRSwipe</a>	618
12.12.4.95	<a href="#">msr_registerCallBk</a>	618
12.12.4.96	<a href="#">msr_registerCallBkp</a>	619
12.12.4.97	<a href="#">msr_startMSRSwipe</a>	619
12.12.4.98	<a href="#">parseMSRData</a>	619
12.12.4.99	<a href="#">pin_registerCallBk</a>	619
12.12.4.100	<a href="#">registerHotplugCallBk</a>	619
12.12.4.101	<a href="#">registerLogCallBk</a>	619
12.12.4.102	<a href="#">SDK_Version</a>	619
12.12.4.103	<a href="#">setAbsoluteLibraryPath</a>	619
12.13	<a href="#">Source_C/libIDT_VP3300_BT.h File Reference</a>	620
12.13.1	<a href="#">Detailed Description</a>	622
12.13.2	<a href="#">Macro Definition Documentation</a>	622
12.13.2.1	<a href="#">IN</a>	622
12.13.2.2	<a href="#">IN_OUT</a>	622
12.13.2.3	<a href="#">OUT</a>	623
12.13.3	<a href="#">Typedef Documentation</a>	623
12.13.3.1	<a href="#">ftpComm_callBack</a>	623
12.13.3.2	<a href="#">httpComm_callBack</a>	623
12.13.3.3	<a href="#">pCMR_callBack</a>	623
12.13.3.4	<a href="#">pCSFS_callBack</a>	623
12.13.3.5	<a href="#">pEMV_callBack</a>	623
12.13.3.6	<a href="#">pMessageHotplug</a>	623
12.13.3.7	<a href="#">pMSR_callBack</a>	623
12.13.3.8	<a href="#">pMSR_callBackp</a>	623
12.13.3.9	<a href="#">pPIN_callBack</a>	624
12.13.3.10	<a href="#">pReadDataLog</a>	624
12.13.3.11	<a href="#">pSendDataLog</a>	624
12.13.3.12	<a href="#">v4Comm_callBack</a>	624
12.13.4	<a href="#">Function Documentation</a>	624
12.13.4.1	<a href="#">comm_registerHTTPCallback</a>	624
12.13.4.2	<a href="#">comm_registerV4Callback</a>	624
12.13.4.3	<a href="#">config_getSerialNumber</a>	624
12.13.4.4	<a href="#">config_getSerialNumber_Len</a>	625
12.13.4.5	<a href="#">ctls_activateTransaction</a>	625
12.13.4.6	<a href="#">ctls_cancelTransaction</a>	626
12.13.4.7	<a href="#">ctls_getAllConfigurationGroups</a>	626

12.13.4.8	<a href="#">ctls_getConfigurationGroup</a>	626
12.13.4.9	<a href="#">ctls_registerCallBk</a>	627
12.13.4.10	<a href="#">ctls_registerCallBkp</a>	627
12.13.4.11	<a href="#">ctls_removeAllApplicationData</a>	627
12.13.4.12	<a href="#">ctls_removeAllCAPK</a>	627
12.13.4.13	<a href="#">ctls_removeApplicationData</a>	627
12.13.4.14	<a href="#">ctls_removeCAPK</a>	627
12.13.4.15	<a href="#">ctls_removeConfigurationGroup</a>	628
12.13.4.16	<a href="#">ctls_retrieveAIDList</a>	628
12.13.4.17	<a href="#">ctls_retrieveApplicationData</a>	628
12.13.4.18	<a href="#">ctls_retrieveCAPK</a>	628
12.13.4.19	<a href="#">ctls_retrieveCAPKList</a>	629
12.13.4.20	<a href="#">ctls_retrieveTerminalData</a>	629
12.13.4.21	<a href="#">ctls_setApplicationData</a>	630
12.13.4.22	<a href="#">ctls_setCAPK</a>	630
12.13.4.23	<a href="#">ctls_setConfigurationGroup</a>	631
12.13.4.24	<a href="#">ctls_setTerminalData</a>	631
12.13.4.25	<a href="#">ctls_startTransaction</a>	631
12.13.4.26	<a href="#">device_activateTransaction</a>	632
12.13.4.27	<a href="#">device_cancelTransaction</a>	634
12.13.4.28	<a href="#">device_close</a>	634
12.13.4.29	<a href="#">device_controlUserInterface</a>	634
12.13.4.30	<a href="#">device_enablePassThrough</a>	635
12.13.4.31	<a href="#">device_getCurrentDeviceType</a>	635
12.13.4.32	<a href="#">device_getFirmwareVersion</a>	635
12.13.4.33	<a href="#">device_getFirmwareVersion_Len</a>	635
12.13.4.34	<a href="#">device_getIDGStatusCodeString</a>	635
12.13.4.35	<a href="#">device_getMerchantRecord</a>	637
12.13.4.36	<a href="#">device_getMerchantRecord_Len</a>	637
12.13.4.37	<a href="#">device_getRTCDateTime</a>	637
12.13.4.38	<a href="#">device_getSDKWaitTime</a>	638
12.13.4.39	<a href="#">device_getThreadStackSize</a>	638
12.13.4.40	<a href="#">device_getTransactionResults</a>	638
12.13.4.41	<a href="#">device_init</a>	638
12.13.4.42	<a href="#">device_isAttached</a>	639
12.13.4.43	<a href="#">device_isConnected</a>	639
12.13.4.44	<a href="#">device_pingDevice</a>	639
12.13.4.45	<a href="#">device_registerCameraCallBk</a>	639
12.13.4.46	<a href="#">device_registerCardStatusFrontSwitchCallBk</a>	639
12.13.4.47	<a href="#">device_registerRKICallBk</a>	639

12.13.4.48	<a href="#">device_SendDataCommandNEO</a>	639
12.13.4.49	<a href="#">device_setBurstMode</a>	640
12.13.4.50	<a href="#">device_setCurrentDevice</a>	640
12.13.4.51	<a href="#">device_setMerchantRecord</a>	641
12.13.4.52	<a href="#">device_setPollMode</a>	641
12.13.4.53	<a href="#">device_setRTCDateTime</a>	642
12.13.4.54	<a href="#">device_setSDKWaitTime</a>	642
12.13.4.55	<a href="#">device_setThreadStackSize</a>	642
12.13.4.56	<a href="#">device_setTransactionExponent</a>	642
12.13.4.57	<a href="#">device_startRKI</a>	642
12.13.4.58	<a href="#">device_startTransaction</a>	643
12.13.4.59	<a href="#">emv_activateTransaction</a>	644
12.13.4.60	<a href="#">emv_allowFallback</a>	644
12.13.4.61	<a href="#">emv_authenticateTransaction</a>	645
12.13.4.62	<a href="#">emv_authenticateTransactionWithTimeout</a>	645
12.13.4.63	<a href="#">emv_cancelTransaction</a>	646
12.13.4.64	<a href="#">emv_completeTransaction</a>	646
12.13.4.65	<a href="#">emv_getAutoAuthenticateTransaction</a>	646
12.13.4.66	<a href="#">emv_getAutoCompleteTransaction</a>	646
12.13.4.67	<a href="#">emv_registerCallBk</a>	646
12.13.4.68	<a href="#">emv_removeAllApplicationData</a>	647
12.13.4.69	<a href="#">emv_removeAllCAPK</a>	647
12.13.4.70	<a href="#">emv_removeAllCRL</a>	647
12.13.4.71	<a href="#">emv_removeApplicationData</a>	647
12.13.4.72	<a href="#">emv_removeCAPK</a>	647
12.13.4.73	<a href="#">emv_removeCRL</a>	648
12.13.4.74	<a href="#">emv_retrieveAIDList</a>	648
12.13.4.75	<a href="#">emv_retrieveApplicationData</a>	648
12.13.4.76	<a href="#">emv_retrieveCAPK</a>	648
12.13.4.77	<a href="#">emv_retrieveCAPKList</a>	649
12.13.4.78	<a href="#">emv_retrieveCRL</a>	649
12.13.4.79	<a href="#">emv_retrieveTerminalData</a>	650
12.13.4.80	<a href="#">emv_setApplicationData</a>	650
12.13.4.81	<a href="#">emv_setApplicationDataTLV</a>	650
12.13.4.82	<a href="#">emv_setAutoAuthenticateTransaction</a>	650
12.13.4.83	<a href="#">emv_setAutoCompleteTransaction</a>	651
12.13.4.84	<a href="#">emv_setCAPK</a>	651
12.13.4.85	<a href="#">emv_setCRL</a>	651
12.13.4.86	<a href="#">emv_setTerminalData</a>	652
12.13.4.87	<a href="#">emv_setTerminalMajorConfiguration</a>	652

12.13.4.88	<a href="#">emv_setTransactionParameters</a>	652
12.13.4.89	<a href="#">emv_startTransaction</a>	653
12.13.4.90	<a href="#">icc_exchangeAPDU</a>	653
12.13.4.91	<a href="#">icc_getICCRewriterStatus</a>	654
12.13.4.92	<a href="#">icc_powerOffICC</a>	654
12.13.4.93	<a href="#">icc_powerOnICC</a>	654
12.13.4.94	<a href="#">msr_cancelMSRSwipe</a>	654
12.13.4.95	<a href="#">msr_registerCallBk</a>	654
12.13.4.96	<a href="#">msr_registerCallBkp</a>	655
12.13.4.97	<a href="#">msr_startMSRSwipe</a>	655
12.13.4.98	<a href="#">parseMSRData</a>	655
12.13.4.99	<a href="#">pin_registerCallBk</a>	655
12.13.4.100	<a href="#">registerHotplugCallBk</a>	655
12.13.4.101	<a href="#">registerLogCallBk</a>	655
12.13.4.102	<a href="#">SDK_Version</a>	655
12.13.4.103	<a href="#">setAbsoluteLibraryPath</a>	655
12.14	<a href="#">Source_C/libIDT_VP3300_COM.h File Reference</a>	656
12.14.1	<a href="#">Detailed Description</a>	658
12.14.2	<a href="#">Macro Definition Documentation</a>	658
12.14.2.1	<a href="#">IN</a>	658
12.14.2.2	<a href="#">IN_OUT</a>	658
12.14.2.3	<a href="#">OUT</a>	659
12.14.3	<a href="#">Typedef Documentation</a>	659
12.14.3.1	<a href="#">ftpComm_callBack</a>	659
12.14.3.2	<a href="#">httpComm_callBack</a>	659
12.14.3.3	<a href="#">pCMR_callBack</a>	659
12.14.3.4	<a href="#">pCSFS_callBack</a>	659
12.14.3.5	<a href="#">pEMV_callBack</a>	659
12.14.3.6	<a href="#">pMessageHotplug</a>	659
12.14.3.7	<a href="#">pMSR_callBack</a>	659
12.14.3.8	<a href="#">pMSR_callBackp</a>	659
12.14.3.9	<a href="#">pPIN_callBack</a>	660
12.14.3.10	<a href="#">pReadDataLog</a>	660
12.14.3.11	<a href="#">pSendDataLog</a>	660
12.14.3.12	<a href="#">v4Comm_callBack</a>	660
12.14.4	<a href="#">Function Documentation</a>	660
12.14.4.1	<a href="#">comm_registerHTTPCallback</a>	660
12.14.4.2	<a href="#">comm_registerV4Callback</a>	660
12.14.4.3	<a href="#">config_getSerialNumber</a>	660
12.14.4.4	<a href="#">config_getSerialNumber_Len</a>	661

12.14.4.5	<a href="#">ctls_activateTransaction</a>	661
12.14.4.6	<a href="#">ctls_cancelTransaction</a>	662
12.14.4.7	<a href="#">ctls_getAllConfigurationGroups</a>	662
12.14.4.8	<a href="#">ctls_getConfigurationGroup</a>	662
12.14.4.9	<a href="#">ctls_registerCallBk</a>	663
12.14.4.10	<a href="#">ctls_registerCallBkp</a>	663
12.14.4.11	<a href="#">ctls_removeAllApplicationData</a>	663
12.14.4.12	<a href="#">ctls_removeAllCAPK</a>	663
12.14.4.13	<a href="#">ctls_removeApplicationData</a>	663
12.14.4.14	<a href="#">ctls_removeCAPK</a>	663
12.14.4.15	<a href="#">ctls_removeConfigurationGroup</a>	664
12.14.4.16	<a href="#">ctls_retrieveAIDList</a>	664
12.14.4.17	<a href="#">ctls_retrieveApplicationData</a>	664
12.14.4.18	<a href="#">ctls_retrieveCAPK</a>	664
12.14.4.19	<a href="#">ctls_retrieveCAPKList</a>	665
12.14.4.20	<a href="#">ctls_retrieveTerminalData</a>	665
12.14.4.21	<a href="#">ctls_setApplicationData</a>	666
12.14.4.22	<a href="#">ctls_setCAPK</a>	666
12.14.4.23	<a href="#">ctls_setConfigurationGroup</a>	667
12.14.4.24	<a href="#">ctls_setTerminalData</a>	667
12.14.4.25	<a href="#">ctls_startTransaction</a>	667
12.14.4.26	<a href="#">device_activateTransaction</a>	668
12.14.4.27	<a href="#">device_cancelTransaction</a>	670
12.14.4.28	<a href="#">device_close</a>	670
12.14.4.29	<a href="#">device_controlUserInterface</a>	670
12.14.4.30	<a href="#">device_enablePassThrough</a>	671
12.14.4.31	<a href="#">device_getCurrentDeviceType</a>	671
12.14.4.32	<a href="#">device_getFirmwareVersion</a>	671
12.14.4.33	<a href="#">device_getFirmwareVersion_Len</a>	671
12.14.4.34	<a href="#">device_getIDGStatusCodeString</a>	671
12.14.4.35	<a href="#">device_getMerchantRecord</a>	673
12.14.4.36	<a href="#">device_getMerchantRecord_Len</a>	673
12.14.4.37	<a href="#">device_getRTCDateTime</a>	673
12.14.4.38	<a href="#">device_getSDKWaitTime</a>	674
12.14.4.39	<a href="#">device_getThreadStackSize</a>	674
12.14.4.40	<a href="#">device_getTransactionResults</a>	674
12.14.4.41	<a href="#">device_init</a>	674
12.14.4.42	<a href="#">device_isAttached</a>	675
12.14.4.43	<a href="#">device_isConnected</a>	675
12.14.4.44	<a href="#">device_pingDevice</a>	675

12.14.4.45	device_registerCameraCallBk	675
12.14.4.46	device_registerCardStatusFrontSwitchCallBk	675
12.14.4.47	device_registerRKICallBk	675
12.14.4.48	device_SendDataCommandNEO	675
12.14.4.49	device_setBurstMode	676
12.14.4.50	device_setCurrentDevice	676
12.14.4.51	device_setMerchantRecord	677
12.14.4.52	device_setPollMode	677
12.14.4.53	device_setRTCDateTime	678
12.14.4.54	device_setSDKWaitTime	678
12.14.4.55	device_setThreadStackSize	678
12.14.4.56	device_setTransactionExponent	678
12.14.4.57	device_startRKI	678
12.14.4.58	device_startTransaction	679
12.14.4.59	emv_activateTransaction	679
12.14.4.60	emv_allowFallback	680
12.14.4.61	emv_authenticateTransaction	680
12.14.4.62	emv_authenticateTransactionWithTimeout	680
12.14.4.63	emv_cancelTransaction	681
12.14.4.64	emv_completeTransaction	681
12.14.4.65	emv_getAutoAuthenticateTransaction	682
12.14.4.66	emv_getAutoCompleteTransaction	682
12.14.4.67	emv_registerCallBk	682
12.14.4.68	emv_removeAllApplicationData	682
12.14.4.69	emv_removeAllCAPK	682
12.14.4.70	emv_removeAllCRL	682
12.14.4.71	emv_removeApplicationData	682
12.14.4.72	emv_removeCAPK	683
12.14.4.73	emv_removeCRL	683
12.14.4.74	emv_retrieveAIDList	683
12.14.4.75	emv_retrieveApplicationData	683
12.14.4.76	emv_retrieveCAPK	684
12.14.4.77	emv_retrieveCAPKList	684
12.14.4.78	emv_retrieveCRL	685
12.14.4.79	emv_retrieveTerminalData	685
12.14.4.80	emv_setApplicationData	685
12.14.4.81	emv_setApplicationDataTLV	686
12.14.4.82	emv_setAutoAuthenticateTransaction	686
12.14.4.83	emv_setAutoCompleteTransaction	686
12.14.4.84	emv_setCAPK	686

12.14.4.85	emv_setCRL	687
12.14.4.86	emv_setTerminalData	687
12.14.4.87	emv_setTerminalMajorConfiguration	688
12.14.4.88	emv_setTransactionParameters	688
12.14.4.89	emv_startTransaction	688
12.14.4.90	icc_exchangeAPDU	689
12.14.4.91	icc_getICCReaderStatus	689
12.14.4.92	icc_powerOffICC	690
12.14.4.93	icc_powerOnICC	690
12.14.4.94	msr_cancelMSRSwipe	690
12.14.4.95	msr_registerCallBk	690
12.14.4.96	msr_registerCallBkp	690
12.14.4.97	msr_startMSRSwipe	690
12.14.4.98	parseMSRData	691
12.14.4.99	pin_registerCallBk	692
12.14.4.100	registerHotplugCallBk	692
12.14.4.101	registerLogCallBk	692
12.14.4.102	SDK_Version	692
12.14.4.103	SetAbsoluteLibraryPath	692
12.15	Source_C/libIDT_VP3300_USB.h File Reference	692
12.15.1	Detailed Description	695
12.15.2	Macro Definition Documentation	695
12.15.2.1	IN	695
12.15.2.2	IN_OUT	695
12.15.2.3	OUT	695
12.15.3	Typedef Documentation	695
12.15.3.1	ftpComm_callBack	695
12.15.3.2	httpComm_callBack	695
12.15.3.3	pCMR_callBack	696
12.15.3.4	pCSFS_callBack	696
12.15.3.5	pEMV_callBack	696
12.15.3.6	pMessageHotplug	696
12.15.3.7	pMSR_callBack	696
12.15.3.8	pMSR_callBackp	696
12.15.3.9	pPIN_callBack	696
12.15.3.10	pReadDataLog	696
12.15.3.11	pSendDataLog	696
12.15.3.12	v4Comm_callBack	697
12.15.4	Function Documentation	697
12.15.4.1	comm_registerHTTPCallback	697

12.15.4.2 comm_registerV4Callback . . . . .	697
12.15.4.3 config_getSerialNumber . . . . .	697
12.15.4.4 config_getSerialNumber_Len . . . . .	697
12.15.4.5 ctls_activateTransaction . . . . .	697
12.15.4.6 ctls_cancelTransaction . . . . .	699
12.15.4.7 ctls_getAllConfigurationGroups . . . . .	699
12.15.4.8 ctls_getConfigurationGroup . . . . .	699
12.15.4.9 ctls_registerCallBk . . . . .	699
12.15.4.10ctls_registerCallBkp . . . . .	699
12.15.4.11ctls_removeAllApplicationData . . . . .	699
12.15.4.12ctls_removeAllCAPK . . . . .	700
12.15.4.13ctls_removeApplicationData . . . . .	700
12.15.4.14ctls_removeCAPK . . . . .	700
12.15.4.15ctls_removeConfigurationGroup . . . . .	700
12.15.4.16ctls_retrieveAIDList . . . . .	700
12.15.4.17ctls_retrieveApplicationData . . . . .	701
12.15.4.18ctls_retrieveCAPK . . . . .	701
12.15.4.19ctls_retrieveCAPKList . . . . .	702
12.15.4.20ctls_retrieveTerminalData . . . . .	702
12.15.4.21ctls_setApplicationData . . . . .	702
12.15.4.22ctls_setCAPK . . . . .	702
12.15.4.23ctls_setConfigurationGroup . . . . .	703
12.15.4.24ctls_setTerminalData . . . . .	703
12.15.4.25ctls_startTransaction . . . . .	704
12.15.4.26device_activateTransaction . . . . .	705
12.15.4.27device_cancelTransaction . . . . .	706
12.15.4.28device_close . . . . .	706
12.15.4.29device_controlUserInterface . . . . .	707
12.15.4.30device_enablePassThrough . . . . .	707
12.15.4.31device_getCurrentDeviceType . . . . .	708
12.15.4.32device_getFirmwareVersion . . . . .	708
12.15.4.33device_getFirmwareVersion_Len . . . . .	708
12.15.4.34device_getIDGStatusCodeString . . . . .	708
12.15.4.35device_getMerchantRecord . . . . .	709
12.15.4.36device_getMerchantRecord_Len . . . . .	710
12.15.4.37device_getRTCDateTime . . . . .	710
12.15.4.38device_getSDKWaitTime . . . . .	711
12.15.4.39device_getThreadStackSize . . . . .	711
12.15.4.40device_getTransactionResults . . . . .	711
12.15.4.41device_init . . . . .	711



12.15.4.42	device_isAttached . . . . .	711
12.15.4.43	device_isConnected . . . . .	712
12.15.4.44	device_pingDevice . . . . .	712
12.15.4.45	device_registerCameraCallBk . . . . .	712
12.15.4.46	device_registerCardStatusFrontSwitchCallBk . . . . .	712
12.15.4.47	device_registerRKICallBk . . . . .	712
12.15.4.48	device_SendDataCommandNEO . . . . .	712
12.15.4.49	device_setBurstMode . . . . .	713
12.15.4.50	device_setCurrentDevice . . . . .	714
12.15.4.51	device_setMerchantRecord . . . . .	714
12.15.4.52	device_setPollMode . . . . .	715
12.15.4.53	device_setRTCDateTime . . . . .	715
12.15.4.54	device_setSDKWaitTime . . . . .	715
12.15.4.55	device_setThreadStackSize . . . . .	715
12.15.4.56	device_setTransactionExponent . . . . .	715
12.15.4.57	device_startRKI . . . . .	716
12.15.4.58	device_startTransaction . . . . .	716
12.15.4.59	emv_activateTransaction . . . . .	716
12.15.4.60	emv_allowFallback . . . . .	717
12.15.4.61	emv_authenticateTransaction . . . . .	717
12.15.4.62	emv_authenticateTransactionWithTimeout . . . . .	718
12.15.4.63	emv_cancelTransaction . . . . .	718
12.15.4.64	emv_completeTransaction . . . . .	718
12.15.4.65	emv_getAutoAuthenticateTransaction . . . . .	719
12.15.4.66	emv_getAutoCompleteTransaction . . . . .	719
12.15.4.67	emv_registerCallBk . . . . .	719
12.15.4.68	emv_removeAllApplicationData . . . . .	719
12.15.4.69	emv_removeAllCAPK . . . . .	719
12.15.4.70	emv_removeAllCRL . . . . .	720
12.15.4.71	emv_removeApplicationData . . . . .	720
12.15.4.72	emv_removeCAPK . . . . .	720
12.15.4.73	emv_removeCRL . . . . .	720
12.15.4.74	emv_retrieveAIDList . . . . .	720
12.15.4.75	emv_retrieveApplicationData . . . . .	721
12.15.4.76	emv_retrieveCAPK . . . . .	721
12.15.4.77	emv_retrieveCAPKList . . . . .	722
12.15.4.78	emv_retrieveCRL . . . . .	722
12.15.4.79	emv_retrieveTerminalData . . . . .	722
12.15.4.80	emv_setApplicationData . . . . .	722
12.15.4.81	emv_setApplicationDataTLV . . . . .	723

12.15.4.82	emv_setAutoAuthenticateTransaction	723
12.15.4.83	emv_setAutoCompleteTransaction	723
12.15.4.84	emv_setCAPK	723
12.15.4.85	emv_setCRL	724
12.15.4.86	emv_setTerminalData	724
12.15.4.87	emv_setTerminalMajorConfiguration	725
12.15.4.88	emv_setTransactionParameters	725
12.15.4.89	emv_startTransaction	726
12.15.4.90	icc_exchangeAPDU	726
12.15.4.91	icc_getICCRReaderStatus	726
12.15.4.92	icc_powerOffICC	727
12.15.4.93	icc_powerOnICC	727
12.15.4.94	msr_cancelMSRSwipe	727
12.15.4.95	msr_registerCallBk	727
12.15.4.96	msr_registerCallBkp	727
12.15.4.97	msr_startMSRSwipe	727
12.15.4.98	parseMSRData	728
12.15.4.99	pin_registerCallBk	728
12.15.4.100	registerHotplugCallBk	728
12.15.4.101	registerLogCallBk	728
12.15.4.102	SDK_Version	728
12.15.4.103	SetAbsoluteLibraryPath	728
12.16	Source_C/libIDT_VP8800.h File Reference	729
12.16.1	Detailed Description	732
12.16.2	Macro Definition Documentation	733
12.16.2.1	IN	733
12.16.2.2	IN_OUT	733
12.16.2.3	OUT	733
12.16.3	Typedef Documentation	733
12.16.3.1	ftpComm_callBack	733
12.16.3.2	httpComm_callBack	733
12.16.3.3	pCMR_callBack	733
12.16.3.4	pCSFS_callBack	733
12.16.3.5	pEMV_callBack	733
12.16.3.6	pLog_callback	733
12.16.3.7	pMessageHotplug	734
12.16.3.8	pMSR_callBack	734
12.16.3.9	pMSR_callBackp	734
12.16.3.10	pPIN_callBack	734
12.16.3.11	pReadDataLog	734

12.16.3.12pSendDataLog . . . . .	734
12.16.3.13v4Comm_callBack . . . . .	734
12.16.4 Function Documentation . . . . .	734
12.16.4.1 comm_registerHTTPCallback . . . . .	734
12.16.4.2 comm_registerV4Callback . . . . .	735
12.16.4.3 config_getSerialNumber . . . . .	736
12.16.4.4 config_getSerialNumber_Len . . . . .	736
12.16.4.5 ctls_activateTransaction . . . . .	736
12.16.4.6 ctls_cancelTransaction . . . . .	737
12.16.4.7 ctls_displayOnlineAuthResult . . . . .	737
12.16.4.8 ctls_getAllConfigurationGroups . . . . .	738
12.16.4.9 ctls_getConfigurationGroup . . . . .	738
12.16.4.10ctls_registerCallBk . . . . .	738
12.16.4.11ctls_registerCallBkp . . . . .	738
12.16.4.12ctls_removeAllApplicationData . . . . .	738
12.16.4.13ctls_removeAllCAPK . . . . .	739
12.16.4.14ctls_removeApplicationData . . . . .	739
12.16.4.15ctls_removeCAPK . . . . .	739
12.16.4.16ctls_removeConfigurationGroup . . . . .	739
12.16.4.17ctls_retrieveAIDList . . . . .	739
12.16.4.18ctls_retrieveApplicationData . . . . .	740
12.16.4.19ctls_retrieveCAPK . . . . .	740
12.16.4.20ctls_retrieveCAPKList . . . . .	741
12.16.4.21ctls_retrieveTerminalData . . . . .	741
12.16.4.22ctls_setApplicationData . . . . .	741
12.16.4.23ctls_setCAPK . . . . .	741
12.16.4.24ctls_setConfigurationGroup . . . . .	742
12.16.4.25ctls_setTerminalData . . . . .	742
12.16.4.26ctls_startTransaction . . . . .	743
12.16.4.27device_activateTransaction . . . . .	744
12.16.4.28device_calibrateParameters . . . . .	745
12.16.4.29device_cancelTransaction . . . . .	745
12.16.4.30device_close . . . . .	745
12.16.4.31device_controlIndicator . . . . .	746
12.16.4.32device_controlUserInterface . . . . .	746
12.16.4.33device_createDirectory . . . . .	747
12.16.4.34device_deleteDirectory . . . . .	747
12.16.4.35device_deleteFile . . . . .	747
12.16.4.36device_enablePassThrough . . . . .	747
12.16.4.37device_enhancedPassthrough . . . . .	749

12.16.4.38	device_getCurrentDeviceType	749
12.16.4.39	device_getDriveFreeSpace	749
12.16.4.40	device_getFirmwareVersion	749
12.16.4.41	device_getFirmwareVersion_Len	750
12.16.4.42	device_getIDGStatusCodeString	750
12.16.4.43	device_getMerchantRecord	751
12.16.4.44	device_getMerchantRecord_Len	752
12.16.4.45	device_getSDKWaitTime	753
12.16.4.46	device_getThreadStackSize	753
12.16.4.47	device_getTransactionResults	753
12.16.4.48	device_init	754
12.16.4.49	device_isAttached	754
12.16.4.50	device_isConnected	754
12.16.4.51	device_listDirectory	754
12.16.4.52	device_pingDevice	754
12.16.4.53	device_registerCameraCallBk	755
12.16.4.54	device_registerCardStatusFrontSwitchCallBk	755
12.16.4.55	device_SendDataCommandNEO	755
12.16.4.56	device_setCurrentDevice	756
12.16.4.57	device_setMerchantRecord	756
12.16.4.58	device_setSDKWaitTime	757
12.16.4.59	device_setThreadStackSize	757
12.16.4.60	device_setTransactionExponent	757
12.16.4.61	device_startTransaction	757
12.16.4.62	device_transferFile	758
12.16.4.63	emv_activateTransaction	758
12.16.4.64	emv_allowFallback	759
12.16.4.65	emv_authenticateTransaction	759
12.16.4.66	emv_authenticateTransactionWithTimeout	760
12.16.4.67	emv_cancelTransaction	760
12.16.4.68	emv_completeTransaction	760
12.16.4.69	emv_getAutoAuthenticateTransaction	761
12.16.4.70	emv_getAutoCompleteTransaction	761
12.16.4.71	emv_getEMVConfigurationCheckValue	761
12.16.4.72	emv_getEMVKernelCheckValue	761
12.16.4.73	emv_getEMVKernelVersion	762
12.16.4.74	emv_getEMVKernelVersion_Len	762
12.16.4.75	emv_registerCallBk	762
12.16.4.76	emv_removeAllApplicationData	762
12.16.4.77	emv_removeAllCAPK	762

12.16.4.78	emv_removeAllCRL	763
12.16.4.79	emv_removeAllExceptions	763
12.16.4.80	emv_removeApplicationData	763
12.16.4.81	emv_removeCAPK	763
12.16.4.82	emv_removeCRL	763
12.16.4.83	emv_removeException	764
12.16.4.84	emv_removeTransactionLog	764
12.16.4.85	emv_retrieveAIDList	764
12.16.4.86	emv_retrieveApplicationData	764
12.16.4.87	emv_retrieveCAPK	765
12.16.4.88	emv_retrieveCAPKList	766
12.16.4.89	emv_retrieveCRL	766
12.16.4.90	emv_retrieveExceptionList	766
12.16.4.91	emv_retrieveExceptionLogStatus	767
12.16.4.92	emv_retrieveTerminalData	767
12.16.4.93	emv_retrieveTransactionLog	767
12.16.4.94	emv_retrieveTransactionLogStatus	768
12.16.4.95	emv_setApplicationData	769
12.16.4.96	emv_setApplicationDataTLV	769
12.16.4.97	emv_setAutoAuthenticateTransaction	769
12.16.4.98	emv_setAutoCompleteTransaction	770
12.16.4.99	emv_setCAPK	770
12.16.4.100	emv_setCRL	771
12.16.4.101	emv_setException	771
12.16.4.102	emv_setTerminalData	771
12.16.4.103	emv_startTransaction	772
12.16.4.104	d_addItemToList	772
12.16.4.105	d_cancelSlideShow	772
12.16.4.106	d_captureSignature	773
12.16.4.107	d_clearDisplay	773
12.16.4.108	d_clearEventQueue	773
12.16.4.109	d_createInputField	773
12.16.4.110	d_createInputField_Len	775
12.16.4.111	d_createList	776
12.16.4.112	d_createList_Len	777
12.16.4.113	d_customDisplayMode	778
12.16.4.114	d_displayButton	778
12.16.4.115	d_displayButton_Len	780
12.16.4.116	d_displayParagraph	781
12.16.4.117	d_displayText	782

12.16.4.118d_displayText_Len . . . . .	783
12.16.4.118d_getInputEvent . . . . .	784
12.16.4.120d_getInputEvent_Len . . . . .	786
12.16.4.121d_getInputFieldValue . . . . .	787
12.16.4.122d_getSelectedItem . . . . .	788
12.16.4.123d_getSelectedItem_Len . . . . .	788
12.16.4.124d_resetInitialState . . . . .	788
12.16.4.125d_setBackgroundImage . . . . .	788
12.16.4.126d_setDisplayImage . . . . .	789
12.16.4.127d_setForeBackColor . . . . .	790
12.16.4.128d_startSlideShow . . . . .	790
12.16.4.129sr_cancelMSRSwipe . . . . .	791
12.16.4.130sr_flushTrackData . . . . .	791
12.16.4.131sr_registerCallBk . . . . .	791
12.16.4.132sr_registerCallBkp . . . . .	791
12.16.4.133sr_startMSRSwipe . . . . .	791
12.16.4.134sr_parseMSRData . . . . .	792
12.16.4.135in_getEncryptedOnlinePIN . . . . .	792
12.16.4.136in_getPAN . . . . .	792
12.16.4.137in_promptCreditDebit . . . . .	792
12.16.4.138in_registerCallBk . . . . .	793
12.16.4.139in_registerHotplugCallBk . . . . .	793
12.16.4.140in_registerLogCallBk . . . . .	793
12.16.4.143SDK_Version . . . . .	793
12.16.4.142in_setAbsoluteLibraryPath . . . . .	793
12.16.4.148s_deleteSSLCert . . . . .	793
12.16.4.144s_getCertChainType . . . . .	794
12.16.4.145s_loadSSLCert . . . . .	794
12.16.4.146s_requestCSR . . . . .	794
12.16.4.147s_revokeSSLCert . . . . .	794
12.16.4.148s_updateRootCertificate . . . . .	795

## Chapter 1

# ID TECH Universal SDK Reference Guide for Linux/Windows/Mac (C++/Java)

ID TECH provides this Universal SDK to drive multiple devices across multiple platforms.

The current version of the SDK supports the USB-HID interface of the listed ID TECH products. For devices that also have RS-232 interfaces, SDK communication support for COM on those devices is under development and will be released at a later date.

This SDK encompasses support for the following devices and platforms. Other ID TECH products are scheduled to be added in an upcoming release. Java supports all the devices that C/C++ supports.

### C/C++

- **Platforms:** Macintosh, Windows, Linux Desktop (x86\_64/amd64), Linux ARM (RaspberryPi)
- **Products:** UniPay 1.5, UniPay III, VP4880, MiniSmartII, BTPayMini, SpectrumPro, Kiosk III, Augusta

### Java

- **Platforms:** Macintosh, Windows, Linux Desktop (x86\_64/amd64), Linux ARM (RaspberryPi)
- **Products:** UniPay 1.5, UniPay III, VP4880, MiniSmartII, BTPayMini, SpectrumPro, Kiosk III, Augusta

### Pre-requisites:

- **Macintosh:** None
- **Windows, Linux:** libusb-1.0

Note:\*\* We recommended installing libusb from the distributors' downloads. If that becomes a challenge, the SDK includes just the libusb library for Windows, Linux x86\_65, and Linux ARM.

## 1.1 Demo Apps

### Pre-Requisites C/C++:

- Eclipse for C/C++
- Windows: MinGW
- Linux ARM on x86\_64 (cross compile): Poky

### Pre-Requisites Java:

- Eclipse for Java Developers
- Java 1.8 or greater

## 1.2 Purpose

This document describes API requirements as well as the interface definitions and requirements for an integrator wishing to integrate it into a payment application.

- [Core Implementation: C/C++](#)
- [Core Implementation: Java](#)
- [Important Security Notice](#)
- [Main Transaction Commands](#)
- [EMV Callback](#)
- [EMV Tag Reference](#)
- [Enumeration Reference](#)
- [Error Code Reference](#)
- [LCD Foreign Language Mapping Table](#)



## Chapter 2

# Important Security Notice

The Payment Card Industry Payment Application Data Security Standard (PCI PA-DSS) is comprised of fourteen requirements that support the Payment Card Industry Data Security Standard (PCI DSS). The PCI Security Standards Council (PCI SSC), which was founded by the major card brands in June 2005, set these requirements in order to protect cardholder payment information. The standards set by the council are enforced by the payment card companies who established the Council: American Express, Discover Financial Services, JCB International, MasterCard Worldwide, and Visa, Inc.

PCI PA-DSS is an evolution of Visas Payment Application Best Practices (PABP), which was based on the Visa Cardholder Information Security Program (CISP). In addition to Visa CISP, PCI DSS combines American Express Data Security Operating Policy (DSOP), Discover Networks Information Security and Compliance (DISC), and MasterCard Site Data Protection (SDP) into a single comprehensive set of security standards. The transition to PCI PA-DSS was announced in April 2008. In early October 2008, PCI PA-DSS Version 1.2 was released to align with the PCI DSS Version 1.2, which was released on October 1, 2008. On January 1, 2011, PCI PA-DSS Version 2.0 was released. This extends the PCI DSS Version 1.2, which was released on October 1, 2008 and is effective as of January 1, 2011.

## 2.1 Applicability

The PCI PA-DSS applies to any payment application that stores, processes, or transmits cardholder data as part of authorization or settlement, unless the application would fall under the merchant's PCI DSS validation. It is important to note that PA-DSS validated payment applications alone do not guarantee PCI DSS compliance for the merchant. The validated payment application must be implemented in a PCI DSS compliant environment. If your application runs on Windows XP, you are required to turn off Windows XP System Restore Points.

## 2.2 What Does PA-DSS Mean to You?

The following table provides opening points to cover in any discussion with merchants on data storage.

	<i>Data Element</i>	<i>Storage Permitted</i>	<i>Protection Required</i>	<i>PCI DSS Req. 3, 4</i>
<b>Cardholder Data</b>	<i>Primary Account Number</i>	Yes	Yes	Yes
	<i>Cardholder Name <sup>1</sup></i>	Yes	Yes <sup>1</sup>	No
	<i>Service Code <sup>1</sup></i>	Yes	Yes <sup>1</sup>	No
	<i>Expiration Date <sup>1</sup></i>	Yes	Yes <sup>1</sup>	No
<b>Sensitive Authentication Data <sup>2</sup></b>	<i>Full Magnetic Stripe Data <sup>3</sup></i>	No	N/A	N/A
	<i>CAV2/CID/CVC2/CVV2</i>	No	N/A	N/A
	<i>PIN/PIN Block</i>	No	N/A	N/A

<sup>1</sup> These data elements must be protected if stored in conjunction with the PAN. This protection should be per PCI DSS requirements for general protection of the cardholder environment. Additionally, other legislation (for example, related to consumer personal data protection, privacy, identity theft, or data security) may require specific protection of this data, or proper disclosure of a company's practices if consumer-related personal data is being collected during the course of business. PCI DSS, however, does not apply if PANs are not stored, processed, or transmitted.

<sup>2</sup> Do not store sensitive authentication data after authorization (even if encrypted).

<sup>3</sup> Full track data from the magnetic stripe, magnetic-stripe image on the chip, or elsewhere.

## 2.3 Third Party Applications

The end-to-end transaction process, beginning with entry into the third party application until the response from the payment engine is returned, must meet the same level of compliance. In order to claim the third party application is end-to-end compliant, the application would need to be submitted to a QSA for a full PA-DSS audit.

The end user and/or P.O.S. developer can integrate and be compliant in the processing portion of a payment transaction. A brief review (given below) of the PA-DSS environmental variables that impact the end user merchant can help the end user merchant obtain and/or maintain PA-DSS compliance. Environmental variables that could prevent passing an audit include without limitation issues involving a secure network connection(s), end user setup location security, users, logging and assigned rights. Remove all testing configurations, samples, and data prior to going into production on your application.

## 2.4 PA-DSS Guidelines

The following PA-DSS Guidelines are being provided by ID TECH as a convenience to its customers. Customers should not rely on these PA-DSS Guidelines, but should instead always refer to the most recent PCI DSS Program Guide published by PCI SSC.

### 1. Sensitive Data Storage Guidelines

Do not retain full magnetic stripe, card validation code or value (CAV2, CID, CVC2, CVV2), or PIN block data.

1.1 Do not store sensitive authentication data after authorization (even if encrypted): Sensitive authentication data includes the data as cited in the following Requirements 1.1.1 through 1.1.3. PCI Data Security Standard Requirement 3.2

Note: By prohibiting storage of sensitive authentication data after authorization, the assumption is that the transaction has completed the authorization process and the customer has received the final transaction approval. After authorization has completed, this sensitive authentication data cannot be stored.

1.1.1 After authorization, do not store the full contents of any track from the magnetic stripe (located on the back of a card, contained in a chip, or elsewhere). This data is alternatively called full track, track, track 1, track 2, and magnetic-stripe data.

In the normal course of business, the following data elements from the magnetic stripe may need to be retained:

- The accountholders name,
- Primary account number (PAN),
- Expiration date, and
- Service code
- To minimize risk, store only those data elements needed for business.

Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.1

1.1.2 After authorization, do not store the card-validation value or code (three-digit or four-digit number printed on the front or back of a payment card) used to verify card-not-present transactions. Note: See PCI DSS and P-A-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.2

1.1.3 After authorization, do not store the personal identification number (PIN) or the encrypted PIN block.

Note: See PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms for additional information. PCI Data Security Standard Requirement 3.2.3

1.1.4 Securely delete any magnetic stripe data, card validation values or codes, and PINs or PIN block data stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example by the list of approved products maintained by the National Security Agency, or by other State or National standards or regulations. PCI Data Security Standard Requirement 3.2

Note: This requirement only applies if previous versions of the payment application stored sensitive authentication data.

1.1.5 Securely delete any sensitive authentication data (pre-authorization data) used for debugging or troubleshooting purposes from log files, debugging files, and other data sources received from customers, to ensure that magnetic stripe data, card validation codes or values, and PINs or PIN block data are not stored on software vendor systems. These data sources must be collected in limited amounts and only when necessary to resolve a problem, encrypted while stored, and deleted immediately after use. PCI Data Security Standard Requirement 3.2

## 2. Protect stored cardholder data

2.1 Software vendor must provide guidance to customers regarding purging of cardholder data after expiration of customer-defined retention period. PCI Data Security Standard Requirement 3.1

2.2 Mask PAN when displayed (the first six and last four digits are the maximum number of digits to be displayed).

Notes:

- This requirement does not apply to those employees and other parties with a legitimate business need to see full PAN;
- This requirement does not supersede stricter requirements in place for displays of cardholder data for example, for point-of-sale (POS) receipts. PCI Data Security Standard Requirement 3.3

2.3 Render PAN, at a minimum, unreadable anywhere it is stored, (including data on portable digital media, backup media, and in logs) by using any of the following approaches:

- One-way hashes based on strong cryptography with associated key management processes and procedures

- Truncation
- Index tokens and pads (pads must be securely stored)
- Strong cryptography with associated key management processes and procedures. The MINIMUM account information that must be rendered unreadable is the PAN. PCI Data Security Standard Requirement 3.4

The PAN must be rendered unreadable anywhere it is stored, even outside the payment application. Note: Strong cryptography is defined in the PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms.

2.4 If disk encryption is used (rather than file- or column-level database encryption), logical access must be managed independently of native operating system access control mechanisms (for example, by not using local user account databases). Decryption keys must not be tied to user accounts. PCI Data Security Standard Requirement 3.4.2

2.5 Payment application must protect cryptographic keys used for encryption of cardholder data against disclosure and misuse. PCI Data Security Standard Requirement 3.5

2.6 Payment application must implement key management processes and procedures for cryptographic keys used for encryption of cardholder data. PCI Data Security Standard Requirement 3.6

2.7 Securely delete any cryptographic key material or cryptogram stored by previous versions of the payment application, in accordance with industry-accepted standards for secure deletion, as defined, for example the list of approved products maintained by the National Security Agency, or by other State or National standards or regulations. These are cryptographic keys used to encrypt or verify cardholder data. PCI Data Security Standard Requirement 3.6

Note: This requirement only applies if previous versions of the payment application used cryptographic key materials or cryptograms to encrypt cardholder data.

### 3. Provide secure authentication features

3.1 The payment application must support and enforce unique user IDs and secure authentication for all administrative access and for all access to cardholder data. Secure authentication must be enforced to all accounts, generated or managed by the application by the completion of installation and for subsequent changes after the "out of the box" installation (defined at PCI DSS Requirements 8.1, 8.2, and 8.5.88.5.15) for all administrative access and for all access to cardholder data. PCI Data Security Standard Requirements 8.1, 8.2, and 8.5.88.5.15

Note: These password controls are not intended to apply to employees who only have access to one card number at a time to facilitate a single transaction. These controls are applicable for access by employees with administrative capabilities, for access to servers with cardholder data, and for access controlled by the payment application. This requirement applies to the payment application and all associated tools used to view or access cardholder data.

3.1.10 If a payment application session has been idle for more than 15 minutes, the application requires the user to re-authenticate. PCI Data Security Standard Requirement 8.5.15.

3.2 Software vendors must provide guidance to customers that all access to PCs, servers, and databases with payment applications must require a unique user ID and secure authentication. PCI Data Security Standard Requirements 8.1 and 8.2

3.3 Render payment application passwords unreadable during transmission and storage, using strong cryptography based on approved standards

Note: Strong cryptography is defined in PCI DSS and PA-DSS Glossary of Terms, Abbreviations, and Acronyms. PCI Data Security Standard Requirement 8.4

### 4. Log payment application activity

4.1 At the completion of the installation process, the out of the box default installation of the payment application must log all user access (especially users with administrative privileges), and be able to link all activities to individual users. PCI Data Security Standard Requirement 10.1

4.2 Payment application must implement an automated audit trail to track and monitor access. PCI Data Security Standard Requirements 10.2 and 10.3

## 5. Develop secure payment applications

5.1 Develop all payment applications in accordance with PCI DSS (for example, secure authentication and logging) and based on industry best practices and incorporate information security throughout the software development life cycle. These processes must include the following: PCI Data Security Standard Requirement 6.3

5.1.1 Live PANS are not used for testing or development. PCI Data Security Standard Requirement 6.4.4.

- Validation of all input (to prevent cross-site scripting, injection flaws, malicious file execution, etc.)
- Validation of proper error handling
- Validation of secure cryptographic storage
- Validation of secure communications
- Validation of proper role-based access control (RBAC)

5.1.2 Separate development/test, and production environments

5.1.3 Removal of test data and accounts before production systems become active development. PCI Data Security Standard Requirement 6.4.4

5.1.4 Review of payment application code prior to release to customers after any significant change, to identify any potential coding vulnerability. Removal of custom payment application accounts, user IDs, and passwords before payment applications are released to customers

Note: This requirement for code reviews applies to all payment application components (both internal and public-facing web applications), as part of the system development life cycle required by PA-DSS Requirement 5.1 and PCI DSS Requirement 6.3. Code reviews can be conducted by knowledgeable internal personnel or third parties.

5.2 Develop all web payment applications (internal and external, and including web administrative access to product) based on secure coding guidelines such as the Open Web Application Security Project Guide. Cover prevention of common coding vulnerabilities in software development processes, to include:

- Injection flaws, with particular emphasis on SQL injection, Cross-site scripting (XSS) OS Command Injection, LDAP and Xpath injection flaws, as well as other injection flaws.
- Buffer Overflow.
- Insecure cryptographic storage.
- Insecure communications.
- Improper error handling.
- All HIGH vulnerabilities as identified in the vulnerability identification process at PA-DSS Requirement 7.1.
- Cross-site scripting (XSS)
- Improper access control such as insecure direct object references, failure to restrict URL access and directory traversal.
- Cross-site request forgery (CSRF)

Note: The vulnerabilities listed in PA-DSS Requirements 5.2.1 through 5.2.9 and in PCI DSS at 6.5.1 through 6.5.9 were current in the OWASP guide when PCI DSS v1.2 / PCI DSS v2.0 (01/01/10) were published. However, if and when the OWASP guide is updated, the current version must be used for these requirements.

5.3 Software vendor must follow change control procedures for all product software configuration changes. PCI Data Security Standard Requirement 6.4. 5. The procedures must include the following:

- Documentation of impact

- Management sign-off by appropriate parties
- Testing functionality to verify the new change(s) does not adversely impact the security of the system. Remove all testing configurations, samples, and data before finalizing the product for production.
- Back-out or product de-installation procedures

5.4 The payment application must not use or require use of unnecessary and insecure services and protocols (for example, NetBIOS, file-sharing, Telnet, unencrypted FTP must be secured via SSH, S-FTP, SSL, IPSec and other technology to implement end to end security). PCI Data Security Standard Requirement 2.2.2

## 6. Protect wireless transmissions

6.1 For payment applications using wireless technology, the wireless technology must be implemented securely. Payment applications using wireless technology must facilitate use of industry best practices (for example, IEEE 802.11i) to implement strong encryption for authentication and transmission. Controls must be in place to protect the implemented wireless network from unknown wireless access points and clients. This includes testing the end users wireless deployment on a quarterly basis to detect unauthorized access points within the system. Change wireless vendor defaults, including but not limited to default wireless encryption keys, passwords, and SNMP community strings. Maintain a detailed updated hardware list. The end to end wireless implementation must be end to end secure. The use of WEP as a security control was prohibited as of 30 June 2010. PCI Data Security Standard Requirements 1.2.3, 2.1.1, 4.1.1, 6.2, 11.1a-e and 11.4a-c.

## 7. Test payment applications to address vulnerabilities

7.1 Software vendors must establish a process to identify newly discovered security vulnerabilities (for example, subscribe to alert services freely available on the Internet) and to test their payment applications for vulnerabilities. Any underlying software or systems that are provided with or required by the payment application (for example, web servers, third-party libraries and programs) must be included in this process. Remove all test configurations, samples, and data after testing and before promoting the changes to production. PCI Data Security Standard Requirement 6.2

7.2 Software vendors must establish a process for timely development and deployment of security patches and upgrades, which includes delivery of updates and patches in a secure manner with a known chain-of-trust, and maintenance of the integrity of patch and update code during delivery and deployment.

## 8. Facilitate secure network implementation

8.1 The payment application must be able to be implemented into a secure network environment. Application must not interfere with use of devices, applications, or configurations required for PCI DSS compliance (for example, payment application cannot interfere with anti-virus protection, firewall configurations, or any other device, application, or configuration required for PCI DSS compliance). PCI Data Security Standard Requirements 1, 3, 4, 5, and 6.

## 9. Cardholder data must never be stored on a server connected to the Internet

9.1 The payment application must be developed such that the database server and web server are not required to be on the same server, nor is the database server required to be in the DMZ with the web server. PCI Data Security Standard Requirement 1.3.7

## 10. Facilitate secure remote software updates

10.1 If payment application updates are delivered securely via remote access into customers systems, software vendors must tell customers to turn on remote-access technologies only when needed for downloads from vendor and to turn off immediately after download completes. Alternatively, if delivered via VPN or other high-speed connection, software vendors must advise customers to properly configure a firewall or a personal firewall product to secure authentication using a two factor authentication mechanism. PCI Data Security Standard Requirement 8.3

10.2 If payment application may be accessed remotely, remote access to the payment application must be authenticated using a two factor authentication mechanism. PCI Data Security Standard Requirement 8.3

10.3 Any remote access into the payment application must be done securely. If vendors, resellers/integrators, or customers can access customers payment applications remotely, the remote access must be implemented securely. PCI Data Security Standard Requirements 1, 8.3 and 12.3.9

## 11. Encrypt sensitive traffic over public networks

11.1 If the payment application sends, or facilitates sending, cardholder data over public networks, the payment application must support use of strong cryptography and security protocols such as SSL/TLS and Internet protocol security (IPSEC) to safeguard sensitive cardholder data during transmission over open, public networks. Examples of open, public networks that are in scope of the PCI DSS are: The Internet Wireless technologies Global System for Mobile Communications (GSM) General Packet Radio Service (GPRS) PCI Data Security Standard Requirement 4.1

11.2 The payment application must never send unencrypted PANs by end-user messaging technologies (for example, e-mail, instant messaging, and chat). PCI Data Security Standard Requirement 4.2

## 12. Encrypt all non-console administrative access

12.1 Instruct customers to encrypt all non-console administrative access using technologies such as SSH, VPN, or SSL/TLS for web-based management and other non-console administrative access. Telnet or remote login must never be used for administrative access. PCI Data Security Standard Requirement 2.3

## 13. Maintain instructional documentation and training programs for customers, resellers, and integrators

13.1 Develop, maintain, and disseminate a PA-DSS Implementation Guide(s) for customers, resellers, and integrators that accomplishes the following:

- Addresses all requirements in this document wherever the PA-DSS Implementation Guide is referenced.
- Includes a review at least annually and updates to keep the documentation current with all major and minor software changes as well as with changes to the requirements in this document.

13.2 Develop and implement training and communication programs to ensure payment application resellers and integrators know how to implement the payment application and related systems and networks according to the PA-DSS Implementation Guide and in a PCI DSS-compliant manner.

- Update the training materials on an annual basis and whenever new payment application versions are released.

## 2.5 More Information

ID TECH Systems, Inc. highly recommends that merchants contact the card association(s) or their processing company and find out exactly what they mandate and/or recommend. Doing so may help merchants protect themselves from fines and fraud.

For more information related to security, visit:

- <http://www.pcisecuritystandards.org>
- <http://www.visa.com/cisp>
- <http://www.sans.org/resources>
- <http://www.microsoft.com/security/default.asp>
- <https://sdp.mastercardintl.com/>
- <http://www.americanexpress.com/merchantspecs>

CAPN questions: [capninfocenter@aexp.com](mailto:capninfocenter@aexp.com)



## Chapter 3

# Main Transaction Commands

The methods below are provided as a reference to the main commands needed to execute an EMV transaction.

### 3.1 EMV Methods

#### Start EMV Transaction

`emv_startTransaction()`

Begins an amount authorization request with the ICC. Returns authorization decision (approved, denied, or go online) in the callback method.

#### Authenticate EMV Transaction

`emv_authenticateTransaction()`

When the results to `emv_startTransaction()` come back as **EMV\_RESULT\_CODE.EMV\_RESULT\_CODE\_AUTHENTICATE\_TRANSACTION**, continuing the EMV transaction requires calling this method.

#### Complete Online EMV Transaction

`emv_completeTransaction()`

If start/authenticate transaction returns **EMV\_RESULT\_CODE.EMV\_RESULT\_CODE\_GO\_ONLINE**, finishing the transaction requires executing `emv_completeTransaction()`.

After receiving a host response, pass host tags (minimum 8A Authorization Response Code) as a parameter.

If there was a communication error with host, finishing the EMV transaction still requires passing "TRUE" for **comm-Error**.

#### Terminal Configuration

`emv_retrieveTerminalData()`

`emv_removeTerminalData()`

`emv_setTerminalData()`

Methods for terminal configuration. When setting terminal data, pass the tags in TLV format.

#### AID Management

`emv_retrieveApplicationData()`

`emv_removeApplicationData()`

`emv_removeAllApplicationData()`

`emv_setApplicationData()`

#### `emv_retrieveAIDList()`

Methods for AID management. When setting the AID, pass the tags in TLV format. When retrieving the AID, receive the results as tags in TLV format.

#### CAPK Management

`emv_retrieveCAPK()`  
`emv_removeCAPK()`  
`emv_removeAllCAPK()`  
`emv_setCAPK()`  
`emv_retrieveCAPKList()`

Methods for Certificate Authority Public Key management. When setting the CAPK, populate and pass the key as a sequence of ordered bytes. When specifying a CAPK to retrieve or remove, populate the name in the byte\* pointer. When retrieving the CAPK list, retrieve the list of RID/Index from the ordered byte stream, 6 bytes each, bytes 1-5 RID, byte 6 index.

#### CRL Management

`emv_removeCRL()`  
`emv_removeAllCRL()`  
`emv_retrieveCRL()`  
`emv_setCRL()`

Methods for Certificate Revocation List management.

#### Kernel Version

`emv_getEMVKernelVersion()`

Method to retrieve the kernel version.

#### Kernel Check Value

`emv_getEMVKernelCheckValue()`

Method to retrieve the kernel Check Value.

#### EMV Configuration Check Value

`emv_getEMVConfigurationCheckValue()`

Method to retrieve the EMV configuration check value.

## 3.2 MSR Methods

#### Start MSR Swipe

`msr_startMSRSwipe()`

Starts a swipe request. Returns card data in the callback method.

#### Cancel MSR Swipe

`msr_cancelMSRSwipe()`

Cancels a swipe request.

## Chapter 4

# Core Implementation: C/C++

IDTechSDK (libIDTechSDK.so-x\_xx\_xxx / libIDTechSDK-x\_xx\_xxx.dll / libIDTechSDK-x\_xx\_xxx.dylib) includes API methods to interface with ID TECH devices. This guide assumes a fair understanding of Eclipse, C, and general Linux, Windows, or Mac programming knowledge.

### 4.1 Integrating with IDTechSDK

- [Import the Necessary Libraries](#)
- [Add Include Statements to Use Libraries](#)
- [Implement the Callback Function](#)
- [Initialize the Target Device](#)

### 4.2 Import the Necessary Libraries

#### Header Files

Communicating with ID TECH devices requires that developers include the following header files in the project's source code folder:

- IDTDef.h
- The appropriate device header file for a single device OR **IDT\_Device.h** to expose all methods for all devices. Usually, using the single device header file is appropriate as it filters out all unrelated methods.

#### Libraries

- libUSB installed on the system (not applicable to Mac).
- Place the IDTechSDK library file the system PATH of the target device.

#### IMPORTANT:

IDTechSDK libraries are distributed with versioning as part of the library name (libIDTechSDK.so-x\_xx\_xxx / libIDTechSDK-x\_xx\_xxx.dll / libIDTechSDK-x\_xx\_xxx.dylib). The system must recognize them as a file WITHOUT the version info ((libIDTechSDK.so / libIDTechSDK.dll / libIDTechSDK.dylib). Accomplish this by either RENAMING the libraries by removing the version info OR create a Symbolic Link pointing to the original libraries and then remove the version info from the symbolic link:

(Symbolic Link) libIDTechSDK.so -> libIDTechSDK.so-x\_xx\_xxx (Compiled Library)

(Symbolic Link) libIDTechSDK.dll -> libIDTechSDK-x\_xx\_xxx.dll (Compiled Library)

(Symbolic Link) libIDTechSDK.dylib -> libIDTechSDK-x\_xx\_xxx.dylib (Compiled Library)

(Symbolic Link) /lib/libIDTechSDK.so -> /home/<USER>/proj/libIDTechSDK.so-x\_xx\_xxx (Compiled Library)

### 4.3 Add Include Statements to Use Libraries

Add a line of code to use the header files for IDTechSDK at the start of the file (a SpectrumPro header file is used here as an example):

```
#include <stdlib.h>
#include <stdio.h>

#include "IDTDef.h"
#include "libIDT_SpectrumPro.h"
```

### 4.4 Implement the Callback Function

There are two callbacks to implement, one for MSR and one for EMV:

```
void MSR_callBack(int type, IDTMSRData cardData){
    printf("\nMSR Callback\n");
    switch (type){
        case MSR_callBack_type_ERR:
            printf("Callback MSR cancelled\n");
            break;
        case MSR_callBack_type_RETURN_CODE:
            printf("Callback MSR data received\n");
            break;
        case MSR_callBack_type_TIMEOUT:
            printf("MSR Callback Timeout\n");
            break;
        default:
            break;
    }
}

void EMV_callBack(int device_type, int device_state, unsigned char * data, int dataLen, IDTTransactionData*
    cardData, EMV_Callback* emvCallback, int transactionResultCode){

    switch(device_state)
    {
        case EMVCallback:
            printf ("EMV Callback\n");
            break;
        case TransactionData:
            printf ("Transaction Data Callback\n");
            break;
        case TransactionFailed:
            printf("Transaction Failed Callback");
            break;
    }
}
```

### 4.5 Initialize SDK and Set the Target Device

Perform [device\\_init\(\)](#) to initialize the SDK, establish the callbacks, and use the [device\\_setCurrentDevice\(\)](#) function to specify the device to use.

```
int main(void) {
    int r = 0;
    printf("Initializing SDK...\n");
    r = device_init();
    if ( r != RETURN_CODE_DO_SUCCESS ) {
        printf(" Fail to init!\n");
        return 0;
    }
    emv_registerCallBk(EMV_callBack);
    msr_registerCallBk(MSR_callBack);
    device_setCurrentDevice(IDT_DEVICE_SPECTRUM_PRO)
    return 0;
}
```

## Chapter 5

# Core Implementation: Java

The **jnibridge.jar** Java library includes API methods to interface with ID TECH devices, accomplished by mapping Java methods onto native C methods used in IDTechSDK (IDTechSDK.so-x\_xx\_xxx / IDTechSDK-x\_xx\_xxx.dll / IDTechSDK-x\_xx\_xxx.dylib). This guide assumes a fair understanding of Eclipse, C, and general Linux, Windows, or Mac programming knowledge.

### 5.1 Integrating with IDTechSDK

- [Add the Necessary Libraries](#)
- [Add Import Statements to Use Libraries](#)
- [Implement the Callback Function](#)
- [Initialize the SDK](#)

### 5.2 Add the Necessary Libraries

#### Java Library

- import jnibridge.jar into your Java application.

#### Native Libraries

- libUSB must be installed on the system (not applicable to Mac).
- Place the IDTechSDK library file the target device's system PATH.

#### IMPORTANT:

IDTechSDK libraries are distributed with versioning as part of the library name (IDTechSDK.so-x\_xx\_xx x/ I- DTechSDK-x\_xx\_xxx.dl / IDTechSDK-x\_xx\_xxx.dylib). The system must recognize them as a file WITHOUT the version info ((IDTechSDK.so / IDTechSDK.dll / IDTechSDK.dylib). Accomplish this by either RENAMING the libraries by removing the version info OR create a Symbolic Link pointing to the original libraries and then remove the version info from the symbolic link:

(Symbolic Link) IDTechSDK.so -> IDTechSDK.so-x\_xx\_xxx (Compiled Library)

(Symbolic Link) IDTechSDK.dll -> IDTechSDK-x\_xx\_xxx.dll (Compiled Library)

(Symbolic Link) IDTechSDK.dylib -> IDTechSDK-x\_xx\_xxx.dylib (Compiled Library)

### 5.3 Add Import Statements to Use Libraries

Add code to use the the Java bridge methods for IDTechSDK at the start of the file (using example of UniPayIII as the target device):

```
import IDTechSDK.ICCReaderStatusStruct;
import IDTechSDK.IDTEMVData;
import IDTechSDK.IDTMSRData;
import IDTechSDK.IDT_UniPayIII;
import IDTechSDK.IDTechSDKBridge;
import IDTechSDK.OnReceiverListener;
import IDTechSDK.ReaderInfo;
import IDTechSDK.ResDataStruct;
import IDTechSDK.StructConfigParameters;
```

## 5.4 Implement the Callback Function

Create a static class named **OnReceiverListenerImp** that implements **OnReceiverListener**. Put in all the required listeners.

NOTE:\*\* although many of the items below are not used in this Native implementation, they still must be defined.

```
static class OnReceiverListenerImp implements OnReceiverListener {
    public void swipeMSRData(IDTMSRData card) {
    }

    public void lcdDisplay(int mode, String[] lines, int timeout) {
    }

    public void emvTransactionData(IDTEMVData emvData) {
    }

    public void deviceConnected() {
    }

    public void deviceDisconnected() {
    }

    public void timeout(int errorCode) {
    }

    public void autoConfigCompleted(StructConfigParameters profile) { } //NOT USED
    public void autoConfigProgress(int progressValue) { } //NOT USED
    public void msgRKICompleted(String MACResult) { } //NOT USED
    public void ICCNotifyInfo(byte[] dataNotify , String strMessage) { } //NOT USED
    public void msgBatteryLow() { } //NOT USED
    public void LoadXMLConfigFailureInfo(int index , String strMessage) { } //NOT USED
    public void msgToConnectDevice() { } //NOT USED
    public void msgAudioVolumeAjustFailed() { } //NOT USED
    public void dataInOutMonitor(byte[] data, boolean isIncoming) {
    }
}
```

## 5.5 Initialize SDK

Create an instance of **OnReceiverListenerImp**, then pass that to a new instance of the device class with which to communicate.

NOTE:\*\* the SDK has a logging function that can be enabled by passing a parameter "2" as initial arguments.

```
public static void main(String[] args) {
    if (args.length > 0){
        System.out.println("\tLogging Level " + args[0]);
        IDTechSDK.IDTechSDKBridge.enableLogging(Integer.valueOf(args[0]));
    }
    OnReceiverListenerImp MessageCallBack = new OnReceiverListenerImp();
    device = new IDT_UniPayIII(MessageCallBack);
}
```

See JNI\_TEST included with SDK for an example of communicating with a UniPay III.

## Chapter 6

# LCD Foreign Language Mapping Table

ID	Message ID	English	French	Spanish	Chinese
0	MSG_NULL	-	-	-	-
1	MSG_AMOUNT	AMOUNT	MONTANT	CANTIDAD	金
2	MSG_AMOUNT_OK	AMOUNT OK?	MONTANT OK	MONTO CORRECTO?	确定金
3	MSG_APPROVED	APPROVED	APPROUVE	APROVADO	通
4	MSG_CALL_YOUR_BANK	CALL YOUR BANK	APPE VOTRE BANQUE	LLAME A SU BANCO	系您的行
5	MSG_CANCEL_OR_ENTER	CANCEL OR ENTER	ANNULE OU ENTRER	CANCEL O ENTRAR	取消或确定
6	MSG_CARD_ERROR	CARD ERROR	ERREUR CARTE	ERROR DE TARJETA	卡
7	MSG_DECLINED	DECLINED	REFUSE	DECLINADO	卡被拒
8	MSG_ENTER_AMOUNT	ENTER AMOUNT	ENTRER MONTANT	INGRESE MONTO	入金
9	MSG_ENTER_PIN	ENTER PIN:	ENTRER PIN:	ENTRAR NPI:	入密
10	MSG_INCORRECT_PIN	INCORRECT PIN	NIP INCORRECT	NPI INCORRECTO	密
11	MSG_ICC_MS-R1	SWIPE OR INSERT	PASSER OU INSERT	MOVER O INSERT	刷卡或插卡
12	MSG_ICC_MS-R2	CARD	CARTE	TARJETA	卡
13	MSG_INSERT_CARD	INSERT CARD	INSERT CARTE	INSERTAR TARJETA	插卡
14	MSG_USE_CHIP_READER	USE CHIP READER UTI	LECTEUR CHIP	USO CHIP LECTOR	使用芯片卡
15	MSG_NOT_ACCEPTED	NOT ACCEPTED	PAS ACCEPTE	DENEGADO	法接受
16	MSG_PIN_OK	GET PIN OK	-	-	密正确
17	MSG_PLEASE_WAIT	PLEASE WAIT...	ATTENDRE...	POR FAVOR ESPERE	等候中



18	MSG_PROCE- SSING_ERRO- R	PROCESSING ERROR	ERREUR DE TRAITE	ERROR PRO- CESANDO	理
19	MSG_USE_M- AGSTRIPE	USE MAGSTRIPE	USAGE MAGSTRIPE	USO DE MAGSTRIPE	使用磁卡
20	MSG_TRY_A- GAIN	TRY AGAIN	REESSAYER	VUELV INTENTARLO	重
21	MSG_ONLINE	GO ONLINE	GO LIGNE	GO LINEA	在
22	MSG_TRANS- ACTION_ERR- OR	TRANSACTION ERR	ERREUR DE TRANS	ERROR DE TRANSAC	交易
23	MSG_TERMIN- ATE	TERMINATE	RESILIER	TERMINAR	止
24	MSG_ADVICE	ADVICE	CONSEILS	CONSEJOS	建
25	MSG_TIMEO- UT	TIME OUT	TIMEOUT	TIEMPO DE ESPERA	超
26	MSG_PROCE- SSING	PROCESSIN- G...	PROCESSU- S...	PROCESAND- O...	理中。。。
27	MSG_PIN_TR- Y_EX	PIN TRY LIMIT EX	PIN TRY DEPASSE	TRY PIN SUPERADA	密次多
28	MSG_ISSUER- _AUTH_FAIL	ISSUER AUTH FAIL	EMETTEUR FAIL	EMISOR FALLA	与卡机构
29	MSG_CONTIN- UE_PROCESS	CONTINUE PROCESS	CONTINUER LA	CONTINUAR PROCES	理
30	MSG_GET_PI- N_ERROR	GET PIN ERROR	GET PIN ERROR	OBTENER PIN ERR	密
31	MSG_GET_PI- N_FAIL	GET PIN FAIL	GET PIN FAIL	OBTENER PIN FALL	取密
32	MSG_NOKEY- _GET_PIN	NO KEY GET PIN	NO KEY GET PIN	NO CLAVE GET PIN	法入密
33	MSG_CANCE- LLED	CANCELLED	ANNULE	CANCELADO	取消
34	MSG_LAST_P- IN_TRY	LAST PIN TRY	-	-	最后一次入密
35	MSG_WELCO- ME	WELCOME	BIENVENUE	BIENVENIDOS	迎使用
36	MSG_AMOUN- T_OTHER	AMOUNT OTHER	MONTANT AUTRES	IMPORTE OTRAS	返
37	MSG_ENTER- _AMOUNT_O- THER	ENTER AMOUNT OTHER	ENTRER MONTANT AUTRES	ENTRAR IMPORTE OTRAS	入返
38	MSG_CAPK_- HASH_VALUE- _FAIL	CAPK HASH VALUE FAIL	CAPK HASH VALEUR FAIL	CAPK HASH VALOR FAIL	公哈希值
64	MSG_TRY_M- SR_AGAIN	TRY MSR AGAIN	-	-	再次使用磁卡
65	MSG_LAST_- MSR_TRY	LAST MSR TRY	-	-	最后一次使用 磁卡
66	MSG_TRY_IC- C_AGAIN	TRY ICC AGAIN	INSERER VOTRE CARTE	INTENTE ICC DE NUEVO	再次使用芯片 卡

67	MSG_REMOV- E_CARD	REMOVE CARD	RETIRER VOTRE CARTE	QUITE TARJETA	移卡
----	----------------------	----------------	---------------------------	------------------	----

## Chapter 7

# Error Code Reference

```
0: "no error, beginning task";
1: "no response from reader";
2: "invalid response data";
3: "time out for task or CMD";
4: "wrong parameter";
5: "SDK is doing MSR or ICC task";
6: "SDK is doing PINPad task";
7: "SDK is doing CTLS task";
8: "SDK is doing EMV task";
9: "SDK is doing Other task";
10: "err response or data";
11: "no reader attached";
12: "mono audio is enabled";
13: "did connection";
14: "audio volume is too low";
15: "task or CMD be canceled";
16: "UF wrong string format";
17: "UF file not found";
18: "UF wrong file format";
19: "Attempt to contact online host failed";
20: "Attempt to perform RKI failed";
22: "Buffer size is not enough";
0x300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
0x400: "Related Key was not loaded.";
0x500: "Key Same.";
0x501: "Key is all zero";
0x502: "TR-31 format error";
0x702: "PAN is Error Key.";
0x705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
0X0C01: "Incorrect Frame Tag";
0X0C02: "Incorrect Frame Type";
0X0C03: "Unknown Frame Type";
0X0C04: "Unknown Command";
0X0C05: "Unknown Sub-Command";
0X0C06: "CRC Error";
0X0C07: "Failed";
0X0C08: "Timeout";
0X0C0A: "Incorrect Parameter";
0X0C0B: "Command Not Supported";
0X0C0C: "Sub-Command Not Supported";
0X0C0D: "Parameter Not Supported / Status Abort Command";
0X0C0F: "Sub-Command Not Allowed";
0X0D01: "Incorrect Header Tag";
0X0D02: "Unknown Command";
0X0D03: "Unknown Sub-Command";
0X0D04: "CRC Error in Frame";
0X0D05: "Incorrect Parameter";
0X0D06: "Parameter Not Supported";
0X0D07: "Mal-formatted Data";
0X0D08: "Timeout";
0X0D0A: "Failed / NACK";
0X0D0B: "Command not Allowed";
0X0D0C: "Sub-Command not Allowed";
0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
0X0D0E: "User Interface Event";
0X0D11: "Communication type not supported, VT-1, burst, etc.";
0X0D12: "Secure interface is not functional or is in an intermediate state.";
0X0D13: "Data field is not mod 8";
0X0D14: "Pad 0x80 not found where expected";
0X0D15: "Specified key type is invalid";
0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
0X0D17: "Hash code problem";
0X0D18: "Could not store the key into the SAM(InstallKey)";
```

```

0X0D19: "Frame is too large";
0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
0X0D1C: "Problem encoding APDU";
0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned
from the SAM(Key Mgr)";
0X0D22: "Improper bit map(ILM)";
0X0D23: "Request Online Authorization";
0X0D24: "ViVOCard3 raw data read successful";
0X0D25: "Message index not available(ILM) ViVOComm activate transaction card type(ViVOComm)";
0X0D26: "Version Information Mismatch(ILM)";
0X0D27: "Not sending commands in correct index message index(ILM)";
0X0D28: "Time out or next expected message not received(ILM)";
0X0D29: "ILM languages not available for viewing(ILM)";
0X0D2A: "Other language not supported(ILM)";
0X0D41: "Unknown Error from SAM";
0X0D42: "Invalid data detected by SAM";
0X0D43: "Incomplete data detected by SAM";
0X0D44: "Reserved";
0X0D45: "Invalid key hash algorithm";
0X0D46: "Invalid key encryption algorithm";
0X0D47: "Invalid modulus length";
0X0D48: "Invalid exponent";
0X0D49: "Key already exists";
0X0D4A: "No space for new RID";
0X0D4B: "Key not found";
0X0D4C: "Crypto not responding";
0X0D4D: "Crypto communication error";
0X0D4E: "Module-specific error for Key Manager";
0X0D4F: "All key slots are full (maximum number of keys has been installed)";
0X0D50: "Auto-Switch OK";
0X0D51: "Auto-Switch failed";
0X0D90: "Account DUKPT Key not exist";
0X0D91: "Account DUKPT Key KSN exhausted";
0x0D00: "This Key had been loaded.";
0x0E00: "Base Time was loaded.";
0x0F00: "Encryption Or Decryption Failed.";
0x1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
0x1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
0x1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
0x30FF: "Security Chip is not connect";
0x3000: "Security Chip is deactivation & Device is In Removal Legally State.";
0x3101: "Security Chip is activation & Device is In Removal Legally State.";
0x5500: "No Admin DUKPT Key.";
0x5501: "Admin DUKPT Key STOP.";
0x5502: "Admin DUKPT Key KSN is Error.";
0x5503: "Get Authentication Code Failed.";
0x5504: "Validate Authentication Code Error.";
0x5505: "Encrypt or Decrypt data failed.";
0x5506: "Not Support the New Key Type.";
0x5507: "New Key Index is Error.";
0x5508: "Step Error.";
0x5509: "KSN Error";
0x550A: "MAC Error.";
0x550B: "Key Usage Error.";
0x550C: "Mode Of Use Error.";
0x550F: "Other Error.";
0x6000: "Save or Config Failed / Or Read Config Error.";
0x6200: "No Serial Number.";
0x6900: "Invalid Command - Protocol is right, but task ID is invalid.";
0x6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
0x6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
0x6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
0x6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the
requirement.";
0x7200: "Device is suspend (MKSK suspend or press password suspend).";
0x7300: "PIN DUKPT is STOP (21 bit 1).";
0x7400: "Device is Busy.";
0xE100: "Can not enter sleep mode";
0xE200: "File has existed";
0xE300: "File has not existed";
0xE313: "IO line low -- Card error after session start";
0xE400: "Open File Error";
0xE500: "SmartCard Error";
0xE600: "Get MSR Card data is error";
0xE700: "Command time out";
0xE800: "File read or write is error";
0xE900: "Active 1850 error!";
0xEA00: "Load bootloader error";
0xEF00: "Protocol Error- STX or ETX or check error.";
0xEB00: "Picture is not exist";
0x2C02: "No Microprocessor ICC seated";
0x2C06: "no card seated to request ATR";
0x2D01: "Card Not Supported,";
0x2D03: "Card Not Supported, wants CRC";
0x690D: "Command not supported on reader without ICC support";

```

---

```

0x8100: "ICC error time out on power-up";
0x8200: "invalid TS character received - Wrong operation step";
0x8300: "Decode MSR Error";
0x8400: "TriMagII no Response";
0x8500: "No Swipe MSR Card";
0x8510: "No Financial Card";
0x8600: "Unsupported F, D, or combination of F and D";
0x8700: "protocol not supported EMV TD1 out of range";
0x8800: "power not at proper level";
0x8900: "ATR length too long";
0x8B01: "EMV invalid TA1 byte value";
0x8B02: "EMV TB1 required";
0x8B03: "EMV Unsupported TB1 only 00 allowed";
0x8B04: "EMV Card Error, invalid BWI or CWI";
0x8B06: "EMV TB2 not allowed in ATR";
0x8B07: "EMV TC2 out of range";
0x8B08: "EMV TC2 out of range";
0x8B09: "per EMV96 TA3 must be > 0xF";
0x8B10: "ICC error on power-up";
0x8B11: "EMV T=1 then TB3 required";
0x8B12: "Card Error, invalid BWI or CWI";
0x8B13: "Card Error, invalid BWI or CWI";
0x8B17: "EMV TC1/TB3 conflict-";
0x8B20: "EMV TD2 out of range must be T=1";
0x8C00: "TCK error";
0xA304: "connector has no voltage setting";
0xA305: "ICC error on power-up invalid (SBLK(IFS)D) exchange";
0xE301: "ICC error after session start";
0xFF00: "Request to go online";
0xFF01: "EMV: Accept the offline transaction";
0xFF02: "EMV: Decline the offline transaction";
0xFF03: "EMV: Accept the online transaction";
0xFF04: "EMV: Decline the online transaction";
0xFF05: "EMV: Application may fallback to magstripe technology";
0xFF06: "EMV: ICC detected that the conditions of use are not satisfied";
0xFF07: "EMV: ICC didn't accept transaction";
0xFF08: "EMV: Transaction was cancelled";
0xFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
0xFF0A: "EMV: Transaction is terminated";
0xFF0B: "EMV: Other EMV Error";
0xFFFF: "NO RESPONSE";
0xF002: "ICC communication timeout";
0xF003: "ICC communication Error";
0xF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
0xF200: "AID List / Application Data is not exist";
0xF201: "Terminal Data is not exist";
0xF202: "TLV format is error";
0xF203: "AID List is full";
0xF204: "Any CA Key is not exist";
0xF205: "CA Key RID is not exist";
0xF206: "CA Key Index it not exist";
0xF207: "CA Key is full";
0xF208: "CA Key Hash Value is Error";
0xF209: "Transaction format error";
0xF20A: "The command will not be processing";
0xF20B: "CRL is not exist";
0xF20C: "CRL number exceed max number";
0xF20D: "Amount, Other Amount, Transaction Type are missing";
0xF20E: "The Identification of algorithm is mistake";
0xF20F: "No Financial Card";
0xF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
0x1001: "INVALID ARG";
0x1002: "FILE_OPEN_FAILED";
0x1003: "FILE_OPERATION_FAILED";
0x2001: "MEMORY_NOT_ENOUGH";
0x3002: "SMARTCARD_FAIL";
0x3003: "SMARTCARD_INIT_FAILED";
0x3004: "FALLBACK_SITUATION";
0x3005: "SMARTCARD_ABSENT";
0x3006: "SMARTCARD_TIMEOUT";
0x5001: "EMV_PARSING_TAGS_FAILED";
0x5002: "EMV_DUPLICATE_CARD_DATA_ELEMENT";
0x5003: "EMV_DATA_FORMAT_INCORRECT";
0x5004: "EMV_NO_TERM_APP";
0x5005: "EMV_NO_MATCHING_APP";
0x5006: "EMV_MISSING_MANDATORY_OBJECT";
0x5007: "EMV_APP_SELECTION_RETRY";
0x5008: "EMV_GET_AMOUNT_ERROR";
0x5009: "EMV_CARD_REJECTED";
0x5010: "EMV_AIP_NOT_RECEIVED";
0x5011: "EMV_AFL_NOT_RECEIVED";
0x5012: "EMV_AFL_LEN_OUT_OF_RANGE";
0x5013: "EMV_SFI_OUT_OF_RANGE";
0x5014: "EMV_AFL_INCORRECT";
0x5015: "EMV_EXP_DATE_INCORRECT";
0x5016: "EMV_EFF_DATE_INCORRECT";
0x5017: "EMV_ISS_COD_TBL_OUT_OF_RANGE";

```

---

```

0x5018: "EMV_CRYPTOGAM_TYPE_INCORRECT";
0x5019: "EMV_PSE_NOT_SUPPORTED_BY_CARD";
0x5020: "EMV_USER_SELECTED_LANGUAGE";
0x5021: "EMV_SERVICE_NOT_ALLOWED";
0x5022: "EMV_NO_TAG_FOUND";
0x5023: "EMV_CARD_BLOCKED";
0x5024: "EMV_LEN_INCORRECT";
0x5025: "CARD_COM_ERROR";
0x5026: "EMV_TSC_NOT_INCREASED";
0x5027: "EMV_HASH_INCORRECT";
0x5028: "EMV_NO_ARC";
0x5029: "EMV_INVALID_ARC";
0x5030: "EMV_NO_ONLINE_COMM";
0x5031: "TRAN_TYPE_INCORRECT";
0x5032: "EMV_APP_NO_SUPPORT";
0x5033: "EMV_APP_NOT_SELECT";
0x5034: "EMV_LANG_NOT_SELECT";
0x5035: "EMV_NO_TERM_DATA";
0x6001: "CVM_TYPE_UNKNOWN";
0x6002: "CVM_AIP_NOT_SUPPORTED";
0x6003: "CVM_TAG_8E_MISSING";
0x6004: "CVM_TAG_8E_FORMAT_ERROR";
0x6005: "CVM_CODE_IS_NOT_SUPPORTED";
0x6006: "CVM_COND_CODE_IS_NOT_SUPPORTED";
0x6007: "NO_MORE_CVM";
0x6008: "PIN_BYPASSED_BEFORE";
0x7001: "PK_BUFFER_SIZE_TOO_BIG";
0x7002: "PK_FILE_WRITE_ERROR";
0x7003: "PK_HASH_ERROR";
0x8001: "NO_CARD HOLDER CONFIRMATION";
0x8002: "GET_ONLINE_PIN";
0xD000: "Data not exist";
0xD001: "Data access error";
0xD100: "RID not exist";
0xD101: "RID existed";
0xD102: "Index not exist";
0xD200: "Maximum exceeded";
0xD201: "Hash error";
0xD205: "System Busy";
0x0E01: "Unable to go online";
0x0E02: "Technical Issue";
0x0E03: "Declined";
0x0E04: "Issuer Referral transaction";
0x0F01: "Decline the online transaction";
0x0F02: "Request to go online";
0x0F03: "Transaction is terminated";
0x0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
0x0F07: "ICC didn't accept transaction";
0x0F0A: "Application may fallback to magstripe technology";
0x0F0C: "Transaction was cancelled";
0x0F0D: "Timeout";
0x0F0F: "Other EMV Error";
0x0F10: "Accept the offline transaction";
0x0F11: "Decline the offline transaction";
0x0F21: "ICC detected the conditions of use are not satisfied";
0x0F22: "No app were found on card matching terminal configuration";
0x0F23: "Terminal file does not exist";
0x0F24: "CAPK file does not exist";
0x0F25: "CRL Entry does not exist";
0x0FFE: "code when blocking is disabled";
0x0FFF: "code when command is not applicable on the selected device";
0xF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
0xBBE0: "CM100 Success";
0xBBE1: "CM100 Parameter Error";
0xBBE2: "CM100 Low Output Buffer";
0xBBE3: "CM100 Card Not Found";
0xBBE4: "CM100 Collision Card Exists";
0xBBE5: "CM100 Too Many Cards Exist";
0xBBE6: "CM100 Saved Data Does Not Exist";
0xBBE8: "CM100 No Data Available";
0xBBE9: "CM100 Invalid CID Returned";
0xBBEA: "CM100 Invalid Card Exists";
0xBBEC: "CM100 Command Unsupported";
0xBBED: "CM100 Error In Command Process";
0xBBEE: "CM100 Invalid Command";

0X9031: "Unknown command";
0X9032: "Wrong parameter (such as the length of the command is incorrect)";

0X9038: "Wait (the command couldnt be finished in BWT)";
0X9039: "Busy (a previously command has not been finished)";
0X903A: "Number of retries over limit";

0X9040: "Invalid Manufacturing system data";
0X9041: "Not authenticated";
0X9042: "Invalid Master DUKPT Key";
0X9043: "Invalid MAC Key";

```

---

```
0X9044: "Reserved for future use";
0X9045: "Reserved for future use";
0X9046: "Invalid DATA DUKPT Key";
0X9047: "Invalid PIN Pairing DUKPT Key";
0X9048: "Invalid DATA Pairing DUKPT Key";
0X9049: "No nonce generated";
0X9949: "No GUID available. Perform getVersion first.";
0X9950: "MAC Calculation unsuccessful. Check BDK value.";

0X904A: "Not ready";
0X904B: "Not MAC data";

0X9050: "Invalid Certificate";
0X9051: "Duplicate key detected";
0X9052: "AT checks failed";
0X9053: "TR34 checks failed";
0X9054: "TR31 checks failed";
0X9055: "MAC checks failed";
0X9056: "Firmware download failed";

0X9060: "Log is full";
0X9061: "Removal sensor unengaged";
0X9062: "Any hardware problems";

0X9070: "ICC communication timeout";
0X9071: "ICC data error (such check sum error)";
0X9072: "Smart Card not powered up";
```

## Chapter 8

# Enumeration Reference

### Common

```

enum DEVICE_TYPE{
    DEVICE_TYPE_UNKNOWN=0,
    IDT_DEVICE_AUGUSTA,
    AUGUSTA_KB,
    IDT_DEVICE_SPECTRUM_PRO,
    IDT_DEVICE_MINISMART_II,
    IDT_DEVICE_UNIPAY,
    IDT_DEVICE_UNIPAY_I_V,
    IDT_DEVICE_UNIPAY_III,
    IDT_DEVICE_L100,
};

enum DEVICE_INTERFACE{
    DEVICE_INTERFACE_UNKNOWN=0,
    PROTOCOL_USBHID,
    PROTOCOL_USBBKB,
    PROTOCOL_SEIRAL,
    PROTOCOL_BLUETOOTH,
};

enum DEVICE_PROTOCOL{
    DEVICE_PROTOCOL_UNKNOWN=0,
    PROTOCOL_NGA,
    PROTOCOL_IDG,
    PROTOCOL_IIP,
};

enum DEVICE_STATUS{
    DEVICE_DISCONNECT=0,
    DEVICE_CONNECTED,
};

enum EMV_LCD_DISPLAY_MODE
{
    EMV_LCD_DISPLAY_MODE_CANCEL = 0,
    EMV_LCD_DISPLAY_MODE_MENU = 1,
    EMV_LCD_DISPLAY_MODE_PROMPT = 2,
    EMV_LCD_DISPLAY_MODE_MESSAGE = 3,
    EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT = 8,
    EMV_LCD_DISPLAY_MODE_CLEAR_SCREEN = 16,
};

enum MSR_callBack_type
{
    MSR_callBack_type_ERR=1,
    MSR_callBack_type_RETURN_CODE,
    MSR_callBack_type_TIMEOUT
};

enum PIN_callBack_type
{
    PIN_callBack_type_ERR=1,
    PIN_callBack_type_RETURN_CODE,
    PIN_callBack_type_TIMEOUT
};

```



```

enum CAPTURE_ENCODE_TYPE
{
    CAPTURE_ENCODE_TYPE_ISOABA,
    CAPTURE_ENCODE_TYPE_AAMVA,
    CAPTURE_ENCODE_TYPE_Other,
    CAPTURE_ENCODE_TYPE_Raw,
    CAPTURE_ENCODE_TYPE_Jis_II,
    CAPTURE_ENCODE_TYPE_Jis_I,
    CAPTURE_ENCODE_TYPE_Jis_II_Security,
    CAPTURE_ENCODE_TYPE_Contactless_Visa_Kernel1,
    CAPTURE_ENCODE_TYPE_Contactless_MasterCard,
    CAPTURE_ENCODE_TYPE_Contactless_Visa_Kernel3,
    CAPTURE_ENCODE_TYPE_Contactless_AmericanExpress,
    CAPTURE_ENCODE_TYPE_Contactless_JCB,
    CAPTURE_ENCODE_TYPE_Contactless_Discover,
    CAPTURE_ENCODE_TYPE_Contactless_UnionPay,
    CAPTURE_ENCODE_TYPE_Contactless_Others,
    CAPTURE_ENCODE_TYPE_Manual_Entry_Enhanced_Mode,
    CAPTURE_ENCODE_TYPE_JisI_II
};

enum CAPTURE_ENCRYPT_TYPE
{
    CAPTURE_ENCRYPT_TYPE_TDES, CAPTURE_ENCRYPT_TYPE_AES, CAPTURE_ENCRYPT_TYPE_NONE
};

enum EMV_PIN_MODE
{
    EMV_PIN_MODE_CANCEL=0, EMV_PIN_MODE_ONLINE_DUKPT=1, EMV_PIN_MODE_ONLINE_MKSK=2, EMV_PIN_MODE_OFFLINE=3,
};

enum EMV_CALLBACK_TYPE
{
    EMV_CALLBACK_TYPE_LCD=1, EMV_CALLBACK_TYPE_PINPAD=2, EMV_CALLBACK_MSR=3,
    EMV_callBack_type_ERR,
    EMV_callBack_type_RETURN_CODE,
};

enum EMV_ENCRYPTION_MODE
{
    EMV_ENCRYPTION_MODE_TDES = 0, EMV_ENCRYPTION_MODE_AES = 1,
};

enum EMV_RESULT_CODE
{
    EMV_RESULT_CODE_APPROVED_OFFLINE = 0,
    EMV_RESULT_CODE_DECLINED_OFFLINE = 1,
    EMV_RESULT_CODE_APPROVED = 2,
    EMV_RESULT_CODE_DECLINED = 3,
    EMV_RESULT_CODE_GO_ONLINE = 4,
    EMV_RESULT_CODE_CALL_YOUR_BANK = 5,
    EMV_RESULT_CODE_NOT_ACCEPTED = 6,
    EMV_RESULT_CODE_FALLBACK_TO_MSR = 7,
    EMV_RESULT_CODE_TIMEOUT = 8,
    EMV_RESULT_CODE_GO_ONLINE_CTLS = 9,
    EMV_RESULT_CODE_AUTHENTICATE_TRANSACTION = 0x0010,
    EMV_RESULT_CODE_TRANSACTION_CANCELED = 0x0012,
    EMV_RESULT_CODE_SWIPE_NON_ICC = 0x11,
    EMV_RESULT_CODE_CTLS_TWO_CARDS = 0x7A,
    EMV_RESULT_CODE_CTLS_TERMINATE = 0x7E,
    EMV_RESULT_CODE_CTLS_TERMINATE_TRY_ANOTHER = 0x7D,
    EMV_RESULT_CODE_UNABLE_TO_REACH_HOST = 0xFF,
    EMV_RESULT_CODE_FILE_ARG_INVALID = 0x1001,
    EMV_RESULT_CODE_FILE_OPEN_FAILED = 0x1002,
    EMV_RESULT_CODE_FILE_OPERATION_FAILED = 0x1003,
    EMV_RESULT_CODE_MEMORY_NOT_ENOUGH = 0x2001,
    EMV_RESULT_CODE_SMARTCARD_OK = 0x3001,
    EMV_RESULT_CODE_SMARTCARD_FAIL = 0x3002,
    EMV_RESULT_CODE_SMARTCARD_INIT_FAILED = 0x3003,
    EMV_RESULT_CODE_FALLBACK_SITUATION = 0x3004,
    EMV_RESULT_CODE_SMARTCARD_ABSENT = 0x3005,
    EMV_RESULT_CODE_SMARTCARD_TIMEOUT = 0x3006,
    EMV_RESULT_CODE_MSR_CARD_ERROR = 0x3007,
    EMV_RESULT_CODE_PARSING_TAGS_FAILED = 0x5001,
    EMV_RESULT_CODE_CARD_DATA_ELEMENT_DUPLICATE = 0x5002,
    EMV_RESULT_CODE_DATA_FORMAT_INCORRECT = 0x5003,
    EMV_RESULT_CODE_APP_NO_TERM = 0x5004,
    EMV_RESULT_CODE_APP_NO_MATCHING = 0x5005,
    EMV_RESULT_CODE_MANDATORY_OBJECT_MISSING = 0x5006,
    EMV_RESULT_CODE_APP_SELECTION_RETRY = 0x5007,
    EMV_RESULT_CODE_AMOUNT_ERROR_GET = 0x5008,
    EMV_RESULT_CODE_CARD_REJECTED = 0x5009,
    EMV_RESULT_CODE_AIP_NOT_RECEIVED = 0x5010,
};

```

```

    EMV_RESULT_CODE_AFL_NOT_RECEIVED = 0x5011,
    EMV_RESULT_CODE_AFL_LEN_OUT_OF_RANGE = 0x5012,
    EMV_RESULT_CODE_SFI_OUT_OF_RANGE = 0x5013,
    EMV_RESULT_CODE_AFL_INCORRECT = 0x5014,
    EMV_RESULT_CODE_EXP_DATE_INCORRECT = 0x5015,
    EMV_RESULT_CODE_EFF_DATE_INCORRECT = 0x5016,
    EMV_RESULT_CODE_ISS_COD_TBL_OUT_OF_RANGE = 0x5017,
    EMV_RESULT_CODE_CRYPTOGAM_TYPE_INCORRECT = 0x5018,
    EMV_RESULT_CODE_PSE_BY_CARD_NOT_SUPPORTED = 0x5019,
    EMV_RESULT_CODE_USER_LANGUAGE_SELECTED = 0x5020,
    EMV_RESULT_CODE_SERVICE_NOT_ALLOWED = 0x5021,
    EMV_RESULT_CODE_NO_TAG_FOUND = 0x5022,
    EMV_RESULT_CODE_CARD_BLOCKED = 0x5023,
    EMV_RESULT_CODE_LEN_INCORRECT = 0x5024,
    EMV_RESULT_CODE_CARD_COM_ERROR = 0x5025,
    EMV_RESULT_CODE_TSC_NOT_INCREASED = 0x5026,
    EMV_RESULT_CODE_HASH_INCORRECT = 0x5027,
    EMV_RESULT_CODE_ARC_NOT_PRESENCE = 0x5028,
    EMV_RESULT_CODE_ARC_INVALID = 0x5029,
    EMV_RESULT_CODE_COMM_NO_ONLINE = 0x5030,
    EMV_RESULT_CODE_TRAN_TYPE_INCORRECT = 0x5031,
    EMV_RESULT_CODE_APP_NO_SUPPORT = 0x5032,
    EMV_RESULT_CODE_APP_NOT_SELECT = 0x5033,
    EMV_RESULT_CODE_LANG_NOT_SELECT = 0x5034,
    EMV_RESULT_CODE_TERM_DATA_NOT_PRESENCE = 0x5035,
    EMV_RESULT_CODE_CVM_TYPE_UNKNOWN = 0x6001,
    EMV_RESULT_CODE_CVM_AIP_NOT_SUPPORTED = 0x6002,
    EMV_RESULT_CODE_CVM_TAG_8E_MISSING = 0x6003,
    EMV_RESULT_CODE_CVM_TAG_8E_FORMAT_ERROR = 0x6004,
    EMV_RESULT_CODE_CVM_CODE_IS_NOT_SUPPORTED = 0x6005,
    EMV_RESULT_CODE_CVM_COND_CODE_IS_NOT_SUPPORTED = 0x6006,
    EMV_RESULT_CODE_CVM_NO_MORE = 0x6007,
    EMV_RESULT_CODE_PIN_BYPASSED_BEFORE = 0x6008,
    EMV_RESULT_CODE_UNKONWN = 0xffff,
};

enum EVENT_TRANSACTION_DATA_Types
{
    EVENT_TRANSACTION_DATA_UNKNOWN, EVENT_TRANSACTION_DATA_CARD_DATA, EVENT_TRANSACTION_DATA_EMV_DATA,
    EVENT_TRANSACTION_DATA_MSR_CANCEL_KEY, EVENT_TRANSACTION_DATA_MSR_BACKSPACE_KEY,
    EVENT_TRANSACTION_DATA_MSR_ENTER_KEY, EVENT_TRANSACTION_DATA_MSR_DATA_ERROR, EVENT_TRANSACTION_PIN_DATA
};

enum EVENT_NOTIFICATION_Types
{
    EVENT_NOTIFICATION_UNKNOWN,
    EVENT_NOTIFICATION_ICC_Card_Not_Seated,
    EVENT_NOTIFICATION_ICC_Card_Seated,
    EVENT_NOTIFICATION_MSR_Swipe_Card,
};

enum CTLS_APPLICATION
{
    CTLS_APPLICATION_NONE = 0,
    CTLS_APPLICATION_MASTERCARD = 1,
    CTLS_APPLICATION_VISA = 2,
    CTLS_APPLICATION_AMEX = 3,
    CTLS_APPLICATION_DISCOVER = 4,
    CTLS_APPLICATION_SPEEDPASS = 5,
    CTLS_APPLICATION_GIFT_CARD = 6,
    CTLS_APPLICATION_DINERS_CLUB = 7,
    CTLS_APPLICATION_EN_ROUTE = 8,
    CTLS_APPLICATION_JCB = 9,
    CTLS_APPLICATION_VIVO_DIAGNOSTIC = 10,
    CTLS_APPLICATION_HID = 11,
    CTLS_APPLICATION_MSR_SWIPE = 12,
    CTLS_APPLICATION_RESERVED = 13,
    CTLS_APPLICATION_DES_FIRE_TRACK_DATA = 14,
    CTLS_APPLICATION_DES_FIRE_RAW_DATA = 15,
    CTLS_APPLICATION_RBS = 17,
    CTLS_APPLICATION_VIVO_COMM = 20,
};

enum DeviceState
{
    TransactionData,
    Connected,
    ConnectionFailed,
    DataReceived,
    DataSent,
    Disconnected,
    SwipeCanceled,
};

```

```

        ToConnect,
        ToSwipe,
        ToTap,
        CommandTimeout,
        SwipeTimeout,
        DeviceTimeout,
        EMVCallback,
        TransactionCancelled,
        TransactionFailed,
        MSRDecodeError,
        DefaultDeviceTypeChange,
        Notification,
        PINpadKeypress,
        PINCancelled,
        PINTimeout
    };

enum RETURN_CODE
{
    RETURN_CODE_DO_SUCCESS = 0,
    RETURN_CODE_ERR_DISCONNECT,
    RETURN_CODE_ERR_CMD_RESPONSE,
    RETURN_CODE_ERR_TIMEDOUT,
    RETURN_CODE_ERR_INVALID_PARAMETER,
    RETURN_CODE_SDK_BUSY_MSR,
    RETURN_CODE_SDK_BUSY_PINPAD,
    RETURN_CODE_SDK_BUSY_CTLs,
    RETURN_CODE_SDK_BUSY_EMV,
    RETURN_CODE_ERR_OTHER,
    RETURN_CODE_FAILED,
    RETURN_CODE_NOT_ATTACHED,
    RETURN_CODE_MONO_AUDIO,
    RETURN_CODE_CONNECTED,
    RETURN_CODE_LOW_VOLUME,
    RETURN_CODE_CANCELED,
    RETURN_CODE_INVALID_STR,
    RETURN_CODE_NO_FILE,
    RETURN_CODE_INVALID_FILE,
    RETURN_CODE_HOST_UNREACHABLE,
    RETURN_CODE_RKI_FAILURE,
    RETURN_CODE_MISSING_DLL,
    RETURN_CODE_ERR_BUF_NOT_ENOUGH,
    RETURN_CODE_P1_INCORRECT_FRAME_TAG = 0X0C01,
    RETURN_CODE_P1_INCORRECT_FRAME_TYPE = 0X0C02,
    RETURN_CODE_P1_UNKNOWN_FRAME_TYPE = 0X0C03,
    RETURN_CODE_P1_UNKNOWN_COMMAND = 0X0C04,
    RETURN_CODE_P1_UNKNOWN_SUB_COMMAND = 0X0C05,
    RETURN_CODE_P1_CRC_ERROR = 0X0C06,
    RETURN_CODE_P1_FAILED = 0X0C07,
    RETURN_CODE_P1_TIMEOUT = 0X0C08,
    RETURN_CODE_P1_INCORRECT_PARAMETER = 0X0C0A,
    RETURN_CODE_P1_COMMAND_NOT_SUPPORTED = 0X0C0B,
    RETURN_CODE_P1_SUB_COMMAND_NOT_SUPPORTED = 0X0C0C,
    RETURN_CODE_P1_STATUS_ABORT_COMMAND = 0X0C0D,
    RETURN_CODE_P1_COMMAND_NOT_ALLOWED = 0X0C0F,
    RETURN_CODE_P2_ = 0X0D01,
    RETURN_CODE_P2_UNKNOWN_COMMAND = 0X0D02,
    RETURN_CODE_P2_UNKNOWN_SUB_COMMAND = 0X0D03,
    RETURN_CODE_P2_CRC_ERROR = 0X0D04,
    RETURN_CODE_P2_INCORRECT_PARAMETER = 0X0D05,
    RETURN_CODE_P2_PARAMETER_NOT_SUPPORTED = 0X0D06,
    RETURN_CODE_P2_MAL_FORMATTED_DATA = 0X0D07,
    RETURN_CODE_P2_TIMEOUT = 0X0D08,
    RETURN_CODE_P2_FAILED = 0X0D0A,
    RETURN_CODE_P2_COMMAND_NOT_ALLOWED = 0X0D0B,
    RETURN_CODE_P2_SUB_COMMAND_NOT_ALLOWED = 0X0D0C,
    RETURN_CODE_P2_BUFFER_OVERFLOW = 0X0D0D,
    RETURN_CODE_P2_USER_INTERFACE_EVENT = 0X0D0E,
    RETURN_CODE_P2_COMM_TYPE_NOT_SUPPORTED = 0X0D11,
    RETURN_CODE_P2_SECURE_INTERFACE_NOT_FUNCTIONAL = 0X0D12,
    RETURN_CODE_P2_DATA_FIELD_NOT_MOD_8 = 0X0D13,
    RETURN_CODE_P2_PADDING_UNEXPECTED = 0X0D14,
    RETURN_CODE_P2_KEY_TYPE_INVALID = 0X0D15,
    RETURN_CODE_P2_COULD_NOT_RETRIEVE_KEY = 0X0D16,
    RETURN_CODE_P2_HASH_CODE_ERROR = 0X0D17,
    RETURN_CODE_P2_COUNDTOT_STORE_KEY = 0X0D18,
    RETURN_CODE_P2_FRAME_TOO_LARGE = 0X0D19,
    RETURN_CODE_P2_RESEND_INITSECURECOMM_COMMAND = 0X0D1A,
    RETURN_CODE_P2_EEPROM_NOT_INITIALIZED = 0X0D1B,
    RETURN_CODE_P2_APDU_ENCODING_ERROR = 0X0D1C,
    RETURN_CODE_P2_SAM_COMM_ERROR = 0X0D20,
    RETURN_CODE_P2_SEQUENCE_COUNTER_ERROR = 0X0D21,
    RETURN_CODE_P2_IMPROPER_BITMAP = 0X0D22,
    RETURN_CODE_P2_REQUEST_ONLINE_AUTHORIZATION = 0X0D23,
    RETURN_CODE_P2_RAW_DATA_READ_SUCCESSFUL = 0X0D24,
    RETURN_CODE_P2_MESSAGE_INDEX_NOT_AVAILABLE = 0X0D25,

```

```

RETURN_CODE_P2_VERSION_INFORMATION_MISMATCH = 0X0D26,
RETURN_CODE_P2_INCORRECT_MESSAGE_INDEX = 0X0D27,
RETURN_CODE_P2_NEXT_MESSAGE_NOT_RECEIVED = 0X0D28,
RETURN_CODE_P2_ILM_LANGUAGE_NOT_AVAILABLE = 0X0D29,
RETURN_CODE_P2_OTHER_LANGUAGE_NOT_SUPPORTED = 0X0D2A,
RETURN_CODE_UNKNOWN_ERROR_FROM_SAM = 0X0D41,
RETURN_CODE_INVALID_DATA_DETECTED_BY_SAM = 0X0D42,
RETURN_CODE_INCOMPLETE_DATA_DETECTED_BY_SAM = 0X0D43,
RETURN_CODE_RESERVED = 0X0D44,
RETURN_CODE_INVALID_KEY_HASH_ALGORITHM = 0X0D45,
RETURN_CODE_INVALID_KEY_ENCRYPTION_ALGORITHM = 0X0D46,
RETURN_CODE_INVALID_MODULUS_LENGTH = 0X0D47,
RETURN_CODE_INVALID_EXPONENT = 0X0D48,
RETURN_CODE_KEY_ALREADY_EXISTS = 0X0D49,
RETURN_CODE_NO_SPACE_FOR_NEW_RID = 0X0D4A,
RETURN_CODE_KEY_NOT_FOUND = 0X0D4B,
RETURN_CODE_CRYPTOTIMEOUT = 0X0D4C,
RETURN_CODE_CRYPTOCOMMUNICATION_ERROR = 0X0D4D,
RETURN_CODE_P2_KEY_MANAGER_ERROR_4E = 0X0D4E,
RETURN_CODE_ALL_KEY_SLOTS_FULL = 0X0D4F,
RETURN_CODE_P2_AUTO_SWITCH_OK = 0X0D50,
RETURN_CODE_P2_AUTO_SWITCH_FAILED = 0X0D51,
RETURN_CODE_P2_DATA_DOES_NOT_EXIST = 0X0D60,
RETURN_CODE_P2_DATA_FULL = 0X0D61,
RETURN_CODE_P2_WRITE_FLASH_ERROR = 0X0D62,
RETURN_CODE_P2_OK_AND_HAVE_NEXT_COMMAND = 0X0D63,
RETURN_CODE_P2_ACCOUNT_DUKPT_KEY_NOT_EXIST = 0X0D90,
RETURN_CODE_P2_ACCOUNT_DUKPT_KEY_KSN_EXHAUSTED = 0X0D91,

RETURN_CODE_UNKNOWN_COMMAND = 0X9031, // Unknown command
RETURN_CODE_WRONG_PARAMETER = 0X9032, // Wrong parameter (such as the length of the command is
    incorrect)

RETURN_CODE_WAIT = 0X9038,
RETURN_CODE_BUSY = 0X9039,
RETURN_CODE_RETRIES_OVER_LIMIT = 0X903A,
RETURN_CODE_TIMEOUT = 0X8100,

RETURN_CODE_INVALID_MAN_SYSTEM_DATA = 0X9040,
RETURN_CODE_NOT_AUTHENTICATED = 0X9041,
RETURN_CODE_INVALID_MASTER_DUKPT_KEY = 0X9042,
RETURN_CODE_INVALID_MAC_KEY = 0X9043,
RETURN_CODE_RESERVED_FOR_FUTURE_USE = 0X9044,
RETURN_CODE_RESERVED_FOR_FUTURE_USE_2 = 0X9045,
RETURN_CODE_INVALID_DATA_DUKPT_KEY = 0X9046,
RETURN_CODE_INVALID_PIN_PAIRING_DUKPT_KEY = 0X9047,
RETURN_CODE_INVALID_DATA_PAIRING_DUKPT_KEY = 0X9048,
RETURN_CODE_NO_NONCE_GENERATED = 0X9049,
RETURN_CODE_NO_GUID_AVAILABLE = 0X9949,
RETURN_CODE_NO_MAC_CALCULATION = 0X9950,

RETURN_CODE_NOT_READY = 0X904A,
RETURN_CODE_MSR_DATA_FAILED = 0X904B,

RETURN_CODE_INVALID_CERTIFICATE = 0X9050,
RETURN_CODE_DUPLICATE_KEY_DETECTED = 0X9051,
RETURN_CODE_AT_CHECKS_FAILED = 0X9052,
RETURN_CODE_TR34_CHECKS_FAILED = 0X9053,
RETURN_CODE_TR31_CHECKS_FAILED = 0X9054,
RETURN_CODE_AMC_CHECKS_FAILED = 0X9055,
RETURN_CODE_FIRMWARE_DOWNLOAD_FAILED = 0X9056,

RETURN_CODE_LOG_IS_FULL = 0X9060,
RETURN_CODE_REMOVAL_SENSOR_UNENGAGED = 0X9061,
RETURN_CODE_HARDWARE_PROBLEM = 0X9062,

RETURN_CODE_ICC_COMMUNICATION_TIMEOUT = 0X9070,
RETURN_CODE_IFF_DATA_ERROR = 0X9071,
RETURN_CODE_SMART_CARD_NOT_POWERED_UP = 0X9072,
RETURN_CODE_NO_AID = 0xF200,
RETURN_CODE_NO_TERMINAL_DATA = 0xF201,
RETURN_CODE_WRONG_TLV_FORMAT = 0xF202,
RETURN_CODE_AID_LIST_FULL = 0xF203,
RETURN_CODE_NO_CA_KEY = 0xF204,
RETURN_CODE_NO_CA_KEY_RID = 0xF205,
RETURN_CODE_NO_CA_KEY_INDEX = 0xF206,
RETURN_CODE_CA_KEY_LIST_FULL = 0xF207,
RETURN_CODE_CA_KEY_HASH_ERROR = 0xF208,
RETURN_CODE_COMMAND_FORMAT_ERROR = 0xF209,
RETURN_CODE_UNEXPECTED_COMMAND = 0xF20A,
RETURN_CODE_NO_CRL = 0xF20B,
RETURN_CODE_CRL_LIST_FULL = 0xF20C,
RETURN_CODE_MISSING_REQUIRED_PARAMETERS = 0xF20D,
RETURN_CODE_CA_INCORRECT_HASH_ALGORITHM = 0xF20E,

RETURN_CODE_EMV_AUTHORIZATION_ACCEPTED = 0x0E00,

```

```
RETURN_CODE_EMV_AUTHORIZATION_UNABLE_TO_GO_ONLINE = 0x0E01,
RETURN_CODE_EMV_AUTHORIZATION_TECHNICAL_ISSUE = 0x0E02,
RETURN_CODE_EMV_AUTHORIZATION_DECLINED = 0x0E03,
RETURN_CODE_EMV_AUTHORIZATION_ISSUER_REFERRAL = 0x0E04,
RETURN_CODE_EMV_APPROVED = 0x0F00,
RETURN_CODE_EMV_DECLINED = 0x0F01,
RETURN_CODE_EMV_GO_ONLINE = 0x0F02,
RETURN_CODE_EMV_FAILED = 0x0F03,
RETURN_CODE_EMV_SYSTEM_ERROR = 0x0F05,
RETURN_CODE_EMV_NOT_ACCEPTED = 0x0F07,
RETURN_CODE_EMV_FALLBACK = 0x0F0A,
RETURN_CODE_EMV_CANCEL = 0x0F0C,
RETURN_CODE_EMV_TIMEOUT = 0x0F0D,
RETURN_CODE_EMV_OTHER_ERROR = 0x0F0F,
RETURN_CODE_EMV_OFFLINE_APPROVED = 0x0F10,
RETURN_CODE_EMV_OFFLINE_DECLINED = 0x0F11,
RETURN_CODE_EMV_NEW_SELECTION = 0x0F21,
RETURN_CODE_EMV_NO_AVAILABLE_APPS = 0x0F22,
RETURN_CODE_EMV_NO_TERMINAL_FILE = 0x0F23,
RETURN_CODE_EMV_NO_CAPK_FILE = 0x0F24,
RETURN_CODE_EMV_NO_CRL_ENTRY = 0x0F25,
RETURN_CODE_BLOCKING_DISABLED = 0x0FFE,
RETURN_CODE_CM100_WITHOUT_ERROR = 0xBBE0,
RETURN_CODE_CM100_PARAMETER = 0xBBE1,
RETURN_CODE_CM100_LOWOUTBUFFER = 0xBBE2,
RETURN_CODE_CM100_CARD_NOT_FOUND = 0xBBE3,
RETURN_CODE_CM100_COLLISION_CARD_EXIST = 0xBBE4,
RETURN_CODE_CM100_TOO_MANY_CARDS_EXIST = 0xBBE5,
RETURN_CODE_CM100_SAVED_DATA_NOT_EXIST = 0xBBE6,
RETURN_CODE_CM100_NO_DATA_AVAILABLE = 0xBBE8,
RETURN_CODE_CM100_INVALID_CID_RETURNED = 0xBBE9,
RETURN_CODE_CM100_INVALID_CARD_EXIST = 0xBBEA,
RETURN_CODE_CM100_COMMAND_UNSUPPORTED = 0xBBEC,
RETURN_CODE_CM100_COMMAND_PROCESS = 0xBBED,
RETURN_CODE_CM100_INVALID_COMMAND = 0xBBEE,
RETURN_CODE_COMMAND_UNAVAILABLE = 0xFFFF
};
```

## Chapter 9

# EMV Callback

During an EMV transaction, ID TECH devices without a built-in LCD display return LCD Display messages as an EMV Callback.

```
void EMV_callback(int device_type, int device_state, unsigned char * data, int dataLen, IDTTransactionData*
cardData, EMV_Callback* emvCallback, int transactionResultCode)
```

There is an **DeviceState** enum in the SDK's EMV Callback that is returned for **device\_state**. When this **DeviceState** is received as **EMVCallback**, it returns an **EMV\_Callback()** class pointer. **EMV\_Callback.callbackType** specifies the type of callback: **EMV\_CALLBACK\_TYPE\_LCD**, **EMV\_CALLBACK\_TYPE\_PINPAD**, or **EMV\_CALLBACK\_TYPE\_MSR**. The device only uses the **EMV\_CALLBACK\_TYPE\_LCD**.

The callback type for LCD display messages is **EMV\_CALLBACK\_TYPE\_LCD**. To determine the type of LCD message, get **EMV\_LCD\_DISPLAY\_MODE** from **IDTechSDK::EMV\_Callback::lcd\_displayMode**:

- 1 - **LCD\_DISPLAY\_MODE\_MENU**: Menu selection, response required with selected menu index #, or 0 to cancel
- 2 - **LCD\_DISPLAY\_MODE\_PROMPT**: Message Prompt, response required 'E' for Enter/Accept, or 'C' for cancel
- 3 - **LCD\_DISPLAY\_MODE\_MESSAGE**: Display Message, no response required
- 8 - **LCD\_DISPLAY\_MODE\_LANGUAGE\_SELECT**: Language selection, response required with selected language index #
- 16 - **LCD\_DISPLAY\_MODE\_CLEAR\_SCREEN**: Request to clear LCD screen of information

The **LCD\_DISPLAY\_MODE\_MESSAGE** and **LCD\_DISPLAY\_MODE\_CLEAR\_SCREEN** do not pause the EMV transaction. These two modes are for displaying a message (no response required) or for clearing the screen.

If the mode is **LCD\_DISPLAY\_MODE\_MENU**, **LCD\_DISPLAY\_MODE\_PROMPT**, or **LCD\_DISPLAY\_MODE\_LANGUAGE\_SELECT**, the provided message must be displayed and the EMV transaction pauses until a response is sent to [emv\\_callbackResponseLCD\(\)](#).

The message to display is **byte[] lcd\_messages**. This contains either a message string or a message ID according to the LCD Foreign Language Mapping Table ([Foreign Language Mapping Table](#)).

## Chapter 10

# EMV Tag Reference

Tag	Description
42	Issuer Identification Number (IIN)
4F	Application Identifier (ADF Name)
50	Application Label
52	Command to perform
56	Track 1 Data
57	Track 2 Equivalent Data
5A	Application Primary Account Number (PAN)
5D	Deleted (see 9D)
5F20	Cardholder Name
5F24	Application Expiration Date
5F28	Issuer Country Code
5F2A	Transaction Currency Code (Default: 08 40)
5F2D	Language Preference
5F30	Service Code
5F34	Application Primary Account Number (PAN) Sequence Number (PSN)
5F36	Transaction Currency Exponent
5F3C	Transaction Reference Currency Code
5F3D	Transaction Reference Currency Exponent
5F50	Issuer URL
5F53	International Bank Account Number (IBAN)
5F54	Bank Identifier Code (BIC)
5F55	Issuer Country Code (alpha2 format)
5F56	Issuer Country Code (alpha3 format)
5F57	Account Type Selection
6F	File Control Information (FCI) Template
61	Application Template
62	File Control Parameters (FCP) Template
70	READ RECORD Response Message Template
71	Issuer Script Template 1
72	Issuer Script Template 2
73	Directory Discretionary Template

77	Response Message Template Format 2
80	Response Message Template Format 1
81	Amount, Authorised (Binary)
82	Application Interchange Profile (AIP)
83	Command Template
84	Dedicated File (DF) Name
86	Issuer Script Command
87	Application Priority Indicator
88	Short File Identifier (SFI)
89	Authorisation Code
8A	Authorization Response Code
8A	Authorisation Response Code (ARC)
8C	Card Risk Management Data Object List 1 (CDOL1)
8D	Card Risk Management Data Object List 2 (CDOL2)
8E	Cardholder Verification Method (CVM) List
8F	Certification Authority Public Key Index (PKI)
90	Issuer Public Key Certificate
91	Issuer Authentication Data
92	Issuer Public Key Remainder
93	Signed Application Data
94	Application File Locator (AFL)
95	Terminal Verification Results (TVR)
97	Transaction Certificate Data Object List (TDOL)
98	Transaction Certificate (TC) Hash Value
99	Transaction Personal Identification Number (PIN) Data
99	Transaction Personal Identification Number (PIN) Data
98	Transaction Certificate (TC) Hash Value
9A	Transaction Date (YYMMDD )
9A	Transaction Date
9B	Transaction Status Information
9B	Transaction Status Information
9C	Transaction Type
9C	Transaction Type
9D	Directory Definition File (DDF) Name
9F01	Acquirer Identifier
9F02	Amount, Authorized (Numeric)
9F03	Amount, Other (Numeric)
9F04	Amount, Other (Binary)
9F05	Application Discretionary Data
9F06	Application Identifier (AID) – terminal
9F07	Application Usage Control (AUC)
9F08	Application Version Number
9F09	Application Version Number (Default: 00 02 )
9F0B	Cardholder Name Extended
9F0D	Issuer Action Code - Default
9F0E	Issuer Action Code - Denial
9F0F	Issuer Action Code - Online



9F10	Issuer Application Data (IAD)
9F11	Issuer Code Table Index
9F12	Application Preferred Name
9F13	Last Online Application Transaction Counter (ATC) Register
9F14	Lower Consecutive Offline Limit
9F15	Merchant Category Code
9F16	Merchant Identifier
9F17	Personal Identification Number (PIN) Try Counter
9F18	Issuer Script Identifier
9F19	Deleted (see 9F49)
9F1A	Terminal Country Code
9F1B	Terminal Floor Limit
9F1C	Terminal Identification
9F1D	Terminal Risk Management Data
9F1E	Interface Device (IFD) Serial Number
9F1F	Track 1 Discretionary Data
9F20	Track 2 Discretionary Data
9F21	Transaction Time (HHMMSS )
9F22	Certification Authority Public Key Index
9F23	Upper Consecutive Offline Limit
9F26	Application Cryptogram (AC)
9F27	Cryptogram Information Data (CID)
9F29	Extended Selection
9F2A	Kernel Identifier
9F2D	Integrated Circuit Card (ICC) PIN Encipherment Public Key Certificate
9F2E	Integrated Circuit Card (ICC) PIN Encipherment Public Key Exponent
9F2F	Integrated Circuit Card (ICC) PIN Encipherment Public Key Remainder
9F32	Issuer Public Key Exponent
9F33	Terminal Capabilities (see below)
9F34	Cardholder Verification Method (CVM) Results
9F35	Terminal Type (see below)
9F36	Application Transaction Counter (ATC)
9F37	Unpredictable Number
9F38	Processing Options Data Object List (PDOL)
9F39	POS Entry Mode (Default: 07)
9F3A	Amount, Reference Currency
9F3B	Application Reference Currency
9F3C	Transaction Reference Currency Code
9F3D	Transaction Reference Currency Exponent
9F40	Additional Terminal Capabilities (see below)
9F41	Transaction Sequence Counter
9F42	Application Currency Code
9F43	Application Reference Currency Exponent
9F44	Application Currency Exponent

9F45	Data Authentication Code
9F46	Integrated Circuit Card (ICC) Public Key Certificate
9F47	Integrated Circuit Card (ICC) Public Key Exponent
9F48	Integrated Circuit Card (ICC) Public Key Remainder
9F49	Dynamic Data Authentication Data Object List (DDOL)
9F4A	Static Data Authentication Tag List (SDA)
9F4B	Signed Dynamic Application Data (SDAD)
9F4C	ICC Dynamic Number
9F4D	Log Entry
9F4E	Merchant Name and Location
9F4E	Merchant Name and Location
9F4F	Log Format
9F50	Offline Accumulator Balance
9F51	Application Currency Code
9F52	Application Default Action (ADA)
9F53	Transaction Category Code
9F54	DS ODS Card
9F55	Geographic Indicator
9F56	Issuer Authentication Indicator
9F57	Issuer Country Code
9F58	Consecutive Transaction Counter Limit (CTCL)
9F59	Consecutive Transaction Counter Upper Limit (CTCUL)
9F5A	Application Program Identifier (Program ID)
9F5B	Issuer Script Results
9F5C	Magstripe Data Object List (MDOL)
9F5D	Available Offline Spending Amount (AOSA)
9F5D	Application Capabilities Information (ACI)
9F5E	Consecutive Transaction International Upper Limit (CTIUL)
9F5E	DS ID
9F5F	DS Slot Availability
9F60	CVC3 (Track1)
9F61	CVC3 (Track2)
9F62	PCVC3 (Track1)
9F64	NATC (Track1)
9F65	PCVC3 (Track2)
9F66	PUNATC (Track2)
9F67	NATC (Track2)
9F68	Card Additional Processes
9F69	UDOL
9F6A	Unpredictable Number (Numeric)
9F6B	Track 2 Data
9F6C	Card Transaction Qualifiers (CTQ)
9F6D	Mag-stripe Application Version Number (Reader)
9F6E	Third Party Data
9F6E	Terminal Transaction Capabilities
9F6F	DS Slot Management Control

9F70	Protected Data Envelope 1
9F71	Protected Data Envelope 2
9F72	Protected Data Envelope 3
9F73	Protected Data Envelope 4
9F74	Protected Data Envelope 5
9F75	Unprotected Data Envelope 1
9F76	Unprotected Data Envelope 2
9F77	Unprotected Data Envelope 3
9F78	Unprotected Data Envelope 4
9F79	Unprotected Data Envelope 5
9F7A	VLP Terminal Support Indicator
9F7B	VLP Terminal Transaction Limit
9F7C	Customer Exclusive Data (CED)
9F7D	DS Summary 1
9F7F	DS Unpredictable Number
A5	File Control Information (FCI) Proprietary Template
BF0C	File Control Information (FCI) Issuer Discretionary Data
BF50	Visa Fleet - CDO
BF60	Integrated Data Storage Record Update Template
C3	Card issuer action code -decline
C4	Card issuer action code -default
C5	Card issuer action code online
C6	PIN Try Limit
C7	CDOL 1 Related Data Length
C8	Card risk management country code
C9	Card risk management currency code
CA	Lower cumulative offline transaction amount
CB	Upper cumulative offline transaction amount
CD	Card Issuer Action Code (PayPass) – Default
CE	Card Issuer Action Code (PayPass) – Online
CF	Card Issuer Action Code (PayPass) – Decline
D1	Currency conversion table
D2	Integrated Data Storage Directory (IDSD)
D3	Additional check table
D5	Application Control
D6	Default ARPC response code
D7	Application Control (PayPass)
D8	AIP (PayPass)
D9	AFL (PayPass)
DA	Static CVC3-TRACK1
DB	Static CVC3-TRACK2
DC	IVCVC3-TRACK1
DD	IVCVC3-TRACK2
DF01	ApplePay VAS Protocol
DF02	ApplePay VAS Failure Report
DF10	Terminal Languages Supported

DF10	Multi Language (Default: "enfr")
DF11	Enable Transaction Logging
DF13	Terminal Action Code - Default
DF14	Terminal Action Code - Denial
DF15	Terminal Action Code - Online
DF17	Threshold Value for Biased Random Selection
DF18	Target Percentage to be Used for Random Selection
DF19	Maximum Target Percentage to be used for Biased Random Selection
DF1F	Last 4 digits of Primary Account Number (PAN)
DF21	Issuer Script Results
DF22	Force Online (1-Enable, 0-Disable)
DF25	Default DDOL (1-Enable, 0-Disable)
DF26	Revocation List Support (Default: Enable - 1)
DF27	Exception File Support (Default: Disable - 0)
DF28	Default TDOL
DF29	Terminal Capabilities - CVM Required
DF2A	Threshold Value for Biased Random Selection(Interac)
DF2B	Maximum Target Percentage for Biased Random Selection (Interac)
DF2C	Target Percentage for Random Selection(Interac)
DF30	Track Data Source
DF31	DD Card Track 1
DF32	DD Card Track 2
DF33	Interac Receipt Required
DF34	TTK Customer - Firmware Version
DF40	Message to be displayed by EMV Kernel on "PIN Try Limit Exceeded" condition
DF41	Message to be displayed by EMV Kernel on "Last PIN Try" condition
DF42	Message to be displayed by EMV Kernel on "Please Try Again" condition
DF43	Message to be displayed by EMV Kernel on "Call Your Bank" condition
DF45	GMEDS Secret Keys
DF46	GMAD MIDs
DF47	ISIS Read Cmd Data
DF48	ISIS Write Data
DF49	ISIS Transaction Data
DF4A	TTK Customer - Current KSN of Data encryption Key
DF4B	TTK Customer - MSR all track data
DF4C	TTK Customer - Masked PAN
DF4D	TTK Customer - Additional POS Info
DF4E	Polling Options
DF4F	TTK Customer - Fallback Reason
DF50	Special Flow
DF51	Amex Terminal Capability
DF52	Transaction CVM

DF55	RID
DF56	Activate Trans for DESFireViVOCComm Flows
DF57	Reader Primary Language
DF57	2nd usage: Remaining Candidates
DF58	Reader Secondary Language
DF5A	TLVExclusion List
DF5B	Terminal Entry Capability
DF5C	RF Deactivate Period
DF5D	D-PAS Issuer Script Response status
DF5E	Transaction Timing Information
DF5F	Encrypted PAN for remote PIN Pad
DF60	Product ID
DF61	Processor ID
DF61	CVMRequiredLimit_JCBScheme
DF62	Main Firmware Build ID
DF63	CB Enhanced DDA Indicator (same block as DF03)
DF64	CB Wave 2 CVM Requirements (same block as DF04)
DF65	Build ID Num (Cxx)
DF65	CB Display Offline Funds Indicator (same block as DF05)
DF65	Serial heartbeat Required
DF66	SVN Number
DF66	CB Terminal Type (same block as 9F35)
DF66	Display Unsupported Card
DF68	Enable/Disable STOP command processing
DF69	ConfigureProprietaryTags
DF6A	Enable/Disable Comm Error Recovery
DF6C	Cubic FTP Phase 2 Mode Options
DF6D	Cubic Mode 3 Match AID
DF6E	Cubic Fixed Fare Amounts
DF6F	Cubic Timestamp Data
DF70	Loyalty Program ID
DF70	Generic Name String
DF71	Value Added Tax 1
DF71	Generic Numeric
DF72	Value Added Tax 2
DF72	Generic Specification String
DF73	Merchant Category Code
DF73	Generic Implementation String
DF74	Discover Optional Features
DF75	Communications Error Message Delay
DF76	TVR from GenAC
DF77	ViVOPay MSR Custom Data Output Tag
DF78	MC Timing Performance Enable
DF79	Card Disable Mask
DF7A	Card Disable Interval
DF7B	Serial Port (UART) Inter-character Timeout Period

DF7C	Auto Switch Feature
DF7D	Track Formatting Feature
DF7F	Improved Collision Detection & Media Removal Feature
DF891B	Poll Mode
DF891C	Interac Retry Limit
DFDE04	MSR Encryption Option
DFEE0C	PPSE Terminate Flags
DFEE12	KID
DFEE15	Application Selection Indicator
DFEE16	DUKPT Key or MKSK Select for Online PIN Encrypted
DFEE17	ICC Terminal Entry Mode
DFEE18	MSR Terminal Entry Mode
DFEE19	Online DOL
DFEE1A	Output data element
DFEE1B	Authorization Request data elements
DFEE1E	Contact Terminal Configuration (see below)
DFEE1F	Issuer script device limit, Range: 0~255 (Default: 128)
DFEE20	ICC Power on detect waiting time. (Unit: Sec) (Default: 60S)
DFEE21	ICC L1 waiting time. (Unit: Sec)(Default: 10 S)
DFEE22	Driver (Menu, Get PIN, Get MSR) Timeout. (Unit: Sec) (see below)
DFEE23	MSR Track Data
DFEE24	Force Acceptance (Default: 00)
DFEE25	ICC Response Code
DFEE26	Encryption StatusInformation
DFEE27	MSR Control
DFEF1A	SmartTap Delimiter
DFEF1E	Encrypted Sensitive Tags
DFEF1F	Auto Authenticate
DFEF20	MAC option in reponse data
DFEF21	BIN
DFEF22	AID
DFEF23	HMAC
DFEF24	HMAC KSN
DFEF25	Output Data Format Select
DFEF26	MSR fallback
DFEF27	Online capability
DFEF28	Disable Encrypt ON
DFEF2C	Terminal AID List
DFEF2E	Terminal Transaction Log
DFEF2F	CUP configuration
DFEF30	White List
DFEF31	Black List
DFEF32	Auto-Switch
DFEF34	Antenna Detection Switch

DDEF35	Communications Watchdog Period
DDEF36	Media Control & Status Tracking
DDEF37	Interface Select
DDEF38	Timeout for Next Command
DDEF39	Network Indicate
DDEF3A	Reader Behavior Mode
DDEF3B	Autopoll Transaction Separation Interval
DDEF40	Ascii-code encryption Tag57 TLV
DDEF41	MAC Verification Data for SRED
DDEF42	MAC VerificationKSN for SRED
DDEF43	Local TZ/DST information.
DDEF44	CombinationOptions
DDEF45	Removal Timeout
DDEF46	ACT Pass Response DOL
DDEF47	CDA Hash Input
DDEF48	Indicate - retrieve transaction result again due to Output RAM is Not enough.
DDEF49	Outcome Parameter Set
DDEF4A	User Interface Request Data
DDEF4B	MSR Equivalent Data Option
DDEF4C	MSR Equivalent Data Track Lengths
DDEF4D	MSR Equivalent Data
DDEF4E	ACT MSD Response DOL
DDEF4F	ACT Decline Response DOL
DDEF50	Terminal Interchange Profile (JCB)
DDEF51	Bypass EMV Completion Output
DDEF52	Re-FallBack times
DDEF53	Dynamic Reader Limits
DDEF54	SmartTap AID Index
DDEF55	Kernel Specific Features
DDEF56	Retry Limit
DDEF57	PPSE Terminate Flags
DDEF59	Terminal Data Setting - Default Amount
DDEF5A	Terminal Data Setting - Tags to Return
DDEF5B	Mask for Tag5A
DDEF5C	Mask for Tag56
DDEF5D	Mask for Tag57
DDEF5E	Mask for Tag9F6B
DDEF5F	Mask for TagFFEE13
DDEF60	Mask for TagFFEE14
DDEF61	Error Code
DDEF62	Allow MSR Swipe data from ICC Card
DDEF63	Tags To Read Yet
DDEF64	Referral Timeout
DDEF6E	USB-KB Output Data Postfix
DDEF6F	Inter-character Delay for USB-KB Interface
DDEF70	PISCES dual interface interference prevention mechanism fine-tune parameters.

DFF71	Waiting ICC insert time
DFF72	Pre-poll card mechanism control in ACT cmd & config setting
DFF73	Transaction Message Type
DFF74	Reference amplitude value
DFF75	Reference delta value
DFF76	Transaction Interface Type to activate
DFF77	Timeout for waiting next command
DFF78	EMV contact L2 display messages option
DFF79	PIN block format (when TDES)
DFF7A	Enable Apple Pay Check
DFF7B	Apple Pay Status
DFF7C	Track Bit Encoding
DFF7D	Re-power on times
DFF7E	Fallback response code list
FF69	ViVOPay Proprietary Tag List
FF70	Serial Finite State Machine Version
FF71	Transaction Finite State Machine Version
FF72	System Information Suite
FF73	Serial Protocol Version
FF74	Serial Protocol Suite
FF75	L1 Paypass Version
FF76	L1 LCR Version
FF77	L2 Card App Version
FF78	L2 Card App Suite
FF79	GMEDs Data
FF79	User Experience Version
FF7A	User Experience Suite
FF7B	ViVOTech Proprietary Suite
FF7C	VIUDS Scheme IDs Supported
FF7D	VIUDS Scheme ID Selection Criteria
FFE0	Registered Application Provider Identifier (RID)
FFE1	Partial Selection Allowed
FFE2	Application Flow
FFE3	Selection Features - GR 1.2.10
FFE4	Group Number / Fallback Group
FFE5	Max AID Length
FFE6	AID Disabled
FFE7	Interface Support
FFE8	Exclude from Processing
FFE9	Kernel ID Transaction Type Group List
FFEA	Default Kernel ID
FFEE01	ViVOPay TLV Group Tag
FFEE02	ViVOPay Pre-PPSE Special Flow Group Tag
FFEE03	ViVOPay Post-PPSE Special Flow Group Tag
FFEE04	M/Chip3 Intermediate Message Data
FFEE05	M/Chip3 Intermediate Message Marker



FFEE06	ApplePay VAS Container
FFEE07	Encrypted Sensitive Tags
FFEE08	Masked Tags
FFEE0A	BIN Range
FFEE0B	AID Range
FFEE0C	White List
FFEE10	ViVOpay MChip Group Tag
FFEE11	ViVOpay Discover Group Tag
FFEE12	KID
FFEE12	Cash Reader Risk Record
FFEE13	Track 1 Data
FFEE13	Cashback Reader Risk Record
FFEE14	Track 2 Data
FFEE14	DRL Record 1
FFEE15	DRL Record 2
FFEE16	DRL Record 3
FFEE17	DRL Record 4
FFEE18	Tags To Write Yet Before GenAC
FFEE19	Tags To Write Yet After GenAC
FFEE1A	Terminal App DET Data
FFEE1C	Unpredictable Number Range
FFEE1D	Sensitive Data Mask
FFEE1E	Group 0 Initialize Flag
FFEE1F	Error Code Table
FFEE20	Restart Deactivation Time
FFF0	Specific Features Switch
FFF1	Terminal Contactless Transaction Limit
FFF2	Terminal IFD
FFF3	Application Capability
FFF4	Visa Reader Risk Flags
FFF6	Torn Transaction Log Clean Interval (minutes)
FFF7	Burst Mode
FFF8	UI Scheme
FFF9	LCD Font Size
FFFA	LCD Delay Time
FFFB	Language Option for LCD
FFFC	Force MagStripe

### 9F33 Terminal Capabilities

Byte 1								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Manual key entry
x	1	x	x	x	x	x	x	Magnetic stripe
x	x	1	x	x	x	x	x	IC with contacts
x	x	x	0	x	x	x	x	RFU
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU
Byte 2								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Plaintext PIN for IC verification
x	1	x	x	x	x	X	x	Enciphered PIN for online verification
x	x	1	x	x	x	X	x	Signature(paper)
x	x	x	1	x	x	X	x	Enciphered PIN for offline verification
x	x	x	x	1	x	X	x	No CVM Required
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	X	0	RFU
Byte 3								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	SDA
X	1	x	x	x	x	x	x	DDA
X	x	1	x	x	x	x	x	Card capture
x	x	x	0	x	x	x	x	RFU

x	x	x	x	1	x	x	X	CDA
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	X	X	x	0	RFU

### 9F40 Additional Terminal Capabilities

Byte 1								
b1	b2	b3	b4	b5	b6	b7	b8	Meaning
1	x	x	x	x	x	x	x	Cash
x	1	x	x	x	x	x	x	Goods
x	x	1	x	x	x	x	x	Services
x	x	x	1	x	x	x	x	Cashback
x	x	x	x	1	x	x	x	Inquiry
x	x	x	x	x	1	x	x	Transfer
x	x	x	x	x	x	1	x	Payment
x	x	x	x	x	x	x	1	Administrative

Byte 2								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Cash Deposit
x	0	x	x	x	x	x	x	RFU
x	x	0	x	x	x	x	x	RFU
x	x	x	0	x	x	x	x	RFU
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

Byte 3								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Numeric keys
x	1	x	x	x	x	x	x	Alphabetic and special characters keys
x	x	1	x	x	x	x	x	Command keys
x	x	x	1	x	x	x	x	Function Keys
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

Byte 4								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Print, attendant
x	1	x	x	x	x	x	x	Print, cardholder
x	x	1	x	x	x	x	x	Display, attendant
x	x	x	1	x	x	x	x	Display, cardholder
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	1	x	Code table 10
x	x	x	x	x	x	x	1	Code table 9

Byte 5								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Code table 8
x	1	x	x	x	x	x	x	Code table 7
x	x	1	x	x	x	x	x	Code table 6
x	x	x	1	x	x	x	x	Code table 5
x	x	x	x	1	x	x	x	Code table 4
x	x	x	x	x	1	x	x	Code table 3
x	x	x	x	x	x	1	x	Code table 2
x	x	x	x	x	x	x	1	Code table 1

### 9F35 Terminal Type

Environment	Financial Institution	Merchant	Cardholder
Attended			
Online only	11	21	
Offline with online capability	12	22	
Offline only	13	23	
Unattended			
Online only	14	24	34
Offline with online capability	15	25	35
Offline only	16	26	36

## DFEE1E Contact Terminal Configuration (Default: F0 DC 3C F0 C2 9E 94 00)

### Byte 1

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Key Pad support
x	1	x	x	x	x	x	x	LCD support
x	x	1	x	x	x	x	x	PIN Pad support
x	x	x	1	x	x	x	x	Print Support
x	x	x	x	0	x	x	x	RFU
x	x	x	x	x	0	x	x	RFU
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	X	0	RFU

### Byte 2

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	PSE support
x	1	x	x	x	x	x	x	Cardholder confirmation
x	x	1	x	x	x	x	x	Preferred display order
x	x	x	1	x	x	x	x	Multi language
x	x	x	x	1	x	x	x	EMV language selection method
x	x	x	x	x	1	x	x	Default DDOL
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

### Byte 3

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	x	x	x	x	x	x	x	RFU
(Revocation of Issuer Public Key Certificate (DF26))								
x	1	x	x	x	x	x	x	Manual action when CA PK loading fails
x	x	1	x	x	x	x	x	CA PK verified with check sum
x	x	x	1	x	x	x	x	Bypass PIN Entry
x	x	x	x	1	x	x	x	Subsequent bypass PIN Entry
x	x	x	x	x	1	x	x	Get data for pin try counter
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

### Byte 4

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Amount before CVM processing
x	1	x	x	x	x	x	x	Floor limit checking
x	x	1	x	x	x	x	x	Random transaction selection
x	x	x	1	x	x	x	x	Velocity checking
x	x	x	x	0	x	x	x	RFU
(Transaction Log (DF11))								
x	x	x	x	x	0	x	x	RFU
(Exception File (DF27))								
x	x	x	x	x	x	0	x	RFU
x	x	x	x	x	x	x	0	RFU

### Byte 5

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	X	Terminal action code support
x	1	x	x	x	x	x	x	Terminal action code can be change
x	x	1	x	x	x	x	x	Terminal action code can be deleted or disable
x	x	x	1	x	x	x	x	Default Action code processing before 1st GAC
x	x	x	x	1	x	x	x	Default Action code processing after 1st GAC
x	x	x	x	x	1	x	x	TAC/IAC default process when unable to go online (Skipped)
x	x	x	x	x	x	1	x	TAC/IAC default process when unable to go online (Normal)
x	x	x	x	x	x	x	0	RFU

### Byte 6

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	Forced Online support
x	1	x	x	x	x	x	x	Forced acceptance support
x	x	1	x	x	x	x	x	Advices support
x	x	x	1	x	x	x	x	Issuer referrals support
X	x	x	x	1	x	x	x	Batch data capture
x	x	x	x	x	1	x	x	Online data capture
X	x	x	x	x	x	1	x	Default TDOL
X	x	x	x	x	x	x	0	RFU

### Byte 7

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	x	x	x	x	x	x	x	amount and pin entered on the same keypad
x	1	x	x	x	x	x	x	ICC/Magstripe reader combined
x	x	1	x	x	x	x	x	Magstripe read first
x	x	x	1	x	x	x	x	Support account type selection
x	x	x	x	1	x	x	x	On fly script processing
x	x	x	x	x	1	x	x	Internal date management
x	x	x	x	x	x	1	x	Reversal Mode
(1)Unable go online								
(2) ARC Error								
0: (3) Online Approved but reader not approved.								
1: (3) Online Approved but card response AAC.								
x	x	x	x	x	x	x	0	RFU

Byte 8								
b8	b7	b6	b5	b4	b3	b2	b1	Meaning
x	x	x	x	x	x	x	x	RFU

**DFEE22 Driver (Menu, Get PIN, Get MSR) Timeout. (Unit: Sec)**

Byte1: Timeout for Menu. (Default: 30 S)  
Byte2: Timeout for Get PIN. (Default: 60 S)  
Byte3: Timeout for Get MSR. (Default: 60 S)

## Chapter 11

# File Index

### 11.1 File List

Here is a list of all documented files with brief descriptions:

Source_C/libIDT_Augusta.h	
Augusta API	48
Source_C/libIDT_Device.h	
Windows C++ API	101
Source_C/libIDT_KioskIII.h	
KioskIII API	262
Source_C/libIDT_L100.h	
L100 API	283
Source_C/libIDT_MiniSmartII.h	
MiniSmartII API	314
Source_C/libIDT_NEO2.h	
NEO2 API	360
Source_C/libIDT_PipReader.h	
PipReader API	453
Source_C/libIDT_SpectrumPro.h	
SpectrumPro API	475
Source_C/libIDT_SREDKey2.h	
SREDKey2 API	516
Source_C/libIDT_UniPayI_V.h	
UniPay 1.5 API	535
Source_C/libIDT_Vendi.h	
Vendi API	560
Source_C/libIDT_VP3300_AJ.h	
VP3300 AJ API	581
Source_C/libIDT_VP3300_BT.h	
VP3300 BT API	620
Source_C/libIDT_VP3300_COM.h	
VP3300 COM API	656
Source_C/libIDT_VP3300_USB.h	
VP3300 USB API	692
Source_C/libIDT_VP8800.h	
VP8800 API	729

## Chapter 12

# File Documentation

### 12.1 Source\_C/libIDT\_Augusta.h File Reference

Augusta API.

```
#include "IDTDef.h"
```

#### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

#### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* pFW_callback )(int, int, int, int, int)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

#### Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callback pEMVf)`
- `void msr_registerCallBk (pMSR_callback pMSRf)`
- `void msr_registerCallBkp (pMSR_callbackp pMSRf)`
- `void pin_registerCallBk (pPIN_callback pPINf)`
- `void device_registerCameraCallBk (pCMR_callback pCMRf)`

- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callBack pCSFSf)
- void [device\\_registerFWCallBk](#) (pFW\_callBack pFWf)
- char \* [SDK\\_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char \*absoluteLibraryPath)
- int [device\\_init](#) ()
- int [device\\_setCurrentDevice](#) (int deviceType)
- int [device\\_close](#) ()
- void [device\\_getResponseCodeString](#) (IN int returnCode, OUT char \*despcriton)
- int [device\\_isConnected](#) ()
- int [device\\_isAttached](#) (int deviceType)
- int [device\\_getFirmwareVersion](#) (OUT char \*firmwareVersion)
- int [device\\_getFirmwareVersion\\_Len](#) (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommand](#) (IN BYTE \*cmd, IN int cmdLen, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_updateFirmware](#) (IN BYTE \*firmwareData, IN int firmwareDataLen, IN char \*firmwareName, IN int encryptionType, IN BYTE \*keyBlob, IN int keyBlobLen)
- int [device\\_rebootDevice](#) ()
- int [device\\_controlLED](#) (byte indexLED, byte control, int intervalOn, int intervalOff)
- int [device\\_controlLED\\_ICC](#) (int controlMode, int interval)
- int [device\\_controlLED\\_MSR](#) (byte control, int intervalOn, int intervalOff)
- int [device\\_controlBeep](#) (int index, int frequency, int duration)
- int [device\\_getDRS](#) (BYTE \*codeDRS, int \*codeDRSLen)
- int [device\\_getKeyStatus](#) (int \*newFormat, BYTE \*status, int \*statusLen)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [config\\_getModelNumber](#) (OUT char \*sNumber)
- int [config\\_getModelNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_setLEDController](#) (int firmwareControlMSRLED, int firmwareControlICCLELED)
- int [config\\_getLEDController](#) (int \*firmwareControlMSRLED, int \*firmwareControlICCLELED)
- int [config\\_setBeeperController](#) (int firmwareControlBeeper)
- int [config\\_getBeeperController](#) (int \*firmwareControlBeeper)
- int [config\\_setEncryptionControl](#) (int msr, int icc)
- int [config\\_getEncryptionControl](#) (int \*msr, int \*icc)
- int [icc\\_enable](#) (IN int withNotification)
- int [icc\\_disable](#) ()
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [icc\\_exchangeEncryptedAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [icc\\_getAPDU\\_KSN](#) (OUT BYTE \*KSN, IN\_OUT int \*inLen)
- int [icc\\_getFunctionStatus](#) (OUT int \*enabled, OUT int \*withNotification)
- int [icc\\_getICCReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_getKeyFormatForICCDUKPT](#) (OUT BYTE \*format)
- int [icc\\_getKeyTypeForICCDUKPT](#) (OUT BYTE \*type)
- int [emv\\_getEMVKernelVersion](#) (OUT char \*version)
- int [emv\\_getEMVKernelVersion\\_Len](#) (OUT char \*version, IN\_OUT int \*versionLen)
- int [emv\\_getEMVKernelCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- int [emv\\_getEMVConfigurationCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- void [emv\\_allowFallback](#) (IN int allow)
- void [emv\\_setAutoAuthenticateTransaction](#) (IN int authenticate)

- void `emv_setAutoCompleteTransaction` (IN int complete)
- int `emv_getAutoAuthenticateTransaction` ()
- int `emv_getAutoCompleteTransaction` ()
- int `emv_startTransaction` (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int `emv_activateTransaction` (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int `emv_authenticateTransaction` (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int `emv_authenticateTransactionWithTimeout` (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int `emv_completeTransaction` (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int `emv_cancelTransaction` ()
- int `emv_retrieveTransactionResult` (IN BYTE \*tags, IN int tagsLen, IDTTransactionData \*cardData)
- int `emv_callbackResponseLCD` (IN int type, byte selection)
- int `emv_callbackResponseMSR` (IN BYTE \*MSR, IN\_OUT int MSRLen)
- int `emv_retrieveApplicationData` (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setApplicationData` (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int `emv_removeApplicationData` (IN BYTE \*AID, IN int AIDLen)
- int `emv_removeAllApplicationData` ()
- int `emv_retrieveAIDList` (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int `emv_retrieveTerminalData` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setTerminalData` (IN BYTE \*tlv, IN int tlvLen)
- int `emv_removeTerminalData` ()
- int `emv_retrieveCAPK` (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int `emv_setCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeAllCAPK` ()
- int `emv_retrieveCAPKList` (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int `emv_retrieveTerminalID` (OUT char \*terminalID)
- int `emv_retrieveTerminalID_Len` (OUT char \*terminalID, IN\_OUT int \*terminalIDLen)
- int `emv_setTerminalID` (IN char \*terminalID)
- int `emv_retrieveCRL` (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int `emv_setCRL` (IN BYTE \*list, IN int lsLen)
- int `emv_removeCRL` (IN BYTE \*list, IN int lsLen)
- int `emv_removeAllCRL` ()
- int `msr_getMSRData` (OUT BYTE \*reData, IN\_OUT int \*reLen)
- int `msr_cancelMSRSwipe` ()
- int `msr_startMSRSwipe` (IN int \_timeout)
- void `parseMSRData` (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)
- int `msr_getKeyFormatForICCDUKPT` (OUT BYTE \*format)
- int `msr_getKeyTypeForICCDUKPT` (OUT BYTE \*type)
- int `msr_setKeyFormatForICCDUKPT` (IN BYTE format)
- int `msr_setKeyTypeForICCDUKPT` (IN BYTE type)
- int `msr_captureMode` (int isBufferMode, int withNotification)
- int `msr_setSetting` (BYTE setting, BYTE \*val, int valLen)
- int `msr_getSetting` (byte setting, BYTE \*value, int \*valueLen)
- int `msr_setSwipeForcedEncryptionOption` (int track1, int track2, int track3, int track3card0)
- int `msr_getSwipeForcedEncryptionOption` (BYTE \*option)
- int `msr_setSwipeMaskOption` (int track1, int track2, int track3)
- int `msr_getSwipeMaskOption` (BYTE \*option)
- int `msr_setExpirationMask` (int mask)
- int `msr_getExpirationMask` (BYTE \*value)
- int `msr_setClearPANID` (BYTE val)
- int `msr_getClearPANID` (BYTE \*value)
- int `msr_disable` ()
- int `pin_cancelPINEntry` ()



### 12.1.1 Detailed Description

Augusta API. Augusta Global API methods.

### 12.1.2 Macro Definition Documentation

#### 12.1.2.1 #define IN

INPUT parameter.

#### 12.1.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.1.2.3 #define OUT

OUTPUT parameter.

### 12.1.3 Typedef Documentation

#### 12.1.3.1 typedef void(\* ftpComm\_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.1.3.2 typedef void(\* httpComm\_callBack)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

#### 12.1.3.3 typedef void(\* pCMR\_callBack)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

#### 12.1.3.4 typedef void(\* pCSFS\_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

#### 12.1.3.5 typedef void(\* pEMV\_callBack)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

#### 12.1.3.6 `typedef void(* pFW_callback)(int, int, int, int, int)`

Define the firmware update callback function to get the status of firmware update

It should be registered using the `device_registerFWCallback`,

#### 12.1.3.7 `typedef void(* pMessageHotplug)(int, int)`

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the `registerHotplugCallback`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.1.3.8 `typedef void(* pMSR_callback)(int, IDTMSRData)`

Define the MSR callback function to get the MSR card data

It should be registered using the `msr_registerCallback`, this callback function is for backward compatibility

#### 12.1.3.9 `typedef void(* pMSR_callback)(int, IDTMSRData *)`

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallback`, this callback function is recommended instead of `pMSR_callback`

#### 12.1.3.10 `typedef void(* pPIN_callback)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallback`,

#### 12.1.3.11 `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallback`,

#### 12.1.3.12 `typedef void(* pSendDataLog)(unsigned char *, int)`

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallback`,

#### 12.1.3.13 `typedef void(* v4Comm_callback)(BYTE, BYTE, BYTE *, int)`

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

### 12.1.4 Function Documentation

#### 12.1.4.1 `int config_getBeeperController ( int * firmwareControlBeeper )`

Get the Beeper Controller Status Set the Beeper controlled Status by software or firmware

## Parameters

<i>firmwareControl-Beeper</i>	1 means firmware control the beeper, 0 means software control beeper.
-------------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.2 int config\_getEncryptionControl ( int \* *msr*, int \* *icc* )

## Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

## Parameters

<i>msr</i>	<ul style="list-style-type: none"> <li>• 1: enabled MSR with Encryption,</li> <li>• 0: disabled MSR with Encryption,</li> </ul>
<i>icc</i>	<ul style="list-style-type: none"> <li>• 1: enabled ICC with Encryption,</li> <li>• 0: disabled ICC with Encryption,</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.3 int config\_getLEDController ( int \* *firmwareControlMSRLED*, int \* *firmwareControlICCLED* )

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

## Parameters

<i>firmwareControl-MSRLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the MSR LED</li> <li>• 0: software control the MSR LED</li> </ul>
-------------------------------	--

<i>firmwareControl-ICCLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the ICC LED</li> <li>• 0: software control the ICC LED</li> </ul>
-------------------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.4 int config\_getModelNumber ( OUT char \* *sNumber* )

Polls device for Model Number

**Parameters**

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.5 int config\_getModelNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

DEPRECATED : please use [config\\_getModelNumber\\_Len\(OUT char\\* \*sNumber\*, IN\\_OUT int \\*\*sNumberLen\*\)](#)

Polls device for Model Number

**Parameters**

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.6 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len\(OUT char\\* \*sNumber\*, IN\\_OUT int \\*\*sNumberLen\*\)](#)

Polls device for Serial Number

**Parameters**

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.7 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.8 int config\_setBeeperController ( int *firmwareControlBeeper* )

Set the Beeper Controller Set the Beeper controlled by software or firmware

## Parameters

<i>firmwareControl-Beeper</i>	1 means firmware control the beeper, 0 means software control beeper.
-------------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.9 int config\_setEncryptionControl ( int *msr*, int *icc* )

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

## Parameters

<i>msr</i>	<ul style="list-style-type: none"> <li>• 1: enable MSR with Encryption,</li> <li>• 0: disable MSR with Encryption,</li> </ul>
<i>icc</i>	<ul style="list-style-type: none"> <li>• 1: enable ICC with Encryption,</li> <li>• 0: disable ICC with Encryption,</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.10 int config\_setLEDController ( int *firmwareControlMSRLED*, int *firmwareControlICCLED* )

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

**Parameters**

<i>firmwareControl-MSRLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the MSR LED</li> <li>• 0: software control the MSR LED</li> </ul>
<i>firmwareControl-ICCLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the ICC LED</li> <li>• 0: software control the ICC LED</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.1.4.11 int device\_close ( )**

Close the device

**Returns**

RETURN\_CODE: 0: success, 0x0A: failed

**12.1.4.12 int device\_controlBeep ( int *index*, int *frequency*, int *duration* )**

Control Beep

Controls the Beeper

**Parameters**

<i>index</i>	For Augusta, must be set to 1 (only one beeper)
<i>frequency</i>	Frequency, range 1000-20000 (suggest minimum 3000)
<i>duration</i>	Duration, in milliseconds (range 1 - 65525)

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.13 int device\_controlLED ( byte *indexLED*, byte *control*, int *intervalOn*, int *intervalOff* )

**Control MSR LED**

Controls the LED for the MSR

**Parameters**

<i>indexLED</i>	For Augusta, must be set to 1 (MSR LED)
<i>control</i>	LED Status: <ul style="list-style-type: none"> <li>• 00: OFF</li> <li>• 01: RED Solid</li> <li>• 02: RED Blink</li> <li>• 11: GREEN Solid</li> <li>• 12: GREEN Blink</li> <li>• 21: BLUE Solid</li> <li>• 22: BLUE Blink</li> </ul>
<i>intervalOn</i>	Blink interval ON, in ms (Range 200 - 2000)
<i>intervalOff</i>	Blink interval OFF, in ms (Range 200 - 2000)

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.14 int device\_controlLED\_ICC ( int *controlMode*, int *interval* )

**Control ICC LED**

Controls the LED for the ICC card slot

**Parameters**

<i>controlMode</i>	0 = off, 1 = solid, 2 = blink
<i>interval</i>	Blink interval, in ms (500 = 500 ms)

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.15 int device\_controlLED\_MSR ( byte *control*, int *intervalOn*, int *intervalOff* )

**Control the MSR LED**

Controls the MSR / ICC LED This API not recommended to control ICC LED

## Parameters

<i>control</i>	<ul style="list-style-type: none"> <li>• 0x00 = off,</li> <li>• 0x01 = RED Solid,</li> <li>• 0x02 = RED Blink,</li> <li>• 0x11 = GREEN Solid,</li> <li>• 0x12 = GREEN Blink,</li> <li>• 0x21 = BLUE Solid,</li> <li>• 0x22 = BLUE Blink,</li> </ul>
<i>intervalOn</i>	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>intervalOff</i>	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.1.4.16 `int device_getCurrentDeviceType ( )`

Get current active device type

## Returns

: return the device type defined as `DEVICE_TYPE` in the `IDTDef.h`

12.1.4.17 `int device_getDRS ( BYTE * codeDRS, int * codeDRSLen )`

Get DRS Status

Gets the status of DRS(Destructive Reset).

## Parameters

<i>codeDRS</i>	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]
----------------	--

[SourceID] [SourceLen] [SourceData] 00 1 01 - Application Error 01 1 01 - Application Error 02 1 0x01 - EMV L2 Configuration Check Value Error 0x02 - Future Key Check Value Error 10 1 01 - Battery Error 11 1 Bit 0 - Tamper Switch 1 (0-No, 1-Error) Bit 1 - Tamper Switch 2 (0-No, 1-Error) Bit 2 - Tamper Switch 3 (0-No, 1-Error) Bit 3 - Tamper Switch 4 (0-No, 1-Error) Bit 4 - Tamper Switch 5 (0-No, 1-Error) Bit 5 - Tamper Switch 6 (0-No, 1-Error)

12 1 01 - TemperatureHigh or Low 13 1 01 - Voltage High or Low 1F 4 Reg31~24bits, Reg23~16bits, Reg15~8bits, Reg7~0bits

## Parameters

<i>codeDRSLen</i>	the length of codeDRS
-------------------	-----------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#) Note: Only support TTK devices



12.1.4.18 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.19 `int device_getFirmwareVersion_Len ( OUT char * firmwareVersion, IN_OUT int * firmwareVersionLen )`

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len\(OUT char\\* firmwareVersion, IN\\_OUT int \\*firmwareVersionLen\)](#)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.20 `int device_getKeyStatus ( int * newFormat, BYTE * status, int * statusLen )`

Get Key Status

Gets the status of loaded keys

## Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len\_L Len\_H, is KeyStatusBlock Number [KeyStatusBlockX] is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

## Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.21 void device\_getResponseCodeString ( IN int *returnCode*, OUT char \* *despcrition* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";

- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";

- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";

- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";

- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";

- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";



- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE\_OPEN\_FAILED";
- 0X1003: "FILE OPERATION\_FAILED";
- 0X2001: "MEMORY\_NOT\_ENOUGH";
- 0X3002: "SMARTCARD\_FAIL";
- 0X3003: "SMARTCARD\_INIT\_FAILED";
- 0X3004: "FALLBACK\_SITUATION";
- 0X3005: "SMARTCARD\_ABSENT";
- 0X3006: "SMARTCARD\_TIMEOUT";
- 0X3012: "EMV\_RESULT\_CODE\_MSR\_CARD\_ERROR\_FALLBACK";
- 0X5001: "EMV\_PARSING\_TAGS\_FAILED";
- 0X5002: "EMV\_DUPLICATE\_CARD\_DATA\_ELEMENT";
- 0X5003: "EMV\_DATA\_FORMAT\_INCORRECT";
- 0X5004: "EMV\_NO\_TERM\_APP";
- 0X5005: "EMV\_NO\_MATCHING\_APP";
- 0X5006: "EMV\_MISSING\_MANDATORY\_OBJECT";
- 0X5007: "EMV\_APP\_SELECTION\_RETRY";
- 0X5008: "EMV\_GET\_AMOUNT\_ERROR";
- 0X5009: "EMV\_CARD\_REJECTED";
- 0X5010: "EMV\_AIP\_NOT\_RECEIVED";
- 0X5011: "EMV\_AFL\_NOT\_RECEIVED";
- 0X5012: "EMV\_AFL\_LEN\_OUT\_OF\_RANGE";
- 0X5013: "EMV\_SFI\_OUT\_OF\_RANGE";
- 0X5014: "EMV\_AFL\_INCORRECT";
- 0X5015: "EMV\_EXP\_DATE\_INCORRECT";
- 0X5016: "EMV\_EFF\_DATE\_INCORRECT";
- 0X5017: "EMV\_ISS\_COD\_TBL\_OUT\_OF\_RANGE";
- 0X5018: "EMV\_CRYPTOGRAM\_TYPE\_INCORRECT";

- 0X5019: "EMV\_PSE\_NOT\_SUPPORTED\_BY\_CARD";
- 0X5020: "EMV\_USER\_SELECTED\_LANGUAGE";
- 0X5021: "EMV\_SERVICE\_NOT\_ALLOWED";
- 0X5022: "EMV\_NO\_TAG\_FOUND";
- 0X5023: "EMV\_CARD\_BLOCKED";
- 0X5024: "EMV\_LEN\_INCORRECT";
- 0X5025: "CARD\_COM\_ERROR";
- 0X5026: "EMV\_TSC\_NOT\_INCREASED";
- 0X5027: "EMV\_HASH\_INCORRECT";
- 0X5028: "EMV\_NO\_ARC";
- 0X5029: "EMV\_INVALID\_ARC";
- 0X5030: "EMV\_NO\_ONLINE\_COMM";
- 0X5031: "TRAN\_TYPE\_INCORRECT";
- 0X5032: "EMV\_APP\_NO\_SUPPORT";
- 0X5033: "EMV\_APP\_NOT\_SELECT";
- 0X5034: "EMV\_LANG\_NOT\_SELECT";
- 0X5035: "EMV\_NO\_TERM\_DATA";
- 0X5039: "EMV\_PIN\_ENTRY\_TIMEOUT";
- 0X6001: "CVM\_TYPE\_UNKNOWN";
- 0X6002: "CVM\_AIP\_NOT\_SUPPORTED";
- 0X6003: "CVM\_TAG\_8E\_MISSING";
- 0X6004: "CVM\_TAG\_8E\_FORMAT\_ERROR";
- 0X6005: "CVM\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6006: "CVM\_COND\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6007: "NO\_MORE\_CVM";
- 0X6008: "PIN\_BYPASSED\_BEFORE";
- 0X7001: "PK\_BUFFER\_SIZE\_TOO\_BIG";
- 0X7002: "PK\_FILE\_WRITE\_ERROR";
- 0X7003: "PK\_HASH\_ERROR";
- 0X8001: "NO\_CARD\_HOLDER\_CONFIRMATION";
- 0X8002: "GET\_ONLINE\_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";

- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected that the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";

- 0XBBEC: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

#### 12.1.4.22 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

##### Returns

SDK wait time in seconds

#### 12.1.4.23 int device\_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

##### Returns

Thread Stack Size

#### 12.1.4.24 int device\_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.25 int device\_isAttached ( int *deviceType* )

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

##### Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

##### Returns

1 if the device is attached, or 0 if the device is not attached

#### 12.1.4.26 int device\_isConnected ( )

Check the device connected status

##### Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

#### 12.1.4.27 int device\_rebootDevice ( )

Reboot Device Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.28 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

#### 12.1.4.29 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

#### 12.1.4.30 void device\_registerFWCallBk ( pFW\_callback pFWf )

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

#### 12.1.4.31 int device\_SendDataCommand ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

##### Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.32 int device\_setCurrentDevice ( int deviceType )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VEND1,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

12.1.4.33 void device\_setSDKWaitTime ( int *waitTime* )

## Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

12.1.4.34 void device\_setThreadStackSize ( int *threadSize* )

## Set Thread Stack Size

Set the stack size setting for newly created threads

12.1.4.35 int device\_updateFirmware ( IN BYTE \* *firmwareData*, IN int *firmwareDataLen*, IN char \* *firmwareName*, IN int *encryptionType*, IN BYTE \* *keyBlob*, IN int *keyBlobLen* )

Update Firmware Updates the firmware of Augusta.

## Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"><li>• For example "Augusta_S_TTK_V1.00.002.fm"</li></ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"><li>• 0 : Plaintext</li><li>• 1 : TDES ECB, PKCS#5 padding</li><li>• 2 : TDES CBC, PKCS#5, IV is all 0</li></ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of keyBlob

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device-State.FirmwareUpdate data = File Progress. Two bytes, with `byte[0]` = current block, and `byte[1]` = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.1.4.36 `int emv_activateTransaction ( IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable



12.1.4.37 void emv\_allowFallback ( IN int *allow* )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.1.4.38 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.39 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>

<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.1.4.40 int emv\_callbackResponseLCD ( IN int *type*, byte *selection* )

**Callback Response LCD Display**

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV\_CALLBACK\_TYPE.EMV\_CALLBACK\_TYPE\_LCD, and lcd\_displayMode = EMV\_LCD\_DISPLAY\_MODE\_MENU, EMV\_LCD\_DISPLAY\_MODE\_PROMPT, or EMV\_LCD\_DISPLAY\_MODE\_LANGUAGE\_SELECT

**Parameters**

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.41 int emv\_callbackResponseMSR ( IN BYTE \* *MSR*, IN\_OUT int *MSRLen* )

**Callback Response MSR Entry**

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV\_CALLBACK\_MSR

**Parameters**

<i>MSR</i>	Swiped track data
<i>MSRLen</i>	the length of Swiped track data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.42 int emv\_cancelTransaction ( )

**Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.1.4.43** `int emv_completeTransaction ( IN int commError, IN BYTE * authCode, IN int authCodeLen, IN BYTE * iad, IN int iadLen, IN BYTE * tlvScripts, IN int tlvScriptsLen, IN BYTE * tlv, IN int tlvLen )`

#### Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

#### Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, <i>authCode</i> , <i>iad</i> , <i>tlvScripts</i> can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of <i>authCode</i>
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of <i>iad</i>
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of <i>tlvScripts</i>
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of <i>tlv</i>

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.1.4.44** `int emv_getAutoAuthenticateTransaction ( )`

Gets auto authenticate value for EMV transactions.

#### Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

**12.1.4.45** `int emv_getAutoCompleteTransaction ( )`

Gets auto complete value for EMV transactions.

#### Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

**12.1.4.46** `int emv_getEMVConfigurationCheckValue ( OUT BYTE * checkValue, IN_OUT int * checkValueLen )`

Get EMV Kernel configuration check value info

#### Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of <i>checkValue</i>

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.1.4.47** `int emv_getEMVKernelCheckValue ( OUT BYTE * checkValue, IN_OUT int * checkValueLen )`

Get EMV Kernel check value info

## Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.1.4.48 `int emv_getEMVKernelVersion ( OUT char * version )`

DEPRECATED : please use `emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)`

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.1.4.49 `int emv_getEMVKernelVersion_Len ( OUT char * version, IN_OUT int * versionLen )`

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.1.4.50 `void emv_registerCallBk ( pEMV_callBack pEMVf )`

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.1.4.51 `int emv_removeAllApplicationData ( )`

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.1.4.52 `int emv_removeAllCAPK ( )`

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

#### 12.1.4.53 int emv\_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.54 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

##### Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.55 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

##### Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.56 int emv\_removeCRL ( IN BYTE \* list, IN int lssLen )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

##### Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.1.4.57 int emv\_removeTerminalData ( )**

Remove Terminal Data

Removes the Terminal Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.1.4.58 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )**

Retrieve AID list

Returns all the AID names installed on the terminal.

**Parameters**

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.1.4.59 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )**

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

**Parameters**

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.60 `int emv_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

**Retrieve Certificate Authority Public Key**

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

**Parameters**

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.61 `int emv_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

**Retrieve the Certificate Authority Public Key list**

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)



12.1.4.62 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *lssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>listLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.63 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

## Retrieve Terminal Data

Retrieves the Terminal Data.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.64 int emv\_retrieveTerminalID ( OUT char \* *terminalID* )

DEPRECATED : please use [emv\\_retrieveTerminalID\\_Len](#)(OUT char\* *terminalID*, IN\_OUT int \**terminalIDLen*)

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.65 int emv\_retrieveTerminalID\_Len ( OUT char \* *terminalID*, IN\_OUT int \* *terminalIDLen* )

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.66 int emv\_retrieveTransactionResult ( IN BYTE \* *tags*, IN int *tagsLen*, IDTTransactionData \* *cardData* )

## Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

## Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDT-TransactionData object

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.67 `int emv_setApplicationData ( IN BYTE * name, IN int nameLen, IN BYTE * tlv, IN int tlvLen )`

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.68 `void emv_setAutoAuthenticateTransaction ( IN int authenticate )`

Enables authenticate for EMV transactions. If a `emv_startTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

12.1.4.69 `void emv_setAutoCompleteTransaction ( IN int complete )`

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

12.1.4.70 `int emv_setCAPK ( IN BYTE * capk, IN int capkLen )`

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.71 int emv\_setCRL ( IN BYTE \* *list*, IN int *lsLen* )

## Set Certificate Revocation List

Sets the CRL

## Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.72 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with device_getResponseCodeString:()
--------------------	---

12.1.4.73 int emv\_setTerminalID ( IN char \* *terminalID* )

Sets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.74 int emv\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C060000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## 12.1.4.75 int icc\_disable ( )

ICC Function enable/disable Disable ICC function

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.76 int `icc_enable` ( IN int *withNotification* )

ICC Function enable/disable Enable ICC function with or without seated notification

## Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> <li>• 1: with notification when ICC seated status changed,</li> <li>• 0: without notification.</li> </ul>
-------------------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.77 int icc\_exchangeAPDU ( IN BYTE \* *c\_APDU*, IN int *cLen*, OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.1.4.78 int icc\_exchangeEncryptedAPDU ( IN BYTE \* *c\_APDU*, IN int *cLen*, OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )

Exchange APDU with encrypted data For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	KSN + encrypted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]
---------------	---

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

## Parameters

<i>cLen</i>	data packet length
<i>reData</i>	response encrypted APDU response. Can be three options:

[00] + [Plaintext R-APDU]

- [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]
- [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]

The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN

is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.



## Parameters

<i>reLen</i>	encrypted APDU response data length
--------------	-------------------------------------

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.79 int icc\_getAPDU\_KSN ( OUT BYTE \* *KSN*, IN\_OUT int \* *inLen* )

## Get APDU KSN

Retrieves the KSN used in ICC Encrypted APDU usage

## Parameters

<i>KSN</i>	Returns the encrypted APDU packet KSN
<i>inLen</i>	KSN data length

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.80 int icc\_getFunctionStatus ( OUT int \* *enabled*, OUT int \* *withNotification* )

Get ICC Function status Get ICC Function status about enable/disable and with or without seated notification

## Parameters

<i>enabled</i>	<ul style="list-style-type: none"> <li>• 1: ICC Function enabled,</li> <li>• 0: means disabled.</li> </ul>
<i>withNotification</i>	1 means with notification when ICC seated status changed. 0 means without notification.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.81 int icc\_getICCRReaderStatus ( OUT BYTE \* *status* )

## Get Reader Status

Returns the reader status

## Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.82 int `icc_getKeyFormatForICCDUKPT` ( OUT BYTE \* *format* )

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

## Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default)</li> <li>• 'AES': Encrypted card data with AES if DUKPT Key had been loaded.</li> <li>• 'NONE': No Encryption.</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.1.4.83 int `icc_getKeyTypeForICCDUKPT ( OUT BYTE * type )`

Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

## Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default)</li> <li>• 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.</li> </ul>
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.1.4.84 int `icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

12.1.4.85 int `icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

## Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

#### 12.1.4.86 int msr\_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.87 int msr\_captureMode ( int *isBufferMode*, int *withNotification* )

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

##### Parameters

<i>isBufferMode</i>	<ul style="list-style-type: none"> <li>• 1: Enter into Buffer mode.</li> <li>• 0: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer</li> </ul>
<i>withNotification</i>	<ul style="list-style-type: none"> <li>• 1: with notification when swiped MSR data.</li> <li>• 0: without notification when swiped MSR data.</li> </ul>

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.1.4.88 int msr\_disable ( )

Disable MSR Disable MSR functions.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.89 int msr\_getClearPANID ( BYTE \* *value* )

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

##### Parameters

<i>value</i>	4901 <Setting value>="". setting Value: Number of digits in clear. Values are char '0' - '6'
--------------	--

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.1.4.90 int msr\_getExpirationMask ( BYTE \* *value* )

Get MSR expiration date mask.

## Parameters

<i>value</i>	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked
--------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.91 int msr\_getKeyFormatForICCDUKPT ( OUT BYTE \* *format* )

## Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

## Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default)</li> <li>• 'AES': Encrypted card data with AES if DUKPT Key had been loaded.</li> <li>• 'NONE': No Encryption.</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.92 int msr\_getKeyTypeForICCDUKPT ( OUT BYTE \* *type* )

## Get Key Type for DUKPT

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

## Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default)</li> <li>• 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.</li> </ul>
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.93 int msr\_getMSRData ( OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )

## Get MSR Data Reads the MSR Data buffer

## Parameters

<i>reData</i>	Card data
<i>reLen</i>	Card data length

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.1.4.94 int msr\_getSetting ( byte *setting*, BYTE \* *value*, int \* *valueLen* )

Get Single MSR Setting value

Returns the encryption used for swipe data

**Parameters**

<i>setting</i>	the msr setting to retrieve.
<i>value</i>	MSR Setting value.
<i>valueLen</i>	the length of value.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.1.4.95 int msr\_getSwipeForcedEncryptionOption ( BYTE \* *option* )

Get MSR Swipe Forced Encryption Option.

**Parameters**

<i>option</i>	8401 <Setting value>="". Setting Value Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.1.4.96 int msr\_getSwipeMaskOption ( BYTE \* *option* )

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

**Parameters**

<i>option</i>	8601 <Setting value>="". Setting Value Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off
---------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.1.4.97 void msr\_registerCallBk ( pMSR\_callback *pMSRf* )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

## 12.1.4.98 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

## 12.1.4.99 int msr\_setClearPANID ( BYTE val )

Set Clear PAN ID.

## Parameters

<i>val</i>	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.
------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.1.4.100 int msr\_setExpirationMask ( int mask )

Set Expiration Masking

Sets the flag to mask the expiration date

## Parameters

<i>mask</i>	TRUE = mask expiration
-------------	------------------------

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.1.4.101 int msr\_setKeyFormatForICCDUKPT ( IN BYTE format )

Set Key Format for DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

## Parameters

<i>format</i>	encryption Encryption Type <ul style="list-style-type: none"> <li>• 00: Encrypt with TDES</li> <li>• 01: Encrypt with AES</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.1.4.102 int msr\_setKeyTypeForICCDUKPT ( IN BYTE type )

Set Key Type for DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

## Parameters

<i>type</i>	Encryption Type <ul style="list-style-type: none"> <li>• 00: Encrypt with Data Key</li> <li>• 01: Encrypt with PIN Key</li> </ul>
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.103 int msr\_setSetting ( BYTE *setting*, BYTE \* *val*, int *valLen* )

Set MSR settings.

## Parameters

<i>setting</i>	the msr setting to set.
<i>val</i>	the value to set to the msr setting.
<i>valLen</i>	the length of val.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.104 int msr\_setSwipeForcedEncryptionOption ( int *track1*, int *track2*, int *track3*, int *track3card0* )

Set MSR Swipe Forced Encryption Option.

## Parameters

<i>track1</i>	Set track1 encryption to true or false.
<i>track2</i>	Set track2 encryption to true or false.
<i>track3</i>	Set track3 encryption to true or false.
<i>track3card0</i>	Set track3 card0 encryption to true or false.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.1.4.105 int msr\_setSwipeMaskOption ( int *track1*, int *track2*, int *track3* )

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

## Parameters

<i>track1</i>	Set track1 mask to true or false.
<i>track2</i>	Set track2 mask to true or false.
<i>track3</i>	Set track3 mask to true or false.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString



**12.1.4.106 int msr\_startMSRSwipe ( IN int *timeout* )**

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.107 void parseMSRData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTMSRData \* *cardData* )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of <i>resData</i>
<i>cardData</i>	the parser result with IDTMSTData structure

12.1.4.108 int pin\_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.1.4.109 void pin\_registerCallBk ( pPIN\_callBack *pPINf* )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.1.4.110 void registerHotplugCallBk ( pMessageHotplug *pMsgHotplug* )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.1.4.111 void registerLogCallBk ( pSendDataLog *pFSend*, pReadDataLog *pFRead* )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.1.4.112 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.1.4.113 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.2 Source\_C/libIDT\_Device.h File Reference

Windows C++ API.

```
#include "IDTDef.h"
```

## Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

## Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pLCD_callback )(int, IDTLCDItem *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* pFW_callback )(int, int, int, int, int)`
- `typedef void(* pRKI_callback )(int, char *)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* pLog_callback )(BYTE, char *)`

## Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, pReadDataLog pFRead)
- void [emv\\_registerCallBk](#) (pEMV\_callback pEMVf)
- void [loyalty\\_registerCallBk](#) (pEMV\_callback pEMVf)
- void [msr\\_registerCallBk](#) (pMSR\_callback pMSRf)
- void [msr\\_registerCallBkp](#) (pMSR\_callbackp pMSRf)
- void [pin\\_registerCallBk](#) (pPIN\_callback pPINf)
- void [lcd\\_registerCallBk](#) (pLCD\_callback pLCDf)
- void [device\\_registerCameraCallBk](#) (pCMR\_callback pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callback pCSFSf)

- void `device_registerFWCallBk` (pFW\_callBack pFWf)
- void `device_registerRKICallBk` (pRKI\_callBack pRKIf)
- char \* `SDK_Version` ()
- int `setAbsoluteLibraryPath` (const char \*absoluteLibraryPath)
- int `device_setConfigPath` (const char \*path)
- int `device_setNEO2DevicesConfigs` (IN const char \*configs, IN int len)
- int `device_init` ()
- int `rs232_device_init` (int deviceType, int port\_number, int brate)
- int `device_setCurrentDevice` (int deviceType)
- int `device_close` ()
- void `device_getResponseCodeString` (IN int returnCode, OUT char \*despcrition)
- void `device_getIDGStatusCodeString` (IN int returnCode, OUT char \*despcrition)
- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `loyalty_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen, IN const int cardType, IN const int iccReadType)
- void `device_setTransactionExponent` (int exponent)
- int `device_activateTransaction` (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `device_cancelTransaction` ()
- int `loyalty_cancelTransaction` ()
- int `device_setCancelTransactionMode` (int mode)
- int `device_cancelTransactionSilent` (int enable)
- int `loyalty_cancelTransactionSilent` (int enable)
- int `device_getDriveFreeSpace` (OUT int \*free, OUT int \*used)
- int `device_listDirectory` (IN char \*directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char \*directory, IN\_OUT int \*directoryLen)
- int `device_createDirectory` (IN char \*directoryName, IN int directoryNameLen)
- int `device_deleteDirectory` (IN char \*dirName, IN int dirNameLen)
- int `device_transferFile` (IN char \*fileName, IN int fileNameLen, IN BYTE \*file, IN int fileLen)
- int `device_deleteFile` (IN char \*fileName, IN int fileNameLen)
- int `device_queryFile` (IN char \*directoryName, IN int directoryNameLen, IN char \*fileName, IN int fileNameLen, OUT int \*isExist, OUT BYTE \*timeStamp, IN\_OUT int \*timeStampLen, OUT char \*fileSize, IN\_OUT int \*fileSizeLen)
- int `device_startListenNotifications` ()
- int `device_stopListenNotifications` ()
- int `device_getFirmwareVersion` (OUT char \*firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int `device_getDateTime` (OUT BYTE \*dateTime)
- int `device_getDateTime_Len` (OUT BYTE \*dateTime, IN\_OUT int \*dateTimeLen)
- int `device_controlLED` (byte indexLED, byte control, int intervalOn, int intervalOff)
- int `device_controlLED_ICC` (int controlMode, int interval)
- int `device_controlLED_MSR` (byte control, int intervalOn, int intervalOff)
- int `device_controlBeep` (int index, int frequency, int duration)
- int `device_getDRS` (BYTE \*codeDRS, int \*codeDRSLen)
- int `device_getKeyStatus` (int \*newFormat, BYTE \*status, int \*statusLen)
- int `device_enterStopMode` ()
- int `device_setSleepModeTime` (int time)
- int `device_verifyBackdoorKey` ()
- int `device_selfCheck` ()
- int `device_pingDevice` ()
- int `device_controlUserInterface` (IN BYTE \*values)
- int `device_controlIndicator` (IN int indicator, IN int enable)
- int `device_getCurrentDeviceType` ()

- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_SendDataCommand](#) (IN BYTE \*cmd, IN int cmdLen, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_SendDataCommandITP](#) (IN BYTE \*cmd, IN int cmdLen, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_rebootDevice](#) ()
- int [device\\_updateFirmware](#) (IN BYTE \*firmwareData, IN int firmwareDataLen, IN char \*firmwareName, IN int encryptionType, IN BYTE \*keyBlob, IN int keyBlobLen)
- int [device\\_getDeviceMemoryUsagelInfo](#) (OUT int \*freeHeapSize, OUT int \*notFreedBlockCnt, OUT int \*min-EverFreeHeapSize)
- int [felica\\_authentication](#) (IN BYTE \*key, IN int keyLen)
- int [felica\\_readWithMac](#) (IN int blockCnt, IN BYTE \*blockList, IN int blockListLen, OUT BYTE \*blockData, OUT int \*blockDataLen)
- int [felica\\_writeWithMac](#) (IN BYTE blockNum, IN BYTE \*blockData, IN int blockDataLen)
- int [felica\\_read](#) (IN BYTE \*serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE \*blockList, IN int blockListLen, OUT BYTE \*blockData, OUT int \*blockDataLen)
- int [felica\\_write](#) (IN BYTE \*serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE \*blockList, IN int blockListLen, IN BYTE \*blockData, IN int blockDataLen, OUT BYTE \*statusFlag, OUT int \*statusFlagLen)
- int [felica\\_poll](#) (IN BYTE \*systemCode, IN int systemCodeLen, OUT BYTE \*respData, OUT int \*respDataLen)
- int [felica\\_SendCommand](#) (IN BYTE \*command, IN int commandLen, OUT BYTE \*respData, OUT int \*respDataLen)
- int [felica\\_requestService](#) (IN BYTE \*nodeCode, IN int nodeCodeLen, OUT BYTE \*respData, OUT int \*respDataLen)
- int [config\\_getModelNumber](#) (OUT char \*sNumber)
- int [config\\_getModelNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [device\\_setSystemLanguage](#) (char \*language)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_setCmdTimeOutDuration](#) (IN int millisecond)
- int [config\\_setLEDController](#) (int firmwareControlMSRLED, int firmwareControlICCLEd)
- int [config\\_getLEDController](#) (int \*firmwareControlMSRLED, int \*firmwareControlICCLEd)
- int [config\\_setBeeperController](#) (int firmwareControlBeeper)
- int [config\\_getBeeperController](#) (int \*firmwareControlBeeper)
- int [config\\_setEncryptionControl](#) (int msr, int icc)
- int [config\\_getEncryptionControl](#) (int \*msr, int \*icc)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_enableL100PassThrough](#) (int enableL100PassThrough)
- int [device\\_getL100PassThroughMode](#) ()
- int [device\\_enhancedPassthrough](#) (IN BYTE \*data, IN int dataLen)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_pollForToken](#) (IN int timeout, OUT BYTE \*respData, IN\_OUT int \*respDataLen)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_pollCardReader](#) (OUT BYTE \*status)
- int [device\\_pollCardReader\\_Len](#) (OUT BYTE \*status, IN\_OUT int \*statusLen)
- int [device\\_getSpectrumProKSN](#) (IN int type, OUT BYTE \*KSN)
- int [device\\_getSpectrumProKSN\\_Len](#) (IN int type, OUT BYTE \*KSN, IN\_OUT int \*KSNLen)
- int [device\\_calibrateParameters](#) (BYTE delta)
- int [device\\_getRTCDateTime](#) (IN BYTE \*dateTime, IN\_OUT int \*dateTimeLen)
- int [device\\_setRTCDateTime](#) (IN BYTE \*dateTime, IN int dateTimeLen)
- int [device\\_configureButtons](#) (IN BYTE done, IN BYTE swipe, IN BYTE delay)
- int [device\\_getButtonConfiguration](#) (OUT BYTE \*done, OUT BYTE \*swipe, OUT BYTE \*delay)

- int [device\\_disableBlueLED](#) ()
- int [device\\_enableBlueLED](#) (IN BYTE \*data, IN int dataLen)
- int [device\\_lcdDisplayClear](#) ()
- int [device\\_enableExternalLCDMessages](#) (IN int enableExtLCDMsg)
- int [device\\_enableRFAntenna](#) (IN int enableAntenna)
- int [device\\_turnOffYellowLED](#) ()
- int [device\\_turnOnYellowLED](#) ()
- int [device\\_buzzerOnOff](#) ()
- int [device\\_lcdDisplayLine1Message](#) (IN BYTE \*message, IN int messageLen)
- int [device\\_lcdDisplayLine2Message](#) (IN BYTE \*message, IN int messageLen)
- int [device\\_startRKI](#) (const char \*caPath)
- int [device\\_startQRCodeScan](#) (IN int \_timeout)
- int [device\\_stopQRCodeScan](#) ()
- int [device\\_startTakingPhoto](#) (IN int \_timeout)
- int [device\\_stopTakingPhoto](#) ()
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- void [device\\_toSDCard](#) (int forSDCard)
- int [icc\\_enable](#) (IN int withNotification)
- int [icc\\_disable](#) ()
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [icc\\_exchangeEncryptedAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [icc\\_getAPDU\\_KSN](#) (OUT BYTE \*KSN, IN\_OUT int \*inLen)
- int [icc\\_getFunctionStatus](#) (OUT int \*enabled, OUT int \*withNotification)
- int [icc\\_getICCRReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_getKeyFormatForICCDUKPT](#) (OUT BYTE \*format)
- int [icc\\_getKeyTypeForICCDUKPT](#) (OUT BYTE \*type)
- int [icc\\_setKeyFormatForICCDUKPT](#) (IN BYTE format)
- int [icc\\_setKeyTypeForICCDUKPT](#) (IN BYTE type)
- int [iso8583\\_get1987Handler](#) (OUT DL\_ISO8583\_HANDLER \*ISOHandler)
- int [iso8583\\_get1993Handler](#) (OUT DL\_ISO8583\_HANDLER \*ISOHandler)
- int [iso8583\\_get2003Handler](#) (OUT DL\_ISO8583\_HANDLER \*ISOHandler)
- int [iso8583\\_getField](#) (IN DL\_UINT16 dataField, IN DL\_ISO8583\_HANDLER \*ISOHandler, OUT DL\_ISO8583\_FIELD\_DEF \*field)
- int [iso8583\\_initializeMessage](#) (OUT DL\_ISO8583\_MSG \*ISOMessage)
- int [iso8583\\_getMessageField](#) (IN DL\_UINT16 dataField, IN DL\_ISO8583\_MSG \*ISOMessage, OUT DL\_ISO8583\_MSG\_FIELD \*messageField)
- int [iso8583\\_setMessageField](#) (IN DL\_UINT16 dataField, IN const DL\_UINT8 \*data, OUT DL\_ISO8583\_MSG \*ISOMessage)
- int [iso8583\\_removeMessageField](#) (IN DL\_UINT16 dataField, OUT DL\_ISO8583\_MSG \*ISOMessage)
- int [iso8583\\_packMessage](#) (IN const DL\_ISO8583\_HANDLER \*ISOHandler, IN const DL\_ISO8583\_MSG \*ISOMessage, OUT DL\_UINT8 \*packedData, OUT DL\_UINT16 \*packedDataLength)
- int [iso8583\\_unpackMessage](#) (IN const DL\_ISO8583\_HANDLER \*ISOHandler, IN const DL\_UINT8 \*packedData, IN DL\_UINT16 packedDataLength, OUT DL\_ISO8583\_MSG \*ISOMessage)
- int [iso8583\\_freeMessage](#) (IN DL\_ISO8583\_MSG \*ISOMessage)
- int [iso8583\\_serializeToXML](#) (IN DL\_ISO8583\_HANDLER \*ISOHandler, IN DL\_ISO8583\_MSG \*ISOMessage, OUT BYTE \*serializedMessage, OUT int \*serializedMessageLength)
- int [iso8583\\_deserializeFromXML](#) (IN BYTE \*serializedMessage, IN int serializedMessageLength, OUT DL\_ISO8583\_HANDLER \*ISOHandler, OUT DL\_ISO8583\_MSG \*ISOMessage)
- int [iso8583\\_displayMessage](#) (IN DL\_ISO8583\_HANDLER \*ISOHandler, IN DL\_ISO8583\_MSG \*ISOMessage)

- int [lcd\\_resetInitialState](#) ()
- int [lcd\\_customDisplayMode](#) (IN int enable)
- int [lcd\\_setForeBackColor](#) (IN BYTE \*foreRGB, IN int foreRGBLen, IN BYTE \*backRGB, IN int backRGBLen)
- int [lcd\\_clearDisplay](#) (IN BYTE control)
- int [lcd\\_captureSignature](#) (IN int timeout)
- int [lcd\\_startSlideShow](#) (IN char \*files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touch-Enable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen)
- int [lcd\\_cancelSlideShow](#) (OUT BYTE \*statusCode, IN\_OUT int \*statusCodeLen)
- int [lcd\\_setDisplayImage](#) (IN char \*file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touch-Enable, IN int clearScreen)
- int [lcd\\_setBackgroundImage](#) (IN char \*file, IN int fileLen, IN int enable)
- int [lcd\\_displayText](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char \*displayText, OUT BYTE \*graphicsID)
- int [lcd\\_displayText\\_Len](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int font-Designation, IN int fontID, IN int screenPosition, IN char \*displayText, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)
- int [lcd\\_displayParagraph](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int font-Designation, IN int fontID, IN int displayProperties, IN char \*displayText)
- int [lcd\\_displayButton](#) (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char \*buttonLabel, IN int buttonTextColorR, IN int buttonTextColor-G, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int button-BackgroundColorB, OUT BYTE \*graphicsID)
- int [lcd\\_displayButton\\_Len](#) (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int font-Designation, IN int fontID, IN int displayPosition, IN char \*buttonLabel, IN int buttonTextColorR, IN int button-TextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)
- int [lcd\\_createList](#) (IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \*graphicsID)
- int [lcd\\_createList\\_Len](#) (IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int font-Designation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScroll-Arrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)
- int [lcd\\_addItemToList](#) (IN BYTE \*listGraphicsID, IN char \*itemName, IN char \*itemID, IN int selected)
- int [lcd\\_getSelectedListItem](#) (IN BYTE \*listGraphicsID, OUT char \*itemID)
- int [lcd\\_getSelectedListItem\\_Len](#) (IN BYTE \*listGraphicsID, OUT char \*itemID, IN\_OUT int \*itemIDLen)
- int [lcd\\_clearEventQueue](#) ()
- int [lcd\\_getInputEvent](#) (IN int timeout, OUT int \*dataReceived, OUT BYTE \*eventType, OUT BYTE \*graphics-ID, OUT BYTE \*eventData)
- int [lcd\\_getInputEvent\\_Len](#) (IN int timeout, OUT int \*dataReceived, OUT BYTE \*eventType, IN\_OUT int \*eventTypeLen, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen, OUT BYTE \*eventData, IN\_OUT int \*eventDataLen)
- int [lcd\\_createInputField](#) (IN BYTE \*specs, IN int specsLen, OUT BYTE \*graphicId)
- int [lcd\\_createInputField\\_Len](#) (IN BYTE \*specs, IN int specsLen, OUT BYTE \*graphicId, IN\_OUT int \*graphic-IdLen)
- int [lcd\\_getInputFieldValue](#) (IN BYTE \*graphicId, OUT BYTE \*retData, IN\_OUT int \*retDataLen)
- int [lcd\\_createScreen](#) (IN char \*screenName, IN int screenNameLen, OUT int \*ScreenID)
- int [lcd\\_destroyScreen](#) (IN char \*screenName, IN int screenNameLen)
- int [lcd\\_getActiveScreen](#) (OUT char \*screenName, IN\_OUT int \*screenNameLen)
- int [lcd\\_showScreen](#) (IN char \*screenName, IN int screenNameLen)
- int [lcd\\_getButtonEvent](#) (OUT int \*screenID, OUT int \*objectID, OUT char \*screenName, IN\_OUT int \*screen-NameLen, OUT char \*objectName, IN\_OUT int \*objectNameLen, OUT int \*isLongPress)
- int [lcd\\_addButton](#) (IN char \*screenName, IN int screenNameLen, IN char \*buttonName, IN int buttonName-Len, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char \*label, IN int labelLen, OUT IDTLCDItem \*returnItem)
- int [lcd\\_addEthernet](#) (IN char \*screenName, IN int screenNameLen, IN char \*objectName, IN int objectName-Len, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem \*returnItem)



- `int lcd_addLED (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem, IN BYTE *LED, IN int LEDLen)`
- `int lcd_addText (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE *color, IN int colorLen, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addImage (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addVideo (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)`
- `int lcd_cloneScreen (IN char *screenName, IN int screenNameLen, IN char *cloneName, IN int cloneNameLen, OUT int *cloneID)`
- `int lcd_updateLabel (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN char *label, IN int labelLen)`
- `int lcd_updateColor (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE *color, IN int colorLen)`
- `int lcd_updatePosition (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord)`
- `int lcd_removeItem (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen)`
- `int lcd_storeScreenInfo ()`
- `int lcd_loadScreenInfo ()`
- `int lcd_clearScreenInfo ()`
- `int lcd_getAllScreens (IN_OUT int *screenNumbers, OUT IDTScreenInfo *screenInfo)`
- `int lcd_getAllObjects (IN char *screenName, IN int screenNameLen, IN_OUT int *objectNumbers, OUT IDTObjectInfo *objectInfo)`
- `int lcd_queryScreenbyName (IN char *screenName, IN int screenNameLen, OUT int *result)`
- `int lcd_queryObjectbyName (IN char *objectName, IN int objectNameLen, IN_OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)`
- `int lcd_queryScreenbyID (IN int screenID, OUT int *result, OUT int *screenName, IN_OUT int *screenNameLen)`
- `int lcd_queryObjectbyID (IN int objectID, OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)`
- `int lcd_setBacklight (IN BYTE backlightVal)`
- `int emv_getEMVKernelVersion (OUT char *version)`
- `int emv_getEMVKernelVersion_Len (OUT char *version, IN_OUT int *versionLen)`
- `int emv_getEMVKernelCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)`
- `int emv_getEMVConfigurationCheckValue (OUT BYTE *checkValue, IN_OUT int *checkValueLen)`
- `void emv_setAutoAuthenticateTransaction (IN int authenticate)`
- `void emv_setAutoCompleteTransaction (IN int complete)`
- `int emv_getAutoAuthenticateTransaction ()`
- `int emv_getAutoCompleteTransaction ()`
- `void emv_allowFallback (IN int allow)`
- `void emv_setTransactionParameters (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen)`
- `int emv_startTransaction (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)`
- `int emv_activateTransaction (IN int timeout, IN BYTE *tags, IN int tagsLen, IN int forceOnline)`
- `int emv_authenticateTransaction (IN BYTE *updatedTLV, IN int updatedTLVLen)`
- `int emv_authenticateTransactionWithTimeout (IN int timeout, IN BYTE *updatedTLV, IN int updatedTLVLen)`
- `int emv_completeTransaction (IN int commError, IN BYTE *authCode, IN int authCodeLen, IN BYTE *iad, IN int iadLen, IN BYTE *tlvScripts, IN int tlvScriptsLen, IN BYTE *tlv, IN int tlvLen)`
- `int emv_cancelTransaction ()`
- `int emv_retrieveTransactionResult (IN BYTE *tags, IN int tagsLen, IDTTransactionData *cardData)`
- `int emv_callbackResponseLCD (IN int type, byte selection)`
- `int emv_callbackResponseMSR (IN BYTE *MSR, IN_OUT int MSRLen)`



- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv\\_removeTerminalData](#) ()
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()
- int [emv\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [emv\\_retrieveTerminalID](#) (OUT char \*terminalID)
- int [emv\\_retrieveTerminalID\\_Len](#) (OUT char \*terminalID, IN\_OUT int \*terminalIDLen)
- int [emv\\_setTerminalID](#) (IN char \*terminalID)
- int [emv\\_retrieveCRL](#) (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int [emv\\_setCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeAllCRL](#) ()
- int [msr\\_clearMSRData](#) ()
- int [msr\\_getMSRData](#) (OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [msr\\_cancelMSRSwipe](#) ()
- int [msr\\_startMSRSwipe](#) (IN int \_timeout)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)
- int [msr\\_getKeyFormatForICCDUKPT](#) (OUT BYTE \*format)
- int [msr\\_getKeyTypeForICCDUKPT](#) (OUT BYTE \*type)
- int [msr\\_setKeyFormatForICCDUKPT](#) (IN BYTE format)
- int [msr\\_setKeyTypeForICCDUKPT](#) (IN BYTE type)
- int [msr\\_captureMode](#) (int isBufferMode, int withNotification)
- int [msr\\_flushTrackData](#) ()
- int [msr\\_setExpirationMask](#) (int mask)
- int [msr\\_getExpirationMask](#) (BYTE \*value)
- int [msr\\_setClearPANID](#) (BYTE val)
- int [msr\\_getClearPANID](#) (BYTE \*value)
- int [msr\\_setSwipeForcedEncryptionOption](#) (int track1, int track2, int track3, int track3card0)
- int [msr\\_getSwipeForcedEncryptionOption](#) (BYTE \*option)
- int [msr\\_setSwipeMaskOption](#) (int track1, int track2, int track3)
- int [msr\\_getSwipeMaskOption](#) (BYTE \*option)
- int [msr\\_disable](#) ()
- int [msr\\_getFunctionStatus](#) (int \*enable, int \*isBufferMode, int \*withNotification)
- int [pin\\_getPIN](#) (IN int mode, IN int PANSource, IN char \*iccPAN, IN int iccPANLen, IN int startTimeout, IN int entryTimeout, IN char \*language, IN int languageLen)
- int [pin\\_cancelPINEntry](#) ()
- int [pin\\_setKeyValues](#) (int mode)
- int [pin\\_getEncryptedOnlinePIN](#) (IN int keyType, IN int timeout)
- int [pin\\_getPAN](#) (IN int getCSC, IN int timeout)
- int [pin\\_promptCreditDebit](#) (IN char \*currencySymbol, IN int currencySymbolLen, IN char \*displayAmount, IN int displayAmountLen, IN int timeout, OUT BYTE \*retData, IN\_OUT int \*retDataLen)
- int [pin\\_getEncryptedPIN](#) (int keyType, char \*PAN, int PANLen, char \*message, int messageLen, int timeout)
- int [pin\\_promptForKeyInput](#) (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int [pin\\_promptForAmountInput](#) (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int [pin\\_getFunctionKey](#) (int timeout)

- int [pin\\_sendBeep](#) (int frequency, int duration)
- int [pin\\_capturePin](#) (IN int timeout, IN int type, IN char \*PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char \*message, IN int messageLen)
- int [pin\\_capturePinExt](#) (IN int type, IN char \*PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char \*message, IN int messageLen, IN char \*verify, IN int verifyLen)
- int [pin\\_promptForNumericKeyWithSwipe](#) (IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char \*line1, IN int line1Len, IN char \*line2, IN int line2Len, BYTE \*signature, IN int signatureLen)
- int [pin\\_promptForNumericKey](#) (IN int timeout, IN int maskInput, IN int minLen, IN int maxLen, IN char \*message, IN int messageLen, BYTE \*signature, IN int signatureLen)
- int [pin\\_inputFromPrompt](#) (BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char \*lang, BYTE promptID, char \*defaultResponse, int defaultResponseLen, int timeout)
- int [pin\\_promptForAmount](#) (IN int timeout, IN int minLen, IN int maxLen, IN char \*message, IN int messageLen, BYTE \*signature, IN int signatureLen)
- int [pin\\_getPanEntry](#) (IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly)
- int [lcd\\_savePrompt](#) (int promptNumber, char \*prompt, int promptLen)
- int [lcd\\_displayPrompt](#) (int promptNumber, int lineNumber)
- int [lcd\\_displayMessage](#) (int lineNumber, char \*message, int messageLen)
- int [lcd\\_enableBacklight](#) (int enable)
- int [lcd\\_getBacklightStatus](#) (int \*enabled)
- int [ws\\_requestCSR](#) (OUT RequestCSR \*csr)
- int [ws\\_loadSSLCert](#) (IN char \*name, IN int nameLen, IN char \*dataDER, IN int dataDERLen)
- int [ws\\_revokeSSLCert](#) (IN char \*name, IN int nameLen)
- int [ws\\_deleteSSLCert](#) (IN char \*name, IN int nameLen)
- int [ws\\_getCertChainType](#) (OUT int \*type)
- int [ws\\_updateRootCertificate](#) (IN char \*name, IN int nameLen, IN char \*dataDER, IN int dataDERLen, IN char \*signature, IN int signatureLen)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()
- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- int [ctls\\_displayOnlineAuthResult](#) (IN int statusCode, IN BYTE \*TLV, IN int TLVLen)
- void [parsePINBlockData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTPINData \*cardData)
- void [parsePINData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTPINData \*cardData)

## 12.2.1 Detailed Description

Windows C++ API. Windows C++ Global API methods.

## 12.2.2 Macro Definition Documentation

### 12.2.2.1 #define IN

INPUT parameter.

### 12.2.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

### 12.2.2.3 #define OUT

OUTPUT parameter.

## 12.2.3 Typedef Documentation

### 12.2.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

### 12.2.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback,

### 12.2.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

### 12.2.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

### 12.2.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

### 12.2.3.6 typedef void(\* pFW\_callback)(int, int, int, int, int)

Define the FW callback function to get the status of the firmware update

It should be registered using the device\_registerFWCallBk,

**12.2.3.7 typedef void(\* pLCD\_callback)(int, IDTLCDItem \*)**

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd\_registerCallBk,

**12.2.3.8 typedef void(\* pLog\_callback)(BYTE, char \*)**

Define the log callback function to receive log messages.

**12.2.3.9 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.2.3.10 typedef void(\* pMSR\_callback)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.2.3.11 typedef void(\* pMSR\_callbackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callback

**12.2.3.12 typedef void(\* pPIN\_callback)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.2.3.13 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

**12.2.3.14 typedef void(\* pRKI\_callback)(int, char \*)**

Define the RKI callback function to get the status of the RKI

It should be registered using the device\_registerRKICallBk,

**12.2.3.15 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

### 12.2.3.16 typedef void(\* v4Comm\_callback)(BYTE, BYTE, BYTE \*, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

## 12.2.4 Function Documentation

### 12.2.4.1 int config\_getBeeperController ( int \* *firmwareControlBeeper* )

Get the Beeper Controller Status - AUGUSTA Set the Beeper controlled Status by software or firmware

#### Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

#### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

### 12.2.4.2 int config\_getEncryptionControl ( int \* *msr*, int \* *icc* )

Get Encryption Control - AUGUSTA

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

#### Parameters

<i>msr</i>	<ul style="list-style-type: none"> <li>• 1: enabled MSR with Encryption,</li> <li>• 0: disabled MSR with Encryption,</li> </ul>
<i>icc</i>	<ul style="list-style-type: none"> <li>• 1: enabled ICC with Encryption,</li> <li>• 0: disabled ICC with Encryption,</li> </ul>

#### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

### 12.2.4.3 int config\_getLEDController ( int \* *firmwareControlMSRLED*, int \* *firmwareControlICCLED* )

Get the LED Controller Status - AUGUSTA Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

## Parameters

<i>firmwareControl-MSRLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the MSR LED</li> <li>• 0: software control the MSR LED</li> </ul>
<i>firmwareControl-ICCLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the ICC LED</li> <li>• 0: software control the ICC LED</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.4 int config\_getModelNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getModelNumber\\_Len\(OUT char\\* sNumber, IN\\_OUT int \\*sNumberLen\)](#)

Polls device for Model Number

## Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.5 int config\_getModelNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Model Number

## Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.6 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len\(OUT char\\* sNumber, IN\\_OUT int \\*sNumberLen\)](#)

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.7 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.2.4.8 `int config_setBeeperController ( int firmwareControlBeeper )`

Set the Beeper Controller - AUGUSTA Set the Beeper controlled by software or firmware

## Parameters

<i>firmwareControl-Beeper</i>	1 means firmware control the beeper, 0 means software control beeper.
-------------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.2.4.9 `int config_setCmdTimeOutDuration ( IN int millisecond )`

Set the timeout duration for regular commands The new timeout value will affect all the functions actually send (sync) commands that doesn't need to wait for a callback function, such as `device_getFirmwareVersion()`, `device_pingDevice()`, `device_SendDataCommandNEO()`, `device_enablePassThrough()`, `device_setBurstMode()`, `device_setPollMode()`, `device_updateFirmware()` ... etc.

## Parameters

<i>millisecond</i>	timeout value in milliseconds
--------------------	-------------------------------

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

#### 12.2.4.10 `int config_setEncryptionControl ( int msr, int icc )`

Set Encryption Control - AUGUSTA

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,



## Parameters

<i>msr</i>	<ul style="list-style-type: none"> <li>• 1: enable MSR with Encryption,</li> <li>• 0: disable MSR with Encryption,</li> </ul>
<i>icc</i>	<ul style="list-style-type: none"> <li>• 1: enable ICC with Encryption,</li> <li>• 0: disable ICC with Encryption,</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.11 int config\_setLEDController ( int *firmwareControlMSRLED*, int *firmwareControlICCLED* )

Set the LED Controller - AUGUSTA Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

## Parameters

<i>firmwareControl-MSRLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the MSR LED</li> <li>• 0: software control the MSR LED</li> </ul>
<i>firmwareControl-ICCLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the ICC LED</li> <li>• 0: software control the ICC LED</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.12 int ctls\_activateTransaction ( IN const int *\_timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1, 2, 3, 4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

**12.2.4.13 int ctls\_cancelTransaction ( )**

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.14 int ctls\_displayOnlineAuthResult ( IN int *statusCode*, IN BYTE \* *TLV*, IN int *TLVLen* )

Display Online Authorization Result Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

## Parameters

<i>statusCode</i>	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
<i>TLV</i>	Optional TLV for AOSA
<i>TLVLen</i>	TLV Length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.15 `int ctls_getAllConfigurationGroups ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

## Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.16 `int ctls_getConfigurationGroup ( IN int group, OUT BYTE * tlv, OUT int * tlvLen )`

Get Configuration Group

Retrieves the Configuration for the specified Group.

## Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.17 `int ctls_removeAllApplicationData ( )`

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.18 `int ctls_removeAllCAPK ( )`

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.19 int ctls\_removeApplicationData ( IN BYTE \* *AID*, IN int *AIDLen* )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.20 int ctls\_removeCAPK ( IN BYTE \* *capk*, IN int *capkLen* )**

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.21 int ctls\_removeConfigurationGroup ( int *group* )**

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

**Parameters**

<i>group</i>	Configuration Group
--------------	---------------------

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

**12.2.4.22 int ctls\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )**

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

**Parameters**

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.23 int ctls\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

##### Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.24 int ctls\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

##### Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>

<i>keyLen</i>	the length of key data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.25 int ctls\_retrieveCAPKList ( OUT BYTE \* keys, IN\_OUT int \* keyLen )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keyLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.26 int ctls\_retrieveTerminalData ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.27 int ctls\_setApplicationData ( IN BYTE \* tlv, IN int tlvLen )

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

**Parameters**

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.28 int ctls\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure



## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.29 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4 or DFEE2D). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.30 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.2.4.31 `int ctls_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F26040000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1, 2, 3, 4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal

- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.2.4.32 int device\_activateTransaction ( IN const int \_timeout, IN BYTE \* tags, IN int tagsLen )

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

##### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of device\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101

9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)

- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1, 2, 3, 4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.2.4.33 int device\_buzzerOnOff ( )

Use this function to make the buzzer beep once

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.34 int device\_calibrateParameters ( BYTE *delta* )

Calibrate reference parameters

##### Parameters

<i>delta</i>	Delta value (0x02 standard default value)
--------------	---

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.35 int device\_cancelTransaction ( )

##### Cancel Transaction

Cancels the currently executing transaction.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.36 int device\_cancelTransactionSilent ( int *enable* )

##### Cancel Transaction Silent

Cancel transaction with or without showing the LCD message

##### Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

##### Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

#### 12.2.4.37 int device\_close ( )

Close the device

##### Returns

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.2.4.38 int device\_configureButtons ( IN BYTE *done*, IN BYTE *swipe*, IN BYTE *delay* )

Configures the buttons on the ViVOpay Vendi reader

##### Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value (<= 30) in seconds

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.39 int device\_controlBeep ( int *index*, int *frequency*, int *duration* )

Control Beep - AUGUSTA

Controls the Beeper

**Parameters**

<i>index</i>	For Augusta, must be set to 1 (only one beeper)
<i>frequency</i>	Frequency, range 1000-20000 (suggest minimum 3000)
<i>duration</i>	Duration, in milliseconds (range 1 - 65525)

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.40 int device\_controlIndicator ( IN int *indicator*, IN int *enable* )****Control Indicators**

Control the reader. If connected, returns success. Otherwise, returns timeout.

**Parameters**

<i>indicator</i>	description as follows: <ul style="list-style-type: none"><li>• 00h: ICC LED</li><li>• 01h: Blue MSR</li><li>• 02h: Red MSR</li><li>• 03h: Green MSR</li></ul>
<i>enable</i>	TRUE = ON, FALSE = OFF

**Returns**

success or error code. Values can be parsed with device\_getResponseCodeString

**See Also**

ErrorCode

**12.2.4.41 int device\_controlLED ( byte *indexLED*, byte *control*, int *intervalOn*, int *intervalOff* )****Control MSR LED - AUGUSTA**

Controls the LED for the MSR

## Parameters

<i>indexLED</i>	For Augusta, must be set to 1 (MSR LED)
<i>control</i>	LED Status: <ul style="list-style-type: none"><li>• 00: OFF</li><li>• 01: RED Solid</li><li>• 02: RED Blink</li><li>• 11: GREEN Solid</li><li>• 12: GREEN Blink</li><li>• 21: BLUE Solid</li><li>• 22: BLUE Blink</li></ul>
<i>intervalOn</i>	Blink interval ON, in ms (Range 200 - 2000)
<i>intervalOff</i>	Blink interval OFF, in ms (Range 200 - 2000)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

intervalOn = 500, int intervalOff = 500

#### 12.2.4.42 int device\_controlLED\_ICC ( int *controlMode*, int *interval* )

Control ICC LED - AUGUSTA

Controls the LED for the ICC card slot

## Parameters

<i>controlMode</i>	0 = off, 1 = solid, 2 = blink
<i>interval</i>	Blink interval, in ms (500 = 500 ms)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.43 int device\_controlLED\_MSR ( byte *control*, int *intervalOn*, int *intervalOff* )

Control the MSR LED - AUGUSTA

Controls the MSR / ICC LED This API not recommended to control ICC LED

## Parameters

<i>control</i>	<ul style="list-style-type: none"><li>• 0x00 = off,</li><li>• 0x01 = RED Solid,</li><li>• 0x02 = RED Blink,</li><li>• 0x11 = GREEN Solid,</li><li>• 0x12 = GREEN Blink,</li><li>• 0x21 = BLUE Solid,</li><li>• 0x22 = BLUE Blink,</li></ul>
<i>intervalOn</i>	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>intervalOff</i>	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

int intervalOn = 500, int intervalOff = 500)

#### 12.2.4.44 int device\_controlUserInterface ( IN BYTE \* values )

Control User Interface - NEO only

Controls the User Interface: Display, Beep, LED



## Parameters

<i>values</i>	<p>Four bytes to control the user interface Byte[0] = LCD Message Messages 00-07 are normally controlled by the reader.</p> <ul style="list-style-type: none"> <li>• 00h: Idle Message (Welcome)</li> <li>• 01h: Present card (Please Present Card)</li> <li>• 02h: Time Out or Transaction cancel (No Card)</li> <li>• 03h: Transaction between reader and card is in the middle (Processing...)</li> <li>• 04h: Transaction Pass (Thank You)</li> <li>• 05h: Transaction Fail (Fail)</li> <li>• 06h: Amount (Amount \$ 0.00 Tap Card)</li> <li>• 07h: Balance or Offline Available funds (Balance \$ 0.00) Messages 08-0B are controlled by the terminal</li> <li>• 08h: Insert or Swipe card (Use Chip &amp; PIN)</li> <li>• 09h: Try Again(Tap Again)</li> <li>• 0Ah: Tells the customer to present only one card (Present 1 card only)</li> <li>• 0Bh: Tells the customer to wait for authentication/authorization (Wait)</li> <li>• FFh: indicates the command is setting the LED/Buzzer only. Byte[1] = Beep Indicator</li> <li>• 00h: No beep</li> <li>• 01h: Single beep</li> <li>• 02h: Double beep</li> <li>• 03h: Three short beeps</li> <li>• 04h: Four short beeps</li> <li>• 05h: One long beep of 200 ms</li> <li>• 06h: One long beep of 400 ms</li> <li>• 07h: One long beep of 600 ms</li> <li>• 08h: One long beep of 800 ms Byte[2] = LED Number</li> <li>• 00h: LED 0 (Power LED) 01h: LED 1</li> <li>• 02h: LED 2</li> <li>• 03h: LED 3</li> <li>• FFh: All LEDs Byte[3] = LED Status</li> <li>• 00h: LED Off</li> <li>• 01h: LED On</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.45 int device\_createDirectory ( IN char \* *directoryName*, IN int *directoryNameLen* )

Create Directory This command adds a subdirectory to the indicated path.

## Parameters

<i>directoryName</i>	Directory Name. The data for this command is ASCII string with the complete path and directory name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>directoryName-Len</i>	Directory Name Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.46 int device\_deleteDirectory ( IN char \* *dirName*, IN int *dirNameLen* )

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

## Parameters

<i>dirName</i>	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
<i>dirNameLen</i>	Directory Name Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.47 int device\_deleteFile ( IN char \* *fileName*, IN int *fileNameLen* )

Delete File This command deletes a file or group of files.

## Parameters

<i>filename</i>	Complete path and file name of the file you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>filenameLen</i>	File Name Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.48 int device\_disableBlueLED ( )

Stops the blue LEDs on the ViVOpay Vendi reader from flashing in left to right sequence and turns the LEDs off, and contactless function is disable at the same time

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.49 int device\_enableBlueLED ( IN BYTE \* *data*, IN int *dataLen* )

Use this function to control the blue LED behavior on the Vendi reader

## Parameters

<i>data</i>	Sequence data Byte 0 (Cycle): 0 = Cycle once, 1 = Repeat Byte 1 (LEDs): LED State Bitmap Byte 2-3 (Duration): Given in multiples of 10 millisecond Byte 4 (LED): LED State Bitmap Byte 5-6 (Duration): Given in multiples of 10 millisecond Byte 7-24 (Additional LED/Durations): Define up to 8 LED and duration pairs
-------------	---

LED State Bitmap: Bit 8: Left blue LED, 0 = off, 1 = on Bit 7: Center Blue LED, 0 = off, 1 = on Bit 6: Right Blue LED 0 = off, 1 = on Bit 5: Yellow LED, 0 = off, 1 = on Bit 4: Reserved for future use Bit 3: Reserved for future use Bit 2: Reserved for future use Bit 1: Reserved for future use

## Parameters

<i>dataLen</i>	Length of the sequence data: 0 or 4 to 25 bytes
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.50 int device\_enableExternalLCDMessages ( IN int *enableExtLCDMsg* )

Enable or disable the external LCD message It will turn off the external LCD messages including EMV transactions. (For the users who only need MSR and/or CTLS transactions.) The function only works for VP5300

## Parameters

<i>enableExtLCD- Msg</i>	1 = ON, 0 = OFF
------------------------------	-----------------

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

ErrorCode

#### 12.2.4.51 int device\_enableL100PassThrough ( int *enableL100PassThrough* )

Enable L100 Pass Through

Enables Pass Through Mode for direct communication to L100 hook up to NEO II device

## Parameters

<i>enableL100- PassThrough</i>	1 = pass through ON, 0 = pass through OFF
------------------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.52 int device\_enablePassThrough ( int *enablePassThrough* )

Start Remote Key Injection - AUGUSTA

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

#### Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString Enable Pass Through - NEO

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

## Parameters

<i>enablePass-Through</i>	1 = pass through ON, 0 = pass through OFF
---------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.53 int device\_enableRFAntenna ( IN int *enableAntenna* )

Enable or disable the RF Antenna

## Parameters

<i>enableAntenna</i>	1 = ON, 0 = OFF
----------------------	-----------------

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

ErrorCode

#### 12.2.4.54 int device\_enhancedPassthrough ( IN BYTE \* *data*, IN int *dataLen* )

Enables pass through mode for ICC. Required when direct ICC commands are required (power on/off ICC, exchange APDU)

## Parameters

<i>data</i>	The data includes Poll Timeout, Flags, Contact Interface to Use, Beep Indicator, LED Status, and Display Strings.
<i>dataLen</i>	length of data

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

ErrorCode

#### 12.2.4.55 int device\_enterStopMode ( )

Enter Stop Mode

Set device enter to stop mode. In stop mode, LCD display and backlight is off. Stop mode reduces power consumption to the lowest possible level. A unit in stop mode can only be woken up by a physical key press.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.56 int device\_getButtonConfiguration ( OUT BYTE \* *done*, OUT BYTE \* *swipe*, OUT BYTE \* *delay* )

Reads the button configuration from the ViVOpay Vendi reader

## Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value in seconds

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.57 int device\_getCurrentDeviceType ( )

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.2.4.58 int device\_getDateTime ( OUT BYTE \* *dateTime* )

DEPRECATED : please use [device\\_getDateTime\\_Len\(OUT BYTE\\* \*dateTime\*, IN\\_OUT int \\*\*dateTimeLen\*\)](#)

Polls device for Date and Time

## Parameters

<i>dateTime</i>	Response returned of Date and Time; needs to have at least 6 bytes of memory
-----------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.59 int device\_getDateTime\_Len ( OUT BYTE \* *dateTime*, IN\_OUT int \* *dateTimeLen* )

Polls device for Date and Time

## Parameters

<i>dateTime</i>	Response returned of Date and Time
<i>dateTime</i>	Length of Date and Time

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.60 int device\_getDeviceMemoryUsageInfo ( OUT int \* *freeHeapSize*, OUT int \* *notFreedBlockCnt*, OUT int \* *minEverFreeHeapSize* )

Get Device Memory Usage Information

## Parameters

<i>freeHeapSize</i>	Free Heap Size: Available heap size
<i>notFreedBlock-Cnt</i>	Memory Not Freed Block Count: Memory in use block count
<i>minEverFree-HeapSize</i>	Minimum Ever Free Heap Size: The lowest ever available heap size

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.61 int device\_getDriveFreeSpace ( OUT int \* *free*, OUT int \* *used* )

Drive Free Space This command returns the free and used disk space on the flash drive.

**Parameters**

<i>free</i>	Free bytes available on device
<i>used</i>	Used bytes on on device

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.62 int device\_getDRS ( BYTE \* *codeDRS*, int \* *codeDRSLen* )

Get DRS Status - AUGUSTA TTK

Gets the status of DRS(Destructive Reset).

**Parameters**

<i>codeDRS</i>	the data format is [DRS SourceBlk Number] [SourceBlk1] ... [SourceBlkN] [DRS SourceBlk Number] is 2 bytes, format is NumL NumH. It is Number of [SourceBlkX] [SourceBlkX] is n bytes, Format is [SourceID] [SourceLen] [SourceData] [SourceID] is 1 byte [SourceLen] is 1 byte, it is length of [SourceData]
----------------	--

[SourceID] [SourceLen] [SourceData] 00 1 01 - Application Error 01 1 01 - Application Error 02 1 0x01 - EMV L2 Configuration Check Value Error 0x02 - Future Key Check Value Error 10 1 01 - Battery Error 11 1 Bit 0 - Tamper Switch 1 (0-No, 1-Error) Bit 1 - Tamper Switch 2 (0-No, 1-Error) Bit 2 - Tamper Switch 3 (0-No, 1-Error) Bit 3 - Tamper Switch 4 (0-No, 1-Error) Bit 4 - Tamper Switch 5 (0-No, 1-Error) Bit 5 - Tamper Switch 6 (0-No, 1-Error)

12 1 01 - TemperatureHigh or Low 13 1 01 - Voltage High or Low 1F 4 Reg31~24bits, Reg23~16bits, Reg15~8bits, Reg7~0bits

**Parameters**

<i>codeDRSLen</i>	the length of codeDRS
-------------------	-----------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#) Note: Only support TTK devices

#### 12.2.4.63 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len\(\)](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version



## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.64 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.65 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcrition* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 01: " Incorrect Header Tag";
- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";

- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVO-Card3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

**12.2.4.66** `int device_getKeyStatus ( int * newFormat, BYTE * status, int * statusLen )`

#### Get Key Status

Gets the status of loaded keys

#### Parameters

---

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len\_L Len\_H, is KeyStatusBlock Number [KeyStatusBlockX] is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

#### Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

#### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.67 int device\_getL100PassThroughMode ( )

Get L100 Pass Through Mode

Get current Pass Through Mode for direct communication to L100 hook up to NEO II device

#### Returns

RETURN\_CODE: return 1 if L100 Pass Through Mode is TRUE, 0 if L100 Pass Through Mode is FALSE

#### 12.2.4.68 int device\_getMerchantRecord ( IN int index, OUT BYTE \* record )

DEPRECATED : please use [device\\_getMerchantRecord\\_Len\(IN int index, OUT BYTE \\* record, IN\\_OUT int \\*recordLen\)](#)

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

12.2.4.69 `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

## Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

12.2.4.70 `void device_getResponseCodeString ( IN int returnCode, OUT char * despcriton )`

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";

- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";

- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";

- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";

- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported, ";
- 0X2D03: "Card Not Supported, wants CRC";



- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";

- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount, Other Amount, Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE\_OPEN\_FAILED";
- 0X1003: "FILE OPERATION\_FAILED";
- 0X2001: "MEMORY\_NOT\_ENOUGH";
- 0X3002: "SMARTCARD\_FAIL";
- 0X3003: "SMARTCARD\_INIT\_FAILED";
- 0X3004: "FALLBACK\_SITUATION";
- 0X3005: "SMARTCARD\_ABSENT";
- 0X3006: "SMARTCARD\_TIMEOUT";
- 0X3012: "EMV\_RESULT\_CODE\_MSR\_CARD\_ERROR\_FALLBACK";

- 0X5001: "EMV\_PARSING\_TAGS\_FAILED";
- 0X5002: "EMV\_DUPLICATE\_CARD\_DATA\_ELEMENT";
- 0X5003: "EMV\_DATA\_FORMAT\_INCORRECT";
- 0X5004: "EMV\_NO\_TERM\_APP";
- 0X5005: "EMV\_NO\_MATCHING\_APP";
- 0X5006: "EMV\_MISSING\_MANDATORY\_OBJECT";
- 0X5007: "EMV\_APP\_SELECTION\_RETRY";
- 0X5008: "EMV\_GET\_AMOUNT\_ERROR";
- 0X5009: "EMV\_CARD\_REJECTED";
- 0X5010: "EMV\_AIP\_NOT\_RECEIVED";
- 0X5011: "EMV\_AFL\_NOT\_RECEIVED";
- 0X5012: "EMV\_AFL\_LEN\_OUT\_OF\_RANGE";
- 0X5013: "EMV\_SFI\_OUT\_OF\_RANGE";
- 0X5014: "EMV\_AFL\_INCORRECT";
- 0X5015: "EMV\_EXP\_DATE\_INCORRECT";
- 0X5016: "EMV\_EFF\_DATE\_INCORRECT";
- 0X5017: "EMV\_ISS\_COD\_TBL\_OUT\_OF\_RANGE";
- 0X5018: "EMV\_CRYPTOGAM\_TYPE\_INCORRECT";
- 0X5019: "EMV\_PSE\_NOT\_SUPPORTED\_BY\_CARD";
- 0X5020: "EMV\_USER\_SELECTED\_LANGUAGE";
- 0X5021: "EMV\_SERVICE\_NOT\_ALLOWED";
- 0X5022: "EMV\_NO\_TAG\_FOUND";
- 0X5023: "EMV\_CARD\_BLOCKED";
- 0X5024: "EMV\_LEN\_INCORRECT";
- 0X5025: "CARD\_COM\_ERROR";
- 0X5026: "EMV\_TSC\_NOT\_INCREASED";
- 0X5027: "EMV\_HASH\_INCORRECT";
- 0X5028: "EMV\_NO\_ARC";
- 0X5029: "EMV\_INVALID\_ARC";
- 0X5030: "EMV\_NO\_ONLINE\_COMM";
- 0X5031: "TRAN\_TYPE\_INCORRECT";
- 0X5032: "EMV\_APP\_NO\_SUPPORT";
- 0X5033: "EMV\_APP\_NOT\_SELECT";
- 0X5034: "EMV\_LANG\_NOT\_SELECT";
- 0X5035: "EMV\_NO\_TERM\_DATA";
- 0X5039: "EMV\_PIN\_ENTRY\_TIMEOUT";

- 0X6001: "CVM\_TYPE\_UNKNOWN";
- 0X6002: "CVM\_AIP\_NOT\_SUPPORTED";
- 0X6003: "CVM\_TAG\_8E\_MISSING";
- 0X6004: "CVM\_TAG\_8E\_FORMAT\_ERROR";
- 0X6005: "CVM\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6006: "CVM\_COND\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6007: "NO\_MORE\_CVM";
- 0X6008: "PIN\_BYPASSED\_BEFORE";
- 0X7001: "PK\_BUFFER\_SIZE\_TOO\_BIG";
- 0X7002: "PK\_FILE\_WRITE\_ERROR";
- 0X7003: "PK\_HASH\_ERROR";
- 0X8001: "NO\_CARD\_HOLDER\_CONFIRMATION";
- 0X8002: "GET\_ONLINE\_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";

- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBECA: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBCEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";

- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

#### 12.2.4.71 int device\_getRTCDateTime ( IN BYTE \* *dateTime*, IN\_OUT int \* *dateTimeLen* )

get RTC date and time of the device

##### Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

##### Returns

success or error code. Values can be parsed with device\_getResponseCodeString

##### See Also

ErrorCode

#### 12.2.4.72 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

##### Returns

SDK wait time in seconds

12.2.4.73 `int device_getSpectrumProKSN ( IN int type, OUT BYTE * KSN )`

DEPRECATED : please use [device\\_getSpectrumProKSN\\_Len](#)(IN int *type*, OUT BYTE \* *KSN*, IN\_OUT int \**KSN-Len*)

Get DUKPT KSN

Returns the KSN for the provided key index

**Parameters**

<i>type</i>	Key type: <ul style="list-style-type: none"><li>• 0: Key Encryption Key (Master Key or KEK)</li><li>• 2: Data Encryption Key (DEK)</li><li>• 5: MAC Key (MAK)</li><li>• 10: RKL Key Encryption Key (REK)</li><li>• 20: HSM DUKPT Key</li></ul>
<i>KSN</i>	Key Serial Number; needs to have at least 10 bytes of memory

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.74 `int device_getSpectrumProKSN_Len ( IN int type, OUT BYTE * KSN, IN_OUT int * KSNLen )`

Get DUKPT KSN

Returns the KSN for the provided key index

**Parameters**

<i>type</i>	Key type: <ul style="list-style-type: none"><li>• 0: Key Encryption Key (Master Key or KEK)</li><li>• 2: Data Encryption Key (DEK)</li><li>• 5: MAC Key (MAK)</li><li>• 10: RKL Key Encryption Key (REK)</li><li>• 20: HSM DUKPT Key</li></ul>
-------------	--

<i>KSN</i>	Key Serial Number
<i>KSNLen</i>	Length of KSN

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.2.4.75 int device\_getThreadStackSize ( )****Get Thread Stack Size**

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.2.4.76 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) or [device\\_getResponseCodeString\(\)](#)

**12.2.4.77 int device\_isAttached ( int deviceType )**

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

**Parameters**

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

**Returns**

1 if the device is attached, or 0 if the device is not attached

**12.2.4.78 int device\_isConnected ( )**

Check the device connected status

**Returns**

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

**12.2.4.79 int device\_lcdDisplayClear ( )**

Use this function to clear the LCD display

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



12.2.4.80 int device\_lcdDisplayLine1Message ( IN BYTE \* *message*, IN int *messageLen* )

Use this function to display text on the LCD display. On the Vendi reader the LCD is a 2-line character display.

## Parameters

<i>message</i>	Valid messages for the first line of text are between 1 and 16 printable characters long. If the text message is greater than 16 bytes but not more than 32 bytes, byte 17 and onward are displayed as a second row of text. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 32 bytes

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.81 int device\_lcdDisplayLine2Message ( IN BYTE \* *message*, IN int *messageLen* )

Use this function to display the message on line 2 of the LCD display. On the Vendi reader the LCD is a 2-line character display.

## Parameters

<i>message</i>	Valid messages are between 1 and 16 printable characters long. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 16 bytes

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.82 int device\_listDirectory ( IN char \* *directoryName*, IN int *directoryNameLen*, IN int *recursive*, IN int *onSD*, OUT char \* *directory*, IN\_OUT int \* *directoryLen* )

List Directory This command retrieves a directory listing of user accessible files from the reader.

## Parameters

<i>directoryName</i>	Directory Name. If null, root directory is listed
<i>directoryNameLen</i>	Directory Name Length. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	TRUE = use flash storage The returned directory information The returned directory information length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.83 int device\_pingDevice ( )

Ping Device - NEO only

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.84 int device\_pollCardReader ( OUT BYTE \* *status* )

DEPRECATED : please use [device\\_pollCardReader\\_Len](#)(OUT BYTE \* status, IN\_OUT int \*statusLen)

Poll Card Reader

Provides information about the state of the Card Reader

## Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> <li>• Bit 0: Device Manufacturing CA data valid</li> <li>• Bit 1: Device Manufacturing Secure data valid</li> <li>• Bit 2: HOST_CR_MASTER_DUKPT Key valid</li> <li>• Bit 3: HOST_CR_MAC Keys valid (Authenticated)</li> <li>• Bit 4: RFU</li> <li>• Bit 5: RFU</li> <li>• Bit 6: DATA_DUKPT Key Valid</li> <li>• Bit 7: Key is initialized (MFK and RSA Key pairs)</li> </ul>
---------------	--

## Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU
- Bit 2: CR\_PINPAD\_MASTER\_DUKPT Key valid
- Bit 3: CR\_PINPAD\_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

## Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

## Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

## Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present

- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.85 `int device_pollCardReader_Len ( OUT BYTE * status, IN_OUT int * statusLen )`

Poll Card Reader

Provides information about the state of the Card Reader

Parameters

<i>status</i>	Six bytes indicating card reader information Byte 0: <ul style="list-style-type: none"><li>• Bit 0: Device Manufacturing CA data valid</li><li>• Bit 1: Device Manufacturing Secure data valid</li><li>• Bit 2: HOST_CR_MASTER_DUKPT Key valid</li><li>• Bit 3: HOST_CR_MAC Keys valid (Authenticated)</li><li>• Bit 4: RFU</li><li>• Bit 5: RFU</li><li>• Bit 6: DATA_DUKPT Key Valid</li><li>• Bit 7: Key is initialized (MFK and RSA Key pairs)</li></ul>
---------------	--

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU

- Bit 2: CR\_PINPAD\_MASTER\_DUKPT Key valid
- Bit 3: CR\_PINPAD\_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present
- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

## Parameters

<i>statusLen</i>	Length of status
------------------	------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.86 `int device_pollForToken ( IN int timeout, OUT BYTE * respData, IN_OUT int * respDataLen )`

Poll for Token

Polls for a PICC

## Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.87 `int device_queryFile ( IN char * directoryName, IN int directoryNameLen, IN char * fileName, IN int fileNameLen, OUT int * isExist, OUT BYTE * timeStamp, IN_OUT int * timeStampLen, OUT char * fileSize, IN_OUT int * fileSizeLen )`

Query File This command checks if the specified file exists in NAND Flash..

## Parameters

<i>directoryName</i>	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>directoryNameLen</i>	Directory Name Length.
<i>fileName</i>	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>fileNameLen</i>	File Name Length.
<i>isExist</i>	File exists: 1, File not exists 0.
<i>timeStamp</i>	Latest time stamp of the file. 6 bytes BCD code if the file exists.
<i>timeStampLen</i>	Length of timeStamp. 6 if the file exists, 0 if the file does not exist.
<i>fileSize</i>	Zero-terminated ASCII string of the file size.
<i>fileSizeLen</i>	Length of fileSize.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.88 `int device_rebootDevice ( )`

Reboot Device - NGA Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.89 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.2.4.90 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.2.4.91 void device\_registerFWCallBk ( pFW\_callback pFWf )

To register the firmware update callback function to get the firmware update status. (Pass NULL to disable the callback.)

12.2.4.92 void device\_registerRKICallBk ( pRKI\_callback pRKIf )

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

12.2.4.93 int device\_selfCheck ( )

Self check for TTK If Self-Test function Failed, then work into De-activation State. If device work into De-activation State, All Sensitive Data will be erased and it need be fixed in Manufacture.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.94 int device\_SendDataCommand ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to NGA device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.95 int device\_SendDataCommandITP ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to ITP device

Sends a command to the device .



## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of ITP command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.2.4.96** int device\_SendDataCommandNEO ( IN int *cmd*, IN int *subCmd*, IN BYTE \* *data*, IN int *dataLen*, OUT BYTE \* *response*, IN\_OUT int \* *respLen* )

Send a Command to NEO device

Sends a command to the NEO device .

## Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.97** int device\_setBurstMode ( IN BYTE *mode* )

Send Burst Mode - NEO

Sets the burst mode for the device.

## Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

[ErrorCode](#)

**12.2.4.98** int device\_setCancelTransactionMode ( int *mode* )

Set Cancel Transaction Mode

Set the cancel transaction mode to be with or without LCD message

## Parameters

<i>mode</i>	0: With LCD message 1: Without LCD message
-------------	--

## Returns

success or error code. 1: Success, 0: Failed

12.2.4.99 int device\_setConfigPath ( const char \* *path* )

Set the path to the config xml file(s) if any

## Parameters

<i>path</i>	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.100 int device\_setCurrentDevice ( int *deviceType* )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	<p>Device to connect to</p> <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 }; </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

12.2.4.101 `int device_setMerchantRecord ( int index, int enabled, char * merchantID, char * merchantURL )`

Set Merchant Record - NEO Sets the merchant record for ApplePay VAS

**Parameters**

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.102 `int device_setNEO2DevicesConfigs ( IN const char * configs, IN int len )`

Pass the content of the config xml file ("NEO2\_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK. It needs to be called before [device\\_init\(\)](#), otherwise the SDK will try to read the config xml file.

**Parameters**

<i>configs</i>	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
<i>len</i>	The length of the string configs. The maximum length is 5000 bytes.

12.2.4.103 `int device_setPollMode ( IN BYTE mode )`

Set Poll Mode - NEO

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

**Parameters**

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.104 `int device_setRTCDateTime ( IN BYTE * dateTime, IN int dateTimeLen )`

set RTC date and time of the device

**Parameters**

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

**Returns**

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

**See Also**

[ErrorCode](#)

#### 12.2.4.105 void device\_setSDKWaitTime ( int *waitTime* )

Set SDK Wait Time

Set the SDK wait time for transactions

Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

#### 12.2.4.106 int device\_setSleepModeTime ( int *time* )

Set Sleep Mode Timer

Set device enter to sleep mode after the given time. In sleep mode, LCD display and backlight is off. Sleep mode reduces power consumption to the lowest possible level. A unit in Sleep mode can only be woken up by a physical key press.

Parameters

<i>time</i>	Enter sleep time value, in second.
-------------	------------------------------------

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.107 int device\_setSystemLanguage ( char \* *language* )

Set Model Number for the device

Parameters

<i>sNumber</i>	Model Number
----------------	--------------

Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString Set System Language Sets the language for the message displayed in the LCD screen

Parameters

<i>language</i>	2-byte ASCII code, can be "EN" or "JP"
-----------------	--

Returns

success or error code. Values can be parsed with device\_getIDGStatusCodeString

See Also

ErrorCode

#### 12.2.4.108 void device\_setThreadStackSize ( int *threadSize* )

Set Thread Stack Size

Set the stack size setting for newly created threads

#### 12.2.4.109 void device\_setTransactionExponent ( int *exponent* )

Sets the transaction exponent to be used with device\_startTransaction. Default value is 2

## Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

## 12.2.4.110 int device\_startListenNotifications ( )

Start Listen Notifications This function enables Card Status and Front Switch notifications.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.111 int device\_startQRCodeScan ( IN int \_timeout )

Start QR Code Scanning

Enables QR Code scanning, waiting for the QR code.

## Parameters

<i>timeout</i>	QR Code Scan Timeout Value. Between 30 and 65536 seconds.
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

## 12.2.4.112 int device\_startRKI ( const char \* caPath )

Start remote key injection.

## Parameters

<i>caPath</i>	The path to ca-certificates.crt
---------------	---------------------------------

## Returns

success or error code.

## See Also

ErrorCode

## 12.2.4.113 int device\_startTakingPhoto ( IN int \_timeout )

Start Taking Photo

Enables the camera to take a photo.

## Parameters

<i>timeout</i>	Photo taking Timeout Value. Between 30 and 65536 seconds.
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.2.4.114 `int device_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)  • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03)  • SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1, 2, 3, 4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal

- - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.2.4.115 int device\_stopListenNotifications ( )

Stop Listen Notifications This function disables Card Status and Front Switch notifications.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.116 int device\_stopQRCodeScan ( )

Stop QR Code Scanning Cancels QR Code scanning request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.117 int device\_stopTakingPhoto ( )

Stop Taking Photo Cancels Photo Taking request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.118 void device\_toSDCard ( int forSDCard )

To SD Card

Set the destination of the file or directory function

## Parameters

<i>forSDCard</i>	0: for internal memory, 1: for SD card
------------------	--

12.2.4.119 `int device_transferFile ( IN char * fileName, IN int fileNameLen, IN BYTE * file, IN int fileLen )`

Transfer File This command transfers a data file to the reader.

## Parameters

<i>fileName</i>	Filename. The data for this command is a ASCII string with the complete path and file name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/)
<i>fileNameLen</i>	File Name Length.
<i>file</i>	The data file.
<i>fileLen</i>	File Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.120 `int device_turnOffYellowLED ( )`

Use this function to turn off the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.121 `int device_turnOnYellowLED ( )`

Use this function to turn on the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs



## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.122** `int device_updateFirmware ( IN BYTE * firmwareData, IN int firmwareDataLen, IN char * firmwareName, IN int encryptionType, IN BYTE * keyBlob, IN int keyBlobLen )`

Update Firmware - NGA Updates the firmware of the Spectrum Pro K21 HUB or Maxq1050.

## Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. Must be one of the following two strings (with appropriate version information) <ul style="list-style-type: none"> <li>• "SP K21 APP Vx.xx.xxx"</li> <li>• "SP MAX APP Vx.xx.xxx"</li> </ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"> <li>• 0 : Plaintext</li> <li>• 1 : TDES ECB, PKCS#5 padding</li> <li>• 2 : TDES CBC, PKCS#5, IV is all 0</li> </ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of keyBlob

## Returns

RETURN\_CODE: Values can be parsed with [errorCode.getErrorString\(\)](#)

Firmware update status is returned in the callback with the following values: sender = SPECTRUM\_PRO state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with byte[0] = current block, and byte[1] = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

**12.2.4.123** `int device_verifyBackdoorKey ( )`

Verify Backdoor Key to Unlock Security

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

Note: The function is only for TTK devices.

12.2.4.124 `int emv_activateTransaction ( IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable.

## 12.2.4.125 void emv\_allowFallback ( IN int allow )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

## 12.2.4.126 int emv\_authenticateTransaction ( IN BYTE \* updatedTLV, IN int updatedTLVLen )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37</li> </ul>
-------------------	---

<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.2.4.127** `int emv_authenticateTransactionWithTimeout ( IN int timeout, IN BYTE * updatedTLV, IN int updatedTLVLen )`

**Authenticate EMV Transaction Request with Timeout**

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

**Parameters**

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.2.4.128** `int emv_callbackResponseLCD ( IN int type, byte selection )`

**Callback Response LCD Display**

Provides menu selection responses to the kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD`, and `lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU`, `EMV_LCD_DISPLAY_MODE_PROMPT`, or `EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT`

**Parameters**

<i>type</i>	If Cancel key pressed during menu selection, then value is <code>EMV_LCD_DISPLAY_MODE_CANCEL</code> . Otherwise, value can be <code>EMV_LCD_DISPLAY_MODE_MENU</code> , <code>EMV_LCD_DISPLAY_MODE_PROMPT</code> , or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code>
<i>selection</i>	If <code>type = EMV_LCD_DISPLAY_MODE_MENU</code> or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code> , provide the selection ID line number. Otherwise, if <code>type = EMV_LCD_DISPLAY_MODE_PROMPT</code> supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.2.4.129 `int emv_callbackResponseMSR ( IN BYTE * MSR, IN_OUT int MSRLen )`

#### Callback Response MSR Entry

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV\_CALLBACK\_MSR

#### Parameters

<i>MSR</i>	Swiped track data
<i>MSRLen</i>	the length of Swiped track data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.130 `int emv_cancelTransaction ( )`

#### Cancel EMV Transaction

Cancels the currently executing EMV transaction.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.131 `int emv_completeTransaction ( IN int commError, IN BYTE * authCode, IN int authCodeLen, IN BYTE * iad, IN int iadLen, IN BYTE * tlvScripts, IN int tlvScriptsLen, IN BYTE * tlv, IN int tlvLen )`

#### Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv\_authenticateTransaction

The tags will be returned in the callback routine.

#### Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

12.2.4.132 `int emv_getAutoAuthenticateTransaction ( )`

Gets auto authenticate value for EMV transactions.

#### Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

#### 12.2.4.133 int emv\_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

##### Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

#### 12.2.4.134 int emv\_getEMVConfigurationCheckValue ( OUT BYTE \* *checkValue*, IN\_OUT int \* *checkValueLen* )

Get EMV Kernel configuration check value info

##### Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

##### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.135 int emv\_getEMVKernelCheckValue ( OUT BYTE \* *checkValue*, IN\_OUT int \* *checkValueLen* )

Get EMV Kernel check value info

##### Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

##### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.136 int emv\_getEMVKernelVersion ( OUT char \* *version* )

DEPRECATED : please use [emv\\_getEMVKernelVersion\\_Len\(OUT char\\* version, IN\\_OUT int \\*versionLen\)](#)

Polls device for EMV Kernel Version

##### Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.137 int emv\_getEMVKernelVersion\_Len ( OUT char \* *version*, IN\_OUT int \* *versionLen* )

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.138 void emv\_registerCallBk ( pEMV\_callback pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

#### 12.2.4.139 int emv\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.140 int emv\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.141 int emv\_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.142 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.143 int emv\_removeCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

##### Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.144 int emv\_removeCRL ( IN BYTE \* *list*, IN int *lsLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

##### Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.145 int emv\_removeTerminalData ( )

Remove Terminal Data

Removes the Terminal Data

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.146 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

Retrieve AID list

Returns all the AID names installed on the terminal.

##### Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.147 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.



## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.148 `int emv_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

## Retrieve Certificate Authority Public Key

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	the length of key data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.149 `int emv_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

## Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.150 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *lssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.151 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.152 int emv\_retrieveTerminalID ( OUT char \* *terminalID* )

DEPRECATED : please use [emv\\_retrieveTerminalID\\_Len](#)(OUT char\* *terminalID*, IN\_OUT int \**terminalIDLen*)

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.153 int emv\_retrieveTerminalID\_Len ( OUT char \* *terminalID*, IN\_OUT int \* *terminalIDLen* )

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.154 `int emv_retrieveTransactionResult ( IN BYTE * tags, IN int tagsLen, IDTTransactionData * cardData )`

## Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

## Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDT-TransactionData object

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.155 `int emv_setApplicationData ( IN BYTE * name, IN int nameLen, IN BYTE * tlv, IN int tlvLen )`

## Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.156 `int emv_setApplicationDataTLV ( IN BYTE * tlv, IN int tlvLen )`

## Set Application Data by TLV

Sets the Application Data as specified by the TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.157 void emv\_setAutoAuthenticateTransaction ( IN int *authenticate* )

Enables authenticate for EMV transactions. If a `emv_startTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

#### 12.2.4.158 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

#### 12.2.4.159 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.160 int emv\_setCRL ( IN BYTE \* *list*, IN int *IsLen* )

Set Certificate Revocation List

Sets the CRL

Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>IsLen</i>	the length of list data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.161 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <a href="#">device_getResponseCodeString()</a>
--------------------	--

12.2.4.162 int emv\_setTerminalID ( IN char \* *terminalID* )

Sets the terminal ID as printable characters .

Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.163 int emv\_setTerminalMajorConfiguration ( IN int *configuration* )

Sets the terminal major configuration in ICS .

**Parameters**

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.164 void emv\_setTransactionParameters ( IN double *amount*, IN double *amtOther*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

**Parameters**

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

<i>tagsLen</i>	the length of tags
----------------	--------------------

12.2.4.165 int emv\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0-C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029-F36959F37

#### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.2.4.166 int felica\_authentication ( IN BYTE \* *key*, IN int *keyLen* )

FeliCa Authentication Provides a key to be used in a follow up FeliCa Read with MAC (3 blocks max) or Write with MAC (1 block max). This command must be executed before each Read w/MAC or Write w/MAC command

#### Parameters

<i>key</i>	16-byte key used for MAC generation of Read or Write with MAC
<i>keyLen</i>	length of key, must be 16 bytes

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.167 int felica\_poll ( IN BYTE \* *systemCode*, IN int *systemCodeLen*, OUT BYTE \* *respData*, OUT int \* *respDataLen* )

FeliCa Poll for Card

Polls for a Felica Card

## Parameters

<i>systemCode</i>	System Code.
<i>systemCodeLen</i>	Length of systemCode. Must be 2 bytes
<i>respData</i>	response data will be stored in respData. Poll response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of systemCode.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.168 `int felica_read ( IN BYTE * serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE * blockList, IN int blockListLen, OUT BYTE * blockData, OUT int * blockDataLen )`

## FeliCa Read

Reads up to 4 blocks.

## Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Maximum 4 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block 16 bytes.
<i>blockDataLen</i>	Length of blockData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.169 `int felica_readWithMac ( IN int blockCnt, IN BYTE * blockList, IN int blockListLen, OUT BYTE * blockData, OUT int * blockDataLen )`

## FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first

## Parameters

<i>blockCnt</i>	Number of blocks in blockList. Maximum 3 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block is 16 bytes.
<i>blockDataLen</i>	Length of blockData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.170 `int felica_requestService ( IN BYTE * nodeCode, IN int nodeCodeLen, OUT BYTE * respData, OUT int * respDataLen )`

## FeliCa Request Service

Request Service for a Felica Card



## Parameters

<i>nodeCode</i>	Node Code List. Each node 2 bytes
<i>nodeCodeLen</i>	Length of nodeCode.
<i>respData</i>	response data will be stored in respData. Response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of respData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.171 `int felica_SendCommand ( IN BYTE * command, IN int commandLen, OUT BYTE * respData, OUT int * respDataLen )`

## FeliCa Send Command

Send a Felica Command

## Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>commandLen</i>	Length of command data
<i>respData</i>	Response data from felica card.
<i>respDataLen</i>	Length of respData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.172 `int felica_write ( IN BYTE * serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE * blockList, IN int blockListLen, IN BYTE * blockData, IN int blockDataLen, OUT BYTE * statusFlag, OUT int * statusFlagLen )`

## FeliCa Write

Writes a block

## Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Currently only support 1 block.
<i>blockList</i>	Block list. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockData.
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.
<i>respData</i>	If successful, the Status Flag (2 bytes) is stored in respData.resData. Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.173 `int felica_writeWithMac ( IN BYTE blockNum, IN BYTE * blockData, IN int blockDataLen )`

## FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

## Parameters

<i>blockNum</i>	Number of block
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.174 int `icc_disable ( )`

ICC Function enable/disable - AUGUSTA Disable ICC function

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.175 int `icc_enable ( IN int withNotification )`

ICC Function enable/disable - AUGUSTA Enable ICC function with or without seated notification

## Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> <li>• 1: with notification when ICC seated status changed,</li> <li>• 0: without notification.</li> </ul>
-------------------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.176 int `icc_exchangeAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

Exchange APDU with plain text - AUGUSTA For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.177 int `icc_exchangeEncryptedAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

Exchange APDU with encrypted data - AUGUSTA For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	KSN + encrypted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]
---------------	---

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

## Parameters

<i>cLen</i>	data packet length
<i>reData</i>	response encrypted APDU response. Can be three options:

[00] + [Plaintext R-APDU]

- [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]
- [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]

The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.

## Parameters

<i>reLen</i>	encrypted APDU response data length
--------------	-------------------------------------

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.178 int icc\_getAPDU\_KSN ( OUT BYTE \* KSN, IN\_OUT int \* inLen )

Get APDU KSN - AUGUSTA

Retrieves the KSN used in ICC Encrypted APDU usage

## Parameters

<i>KSN</i>	Returns the encrypted APDU packet KSN
<i>inLen</i>	KSN data length

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.179 int icc\_getFunctionStatus ( OUT int \* enabled, OUT int \* withNotification )

Get ICC Function status - AUGUSTA Get ICC Function status about enable/disable and with or without seated notification

**Parameters**

<i>enabled</i>	<ul style="list-style-type: none"> <li>• 1: ICC Function enabled,</li> <li>• 0: means disabled.</li> </ul>
<i>withNotification</i>	1 means with notification when ICC seated status changed. 0 means without notification.

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.180 int icc\_getICCReaderStatus ( OUT BYTE \* *status* )

Get Reader Status - AUGUSTA

Returns the reader status

**Parameters**

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.181 int icc\_getKeyFormatForICCDUKPT ( OUT BYTE \* *format* )

Get Key Format For DUKPT - AUGUSTA

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

**Parameters**

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default)</li> <li>• 'AES': Encrypted card data with AES if DUKPT Key had been loaded.</li> <li>• 'NONE': No Encryption.</li> </ul>
---------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.2.4.182 int icc\_getKeyTypeForICCDUKPT ( OUT BYTE \* *type* )

Get Key Type for DUKPT - AUGUSTA

Specifies the key type used for ICC DUKPT encryption This applies to both MSR and ICC

## Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded. (default)</li> <li>• 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.</li> </ul>
-------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.183 int `icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

12.2.4.184 int `icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

## Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.2.4.185 int `icc_setKeyFormatForICCDUKPT ( IN BYTE format )`

Set Key Format for DUKPT - AUGUSTA

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

## Parameters

<i>format</i>	encryption Encryption Type <ul style="list-style-type: none"> <li>• 00: Encrypt with TDES</li> <li>• 01: Encrypt with AES</li> <li>• 02: Encrypt with TransArmor - AUGUSTA only</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.2.4.186 int `icc_setKeyTypeForICCDUKPT` ( IN BYTE *type* )

Set Key Type for DUKPT Key - AUGUSTA

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

##### Parameters

<i>type</i>	Encryption Type <ul style="list-style-type: none"> <li>• 00: Encrypt with Data Key</li> <li>• 01: Encrypt with PIN Key</li> </ul>
-------------	---

##### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.2.4.187 int `iso8583_deserializeFromXML` ( IN BYTE \* *serializedMessage*, IN int *serializedMessageLength*, OUT DL\_ISO8583\_HANDLER \* *ISOHandler*, OUT DL\_ISO8583\_MSG \* *ISOMessage* )

Deserialize the XML-formatted ISO8583 message.

##### Parameters

<i>serialized-Message</i>	- The XML-formatted message
<i>serialized-MessageLength</i>	- The length of the XML-formatted message
<i>ISOHandler</i>	- A null ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.188 int `iso8583_displayMessage` ( IN DL\_ISO8583\_HANDLER \* *ISOHandler*, IN DL\_ISO8583\_MSG \* *ISOMessage* )

Display the messages in a formatted manner on the screen for verifying results.

##### Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.189 int `iso8583_freeMessage` ( IN DL\_ISO8583\_MSG \* *ISOMessage* )

Deallocate the ISO8583 message structure's memory.

## Parameters

<i>ISOMessage</i>	- The ISO8583 message structure
-------------------	---------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.190 `int iso8583_get1987Handler ( OUT DL_ISO8583_HANDLER * ISOHandler )`

Get the ISO8583 1987 version handler.

## Parameters

<i>ISOHandler</i>	A handler with knowledge of the ISO8583 1987 version fields
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.191 `int iso8583_get1993Handler ( OUT DL_ISO8583_HANDLER * ISOHandler )`

Get the ISO8583 1993 version handler.

## Parameters

<i>ISOHandler</i>	A handler with knowledge of the ISO8583 1993 version fields
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.192 `int iso8583_get2003Handler ( OUT DL_ISO8583_HANDLER * ISOHandler )`

Get the ISO8583 2003 version handler.

## Parameters

<i>ISOHandler</i>	A handler with knowledge of the ISO8583 2003 version fields
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.193 `int iso8583_getField ( IN DL_UINT16 dataField, IN DL_ISO8583_HANDLER * ISOHandler, OUT DL_ISO8583_FIELD_DEF * field )`

Get the specified field's information using the data field.

## Parameters

<i>dataField</i>	- The data field number
------------------	-------------------------

<i>ISOHandler</i>	- The ISO8583 handler
<i>field</i>	- The requested field

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.194 `int iso8583_getMessageField ( IN DL_UINT16 dataField, IN DL_ISO8583_MSG * ISOMessage, OUT DL_ISO8583_MSG_FIELD * messageField )`

Get the specified message field using the data field.

#### Parameters

<i>dataField</i>	- The data field number
<i>ISOMessage</i>	- The ISO8583 message structure
<i>messageField</i>	- The requested message field

#### Returns

0 if the if the setting was applied; otherwise, return -1 on failure

12.2.4.195 `int iso8583_initializeMessage ( OUT DL_ISO8583_MSG * ISOMessage )`

Initialize the ISO8583 message structure.

#### Parameters

<i>ISOMessage</i>	- The initialized ISO8583 message structure
-------------------	---

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.196 `int iso8583_packMessage ( IN const DL_ISO8583_HANDLER * ISOHandler, IN const DL_ISO8583_MSG * ISOMessage, OUT DL_UINT8 * packedData, OUT DL_UINT16 * packedDataLength )`

Pack the message fields into an array.

#### Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure
<i>packedData</i>	- The packaged data
<i>packedDataLength</i>	- The packaged data's length

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.197 `int iso8583_removeMessageField ( IN DL_UINT16 dataField, OUT DL_ISO8583_MSG * ISOMessage )`

Remove the specified message field.



## Parameters

<i>dataField</i>	- The data field number
<i>ISOMessage</i>	- The ISO8583 message structure

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.198 `int iso8583_serializeToXML ( IN DL_ISO8583_HANDLER * ISOHandler, IN DL_ISO8583_MSG * ISOMessage, OUT BYTE * serializedMessage, OUT int * serializedMessageLength )`

Serialize the message fields into an XML format.

## Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>ISOMessage</i>	- The ISO8583 message structure
<i>serialized-Message</i>	- The XML-formatted message
<i>serialized-MessageLength</i>	- The XML message's length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.199 `int iso8583_setMessageField ( IN DL_UINT16 dataField, IN const DL_UINT8 * data, OUT DL_ISO8583_MSG * ISOMessage )`

Set the specified message field.

## Parameters

<i>dataField</i>	- The data field number
<i>data</i>	- The data to apply
<i>ISOMessage</i>	- The ISO8583 message structure

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.200 `int iso8583_unpackMessage ( IN const DL_ISO8583_HANDLER * ISOHandler, IN const DL_UINT8 * packedData, IN DL_UINT16 packedDataLength, OUT DL_ISO8583_MSG * ISOMessage )`

Unpack the message field array into the ISO8583 message structure.

## Parameters

<i>ISOHandler</i>	- The ISO8583 handler
<i>packedData</i>	- The packaged data
<i>packedData- Length</i>	- The packaged data's length
<i>ISOMessage</i>	- The ISO8583 message structure

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.201 `int lcd_addButton ( IN char * screenName, IN int screenNameLen, IN char * buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char * label, IN int labelLen, OUT IDTLCDItem * returnItem )`

#### Add Button

Adds a button to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

#### Parameters

<i>screenName</i>	Screen name that will be the target of add button
<i>screenNameLen</i>	Length of <i>screenName</i>
<i>buttonName</i>	Button name that will be the target of add button
<i>buttonNameLen</i>	Length of <i>buttonName</i>
<i>type</i>	Button Type <ul style="list-style-type: none"> <li>• Large = 0x01</li> <li>• Medium = 0x02</li> <li>• Invisible = 0x03 (70px by 60 px)</li> </ul>
<i>alignment</i>	Position for Button <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x andy</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Button, range 0-271
<i>yCord</i>	y-coordinate for Button, range 0-479
<i>label</i>	Label to show on the button. Required for Large/Medium buttons. Not used for Invisible buttons.
<i>labelLen</i>	Length of label
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created button

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.202 `int lcd_addEthernet ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem * returnItem )`

#### Add Ethernet Settings

Adds an Ethernet settings to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add widget
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add widget
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for widget <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for widget, range 0-271
<i>yCord</i>	y-coordinate for widget, range 0-479
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.2.4.203 `int lcd_addImage ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char * filename, IN int filenameLen, OUT IDTLCDItem * returnItem )`

## Add Image

Adds a image to a selected screen. Must execute lcd\_createScreen first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add image
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add image
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Image <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Image, range 0-271
<i>yCord</i>	y-coordinate for Image, range 0-479
<i>filename</i>	Filename of the image. Must be available in device memory.
<i>filenameLen</i>	Length of filename
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created image

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.2.4.204 `int lcd_addItemToList ( IN BYTE * listGraphicsID, IN char * itemName, IN char * itemID, IN int selected )`

Adds an item to an existing list.

Custom Display Mode must be enabled for custom text.

## Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemName</i>	Item name (Maximum: 127 characters)
<i>itemID</i>	Identifier for the item (Maximum: 31 characters)
<i>selected</i>	If the item should be selected

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

12.2.4.205 `int lcd_addLED ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem * returnItem, IN BYTE * LED, IN int LEDLen )`

Add LED

Adds a LED widget to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add LED
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add LED
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for LED <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x andy</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for LED, range 0-271
<i>yCord</i>	y-coordinate for LED, range 0-479
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget
<i>LED</i>	Must be 4 bytes, LED 0 = byte 0, LED 1 = byte 1, LED 2 = byte 2, LED 3 = byte 3 <ul style="list-style-type: none"> <li>• Value 0 = LED OFF</li> <li>• Value 1 = LED Green</li> <li>• Value 2 = LED Yellow</li> <li>• Value 3 = LED Red</li> </ul>
<i>LEDLen</i>	Length of LED

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.2.4.206 int lcd\_addText ( IN char \* *screenName*, IN int *screenNameLen*, IN char \* *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *xCord*, IN int *yCord*, IN int *width*, IN int *height*, IN BYTE *fontID*, IN BYTE \* *color*, IN int *colorLen*, IN char \* *label*, IN int *labelLen*, OUT IDTLCDItem \* *returnItem* )

## Add text

Adds a text component to a selected screen. Must execute lcd\_createScreen first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add text
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add text
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Text <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x andy</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Text, range 0-271
<i>yCord</i>	y-coordinate for Text, range 0-479
<i>width</i>	Width of text area
<i>height</i>	Height of text area
<i>fontID</i>	Font ID
<i>color</i>	Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000 <ul style="list-style-type: none"> <li>• Byte 0 = B</li> <li>• Byte 1 = G</li> <li>• Byte 2 = R</li> <li>• Byte 3 = Reserved</li> </ul>
<i>colorLen</i>	Length of color
<i>label</i>	Label to show on the text
<i>labelLen</i>	Length of label
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created text area

### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

Font ID	Typography Name	Font	Size
0	RoundBold_12	RoundBold.ttf	12
1	RoundBold_18	RoundBold.ttf	18
2	RoundBold_24	RoundBold.ttf	24
3	RoundBold_36	RoundBold.ttf	36
4	RoundBold_48	RoundBold.ttf	48
5	RoundBold_60	RoundBold.ttf	60
6	RoundBold_72	RoundBold.ttf	72
7	RoundCondensedBold_12	RoundCondensedBold.ttf	12
8	RoundCondensedBold_18	RoundCondensedBold.ttf	18
9	RoundCondensedBold_24	RoundCondensedBold.ttf	24

10	RoundCondensedBold_ - 36	RoundCondensedBold.ttf	36
11	RoundCondensedBold_ - 48	RoundCondensedBold.ttf	48
12	RoundCondensedBold_ - 60	RoundCondensedBold.ttf	60
13	RoundCondensedBold_ - 72	RoundCondensedBold.ttf	72
14	RoundCondensed-Medium_ 12	RoundCondensed-Medium_0.ttf	12
15	RoundCondensed-Medium_ 18	RoundCondensed-Medium_0.ttf	18
16	RoundCondensed-Medium_ 24	RoundCondensed-Medium_0.ttf	24
17	RoundCondensed-Medium_ 36	RoundCondensed-Medium_0.ttf	36
18	RoundCondensed-Medium_ 48	RoundCondensed-Medium_0.ttf	48
19	RoundCondensed-Medium_ 60	RoundCondensed-Medium_0.ttf	60
20	RoundCondensed-Medium_ 72	RoundCondensed-Medium_0.ttf	72
21	RoundCondensed-Semibold_ 12	RoundCondensed-Semibold.ttf	12
22	RoundCondensed-Semibold_ 18	RoundCondensed-Semibold.ttf	18
23	RoundCondensed-Semibold_ 24	RoundCondensed-Semibold.ttf	24
24	RoundCondensed-Semibold_ 36	RoundCondensed-Semibold.ttf	36
25	RoundCondensed-Semibold_ 48	RoundCondensed-Semibold.ttf	48
26	RoundCondensed-Semibold_ 60	RoundCondensed-Semibold.ttf	60
27	RoundCondensed-Semibold_ 72	RoundCondensed-Semibold.ttf	72
28	RoundMedium_ 12	RoundMedium.ttf	12
29	RoundMedium_ 18	RoundMedium.ttf	18
30	RoundMedium_ 24	RoundMedium.ttf	24
31	RoundMedium_ 36	RoundMedium.ttf	36
32	RoundMedium_ 48	RoundMedium.ttf	48
33	RoundMedium_ 60	RoundMedium.ttf	60
34	RoundMedium_ 72	RoundMedium.ttf	72
35	RoundSemibold_ 12	RoundSemibold.ttf	12
36	RoundSemibold_ 18	RoundSemibold.ttf	18
37	RoundSemibold_ 24	RoundSemibold.ttf	24
38	RoundSemibold_ 36	RoundSemibold.ttf	36
39	RoundSemibold_ 48	RoundSemibold.ttf	48
40	RoundSemibold_ 60	RoundSemibold.ttf	60
41	RoundSemibold_ 72	RoundSemibold.ttf	72

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16



Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.2.4.207 `int lcd_addVideo ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char * filename, IN int filenameLen, OUT IDTLCDItem * returnItem )`

#### Add Video

Adds a video to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

#### Parameters

<i>screenName</i>	Screen name that will be the target of add video
<i>screenNameLen</i>	Length of <i>screenName</i>
<i>objectName</i>	Object name that will be the target of add video
<i>objectNameLen</i>	Length of <i>objectName</i>
<i>alignment</i>	Position for Video <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Video, range 0-271
<i>yCord</i>	y-coordinate for Video, range 0-479
<i>filename</i>	Filename of the video. Must be available in the sd card.
<i>filenameLen</i>	Length of filename
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created video

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20
video	1

12.2.4.208 `int lcd_cancelSlideShow ( OUT BYTE * statusCode, IN_OUT int * statusCodeLen )`

Cancel slide show Cancel the slide show currently running

## Parameters

<i>statusCode</i>	If the return code is not Success (0), the kernel may return a four-byte Extended Status Code
<i>statusCodeLen</i>	the length of the Extended Status Code (should be 4 bytes)

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.209 int lcd\_captureSignature ( IN int *timeout* )

Enables Signature Capture This command executes the signature capture screen. Once a signature is captured, it is sent to the callback with DeviceState.Signature, and the data will contain a .png of the signature

## Parameters

<i>timeout</i>	Timeout waiting for the signature capture
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.210 int lcd\_clearDisplay ( IN BYTE *control* )

Clear Display Command to clear the display screen on the reader. It returns the display to the currently defined background color and terminates all events

## Parameters

<i>control</i>	for L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.211 int lcd\_clearEventQueue ( )

Removes all entries from the event queue.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.212 int lcd\_clearScreenInfo ( )

Clear Screen Info

Clear all current screen information in RAM and flash. And then show 'power-on screen'

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

12.2.4.213 `int lcd_cloneScreen ( IN char * screenName, IN int screenNameLen, IN char * cloneName, IN int cloneNameLen,  
OUT int * cloneID )`

Clone Screen

Clones an existing screen.

## Parameters

<i>screenName</i>	Screen name to clone.
<i>screenNameLen</i>	Length of screenName.
<i>cloneName</i>	Name of the cloned screen.
<i>cloneNameLen</i>	Length of cloneName.
<i>cloneID</i>	Screen ID of the cloned screen

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.214 int lcd\_createInputField ( IN BYTE \* *specs*, IN int *specsLen*, OUT BYTE \* *graphicId* )

DEPRECATED : please use [lcd\\_createInputField\\_Len\(IN BYTE \\*specs, IN int specsLen, OUT BYTE \\*graphicId, IN\\_OUT int \\*graphicIdLen\)](#)

Create an input field on the screen.

## Parameters

<i>specs</i>	The specs of the input field:
--------------	-------------------------------

Length (bytes)	Description
2 - 4 -----	X coordinate in pixels, zero terminated ASCII -----
2 - 4 -----	Y coordinate in pixels, zero terminated ASCII -----
2 - 4 -----	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width. -----
2 - 4 -----	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height. -----
2 -----	Font designation. Default font = 1, zero terminated ASCII -----
2 - 3 -----	Zero terminated ASCII Font ID -----
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll

	from the right to the left.
	Bit 2 - 15 Reserved
-----	-----
1 or 9	Foreground color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Background color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Border color, zero terminated ASCII hexadecimal
-----	-----
1 - 65	Prefill String, zero terminated ASCII
-----	-----
1 - 65	Format String, zero terminated ASCII

**Parameters**

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.215** int lcd\_createInputField\_Len ( IN BYTE \* *specs*, IN int *specsLen*, OUT BYTE \* *graphicId*, IN\_OUT int \* *graphicIdLen* )

Create an input field on the screen.

**Parameters**

<i>specs</i>	The specs of the input field:
--------------	-------------------------------

Length (bytes)	Description
2 - 4	X coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Y coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
-----	-----
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
-----	-----
2	Font designation. Default font = 1, zero terminated ASCII
-----	-----

2 - 3	Zero terminated ASCII Font ID
-----	-----
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
	Bit 2 - 15 Reserved
-----	-----
1 or 9	Foreground color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Background color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Border color, zero terminated ASCII hexadecimal
-----	-----
1 - 65	Prefill String, zero terminated ASCII
-----	-----
1 - 65	Format String, zero terminated ASCII

**Parameters**

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)
<i>graphicsIDLen</i>	Length of graphicsID

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.216 int lcd\_createList ( IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \* graphicsID )

DEPRECATED : please use lcd\_createList\_Len(IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)

Creates a display list.

**Parameters**

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels

<i>numOfColumns</i>	Number of columns to display																																																									
<i>numOfRows</i>	Number of rows to display																																																									
<i>fontDesignation</i>	Font Designation 1 - Default font																																																									
<i>fontID</i>	Font styling <table><tr><th>Font ID</th><th>Height in pixels</th><th>Font Properties</th></tr><tr><td>1</td><td>13</td><td>Regular</td></tr><tr><td>2</td><td>17</td><td>Regular</td></tr><tr><td>3</td><td>17</td><td>Bold</td></tr><tr><td>4</td><td>22</td><td>Regular</td></tr><tr><td>5</td><td>20</td><td>Regular</td></tr><tr><td>6</td><td>20</td><td>Bold</td></tr><tr><td>7</td><td>29</td><td>Regular</td></tr><tr><td>8</td><td>38</td><td>Regular</td></tr><tr><td>9</td><td>38</td><td>Bold</td></tr><tr><td>10</td><td>58</td><td>Regular</td></tr><tr><td>11</td><td>58</td><td>Bold, mono-space</td></tr><tr><td>12</td><td>14</td><td>Regular, mono-space, 8 pixels wide</td></tr><tr><td>13</td><td>15</td><td>Regular, mono-space, 9 pixels wide</td></tr><tr><td>14</td><td>17</td><td>Regular, mono-space, 9 pixels wide</td></tr><tr><td>15</td><td>20</td><td>Regular, mono-space, 11 pixels wide</td></tr><tr><td>16</td><td>21</td><td>Regular, mono-space, 12 pixels wide</td></tr><tr><td>17</td><td>25</td><td>Regular, mono-space, 14 pixels wide</td></tr><tr><td>18</td><td>30</td><td>Regular, mono-space, 17 pixels wide</td></tr></table>	Font ID	Height in pixels	Font Properties	1	13	Regular	2	17	Regular	3	17	Bold	4	22	Regular	5	20	Regular	6	20	Bold	7	29	Regular	8	38	Regular	9	38	Bold	10	58	Regular	11	58	Bold, mono-space	12	14	Regular, mono-space, 8 pixels wide	13	15	Regular, mono-space, 9 pixels wide	14	17	Regular, mono-space, 9 pixels wide	15	20	Regular, mono-space, 11 pixels wide	16	21	Regular, mono-space, 12 pixels wide	17	25	Regular, mono-space, 14 pixels wide	18	30	Regular, mono-space, 17 pixels wide
Font ID	Height in pixels	Font Properties																																																								
1	13	Regular																																																								
2	17	Regular																																																								
3	17	Bold																																																								
4	22	Regular																																																								
5	20	Regular																																																								
6	20	Bold																																																								
7	29	Regular																																																								
8	38	Regular																																																								
9	38	Bold																																																								
10	58	Regular																																																								
11	58	Bold, mono-space																																																								
12	14	Regular, mono-space, 8 pixels wide																																																								
13	15	Regular, mono-space, 9 pixels wide																																																								
14	17	Regular, mono-space, 9 pixels wide																																																								
15	20	Regular, mono-space, 11 pixels wide																																																								
16	21	Regular, mono-space, 12 pixels wide																																																								
17	25	Regular, mono-space, 14 pixels wide																																																								
18	30	Regular, mono-space, 17 pixels wide																																																								
<i>verticalScroll-ArrowsVisible</i>	Display vertical scroll arrows by default																																																									
<i>borederedList-Items</i>	Draw border around list items																																																									
<i>borederedScroll-Arrows</i>	Draw border around scroll arrows (if visible)																																																									
<i>touchSensitive</i>	List items are touch enabled																																																									
<i>automatic-Scrolling</i>	Enable automatic scrolling of list when new items exceed display area																																																									
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory																																																									

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.217 int lcd\_createList\_Len ( IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \* graphicsID, IN\_OUT int \* graphicsIDLen )

Creates a display list.

## Parameters

<i>posX</i>	X coordinate in pixels																																																																																
<i>posY</i>	Y coordinate in pixels																																																																																
<i>numOfColumns</i>	Number of columns to display																																																																																
<i>numOfRows</i>	Number of rows to display																																																																																
<i>fontDesignation</i>	Font Designation 1 - Default font																																																																																
<i>fontID</i>	Font styling <table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															
<i>verticalScroll-ArrowsVisible</i>	Display vertical scroll arrows by default																																																																																
<i>borederedList-Items</i>	Draw border around list items																																																																																
<i>borederedScroll-Arrows</i>	Draw border around scroll arrows (if visible)																																																																																
<i>touchSensitive</i>	List items are touch enabled																																																																																
<i>automatic-Scrolling</i>	Enable automatic scrolling of list when new items exceed display area																																																																																
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)																																																																																
<i>graphicsIDLen</i>	Length of graphicsID (optional)																																																																																

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.218 `int lcd_createScreen ( IN char * screenName, IN int screenNameLen, OUT int * ScreenID )`

## Create Screen

Creates a new screen.

## Parameters

<i>screenName</i>	Screen name to use.
<i>screenNameLen</i>	Length of screenName.
<i>screenID</i>	Screen ID that was created.

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`



#### 12.2.4.219 int lcd\_customDisplayMode ( IN int *enable* )

Custom Display Mode Controls the LCD display mode to custom display. Keyboard entry is limited to the Cancel, Clear, Enter and the function keys, if present. PIN entry is not permitted while the reader is in Custom Display Mode

## Parameters

<i>enable</i>	TRUE = enabled, FALSE = disabled
---------------	----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.220 `int lcd_destroyScreen ( IN char * screenName, IN int screenNameLen )`

## Destroy Screen

Destroys a previously created inactive screen. The screen cannot be active

## Parameters

<i>screenName</i>	Screen name to destroy. The screen number is assigned with <code>lcd_createScreen</code> .
<i>screenNameLen</i>	Length of <i>screenName</i> .

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.221 `int lcd_displayButton ( IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char * buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE * graphicsID )`

DEPRECATED : please use `lcd_displayButton_Len(IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char *buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen)`

Displays an interactive button.

## Parameters

<i>posX</i>	X coordinate in pixels																																																									
<i>posY</i>	Y coordinate in pixels																																																									
<i>buttonWidth</i>	Width of the button																																																									
<i>buttonHeight</i>	Height of the button																																																									
<i>fontDesignation</i>	Font designation 1 - Default																																																									
<i>Font</i>	<div>ID Font styling</div> <table><thead><tr><th>Font ID</th><th>Height in pixels</th><th>Font Properties</th></tr></thead><tbody><tr><td>1</td><td>13</td><td>Regular</td></tr><tr><td>2</td><td>17</td><td>Regular</td></tr><tr><td>3</td><td>17</td><td>Bold</td></tr><tr><td>4</td><td>22</td><td>Regular</td></tr><tr><td>5</td><td>20</td><td>Regular</td></tr><tr><td>6</td><td>20</td><td>Bold</td></tr><tr><td>7</td><td>29</td><td>Regular</td></tr><tr><td>8</td><td>38</td><td>Regular</td></tr><tr><td>9</td><td>38</td><td>Bold</td></tr><tr><td>10</td><td>58</td><td>Regular</td></tr><tr><td>11</td><td>58</td><td>Bold, mono-space</td></tr><tr><td>12</td><td>14</td><td>Regular, mono-space, 8 pixels wide</td></tr><tr><td>13</td><td>15</td><td>Regular, mono-space, 9 pixels wide</td></tr><tr><td>14</td><td>17</td><td>Regular, mono-space, 9 pixels wide</td></tr><tr><td>15</td><td>20</td><td>Regular, mono-space, 11 pixels wide</td></tr><tr><td>16</td><td>21</td><td>Regular, mono-space, 12 pixels wide</td></tr><tr><td>17</td><td>25</td><td>Regular, mono-space, 14 pixels wide</td></tr><tr><td>18</td><td>30</td><td>Regular, mono-space, 17 pixels wide</td></tr></tbody></table>	Font ID	Height in pixels	Font Properties	1	13	Regular	2	17	Regular	3	17	Bold	4	22	Regular	5	20	Regular	6	20	Bold	7	29	Regular	8	38	Regular	9	38	Bold	10	58	Regular	11	58	Bold, mono-space	12	14	Regular, mono-space, 8 pixels wide	13	15	Regular, mono-space, 9 pixels wide	14	17	Regular, mono-space, 9 pixels wide	15	20	Regular, mono-space, 11 pixels wide	16	21	Regular, mono-space, 12 pixels wide	17	25	Regular, mono-space, 14 pixels wide	18	30	Regular, mono-space, 17 pixels wide
Font ID	Height in pixels	Font Properties																																																								
1	13	Regular																																																								
2	17	Regular																																																								
3	17	Bold																																																								
4	22	Regular																																																								
5	20	Regular																																																								
6	20	Bold																																																								
7	29	Regular																																																								
8	38	Regular																																																								
9	38	Bold																																																								
10	58	Regular																																																								
11	58	Bold, mono-space																																																								
12	14	Regular, mono-space, 8 pixels wide																																																								
13	15	Regular, mono-space, 9 pixels wide																																																								
14	17	Regular, mono-space, 9 pixels wide																																																								
15	20	Regular, mono-space, 11 pixels wide																																																								
16	21	Regular, mono-space, 12 pixels wide																																																								
17	25	Regular, mono-space, 14 pixels wide																																																								
18	30	Regular, mono-space, 17 pixels wide																																																								
<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap																																																									
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)																																																									
<i>buttonTextColor-R</i>	- Red component for foreground color (0 - 255)																																																									
<i>buttonTextColor-G</i>	- Green component for foreground color (0 - 255)																																																									
<i>buttonTextColor-B</i>	- Blue component for foreground color (0 - 255)																																																									
<i>button-Background-ColorR</i>	- Red component for background color (0 - 255)																																																									
<i>button-Background-ColorG</i>	- Green component for background color (0 - 255)																																																									

<i>button-Background-ColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.222 `int lcd_displayButton_Len ( IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char * buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE * graphicsID, IN_OUT int * graphicsIDLen )`

Displays an interactive button.

## Parameters

<i>posX</i>	X coordinate in pixels																																																																																
<i>posY</i>	Y coordinate in pixels																																																																																
<i>buttonWidth</i>	Width of the button																																																																																
<i>buttonHeight</i>	Height of the button																																																																																
<i>fontDesignation</i>	Font designation 1 - Default																																																																																
<i>Font</i>	ID Font styling																																																																																
	<table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															

<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)
<i>buttonTextColor-R</i>	- Red component for foreground color (0 - 255)
<i>buttonTextColor-G</i>	- Green component for foreground color (0 - 255)
<i>buttonTextColor-B</i>	- Blue component for foreground color (0 - 255)
<i>button-Background-ColorR</i>	- Red component for background color (0 - 255)
<i>button-Background-ColorG</i>	- Green component for background color (0 - 255)
<i>button-Background-ColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.223 int lcd\_displayMessage ( int lineNumber, char \* message, int messageLen )

Display Message on Line

Displays a message on a display line.

**Parameters**

<i>lineNumber</i>	Line number to display message on (1-4)
<i>message</i>	Message to display
<i>messageLen</i>	length of message

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.224 int lcd\_displayParagraph ( IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char \* displayText )

Displays text with scroll feature.

Custom Display Mode must be enabled.

## Parameters

<i>posX</i>	X coordinate in pixels		
<i>posY</i>	Y coordinate in pixels		
<i>displayWidth</i>	Width of the display area in pixels (Minimum: 40px) 0 or NULL - Use the full width to display text		
<i>displayHeight</i>	Height of the display area in pixels (Minimum: 100px) 0 or NULL - Use the full height to display text		
<i>fontDesignation</i>	Font designation 1 - Default		
<i>fontID</i>	Font styling		
	Font ID	Height in pixels	Font Properties
	-----	-----	-----
	1	13	Regular
	2	17	Regular
	3	17	Bold
	4	22	Regular
	5	20	Regular
	6	20	Bold
	7	29	Regular
	8	38	Regular
	9	38	Bold
	10	58	Regular
	11	58	Bold, mono-space
	12	14	Regular, mono-space, 8 pixels wide
	13	15	Regular, mono-space, 9 pixels wide
	14	17	Regular, mono-space, 9 pixels wide
	15	20	Regular, mono-space, 11 pixels wide
	16	21	Regular, mono-space, 12 pixels wide
17	25	Regular, mono-space, 14 pixels wide	
18	30	Regular, mono-space, 17 pixels wide	
<i>display-Properties</i>	Display properties for the text 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Center on screen without clearing screen 5 - Center on screen after clearing screen		
<i>displayText</i>	Display text (Maximum: 3999 characters)		

12.2.4.225 `int lcd_displayPrompt ( int promptNumber, int lineNumber )`

Display Prompt on Line

Displays a message prompt from L100 memory.

## Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>lineNumber</i>	Line number to display message prompt (1-4)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.226 `int lcd_displayText ( IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char * displayText, OUT BYTE * graphicsID )`

DEPRECATED : please use lcd\_displayText\_Len(IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char \*displayText, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

## Parameters

<i>posX</i>	X coordinate in pixels																																																												
<i>posY</i>	Y coordinate in pixels																																																												
<i>displayWidth</i>	Width of the display area in pixels (optional)																																																												
<i>displayHeight</i>	Height of the display area in pixels (optional)																																																												
<i>fontDesignation</i>	Font designation 1 - Default																																																												
<i>Font</i>	<div>ID Font styling</div> <table><tr><th>Font ID</th><th>Height in pixels</th><th>Font Properties</th></tr><tr><td>-----</td><td>-----</td><td>-----</td></tr><tr><td>1</td><td>13</td><td>Regular</td></tr><tr><td>2</td><td>17</td><td>Regular</td></tr><tr><td>3</td><td>17</td><td>Bold</td></tr><tr><td>4</td><td>22</td><td>Regular</td></tr><tr><td>5</td><td>20</td><td>Regular</td></tr><tr><td>6</td><td>20</td><td>Bold</td></tr><tr><td>7</td><td>29</td><td>Regular</td></tr><tr><td>8</td><td>38</td><td>Regular</td></tr><tr><td>9</td><td>38</td><td>Bold</td></tr><tr><td>10</td><td>58</td><td>Regular</td></tr><tr><td>11</td><td>58</td><td>Bold, mono-space</td></tr><tr><td>12</td><td>14</td><td>Regular, mono-space, 8 pixels wide</td></tr><tr><td>13</td><td>15</td><td>Regular, mono-space, 9 pixels wide</td></tr><tr><td>14</td><td>17</td><td>Regular, mono-space, 9 pixels wide</td></tr><tr><td>15</td><td>20</td><td>Regular, mono-space, 11 pixels wide</td></tr><tr><td>16</td><td>21</td><td>Regular, mono-space, 12 pixels wide</td></tr><tr><td>17</td><td>25</td><td>Regular, mono-space, 14 pixels wide</td></tr><tr><td>18</td><td>30</td><td>Regular, mono-space, 17 pixels wide</td></tr></table>	Font ID	Height in pixels	Font Properties	-----	-----	-----	1	13	Regular	2	17	Regular	3	17	Bold	4	22	Regular	5	20	Regular	6	20	Bold	7	29	Regular	8	38	Regular	9	38	Bold	10	58	Regular	11	58	Bold, mono-space	12	14	Regular, mono-space, 8 pixels wide	13	15	Regular, mono-space, 9 pixels wide	14	17	Regular, mono-space, 9 pixels wide	15	20	Regular, mono-space, 11 pixels wide	16	21	Regular, mono-space, 12 pixels wide	17	25	Regular, mono-space, 14 pixels wide	18	30	Regular, mono-space, 17 pixels wide
Font ID	Height in pixels	Font Properties																																																											
-----	-----	-----																																																											
1	13	Regular																																																											
2	17	Regular																																																											
3	17	Bold																																																											
4	22	Regular																																																											
5	20	Regular																																																											
6	20	Bold																																																											
7	29	Regular																																																											
8	38	Regular																																																											
9	38	Bold																																																											
10	58	Regular																																																											
11	58	Bold, mono-space																																																											
12	14	Regular, mono-space, 8 pixels wide																																																											
13	15	Regular, mono-space, 9 pixels wide																																																											
14	17	Regular, mono-space, 9 pixels wide																																																											
15	20	Regular, mono-space, 11 pixels wide																																																											
16	21	Regular, mono-space, 12 pixels wide																																																											
17	25	Regular, mono-space, 14 pixels wide																																																											
18	30	Regular, mono-space, 17 pixels wide																																																											
<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen																																																												
<i>displayText</i>	Display text (Maximum: 1900 characters)																																																												
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory																																																												

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.227 int lcd\_displayText\_Len ( IN int *posX*, IN int *posY*, IN int *displayWidth*, IN int *displayHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *screenPosition*, IN char \* *displayText*, OUT BYTE \* *graphicsID*, IN\_OUT int \* *graphicsIDL* )

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

## Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (optional)



<i>displayHeight</i>	Height of the display area in pixels (optional)																																																																																
<i>fontDesignation</i>	Font designation 1 - Default																																																																																
<i>Font</i>	ID Font styling <table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															
<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen																																																																																
<i>displayText</i>	Display text (Maximum: 1900 characters)																																																																																
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)																																																																																
<i>graphicsIDLen</i>	Length of graphicsID (optional)																																																																																

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.228 int lcd\_enableBacklight ( int enable )

Enable/Disable LCD Backlight Trns on/off the LCD back lighting.

**Parameters**

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
---------------	--

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.229 int lcd\_getActiveScreen ( OUT char \* screenName, IN\_OUT int \* screenNameLen )

Get Active Screen

Returns the active screen ID.

**Parameters**

<i>screenName</i>	Screen name this is active.
<i>screenNameLen</i>	Length of screenName.

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.230 `int lcd_getAllObjects ( IN char * screenName, IN int screenNameLen, IN_OUT int * objectNumbers, OUT IDTObjectInfo * objectInfo )`

Get All Objects on Screen

Get all created objects' name on certain screen

**Parameters**

<i>screenName</i>	Screen name to get all objects
<i>screenNameLen</i>	Length of screenName
<i>objectNumbers</i>	Number of created objects
<i>returnObjects</i>	Array of all created objects each element includes -objectID of a created object -objectName of a created object -objectNameLen of objectName

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.231 `int lcd_getAllScreens ( IN_OUT int * screenNumbers, OUT IDTScreenInfo * screenInfo )`

Get All Screens

Get all created screens' name

**Parameters**

<i>screenNumbers</i>	Number of created screens
<i>screenInfo</i>	Array of all created screens each element includes -screenID of a created screen -screen-Name of a created screen -screenNameLen of screenName

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.232 `int lcd_getBacklightStatus ( int * enabled )`

Get Backlight Status

Returns the status of the LCD back lighting.

**Parameters**

<i>enabled</i>	1 = Backlight is ON, 0 = Backlight is OFF
----------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.233 `int lcd_getButtonEvent ( OUT int * screenID, OUT int * objectID, OUT char * screenName, IN_OUT int * screenNameLen, OUT char * objectName, IN_OUT int * objectNameLen, OUT int * isLongPress )`

Get Button Event

Reports back the ID of the button that encountered a click event after the last Get Button Event.

Parameters

<i>screenID</i>	Screen ID of the last clicked button
<i>objectID</i>	Button ID of the last clicked button
<i>screenName</i>	Screen Name of the last clicked button
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Button Name of the last clicked button
<i>objectNameLen</i>	Length of objectName
<i>isLongPress</i>	1 = Long Press, 0 = Short Press
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices)

Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.234 `int lcd_getInputEvent ( IN int timeout, OUT int * dataReceived, OUT BYTE * eventType, OUT BYTE * graphicsID, OUT BYTE * eventData )`

DEPRECATED : please use `lcd_getInputEvent_Len(IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)`

Requests input from the reader.

Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable Byte 1: State (1 = Pressed, other values RFU) Byte 2 - n: Null terminated caption
-----	-----	-----
Checkbox Event	00030001h	Length = 1 byte Byte 1: State (1 = Checked, 0 = Unchecked)
-----	-----	-----
Line Item Event	00030002h	Length = 5 bytes Byte 1: State (1 = Item Selected, other values RFU) Byte 2 - n: Caption of the selected item

-----	-----	-----
Keypad Event	00030003h	Length - 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release
		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
-----	-----	-----
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
-----	-----	-----
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
-----	-----	-----
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes

-----	-----	-----
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name
		Byte 34 - 65: Radio button caption

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.235 int lcd\_getInputEvent\_Len ( IN int *timeout*, OUT int \* *dataReceived*, OUT BYTE \* *eventType*, IN\_OUT int \* *eventTypeLen*, OUT BYTE \* *graphicsID*, IN\_OUT int \* *graphicsIDLen*, OUT BYTE \* *eventData*, IN\_OUT int \* *eventDataLen* )

Requests input from the reader.

#### Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>eventTypeLen</i>	Length of eventType
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>graphicsIDLen</i>	length of graphicID
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable
		Byte 1: State (1 = Pressed, other values RFU)
		Byte 2 - n: Null terminated caption
-----	-----	-----
Checkbox Event	00030001h	Length = 1 byte
		Byte 1: State (1 = Checked, 0 = Unchecked)
-----	-----	-----
Line Item Event	00030002h	Length = 5 bytes
		Byte 1: State (1 = Item Selected, other values RFU)
		Byte 2 - n: Caption of the selected item
-----	-----	-----
Keypad Event	00030003h	Length - 3 bytes
		Byte 1: State (1 = key pressed, 2 = key released, other values RFU)
		Byte 2 - 3: Key pressed and Key release

		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
-----	-----	-----
-----	-----	-----
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
-----	-----	-----
-----	-----	-----
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
-----	-----	-----
-----	-----	-----
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
-----	-----	-----
-----	-----	-----
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name

		Byte 34 - 65: Radio button caption
--	--	------------------------------------

## Parameters

<i>eventDataLen</i>	Length of eventData
---------------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.236 `int lcd_getInputFieldValue ( IN BYTE * graphicId, OUT BYTE * retData, IN_OUT int * retDataLen )`

Get the keypad data that was entered into the specified Input Field.

## Parameters

<i>graphicsID</i>	The graphicID of the input field (required to be 4 bytes)
<i>retData</i>	return keypad data
<i>retDataLen</i>	The length of retData

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.237 `int lcd_getSelectedListItem ( IN BYTE * listGraphicsID, OUT char * itemID )`

DEPRECATED : please use [lcd\\_getSelectedListItem\\_Len\(IN BYTE \\*listGraphicsID, OUT char \\*itemID, IN\\_OUT int \\*itemIDLen\)](#)

Retrieves the selected item's ID.

## Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'

12.2.4.238 `int lcd_getSelectedListItem_Len ( IN BYTE * listGraphicsID, OUT char * itemID, IN_OUT int * itemIDLen )`

Retrieves the selected item's ID.

## Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'
<i>itemIDLen</i>	Length of itemID

12.2.4.239 `int lcd_loadScreenInfo ( )`

Load Screen Info

Load all current screen information from RAM to flash

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

12.2.4.240 int lcd\_queryObjectbyID ( IN int *objectID*, OUT int \* *objectNumbers*, OUT IDTScreenInfo \* *screenInfo* )

#### Queery Object by ID

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object ID

##### Parameters

<i>objectID</i>	ID of the checked object
<i>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	screen names containing the item

##### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.241 int lcd\_queryObjectbyName ( IN char \* *objectName*, IN int *objectNameLen*, IN\_OUT int \* *objectNumbers*, OUT IDTScreenInfo \* *screenInfo* )

#### Queery Object by Name

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object name

##### Parameters

<i>objectName</i>	Name of the checked object
<i>objectNameLen</i>	Length of objectName
<i>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	Array of all the screen names that contain the object

##### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.242 int lcd\_queryScreenbyID ( IN int *screenID*, OUT int \* *result*, OUT int \* *screenName*, IN\_OUT int \* *screenNameLen* )

#### Queery Screen by ID

Check if the given screen exists or not

##### Parameters

<i>screenID</i>	ID of the checked screen
<i>result</i>	the checking result
<i>screenName</i>	Name of the checked screen
<i>screenNameLen</i>	Length of screenName
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices)

##### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.243 int lcd\_queryScreenbyName ( IN char \* *screenName*, IN int *screenNameLen*, OUT int \* *result* )

#### Queery Screen by Name

Check if the given screen exists or not



## Parameters

<i>screenName</i>	Name of the checked screen
<i>screenNameLen</i>	Length of screenName
<i>result</i>	the checking result

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.2.4.244 void lcd\_registerCallBk ( pLCD\_callBack pLCDf )

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

## 12.2.4.245 int lcd\_removeItem ( IN char \* screenName, IN int screenNameLen, IN char \* objectName, IN int objectNameLen )

## Removed Item

Removes a component.

## Parameters

<i>screenName</i>	Screen name where to remove the target from.
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.2.4.246 int lcd\_resetInitialState ( )

Reset to Initial State. This command places the reader UI into the idle state and displays the appropriate idle display.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.247 int lcd\_savePrompt ( int promptNumber, char \* prompt, int promptLen )

## Save Prompt

Saves a message prompt to L100 memory.

## Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>prompt</i>	Prompt string (up to 20 characters)
<i>promptLen</i>	length of prompt

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.248 `int lcd_setBackgroundImage ( IN char * file, IN int fileLen, IN int enable )`

Set Background Image You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

## Parameters

<i>file</i>	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
<i>fileLen</i>	Length of files
<i>enable</i>	TRUE = Use Background Image, FALSE = Use Background Color

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.249 int lcd\_setBacklight ( IN BYTE *backlightVal* )

## Set Backlight

Set backlight percentage. If the percent > 100, it will be rejected. If 0 < percent < 10, backlight percent will be set to 10. If percent == 0, backlight will be turned off

## Parameters

<i>backlightVal</i>	Backlight percent value to be sat
---------------------	-----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

12.2.4.250 int lcd\_setDisplayImage ( IN char \* *file*, IN int *fileLen*, IN int *posX*, IN int *posY*, IN int *posMode*, IN int *touchEnable*, IN int *clearScreen* )

Set Display Image You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

## Parameters

<i>file</i>	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
<i>fileLen</i>	Length of files
<i>posX</i>	X coordinate in pixels, Range 0-271
<i>posY</i>	Y coordinate in pixels, Range 0-479
<i>posMode</i>	Position Mode <ul style="list-style-type: none"> <li>• 0 = Center on Line Y</li> <li>• 1 = Display at (X, Y)</li> <li>• 2 - Center on screen</li> </ul>

<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>clearScreen</i>	TRUE = Clear screen

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.251 int lcd\_setForeBackColor ( IN BYTE \* *foreRGB*, IN int *foreRGBLen*, IN BYTE \* *backRGB*, IN int *backRGBLen* )

Set Foreground and Background Color This command sets the foreground and background colors of the LCD.

## Parameters

<i>foreRGB</i>	Foreground RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>Length</i>	of foreRGB. Must be 3.
<i>backRGB</i>	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>Length</i>	of backRGB. Must be 3.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.252 int lcd\_showScreen ( IN char \* *screenName*, IN int *screenNameLen* )

## Show Screen

Displays and makes active a previously created screen.

## Parameters

<i>screenName</i>	Screen to display. The screen name is defined with <code>lcd_createScreen</code> .
<i>screenNameLen</i>	Length of <i>screenName</i> .

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.2.4.253 int lcd\_startSlideShow ( IN char \* *files*, IN int *filesLen*, IN int *posX*, IN int *posY*, IN int *posMode*, IN int *touchEnable*, IN int *recursion*, IN int *touchTerminate*, IN int *delay*, IN int *loops*, IN int *clearScreen* )

Start slide show You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to this commands. Image files must be in .bmp or .png format.

## Parameters

<i>files</i>	Complete paths and file names of the files you want to use, separated by commas. If a directory is specified, all files in the directory are displayed
<i>filesLen</i>	Length of <i>files</i>
<i>posX</i>	X coordinate in pixels, Range 0-271
<i>posY</i>	Y coordinate in pixels, Range 0-479

<i>posMode</i>	Position Mode <ul style="list-style-type: none"> <li>• 0 = Center on Line Y</li> <li>• 1 = Display at (X, Y)</li> <li>• 2 - Center on screen</li> </ul>
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>recursion</i>	TRUE = Recursively follow directorys in list
<i>touchTerminate</i>	TRUE = Terminate slideshow on touch (if touch enabled)
<i>delay</i>	Number of seconds between image displays
<i>loops</i>	Number of display loops. A zero indicates continuous display
<i>clearScreen</i>	TRUE = Clear screen

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.254 int lcd\_storeScreenInfo ( )

**Store Screen Info**

Store all current screen information from RAM to flash

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.255 int lcd\_updateColor ( IN char \* *screenName*, IN int *screenNameLen*, IN char \* *objectName*, IN int *objectNameLen*, IN BYTE \* *color*, IN int *colorLen* )

**Update Color**

Updates the component color, or updates the LED colors if specifying LCD component

**Parameters**

<i>screenName</i>	Screen name that will be the target of update color
<i>screenNameLen</i>	Length of screenName.
<i>objecName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>color</i>	Non LCD Widget: Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000 <ul style="list-style-type: none"> <li>• Byte 0 = B</li> <li>• Byte 1 = G</li> <li>• Byte 2 = R</li> <li>• Byte 3 = Reserved LCD Widget: Four bytes for LED color, byte 0 = LED 0, byte 1 = LED 1, byte 2 = LED2, byte 3 = LED 3</li> <li>• Value 0 = LED OFF</li> <li>• Value 1 = LED Green</li> <li>• Value 2 = LED Yellow</li> <li>• Value 3 = LED Red</li> </ul>
<i>colorLen</i>	Length of color.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.256 int lcd\_updateLabel ( IN char \* *screenName*, IN int *screenNameLen*, IN char \* *objectName*, IN int *objectNameLen*, IN char \* *label*, IN int *labelLen* )

## Update Label

Updates the component label.

## Parameters

<i>screenName</i>	Screen name that will be the target of update label
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>label</i>	Label to show on the component
<i>labelLen</i>	Length of label.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.2.4.257 int lcd\_updatePosition ( IN char \* *screenName*, IN int *screenNameLen*, IN char \* *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *new\_xCord*, IN int *new\_yCord* )

## Update Position

Updates the component position.

## Parameters

<i>screenName</i>	Screen Name that will be the target of update position
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>alignment</i>	Alignment for the target <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>

<i>new_xCord</i>	x-coordinate for Text, range 0-271
<i>new_yCord</i>	y-coordinate for Text, range 0-479

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.2.4.258 int loyalty\_cancelTransaction ( )**

Cancel Loyalty Transaction Only for VP6800

Cancels the currently executing transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.2.4.259 int loyalty\_cancelTransactionSilent ( int enable )**

Cancel Loyalty Transaction Silent Only for VP6800

Cancel transaction with or without showing the LCD message

**Parameters**

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

**Returns**

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

**12.2.4.260 void loyalty\_registerCallBk ( pEMV\_callback pEMVf )**

To register the loyalty callback function to get the EMV processing response. (Pass NULL to disable the callback.)  
Only for VP6800

**12.2.4.261 int loyalty\_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \* tags, IN int tagsLen, IN const int cardType, IN const int iccReadType )**

Start Loyalty Transaction Request Only for VP6800

Authorizes the transaction for an MSR/ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>amount</i>	Transaction amount value (tag value 9F02)  • SEE IMPORTANT NOTE BELOW
---------------	---

<i>amtOther</i>	Other amount value, if any (tag value 9F03)  • SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

#### Parameters

<i>cardType</i>	Set available card to accept. 0 = MSR only. 1 = MSR and ICC.
<i>iccReadType</i>	In case of ICC reading, this is how to behave. 0 = Same as device_startTransaction 1 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of device\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1, 2, 3, 4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay



- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.2.4.262 int msr\_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.263 int msr\_captureMode ( int *isBufferMode*, int *withNotification* )

Set MSR Capture Mode.

For Non-SRED Augusta Only

Switch between Auto mode and Buffer mode. Auto mode only available on KB interface

##### Parameters

<i>isBufferMode</i>	<ul style="list-style-type: none"> <li>• 1: Enter into Buffer mode.</li> <li>• 0: Enter into Auto mode. KB mode only. Swipes automatically captured and sent to keyboard buffer</li> </ul>
<i>withNotification</i>	<ul style="list-style-type: none"> <li>• 1: with notification when swiped MSR data.</li> <li>• 0: without notification when swiped MSR data.</li> </ul>

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.264 int msr\_clearMSRData ( )

Clear MSR Data Clears the MSR Data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.265 int msr\_disable ( )

Disable MSR Disable MSR functions.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.266 int msr\_flushTrackData ( )

Flush Track Data Clears any track data being retained in memory by future PIN Block request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.2.4.267 int msr\_getClearPANID ( BYTE \* value )

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

## Parameters

<i>value</i>	4901 <Setting value>="". setting Value: Number of digits in clear. Values are char '0' - '6'
--------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

## 12.2.4.268 int msr\_getExpirationMask ( BYTE \* value )

Get MSR expiration date mask.

## Parameters

<i>value</i>	5001 <Setting value>="". setting Value: '0' = masked, '1' = not-masked
--------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

## 12.2.4.269 int msr\_getFunctionStatus ( int \* enable, int \* isBufferMode, int \* withNotification )

Get MSR Function Status.

Gets the MSR function status

## Parameters

<i>enable</i>	1 = MSR enabled, 0 = MSR disabled
<i>isBufferMode</i>	1 = buffer mode, 0 = auto mode

<i>withNotification</i>	1 = with notification, 0 = without notification
-------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.270 int msr\_getKeyFormatForICCDUKPT ( OUT BYTE \* *format* )****Get Key Format For DUKPT**

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

**Parameters**

<i>format</i>	Response returned from method: <ul style="list-style-type: none"><li>• 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded. (default)</li><li>• 'AES': Encrypted card data with AES if DUKPT Key had been loaded.</li><li>• 'NONE': No Encryption.</li></ul>
---------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.271 int msr\_getKeyTypeForICCDUKPT ( OUT BYTE \* *type* )****Get Key Type for DUKPT**

Specifies the key type used for ICC DUKPT encryption. This applies to both MSR and ICC.

**Parameters**

<i>type</i>	Response returned from method: <ul style="list-style-type: none"><li>• 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded. (default)</li><li>• 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.</li></ul>
-------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.272 int msr\_getMSRData ( OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )****Get MSR Data Reads the MSR Data buffer****Parameters**

<i>reData</i>	Card data
---------------	-----------

<i>reLen</i>	Card data length
--------------	------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.273 int msr\_getSwipeForcedEncryptionOption ( BYTE \* *option* )

Get MSR Swipe Forced Encryption Option.

**Parameters**

<i>option</i>	8401 <Setting value>="". Setting Value Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.274 int msr\_getSwipeMaskOption ( BYTE \* *option* )

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

**Parameters**

<i>option</i>	8601 <Setting value>="". Setting Value Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off
---------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.2.4.275 void msr\_registerCallBk ( pMSR\_callBack *pMSRf* )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.2.4.276 void msr\_registerCallBkp ( pMSR\_callBackp *pMSRf* )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.2.4.277 int msr\_setClearPANID ( BYTE *val* )

Set Clear PAN ID.

**Parameters**

<i>val</i>	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.
------------	--

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

**12.2.4.278 int msr\_setExpirationMask ( int *mask* )**

Set Expiration Masking

Sets the flag to mask the expiration date

**Parameters**

<i>mask</i>	TRUE = mask expiration
-------------	------------------------

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.279 int msr\_setKeyFormatForICCDUKPT ( IN BYTE *format* )**

Set Key Format for DUKPT

Sets how data will be encrypted, with either TDES or AES (if DUKPT key loaded) This applies to both MSR and ICC

**Parameters**

<i>format</i>	encryption Encryption Type <ul style="list-style-type: none"><li>• 00: Encrypt with TDES</li><li>• 01: Encrypt with AES</li></ul>
---------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.280 int msr\_setKeyTypeForICCDUKPT ( IN BYTE *type* )**

Set Key Type for DUKPT Key

Sets which key the data will be encrypted with, with either Data Key or PIN key (if DUKPT key loaded) This applies to both MSR and ICC

**Parameters**

<i>type</i>	Encryption Type <ul style="list-style-type: none"><li>• 00: Encrypt with Data Key</li><li>• 01: Encrypt with PIN Key</li></ul>
-------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.2.4.281 int msr\_setSwipeForcedEncryptionOption ( int *track1*, int *track2*, int *track3*, int *track3card0* )**

Set MSR Swipe Forced Encryption Option.

## Parameters

<i>track1</i>	Set track1 encryption to true or false.
<i>track2</i>	Set track2 encryption to true or false.
<i>track3</i>	Set track3 encryption to true or false.
<i>track3card0</i>	Set track3 card0 encryption to true or false.

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.282 `int msr_setSwipeMaskOption ( int track1, int track2, int track3 )`

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

## Parameters

<i>track1</i>	Set track1 mask to true or false.
<i>track2</i>	Set track2 mask to true or false.
<i>track3</i>	Set track3 mask to true or false.

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.283 `int msr_startMSRSwipe ( IN int timeout )`

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `MSR_callBack()`

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.2.4.284 `void parseMSRData ( IN BYTE * resData, IN int resLen, IN_OUT IDTMSRData * cardData )`

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

12.2.4.285 `void parsePINBlockData ( IN BYTE * resData, IN int resLen, IN_OUT IDTPINData * cardData )`

Parse the PIN block data from the buffer into IDTPINData structure

## Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

12.2.4.286 void parsePINData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTPINData \* *cardData* )

Parse the PIN data from the buffer into IDTPINData structure

## Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

12.2.4.287 int pin\_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.288 int pin\_capturePin ( IN int *timeout*, IN int *type*, IN char \* *PAN*, IN int *PANLen*, IN int *minPIN*, IN int *maxPIN*, IN char \* *message*, IN int *messageLen* )

Capture PIN

## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> <li>• 00h = MKSK to encrypt PIN, Internal PAN (from MSR)</li> <li>• 01h = DUKPT to encrypt PIN, Internal PAN (from MSR)</li> <li>• 10h = MKSK to encrypt PIN, External Plaintext PAN</li> <li>• 11h = DUKPT to encrypt PIN, External Plaintext PAN</li> <li>• 20h = MKSK to encrypt PIN, External Ciphertext PAN</li> <li>• 21h = DUKPT to encrypt PIN, External Ciphertext PAN</li> </ul>

<i>PAN</i>	Personal Account Number (if internal, value is 0)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message
<i>messageLen</i>	Length of message

#### Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

12.2.4.289 `int pin_capturePinExt ( IN int type, IN char * PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char * message, IN int messageLen, IN char * verify, IN int verifyLen )`

- Capture PIN Ext

#### Parameters

<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> <li>• 00h: MKSK to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 01h: DUKPT to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 10h: MKSK to encrypt PIN, External Plaintext PAN</li> <li>• 11h: DUKPT to encrypt PIN, External Plaintext PAN</li> <li>• 20h: MKSK to encrypt PIN, External Ciphertext PAN (for PIN pad only)</li> <li>• 21h: DUKPT to encrypt PIN, External Ciphertext PAN (for PIN pad only)</li> <li>• 80h: MKSK to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 81h: DUKPT to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 90h: MKSK to encrypt PIN, External Plaintext PAN, Verify PIN</li> <li>• 91h: DUKPT to encrypt PIN, External Plaintext PAN, Verify PIN</li> </ul>
-------------	--



<i>PAN</i>	Personal Account Number (if internal, value is 0)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>messageLen</i>	Length of 1st scenario LCD message, valid in 00h~21h (0~33).If no LCD message input, length is 00h, and display default msg "PLEASE ENTER PIN"
<i>verify</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>verifyLen</i>	Length of 2nd Scenario LCD message.valid in 00h~21h (0~33).This field is present only when PAN and Key Type has Verify PIN.If no LCD message input, length is 00h, and display default msg " ENTER PIN AGAIN"

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.2.4.290 int pin\_getEncryptedOnlinePIN ( IN int *keyType*, IN int *timeout* )

**Get Encrypted DUKPT PIN**

Requests PIN Entry for online authorization. PIN block and KSN returned in callback function DeviceState.- TransactionData with cardData.pin\_pinblock. A swipe must be captured first before this function can execute

**Parameters**

<i>keyType</i>	PIN block key type. Valid values 0, 3 for TDES, 4 for AES
<i>timeout</i>	PIN entry timeout

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.291 int pin\_getEncryptedPIN ( int *keyType*, char \* *PAN*, int *PANLen*, char \* *message*, int *messageLen*, int *timeout* )

**Get Encrypted PIN**

Requests PIN Entry

**Parameters**

<i>keyType</i>	<ul style="list-style-type: none"> <li>• 0x00- MKSK-TDES: External Plaintext PAN</li> <li>• 0x01- DUKPT-TDES: External Plaintext PAN</li> <li>• 0x10 MKSK-TDES: External Ciphertext PAN</li> <li>• 0x11 DUKPT-TDES: External Ciphertext PAN</li> </ul>
<i>PAN</i>	Account Number
<i>PANLen</i>	length of PAN
<i>message</i>	Message to display
<i>messageLen</i>	length of message
<i>timeout</i>	PIN entry timeout

**Returns**

RETURN\_CODE: Values can be parsed with [errorCode.getErrorString\(\)](#)

#### 12.2.4.292 int pin\_getFunctionKey ( int timeout )

##### Get Function Key

Captures a function key

```
- Backspace = B
- Cancel = C
- Enter = E
- * = *
- # = #
- Help = ?
- Function Key 1 = F1
- Function Key 2 = F2
- Function Key 3 = F3

@param timeout Timeout, in seconds
```

##### Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

#### 12.2.4.293 int pin\_getPAN ( IN int getCSC, IN int timeout )

##### Get PAN

Requests PAN Entry on pinpad

##### Parameters

<i>getCSC</i>	Include Customer Service Code (also known as CVV, CVC)
<i>timeout</i>	PAN entry timeout

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.294 int pin\_getPanEntry ( IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly )

##### Get Pan

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

##### Parameters

<i>csc</i>	Request CSS
<i>expDate</i>	Request Expiration Date
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>mod10CK</i>	Validate entered PAN passes MOD-10 checking before accepting
<i>timeout</i>	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
<i>encPANOnly</i>	Output only encrypted PAN

##### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.2.4.295 `int pin_getPIN ( IN int mode, IN int PANSource, IN char * iccPAN, IN int iccPANLen, IN int startTimeout, IN int entryTimeout, IN char * language, IN int languageLen )`

Get Encrypted PIN

Requests PIN Entry

Parameters

<i>mode</i>	<ul style="list-style-type: none"> <li>• 0x00- Cancel: Cancels PIN entry = also can execute <a href="#">pin_cancelPINEntry()</a>. All other parameters for this method will be ignored</li> <li>• 0x01- Online PIN DUKPT</li> <li>• 0x02- Online PIN MKSK</li> <li>• 0x03- Offline PIN (No need to define PAN Source or ICC PAN)</li> </ul>
<i>PANSource</i>	<ul style="list-style-type: none"> <li>• 0x00- ICC: PAN Captured from ICC and must be provided in iccPAN parameter</li> <li>• 0x01- MSR: PAN Captured from MSR swipe and will be inserted by Spectrum Pro. No need to provide iccPAN parameter.</li> </ul>
<i>iccPAN</i>	PAN captured from ICC. When PAN captured from MSR, this parameter will be ignored
<i>iccPANLen</i>	the length of iccPAN
<i>startTimeout</i>	The amount of time allowed to start PIN entry before timeout
<i>entryTimeout</i>	The amount of time to enter the PIN after first digit selected before timeout
<i>language</i>	Valid values "EN" for English, "ES" for Spanish, "ZH" for Chinese, "FR" for French
<i>languageLen</i>	the length of language

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.296 `int pin_inputFromPrompt ( BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char * lang, BYTE promptID, char * defaultResponse, int defaultResponseLen, int timeout )`

Get PIN Input from Prompt Results returned to PIN Callback. If successful function key capture, data is returned as IDTPINData.keyString.

## Parameters

<i>mask</i>	0 = no masking, 1 = Display "*" for numeric key according to Pre-Cleartext and Post-Cleartext display
<i>preClearText</i>	Range 0-6 Characters to mask at start of text if masking is on;
<i>postClearText</i>	Range 0-6 Characters to mask at end of text if masking is on;
<i>minLen</i>	Minimum number of digits required to input
<i>maxLen</i>	Maximum number of digits allowed to be input
<i>lang</i>	Valid values; "EN" is English display message "JP" is Japanese display message "ES" is Spanish display message "FR" is French display message "ZH" is Chinese display message "PT" is Portuguese display message
<i>promptID</i>	Valid values: 0x00: Enter Phone Number 0x01: Enter IP Address 0x02: Enter Subnet Mask 0x03: Enter Default Gateway 0x04: Enter Odometer Reading/Mileage 0x05: Enter Employee ID 0x06: Enter Password for Default Factory Setting 0x07: Enter Email Address (Full keyboard)
<i>defaultResponse</i>	Default String on input field
<i>defaultResponseLen</i>	Length of defaultResponse
<i>timeout</i>	Timeout, in seconds

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

12.2.4.297 `int pin_promptCreditDebit ( IN char * currencySymbol, IN int currencySymbolLen, IN char * displayAmount, IN int displayAmountLen, IN int timeout, OUT BYTE * retData, IN_OUT int * retDataLen )`

## Prompt for Credit or Debit

Requests prompt for Credit or Debit. Response returned in callback function as `DeviceState.MenuItem` with data `MENU_SELECTION_CREDIT = 0`, `MENU_SELECTION_DEBIT = 1`

## Parameters

<i>currencySymbol</i>	Allowed values are \$ (0x24), ???(0xA5), ???(0xA3), ???(0xA4), or NULL
<i>currencySymbolLen</i>	length of currencySymbol
<i>displayAmount</i>	Amount to display (can be NULL)
<i>displayAmountLen</i>	length of displayAmount
<i>timeout</i>	Menu entry timeout. Valid values 2-20 seconds
<i>retData</i>	If successful, the return code is zero and the data is 1 byte (0: Credit 1: Debit). If the return code is not zero, it may be a four-byte Extended Status Code
<i>currencySymbolLen</i>	length of currencySymbol

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.2.4.298 `int pin_promptForAmount ( IN int timeout, IN int minLen, IN int maxLen, IN char * message, IN int messageLen, BYTE * signature, IN int signatureLen )`

## Capture Amount

## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	LCD Message
<i>messageLen</i>	Length of message
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> <li>1. Calculate 32 bytes Hash for ?斤??Display Flag&gt;&lt;Key max="" length=""&gt;=""&gt;&lt; Key Min Length&gt;&lt;Plaintext display="" message=""&gt;=""&gt;?斤??</li> <li>2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data</li> <li>3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature</li> </ol>
<i>signatureLen</i>	Length of signature

## Returns

RETURN\_CODE: Values can be parsed with errorCode.getErrorString()

12.2.4.299 int pin\_promptForAmountInput ( int messageID, int languageID, int minLen, int maxLen, int timeout )

## Prompt for Amount Input

Prompts for amount input using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	N?MERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN?A	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N?MERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP N?MERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE?CULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR

17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	N?MERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N?MERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N?MERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN?A	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VE?CULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	N?MERO DO VE?CULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USU?RIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION?RIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N?MERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO

44	ODOMETER NUMBER	N?MERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE?CULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE?CULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER?NCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N?MERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGE- M	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUCT
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCTOR #	ENT # CONDUCTEUR
67	VEHICLE #	VE?CULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE?CULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N?MERO DEPT	NUMERO DEPTO	No DEPARTEMENT

72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N?MERO DE LICEN?A	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN?A #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN?A #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER N?MERO DE CART?O	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN?CIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N?MERO DE EMISS?O	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN?CIO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```

@param messageID Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param minLen Minimum input length. Cannot be less than 1
@param maxLen Maximum input length. Cannot be greater than 15
@param timeout Timeout value, in seconds

```

```

@return RETURN_CODE: Values can be parsed with errorCode.getErrorString()

```

#### 12.2.4.300 int pin\_promptForKeyInput ( int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout )

Prompt for Key Input

Prompts for a numeric key using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ



6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	N?MERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICEN?A	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N?MERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP N?MERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VE?CULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	N?MERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	N?MERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N?MERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN?A	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VE?CULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	N?MERO DO VE?CULO	VEHICULO NUMERO	No VEHICULE

36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USU?RIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION?RIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N?MERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	N?MERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE?CULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE?CULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER?NCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N?MERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGE-M	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT

62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCTOR #	ENT # CONDUCTEUR
67	VEHICLE #	VE?CULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE?CULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N?MERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N?MERO DE LICEN?A	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN?A #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN?A #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER N?MERO DE CART?O	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN?CIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N?MERO DE EMISS?O	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE IN?CIO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

@param messageID Message (1-90)

@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt

@param maskInput 1 = entry is masked with '\*', 0 = entry is displayed on keypad

@param minLen Minimum input length. Cannot be less than 1

@param maxLen Maximum input length. Cannot be greater than 16

@param timeout Timeout value, in seconds

@return RETURN\_CODE: Values can be parsed with errorCode.getErrorString()

12.2.4.301 int pin\_promptForNumericKey ( IN int *timeout*, IN int *maskInput*, IN int *minLen*, IN int *maxLen*, IN char \* *message*, IN int *messageLen*, BYTE \* *signature*, IN int *signatureLen* )

Capture Numeric Input

## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>maskInput</i>	<ul style="list-style-type: none"> <li>• 0 = Display numeric for numeric key on LCD</li> <li>• 1 = Display ?? for numeric key on LCD</li> </ul>
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	Plaintext Display Message. - 16 chars max
<i>messageLen</i>	Length of message
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> <li>1. Calculate 32 bytes Hash for ?&lt;Display Flag&gt;&lt;Key max="" length&gt;=""&gt;&lt; Key Min Length&gt;&lt;Plaintext display="" message&gt;=""&gt;?</li> <li>2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data</li> <li>3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature</li> </ol>
<i>signatureLen</i>	Length of signature

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

```
12.2.4.302 int pin_promptForNumericKeyWithSwipe ( IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char
* line1, IN int line1Len, IN char * line2, IN int line2Len, BYTE * signature, IN int signatureLen )
```

## Capture Numeric Input

## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>function</i>	<ul style="list-style-type: none"> <li>• 0x00 = Plaintext Input</li> <li>• 0x01 = Masked Input</li> <li>• 0x02 = Delayed Masking Input</li> <li>• 0x10 = Plaintext Input + MSR Active</li> <li>• 0x11 = Masked Input + MSR Active</li> <li>• 0x12 = Delayed Masking Input + MSR Active</li> </ul>
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>line1</i>	Line 1 of LCD Message - 16 chars max
<i>line1Len</i>	Length of line1
<i>line2</i>	Line 2 of LCD Message - 16 chars max
<i>line2Len</i>	Length of line2
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> <li>1. Calculate 32 bytes Hash for ?<math>\text{斤}</math>??Display Flag&gt;&lt;Key max="" length=""&gt;=""&gt;&lt; Key Min Length&gt;&lt;Plaintext display="" message=""&gt;=""&gt;?<math>\text{斤}</math>??</li> <li>2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data</li> <li>3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature</li> </ol>
<i>signatureLen</i>	Length of signature

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

#### 12.2.4.303 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

#### 12.2.4.304 int pin\_sendBeep ( int frequency, int duration )

Send Beep

Executes a beep request.

## Parameters

<i>frequency</i>	Frequency, range 200-20000Hz Not used for NEO 2 devices
<i>duration</i>	Duration, range 16-65535ms Not used for NEO 2 devices

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.2.4.305 int pin\_setKeyValues ( int mode )

Set Key Values

Set return key values on or off

## Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.306 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

#### 12.2.4.307 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

#### 12.2.4.308 int rs232\_device\_init ( int deviceType, int port\_number, int brate )

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

## Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

#### Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

## Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.2.4.309 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string



12.2.4.310 `int setAbsoluteLibraryPath ( const char * absoluteLibraryPath )`

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.2.4.311 `int ws_deleteSSLCert ( IN char * name, IN int nameLen )`

Delete SSL Certificate Deletes a SSL Certificate by name

## Parameters

<i>name</i>	Name of certificate to delete
<i>nameLen</i>	Certificate Name Length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.312 `int ws_getCertChainType ( OUT int * type )`

Get Certificate Chain Type Returns indicator for using test/production certificate chain

## Parameters

<i>type</i>	0 = test certificate chain, 1 = production certificate chain
-------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.313 `int ws_loadSSLCert ( IN char * name, IN int nameLen, IN char * dataDER, IN int dataDERLen )`

Load SSL Certificate Loads a SSL certificate

## Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.314 `int ws_requestCSR ( OUT RequestCSR * csr )`

Request CSR Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

## Parameters

<i>csr</i>	RequestCSR structure to return the data
------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.315 int ws\_revokeSSLCert ( IN char \* *name*, IN int *nameLen* )

Revoke SSL Certificate Revokes a SSL Certificate by name

## Parameters

<i>name</i>	Name of certificate to revoke
<i>nameLen</i>	Certificate Name Length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.2.4.316 int ws\_updateRootCertificate ( IN char \* *name*, IN int *nameLen*, IN char \* *dataDER*, IN int *dataDERLen*, IN char \* *signature*, IN int *signatureLen* )

Update Root Certificate Updates the root certificate

## Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length
<i>signature</i>	Future Root CA signed (RSASSA PSS SHA256) by current Root CA
<i>signature</i>	length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.3 Source\_C/libIDT\_KioskIII.h File Reference

KioskIII API.

```
#include "IDTDef.h"
```

## Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

## Typedefs

- typedef void(\* [pMessageHotplug](#) )(int, int)
- typedef void(\* [pSendDataLog](#) )(unsigned char \*, int)
- typedef void(\* [pReadDataLog](#) )(unsigned char \*, int)
- typedef void(\* [pEMV\\_callback](#) )(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)
- typedef void(\* [pMSR\\_callback](#) )(int, IDTMSRData)
- typedef void(\* [pMSR\\_callback](#) )(int, IDTMSRData \*)
- typedef void(\* [pPIN\\_callback](#) )(int, IDTPINData \*)
- typedef void(\* [pCMR\\_callback](#) )(int, IDTCMRData \*)
- typedef void(\* [pCSFS\\_callback](#) )(BYTE status)
- typedef void(\* [ftpComm\\_callback](#) )(int, int, int)
- typedef void(\* [httpComm\\_callback](#) )(BYTE \*, int)
- typedef void(\* [v4Comm\\_callback](#) )(BYTE, BYTE, BYTE \*, int)

## Functions

- void [registerHotplugCallBk](#) ([pMessageHotplug](#) pMsgHotplug)
- void [registerLogCallBk](#) ([pSendDataLog](#) pFSend, [pReadDataLog](#) pFRead)
- void [emv\\_registerCallBk](#) ([pEMV\\_callback](#) pEMVf)
- void [ctls\\_registerCallBk](#) ([pMSR\\_callback](#) pCTLSf)
- void [ctls\\_registerCallBkp](#) ([pMSR\\_callback](#) pCTLSf)
- void [pin\\_registerCallBk](#) ([pPIN\\_callback](#) pPINf)
- void [device\\_registerCameraCallBk](#) ([pCMR\\_callback](#) pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) ([pCSFS\\_callback](#) pCSFSf)
- char \* [SDK\\_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char \*absoluteLibraryPath)
- int [device\\_init](#) ()
- int [rs232\\_device\\_init](#) (int deviceType, int port\_number, int brate)
- int [device\\_setCurrentDevice](#) (int deviceType)
- int [device\\_close](#) ()
- void [device\\_getIDGStatusCodeString](#) (IN int returnCode, OUT char \*despcrition)
- int [device\\_isConnected](#) ()
- int [device\\_isAttached](#) (int deviceType)
- int [device\\_getFirmwareVersion](#) (OUT char \*firmwareVersion)
- int [device\\_getFirmwareVersion\\_Len](#) (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int [device\\_pingDevice](#) ()
- int [device\\_controlUserInterface](#) (IN BYTE \*values)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)

- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()
- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.3.1 Detailed Description

KioskIII API. KioskIII Global API methods.

### 12.3.2 Macro Definition Documentation

#### 12.3.2.1 #define IN

INPUT parameter.

#### 12.3.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.3.2.3 #define OUT

OUTPUT parameter.

### 12.3.3 Typedef Documentation

#### 12.3.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.3.3.2 `typedef void(* httpComm_callBack)(BYTE *, int)`

Define the comm callback function to get the async url data

It should be registered using the `comm_registerHTTPCallback`

#### 12.3.3.3 `typedef void(* pCMR_callBack)(int, IDTCMRData *)`

Define the camera callback function to get the image data

It should be registered using the `device_registerCameraCallBk`,

#### 12.3.3.4 `typedef void(* pCSFS_callBack)(BYTE status)`

Define the card status and front switch callback function to get card and front switch status

It should be registered using the `device_registerCardStatusFrontSwitchCallBk`,

#### 12.3.3.5 `typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the `emv_registerCallBk`,

#### 12.3.3.6 `typedef void(* pMessageHotplug)(int, int)`

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.3.3.7 `typedef void(* pMSR_callBack)(int, IDTMSRData)`

Define the MSR callback function to get the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

#### 12.3.3.8 `typedef void(* pMSR_callBackp)(int, IDTMSRData *)`

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

#### 12.3.3.9 `typedef void(* pPIN_callBack)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

#### 12.3.3.10 `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

#### 12.3.3.11 typedef void(\* pSendDataLog)(unsigned char \*, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

#### 12.3.3.12 typedef void(\* v4Comm\_callback)(BYTE, BYTE, BYTE \*, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

### 12.3.4 Function Documentation

#### 12.3.4.1 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

#### 12.3.4.2 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

#### 12.3.4.3 int ctls\_activateTransaction ( IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of ctls\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DFO1 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

**12.3.4.4 int ctls\_cancelTransaction ( )**

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



**12.3.4.5 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )**

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

**Parameters**

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.6 int ctls\_getConfigurationGroup ( IN int group, OUT BYTE \* tlv, OUT int \* tlvLen )**

Get Configuration Group

Retrieves the Configuration for the specified Group.

**Parameters**

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.7 void ctls\_registerCallBk ( pMSR\_callBack pCTLSf )**

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

**12.3.4.8 void ctls\_registerCallBkp ( pMSR\_callBackp pCTLSf )**

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

**12.3.4.9 int ctls\_removeAllApplicationData ( )**

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.10 int ctls\_removeAllCAPK ( )**

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.11 int `ctls_removeApplicationData` ( IN BYTE \* *AID*, IN int *AIDLen* )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.12 int ctls\_removeCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.13 int ctls\_removeConfigurationGroup ( int *group* )

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

## Parameters

<i>group</i>	Configuration Group
--------------	---------------------

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.3.4.14 int ctls\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.15 int ctls\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.16 int ctls\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>

<i>keyLen</i>	the length of key data buffer •
---------------	------------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.17 int ctls\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keyLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keyLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.18 int ctls\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.19 int ctls\_setApplicationData ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

**Parameters**

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.20 int ctls\_setCAPK ( IN BYTE \* capk, IN int capkLen )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

##### Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.21 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

##### Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.22 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.3.4.23 `int ctls_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal

- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.3.4.24 int device\_close ( )

Close the device

##### Returns

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.3.4.25 int device\_controlUserInterface ( IN BYTE \* values )

##### Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
```



```

- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On

```

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.26 int device\_enablePassThrough ( int *enablePassThrough* )****Start Remote Key Injection**

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

**Enable Pass Through - NEO**

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

**Parameters**

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.27 int device\_getCurrentDeviceType ( )****Get current active device type****Returns**

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

**12.3.4.28 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )**

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

**Parameters**

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.29 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )**

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.30 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.3.4.31 int device\_getMerchantRecord ( IN int *index*, OUT BYTE \* *record* )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int *index*, OUT BYTE \* *record*, IN\_OUT int \**recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i> ;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

See Also

ErrorCode

#### 12.3.4.32 int device\_getMerchantRecord\_Len ( IN int *index*, OUT BYTE \* *record*, IN\_OUT int \* *recordLen* )

Get Merchant Record

Gets the merchant record for the device.

**Parameters**

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**See Also**

[ErrorCode](#)

**12.3.4.33 int device\_getSDKWaitTime ( )**

Get SDK Wait Time

Get the SDK wait time for transactions

**Returns**

SDK wait time in seconds

**12.3.4.34 int device\_getThreadStackSize ( )**

Get Thread Stack Size

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.3.4.35 int device\_getTransactionResults ( IDTMSRData \* *cardData* )**

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

**Parameters**

<i>cardData</i>	The transaction results
-----------------	-------------------------

**Returns**

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

**See Also**

[ErrorCode](#)

**12.3.4.36 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.37 int device\_isAttached ( int *deviceType* )**

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

**Parameters**

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

**Returns**

1 if the device is attached, or 0 if the device is not attached

**12.3.4.38 int device\_isConnected ( )**

Check the device connctected status

**Returns**

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

**12.3.4.39 int device\_pingDevice ( )**

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.40 void device\_registerCameraCallBk ( pCMR\_callback *pCMRf* )**

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

**12.3.4.41 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback *pCSFSf* )**

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

**12.3.4.42 int device\_SendDataCommandNEO ( IN int *cmd*, IN int *subCmd*, IN BYTE \* *data*, IN int *dataLen*, OUT BYTE \* *response*, IN\_OUT int \* *respLen* )**

Send a Command to NEO device

Sends a command to the NEO device .

**Parameters**

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.3.4.43 int device\_setBurstMode ( IN BYTE mode )****Send Burst Mode - NEO**

Sets the burst mode for the device.

**Parameters**

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

**See Also**

[ErrorCode](#)

**12.3.4.44 int device\_setCurrentDevice ( int deviceType )**

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.3.4.45 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record - NEO Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.3.4.46 int device\_setPollMode ( IN BYTE *mode* )

Set Poll Mode - NEO

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

## Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.3.4.47 void device\_setSDKWaitTime ( int waitTime )

## Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

## 12.3.4.48 void device\_setThreadStackSize ( int threadSize )

## Set Thread Stack Size

Set the stack size setting for newly created threads

## 12.3.4.49 void emv\_registerCallBk ( pEMV\_callback pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

## 12.3.4.50 void parseMSRData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTMSRData \* cardData )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

## 12.3.4.51 void pin\_registerCallBk ( pPIN\_callback pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

## 12.3.4.52 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

## 12.3.4.53 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.



12.3.4.54 `int rs232_device_init ( int deviceType, int port_number, int brate )`

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

## Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

## Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

## Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.3.4.55 `char* SDK_Version ( )`

To Get SDK version

## Returns

return the SDK version string

12.3.4.56 `int setAbsoluteLibraryPath ( const char * absoluteLibraryPath )`

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.4 Source\_C/libIDT\_L100.h File Reference

L100 API.

```
#include "IDTDef.h"
```

## Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

## Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callBack )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack )(int, IDTMSRData)`
- `typedef void(* pMSR_callBackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callBack )(int, IDTPINData *)`
- `typedef void(* pCMR_callBack )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack )(BYTE status)`
- `typedef void(* pFW_callBack )(int, int, int, int, int)`
- `typedef void(* ftpComm_callBack )(int, int, int)`
- `typedef void(* httpComm_callBack )(BYTE *, int)`
- `typedef void(* v4Comm_callBack )(BYTE, BYTE, BYTE *, int)`

## Functions

- void `registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- void `registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- void `emv_registerCallBk (pEMV_callBack pEMVf)`
- void `msr_registerCallBk (pMSR_callBack pMSRf)`
- void `msr_registerCallBkp (pMSR_callBackp pMSRf)`
- void `pin_registerCallBk (pPIN_callBack pPINf)`
- void `device_registerCameraCallBk (pCMR_callBack pCMRf)`
- void `device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- void `device_registerFWCallBk (pFW_callBack pFWf)`
- char \* `SDK_Version ()`
- int `setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- int `device_init ()`
- int `device_setCurrentDevice (int deviceType)`
- int `device_close ()`
- void `device_getResponseCodeString (IN int returnCode, OUT char *despcrition)`
- int `device_isConnected ()`
- int `device_isAttached (int deviceType)`
- int `device_getFirmwareVersion (OUT char *firmwareVersion)`
- int `device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`
- int `device_getDateTime (OUT BYTE *dateTime)`
- int `device_getDateTime_Len (OUT BYTE *dateTime, IN_OUT int *dateTimeLen)`
- int `device_getCurrentDeviceType ()`
- int `device_SendDataCommand (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)`
- int `device_updateFirmware (IN BYTE *firmwareData, IN int firmwareDataLen, IN char *firmwareName, IN int encryptionType, IN BYTE *keyBlob, IN int keyBlobLen)`
- int `device_rebootDevice ()`
- int `device_getKeyStatus (int *newFormat, BYTE *status, int *statusLen)`
- int `device_enterStopMode ()`
- int `device_setSleepModeTime (int time)`

- int [config\\_getModelNumber](#) (OUT char \*sNumber)
- int [config\\_getModelNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [pin\\_getEncryptedPIN](#) (int keyType, char \*PAN, int PANLen, char \*message, int messageLen, int timeout)
- int [pin\\_promptForKeyInput](#) (int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout)
- int [pin\\_promptForAmountInput](#) (int messageID, int languageID, int minLen, int maxLen, int timeout)
- int [pin\\_getFunctionKey](#) (int timeout)
- int [pin\\_sendBeep](#) (int frequency, int duration)
- int [pin\\_setKeyValues](#) (int mode)
- int [lcd\\_savePrompt](#) (int promptNumber, char \*prompt, int promptLen)
- int [lcd\\_displayPrompt](#) (int promptNumber, int lineNumber)
- int [lcd\\_displayMessage](#) (int lineNumber, char \*message, int messageLen)
- int [lcd\\_enableBacklight](#) (int enable)
- int [lcd\\_getBacklightStatus](#) (int \*enabled)

### 12.4.1 Detailed Description

L100 API. L100 Global API methods.

### 12.4.2 Macro Definition Documentation

#### 12.4.2.1 #define IN

INPUT parameter.

#### 12.4.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.4.2.3 #define OUT

OUTPUT parameter.

### 12.4.3 Typedef Documentation

#### 12.4.3.1 typedef void(\* ftpComm\_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.4.3.2 typedef void(\* httpComm\_callBack)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

#### 12.4.3.3 `typedef void(* pCMR_callback)(int, IDTCMRData *)`

Define the camera callback function to get the image data

It should be registered using the `device_registerCameraCallBk`,

#### 12.4.3.4 `typedef void(* pCSFS_callback)(BYTE status)`

Define the card status and front switch callback function to get card and front switch status

It should be registered using the `device_registerCardStatusFrontSwitchCallBk`,

#### 12.4.3.5 `typedef void(* pEMV_callback)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the `emv_registerCallBk`,

#### 12.4.3.6 `typedef void(* pFW_callback)(int, int, int, int, int)`

Define the firmware update callback function to get the status of firmware update

It should be registered using the `device_registerFWCallBk`,

#### 12.4.3.7 `typedef void(* pMessageHotplug)(int, int)`

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.4.3.8 `typedef void(* pMSR_callback)(int, IDTMSRData)`

Define the MSR callback function to get the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

#### 12.4.3.9 `typedef void(* pMSR_callbackp)(int, IDTMSRData *)`

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callback`

#### 12.4.3.10 `typedef void(* pPIN_callback)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

#### 12.4.3.11 `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

#### 12.4.3.12 typedef void(\* pSendDataLog)(unsigned char \*, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

#### 12.4.3.13 typedef void(\* v4Comm\_callback)(BYTE, BYTE, BYTE \*, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

### 12.4.4 Function Documentation

#### 12.4.4.1 int config\_getModelNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getModelNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.4.4.2 int config\_getModelNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.4.4.3 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.4.4.4 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.4.4.5 `int device_close ( )`

Close the device

## Returns

RETURN\_CODE: 0: success, 0x0A: failed

12.4.4.6 `int device_enterStopMode ( )`

Enter Stop Mode

Set device enter to stio mode. In stop mode, LCD display and backlight is off. Stop mode reduces power consumption to the lowest possible level. A unit in stop mode can only be woken up by a physical key press.

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.4.4.7 `int device_getCurrentDeviceType ( )`

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.4.4.8 `int device_getDateTime ( OUT BYTE * dateTime )`

Polls device for Date and Time

## Parameters

<i>dateTime</i>	Response returned of Date and Time; needs to have at least 6 bytes of memory
-----------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.4.4.9 `int device_getDateTime_Len ( OUT BYTE * dateTime, IN_OUT int * dateTimeLen )`

Polls device for Date and Time

## Parameters

<i>dateTime</i>	Response returned of Date and Time
<i>dateTime</i>	Length of Date and Time

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.4.4.10 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.4.4.11 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.4.4.12 int device\_getKeyStatus ( int \* *newFormat*, BYTE \* *status*, int \* *statusLen* )

Get Key Status

Gets the status of loaded keys

## Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len\_L Len\_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

#### Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

#### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.4.4.13 void device\_getResponseCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

#### Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

#### Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";



- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";

- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";

- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";

- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";

- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";

- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE\_OPEN\_FAILED";
- 0X1003: "FILE OPERATION\_FAILED";
- 0X2001: "MEMORY\_NOT\_ENOUGH";
- 0X3002: "SMARTCARD\_FAIL";
- 0X3003: "SMARTCARD\_INIT\_FAILED";
- 0X3004: "FALLBACK\_SITUATION";
- 0X3005: "SMARTCARD\_ABSENT";
- 0X3006: "SMARTCARD\_TIMEOUT";
- 0X3012: "EMV\_RESULT\_CODE\_MSR\_CARD\_ERROR\_FALLBACK";
- 0X5001: "EMV\_PARSING\_TAGS\_FAILED";
- 0X5002: "EMV\_DUPLICATE\_CARD\_DATA\_ELEMENT";
- 0X5003: "EMV\_DATA\_FORMAT\_INCORRECT";
- 0X5004: "EMV\_NO\_TERM\_APP";
- 0X5005: "EMV\_NO\_MATCHING\_APP";
- 0X5006: "EMV\_MISSING\_MANDATORY\_OBJECT";
- 0X5007: "EMV\_APP\_SELECTION\_RETRY";
- 0X5008: "EMV\_GET\_AMOUNT\_ERROR";
- 0X5009: "EMV\_CARD\_REJECTED";
- 0X5010: "EMV\_AIP\_NOT\_RECEIVED";

- 0X5011: "EMV\_AFL\_NOT\_RECEIVED";
- 0X5012: "EMV\_AFL\_LEN\_OUT\_OF\_RANGE";
- 0X5013: "EMV\_SFI\_OUT\_OF\_RANGE";
- 0X5014: "EMV\_AFL\_INCORRECT";
- 0X5015: "EMV\_EXP\_DATE\_INCORRECT";
- 0X5016: "EMV\_EFF\_DATE\_INCORRECT";
- 0X5017: "EMV\_ISS\_COD\_TBL\_OUT\_OF\_RANGE";
- 0X5018: "EMV\_CRYPTOGram\_TYPE\_INCORRECT";
- 0X5019: "EMV\_PSE\_NOT\_SUPPORTED\_BY\_CARD";
- 0X5020: "EMV\_USER\_SELECTED\_LANGUAGE";
- 0X5021: "EMV\_SERVICE\_NOT\_ALLOWED";
- 0X5022: "EMV\_NO\_TAG\_FOUND";
- 0X5023: "EMV\_CARD\_BLOCKED";
- 0X5024: "EMV\_LEN\_INCORRECT";
- 0X5025: "CARD\_COM\_ERROR";
- 0X5026: "EMV\_TSC\_NOT\_INCREASED";
- 0X5027: "EMV\_HASH\_INCORRECT";
- 0X5028: "EMV\_NO\_ARC";
- 0X5029: "EMV\_INVALID\_ARC";
- 0X5030: "EMV\_NO\_ONLINE\_COMM";
- 0X5031: "TRAN\_TYPE\_INCORRECT";
- 0X5032: "EMV\_APP\_NO\_SUPPORT";
- 0X5033: "EMV\_APP\_NOT\_SELECT";
- 0X5034: "EMV\_LANG\_NOT\_SELECT";
- 0X5035: "EMV\_NO\_TERM\_DATA";
- 0X5039: "EMV\_PIN\_ENTRY\_TIMEOUT";
- 0X6001: "CVM\_TYPE\_UNKNOWN";
- 0X6002: "CVM\_AIP\_NOT\_SUPPORTED";
- 0X6003: "CVM\_TAG\_8E\_MISSING";
- 0X6004: "CVM\_TAG\_8E\_FORMAT\_ERROR";
- 0X6005: "CVM\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6006: "CVM\_COND\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6007: "NO\_MORE\_CVM";
- 0X6008: "PIN\_BYPASSED\_BEFORE";
- 0X7001: "PK\_BUFFER\_SIZE\_TOO\_BIG";
- 0X7002: "PK\_FILE\_WRITE\_ERROR";

- 0X7003: "PK\_HASH\_ERROR";
- 0X8001: "NO\_CARD HOLDER\_CONFIRMATION";
- 0X8002: "GET\_ONLINE\_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";



- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBECA: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XBBEF: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";

- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

#### 12.4.4.14 int device\_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.4.4.15 int device\_isAttached ( int *deviceType* )

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

##### Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

##### Returns

1 if the device is attached, or 0 if the device is not attached

#### 12.4.4.16 int device\_isConnected ( )

Check the device connected status

##### Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

#### 12.4.4.17 int device\_rebootDevice ( )

Reboot Device Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.4.4.18 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.4.4.19 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.4.4.20 void device\_registerFWCallBk ( pFW\_callBack pFWf )

To register the firmware update callback function to get the status of firmware update. (Pass NULL to disable the callback.)

12.4.4.21 int device\_SendDataCommand ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.4.4.22 int device\_setCurrentDevice ( int deviceType )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VEND1,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

## 12.4.4.23 int device\_setSleepModeTime ( int time )

## Set Sleep Mode Timer

Set device enter to sleep mode after the given time. In sleep mode, LCD display and backlight is off. Sleep mode reduces power consumption to the lowest possible level. A unit in Sleep mode can only be woken up by a physical key press.

## Parameters

<i>time</i>	Enter sleep time value, in second.
-------------	------------------------------------

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.4.4.24 int device\_updateFirmware ( IN BYTE \* firmwareData, IN int firmwareDataLen, IN char \* firmwareName, IN int encryptionType, IN BYTE \* keyBlob, IN int keyBlobLen )

Update Firmware Updates the firmware of Augusta.

## Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"> <li>• For example "Augusta_S_TTK_V1.00.002.fm"</li> </ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"> <li>• 0 : Plaintext</li> <li>• 1 : TDES ECB, PKCS#5 padding</li> <li>• 2 : TDES CBC, PKCS#5, IV is all 0</li> </ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of keyBlob

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device-State.FirmwareUpdate data = File Progress. Two bytes, with `byte[0]` = current block, and `byte[1]` = total blocks. 0x0310 = block 3 of 16 `transactionResultCode`:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

#### 12.4.4.25 void emv\_registerCallBk ( pEMV\_callback pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

#### 12.4.4.26 int lcd\_displayMessage ( int lineNumber, char \* message, int messageLen )

Display Message on Line

Displays a message on a display line.

## Parameters

<i>lineNumber</i>	Line number to display message on (1-4)
<i>message</i>	Message to display
<i>messageLen</i>	length of message

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.4.4.27 int lcd\_displayPrompt ( int promptNumber, int lineNumber )

Display Prompt on Line

Displays a message prompt from L100 memory.

## Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>lineNumber</i>	Line number to display message prompt (1-4)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.4.4.28 int lcd\_enableBacklight ( int *enable* )

Enable/Disable LCD Backlight

Turns on/off the LCD back lighting.

## Parameters

<i>enable</i>	TRUE = turn ON backlight, FALSE = turn OFF backlight
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.4.4.29 int lcd\_getBacklightStatus ( int \* *enabled* )

Get Backlight Status

Returns the status of the LCD back lighting.

## Parameters

<i>enabled</i>	1 = Backlight is ON, 0 = Backlight is OFF
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.4.4.30 int lcd\_savePrompt ( int *promptNumber*, char \* *prompt*, int *promptLen* )

Save Prompt

Saves a message prompt to L100 memory.

## Parameters

<i>promptNumber</i>	Prompt number (0-9)
<i>prompt</i>	Prompt string (up to 20 characters)
<i>promptLen</i>	length of prompt

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.4.4.31 void msr\_registerCallBk ( pMSR\_callback *pMSRf* )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

## 12.4.4.32 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

## 12.4.4.33 int pin\_getEncryptedPIN ( int keyType, char \* PAN, int PANLen, char \* message, int messageLen, int timeout )

Get Encrypted PIN

Requests PIN Entry

Parameters

<i>keyType</i>	<ul style="list-style-type: none"> <li>• 0x00- MKSK-TDES: External Plaintext PAN</li> <li>• 0x01- DUKPT-TDES: External Plaintext PAN</li> <li>• 0x10 MKSK-TDES: External Ciphertext PAN</li> <li>• 0x11 DUKPT-TDES: External Ciphertext PAN</li> </ul>
<i>PAN</i>	Account Number
<i>PANLen</i>	length of PAN
<i>message</i>	Message to display
<i>messageLen</i>	length of message
<i>timeout</i>	PIN entry timeout

Returns

RETURN\_CODE: Values can be parsed with errorCode.getErrorString()

## 12.4.4.34 int pin\_getFunctionKey ( int timeout )

Get Function Key

Captures a function key

- Backspace = B
- Cancel = C
- Enter = E
- \* = \*
- # = #
- Help = ?
- Function Key 1 = F1
- Function Key 2 = F2
- Function Key 3 = F3

Parameters

<i>timeout</i>	Timeout, in seconds
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

12.4.4.35 `int pin_promptForAmountInput ( int messageId, int languageID, int minLen, int maxLen, int timeout )`

## Prompt for Amount Input

Prompts for amount input using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVOR REENTRAR	POR FAVOR REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	NUMERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICENÇA	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	NUMERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP NUMERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL
15	VEHICLE ID	ID VEICULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMENT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	NUMERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	NUMERO DE FROTA	FLOTA NUMERO	No PARC AUTO



25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	N <sup>筋</sup> ERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICEN <sup>斤</sup> 墩	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VE <sup>斤</sup> ULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	N <sup>筋</sup> ERO DO VE <sup>斤</sup> ULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USU 筋 <sup>秣</sup> IO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCION 筋 <sup>秣</sup> IO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER N <sup>筋</sup> ERO ID	INGRESE NUMERO ID	ENTREZ No ID
42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	N <sup>筋</sup> ERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE <sup>斤</sup> ULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE <sup>斤</sup> ULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA

51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER 斤 憐CIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N 筋ERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGE-M	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCTOR #	ENT # CONDUCTEUR
67	VEHICLE #	VE 斤ULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE 斤ULO #	INGRESE VEHICULO #	ENT # VEHICULE
69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	N 筋ERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	N 筋ERO DE LICEN 斤墩	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICEN 斤墩 #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICEN 斤墩 #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID

81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER N <sup>o</sup> ERO DE CART <sup>o</sup>	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YYMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE IN c <sup>o</sup>	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	N <sup>o</sup> ERO DE EMISS <sup>o</sup>	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE INc <sup>o</sup>	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```

@param messageID Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param minLen Minimum input length. Cannot be less than 1
@param maxLen Maximum input length. Cannot be greater than 15
@param timeout Timeout value, in seconds

```

```

@return RETURN_CODE: Values can be parsed with errorCode.getErrorString()

```

#### 12.4.4.36 int pin\_promptForKeyInput ( int messageID, int languageID, int maskInput, int minLen, int maxLen, int timeout )

Prompt for Key Input

Prompts for a numeric key using the secure message according to the following table

Msg Id	English Prompt	Portuguese Prompt	Spanish Prompt	French Prompt
1	ENTER	ENTER	INGRESE	ENTREZ
2	REENTER	RE-INTRODUZIR	REINGRESE	RE-ENTREZ
3	ENTER YOUR	INTRODUZIR O SEU	INGRESE SU	ENTREZ VOTRE
4	REENTER YOUR	RE-INTRODUZIR O SEU	REINGRESE SU	RE-ENTREZ VOTRE
5	PLEASE ENTER	POR FAVOR DIGITE	POR FAVOR INGRESE	SVP ENTREZ
6	PLEASE REENTER	POR FAVO REENTRAR	POR FAVO REINGRESE	SVP RE-ENTREZ
7	PO NUMBER	N <sup>o</sup> ERO PO	NUMERO PO	No COMMANDE
8	DRIVER ID	LICENc <sup>o</sup>	LICENCIA	ID CONDUCTEUR
9	ODOMETER	ODOMETER	ODOMETRO	ODOMETRE
10	ID NUMBER	N <sup>o</sup> ERO ID	NUMERO ID	No IDENT
11	EQUIP CODE	EQUIP CODE	CODIGO EQUIP	CODE EQUIPEMENT
12	DRIVERS ID	DRIVER ID	ID CONDUCTOR	ID CONDUCTEUR
13	JOB NUMBER	EMP N <sup>o</sup> ERO	NUMERO EMP	No TRAVAIL
14	WORK ORDER	TRABALHO ORDEM	ORDEN TRABAJO	FICHE TRAVAIL

15	VEHICLE ID	ID VEICULO	ID VEHICULO	ID VEHICULE
16	ENTER DRIVER	ENTER DRIVER	INGRESE CONDUCTOR	ENTR CONDUCTEUR
17	ENTER DEPT	ENTER DEPT	INGRESE DEPT	ENTR DEPARTEMNT
18	ENTER PHONE	ADICIONAR PHONE	INGRESE TELEFONO	ENTR No TELEPH
19	ENTER ROUTE	ROUTE ADD	INGRESE RUTA	ENTREZ ROUTE
20	ENTER FLEET	ENTER FROTA	INGRESE FLOTA	ENTREZ PARC AUTO
21	ENTER JOB ID	ENTER JOB ID	INGRESE ID TRABAJO	ENTR ID TRAVAIL
22	ROUTE NUMBER	NÚMERO PATH	RUTA NUMERO	No ROUTE
23	ENTER USER ID	ENTER USER ID	INGRESE ID USUARIO	ID UTILISATEUR
24	FLEET NUMBER	NÚMERO DE FROTA	FLOTA NUMERO	No PARC AUTO
25	ENTER PRODUCT	ADICIONAR PRODUTO	INGRESE PRODUCTO	ENTREZ PRODUIT
26	DRIVER NUMBER	NÚMERO DRIVER	CONDUCTOR NUMERO	No CONDUCTEUR
27	ENTER LICENSE	ENTER LICENÇA	INGRESE LICENCIA	ENTREZ PERMIS
28	ENTER FLEET NO	ENTER NRO FROTA	INGRESE NRO FLOTA	ENT No PARC AUTO
29	ENTER CAR WASH	WASH ENTER	INGRESE LAVADO	ENTREZ LAVE-AUTO
30	ENTER VEHICLE	ENTER VEICULO	INGRESE VEHICULO	ENTREZ VEHICULE
31	ENTER TRAILER	TRAILER ENTER	INGRESE TRAILER	ENTREZ REMORQUE
32	ENTER ODOMETER	ENTER ODOMETER	INGRESE ODOMETRO	ENTREZ ODOMETRE
33	DRIVER LICENSE	CARTEIRA DE MOTORISTA	LICENCIA CONDUCTOR	PERMIS CONDUIRE
34	ENTER CUSTOMER	ENTER CLIENTE	INGRESE CLIENTE	ENTREZ CLIENT
35	VEHICLE NUMBER	NÚMERO DO VEICULO	VEHICULO NUMERO	No VEHICULE
36	ENTER CUST DATA	ENTER CLIENTE INFO	INGRESE INFO CLIENTE	INFO CLIENT
37	REENTER DRIVID	REENTRAR DRIVER ID	REINGRESE ID CHOFE	RE-ENTR ID COND
38	ENTER USER DATA	ENTER INFO USUARIO	INGRESE INFO USUARIO	INFO UTILISATEUR
39	ENTER CUST CODE	ENTER CODE. CLIENTE	INGRESE COD. CLIENTE	ENTR CODE CLIENT
40	ENTER EMPLOYEE	ENTER FUNCIONARIO	INGRESE EMPLEADO	ENTREZ EMPLOYE
41	ENTER ID NUMBER	ENTER NÚMERO ID	INGRESE NUMERO ID	ENTREZ No ID

42	ENTER DRIVER ID	ENTER ID DRIVER	INGRESE ID CONDUCTOR	No CONDUCTEUR
43	ENTER FLEET PIN	ENTER PIN FROTA	INGRESE PIN DE FLOTA	NIP PARC AUTO
44	ODOMETER NUMBER	N <sup>筋</sup> ERO ODOMETER	ODOMETRO NUMERO	No ODOMETRE
45	ENTER DRIVER LIC	ENTER DRIVER LIC	INGRESE LIC CONDUCTOR	PERMIS CONDUIRE
46	ENTER TRAILER NO	NRO TRAILER ENTER	INGRESE NRO TRAILER	ENT No REMORQUE
47	REENTER VEHICLE	REENTRAR VE <sup>斤</sup> ULO	REINGRESE VEHICULO	RE-ENTR VEHICULE
48	ENTER VEHICLE ID	ENTER VE <sup>斤</sup> ULO ID	INGRESE ID VEHICULO	ENTR ID VEHICULE
49	ENTER BIRTH DATE	INSERIR DATA NAC	INGRESE FECHA NAC	ENT DT NAISSANCE
50	ENTER DOB MMDDYY	ENTER FDN MMDDYY	INGRESE FDN MMDDAA	NAISSANCE MMJJAA
51	ENTER FLEET DATA	ENTER FROTA INFO	INGRESE INFO DE FLOTA	INFO PARC AUTO
52	ENTER REFERENCE	ENTER REFER <sup>斤</sup> ENCIA	INGRESE REFERENCIA	ENTREZ REFERENCE
53	ENTER AUTH NUMBR	ENTER N <sup>筋</sup> ERO AUT	INGRESE NUMERO AUT	No AUTORISATION
54	ENTER HUB NUMBER	ENTER HUB NRO	INGRESE NRO HUB	ENTREZ No NOYAU
55	ENTER HUBOMETER	MEDIDA PARA ENTRAR HUB	INGRESE MEDIDO DE HUB	COMPTEUR NOYAU
56	ENTER TRAILER ID	TRAILER ENTER ID	INGRESE ID TRAILER	ENT ID REMORQUE
57	ODOMETER READING	QUILOMETRAGE-M	LECTURA ODOMETRO	LECTURE ODOMETRE
58	REENTER ODOMETER	REENTRAR ODOMETER	REINGRESE ODOMETRO	RE-ENT ODOMETRE
59	REENTER DRIV. ID	REENTRAR DRIVER ID	REINGRESE ID CHOFER	RE-ENT ID CONDUC
60	ENTER CUSTOMER ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
61	ENTER CUST. ID	ENTER CLIENTE ID	INGRESE ID CLIENTE	ENTREZ ID CLIENT
62	ENTER ROUTE NUM	ENTER NUM ROUTE	INGRESE NUM RUTA	ENT No ROUTE
63	ENTER FLEET NUM	FROTA ENTER NUM	INGRESE NUM FLOTA	ENT No PARC AUTO
64	FLEET PIN	FROTA PIN	PIN DE FLOTA	NIP PARC AUTO
65	DRIVER #	DRIVER #	CONDUCTOR #	CONDUCTEUR
66	ENTER DRIVER #	ENTER DRIVER #	INGRESE CONDUCTOR #	ENT # CONDUCTEUR
67	VEHICLE #	VE <sup>斤</sup> ULO #	VEHICULO #	# VEHICULE
68	ENTER VEHICLE #	ENTER VE <sup>斤</sup> ULO #	INGRESE VEHICULO #	ENT # VEHICULE

69	JOB #	TRABALHO #	TRABAJO #	# TRAVAIL
70	ENTER JOB #	ENTER JOB #	INGRESE TRABAJO #	ENTREZ # TRAVAIL
71	DEPT NUMBER	NÚMERO DEPT	NUMERO DEPTO	No DEPARTEMENT
72	DEPARTMENT #	DEPARTAMENTO #	DEPARTAMENTO #	DEPARTEMENT
73	ENTER DEPT #	ENTER DEPT #	INGRESE DEPTO #	ENT# DEPARTEMENT
74	LICENSE NUMBER	NÚMERO DE LICENÇA	NUMERO LICENCIA	No PERMIS
75	LICENSE #	LICENÇA #	LICENCIA #	# PERMIS
76	ENTER LICENSE #	ENTER LICENÇA #	INGRESE LICENCIA #	ENTREZ # PERMIS
77	DATA	INFO	INFO	INFO
78	ENTER DATA	ENTER INFO	INGRESE INFO	ENTREZ INFO
79	CUSTOMER DATA	CLIENTE INFO	INFO CLIENTE	INFO CLIENT
80	ID #	ID #	ID #	# ID
81	ENTER ID #	ENTER ID #	INGRESE ID #	ENTREZ # ID
82	USER ID	USER ID	ID USUARIO	ID UTILISATEUR
83	ROUTE #	ROUTE #	RUTA #	# ROUTE
84	ENTER ROUTE #	ADD ROUTE #	INGRESE RUTA #	ENTREZ # ROUTE
85	ENTER CARD NUM	ENTER NÚMERO DE CARTÃO	INGRESE NUM TARJETA	ENTREZ NO CARTE
86	EXP DATE(YMM)	VALIDADE VAL (AAMM)	FECHA EXP (AAMM)	DATE EXPIR(AAMM)
87	PHONE NUMBER	TELEFONE	NUMERO TELEFONO	NO TEL
88	CVV START DATE	CVV DATA DE INÍCIO	CVV FECHA INICIO	CVV DATE DE DEBUT
89	ISSUE NUMBER	NÚMERO DE EMISSÃO	NUMERO DE EMISION	NO DEMISSION
90	START DATE (MMYY)	DATA DE INÍCIO (AAMM)	FECHA INICIO (AAMM)	DATE DE DEBUT-AAMM

```

@param messageID Message (1-90)
@param languageID 0=English Prompt, 1=Portuguese Prompt, 2=Spanish Prompt, 3=French Prompt
@param maskInput 1 = entry is masked with '*', 0 = entry is displayed on keypad
@param minLen Minimum input length. Cannot be less than 1
@param maxLen Maximum input length. Cannot be greater than 16
@param timeout Timeout value, in seconds

```

```

@return RETURN_CODE: Values can be parsed with errorCode.getErrorString()

```

#### 12.4.4.37 void pin\_registerCallBk ( pPIN\_callback pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

#### 12.4.4.38 int pin\_sendBeep ( int frequency, int duration )

Send Beep

Executes a beep request.

## Parameters

<i>frequency</i>	Frequency, range 200-20000Hz Not used for NEO 2 devices
<i>duration</i>	Duration, range 16-65535ms Not used for NEO 2 devices

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

## 12.4.4.39 int pin\_setKeyValues ( int mode )

## Set Key Values

Set return key values on or off

## Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.4.4.40 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

## 12.4.4.41 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

## 12.4.4.42 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

## 12.4.4.43 int setAbsoluteLibraryPath ( const char \* absoluteLibraryPath )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.5 Source\_C/libIDT\_MiniSmartII.h File Reference

MiniSmartII API.

```
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackk )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

### Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callback pEMVf)`
- `void msr_registerCallBk (pMSR_callback pMSRf)`
- `void msr_registerCallBkp (pMSR_callbackk pMSRf)`
- `void pin_registerCallBk (pPIN_callback pPINf)`
- `void device_registerCameraCallBk (pCMR_callback pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callback pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callback cBack)`
- `void comm_registerV4Callback (v4Comm_callback cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int rs232_device_init (int deviceType, int port_number, int brate)`
- `int device_setCurrentDevice (int deviceType)`
- `int device_close ()`
- `void device_getResponseCodeString (IN int returnCode, OUT char *despcriton)`
- `int device_isConnected ()`
- `int device_isAttached (int deviceType)`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`
- `int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`
- `int device_getCurrentDeviceType ()`
- `int device_SendDataCommand (IN BYTE *cmd, IN int cmdLen, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)`



- int [device\\_updateFirmware](#) (IN BYTE \*firmwareData, IN int firmwareDataLen, IN char \*firmwareName, IN int encryptionType, IN BYTE \*keyBlob, IN int keyBlobLen)
- int [device\\_rebootDevice](#) ()
- int [device\\_controlLED](#) (byte indexLED, byte control, int intervalOn, int intervalOff)
- int [device\\_controlLED\\_ICC](#) (int controlMode, int interval)
- int [device\\_controlLED\\_MSR](#) (byte control, int intervalOn, int intervalOff)
- int [device\\_controlBeep](#) (int index, int frequency, int duration)
- int [device\\_getKeyStatus](#) (int \*newFormat, BYTE \*status, int \*statusLen)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [config\\_getModelNumber](#) (OUT char \*sNumber)
- int [config\\_getModelNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_setLEDController](#) (int firmwareControlMSRLED, int firmwareControlICCLEd)
- int [config\\_getLEDController](#) (int \*firmwareControlMSRLED, int \*firmwareControlICCLEd)
- int [config\\_setBeeperController](#) (int firmwareControlBeeper)
- int [config\\_getBeeperController](#) (int \*firmwareControlBeeper)
- int [config\\_setEncryptionControl](#) (int msr, int icc)
- int [config\\_getEncryptionControl](#) (int \*msr, int \*icc)
- int [icc\\_enable](#) (IN int withNotification)
- int [icc\\_disable](#) ()
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [icc\\_exchangeEncryptedAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [icc\\_getAPDU\\_KSN](#) (OUT BYTE \*KSN, IN\_OUT int \*inLen)
- int [icc\\_getFunctionStatus](#) (OUT int \*enabled, OUT int \*withNotification)
- int [icc\\_getICCReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_getKeyFormatForICCDUKPT](#) (OUT BYTE \*format)
- int [icc\\_getKeyTypeForICCDUKPT](#) (OUT BYTE \*type)
- int [emv\\_getEMVKernelVersion](#) (OUT char \*version)
- int [emv\\_getEMVKernelVersion\\_Len](#) (OUT char \*version, IN\_OUT int \*versionLen)
- int [emv\\_getEMVKernelCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- int [emv\\_getEMVConfigurationCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- void [emv\\_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv\\_setAutoCompleteTransaction](#) (IN int complete)
- int [emv\\_getAutoAuthenticateTransaction](#) ()
- int [emv\\_getAutoCompleteTransaction](#) ()
- void [emv\\_allowFallback](#) (IN int allow)
- int [emv\\_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveTransactionResult](#) (IN BYTE \*tags, IN int tagsLen, IDTTransactionData \*cardData)
- int [emv\\_callbackResponseLCD](#) (IN int type, byte selection)
- int [emv\\_callbackResponseMSR](#) (IN BYTE \*MSR, IN\_OUT int MSRLen)
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)

- int `emv_removeApplicationData` (IN BYTE \*AID, IN int AIDLen)
- int `emv_removeAllApplicationData` ()
- int `emv_retrieveAIDList` (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int `emv_retrieveTerminalData` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setTerminalData` (IN BYTE \*tlv, IN int tlvLen)
- int `emv_removeTerminalData` ()
- int `emv_retrieveCAPK` (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int `emv_setCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeAllCAPK` ()
- int `emv_retrieveCAPKList` (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int `emv_retrieveTerminalID` (OUT char \*terminalID)
- int `emv_retrieveTerminalID_Len` (OUT char \*terminalID, IN\_OUT int \*terminalIDLen)
- int `emv_setTerminalID` (IN char \*terminalID)
- int `emv_retrieveCRL` (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int `emv_setCRL` (IN BYTE \*list, IN int lsLen)
- int `emv_removeCRL` (IN BYTE \*list, IN int lsLen)
- int `emv_removeAllCRL` ()

### 12.5.1 Detailed Description

MiniSmartII API. MiniSmartII Global API methods.

### 12.5.2 Macro Definition Documentation

#### 12.5.2.1 #define IN

INPUT parameter.

#### 12.5.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.5.2.3 #define OUT

OUTPUT parameter.

### 12.5.3 Typedef Documentation

#### 12.5.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.5.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the `comm_registerHTTPCallback`

**12.5.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)**

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

**12.5.3.4 typedef void(\* pCSFS\_callback)(BYTE status)**

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

**12.5.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)**

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

**12.5.3.6 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.5.3.7 typedef void(\* pMSR\_callback)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.5.3.8 typedef void(\* pMSR\_callbackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callback

**12.5.3.9 typedef void(\* pPIN\_callback)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.5.3.10 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

**12.5.3.11 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

#### 12.5.3.12 typedef void(\* v4Comm\_callback)(BYTE, BYTE, BYTE \*, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

### 12.5.4 Function Documentation

#### 12.5.4.1 void comm\_registerHTTPCallback ( httpComm\_callback cBack )

Register Comm HTTP Async Callback

##### Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

#### 12.5.4.2 void comm\_registerV4Callback ( v4Comm\_callback cBack )

Register External V4 Protocol commands Callback

##### Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

#### 12.5.4.3 int config\_getBeeperController ( int \* firmwareControlBeeper )

Get the Beeper Controller Status Set the Beeper controlled Status by software or firmware

##### Parameters

<i>firmwareControl-Beeper</i>	1 means firmware control the beeper, 0 means software control beeper.
-------------------------------	---

##### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.4 int config\_getEncryptionControl ( int \* msr, int \* icc )

Get Encryption Control

Get Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,
- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

##### Parameters

<i>msr</i>	<ul style="list-style-type: none"> <li>• 1: enabled MSR with Encryption,</li> <li>• 0: disabled MSR with Encryption,</li> </ul>
<i>icc</i>	<ul style="list-style-type: none"> <li>• 1: enabled ICC with Encryption,</li> <li>• 0: disabled ICC with Encryption,</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.5 int config\_getLEDController ( int \* *firmwareControlMSRLED*, int \* *firmwareControlICCLEd* )

Get the LED Controller Status Get the MSR / ICC LED controlled status by software or firmware NOTE: The ICC LED always controlled by software.

**Parameters**

<i>firmwareControl-MSRLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the MSR LED</li> <li>• 0: software control the MSR LED</li> </ul>
<i>firmwareControl-ICCLEd</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the ICC LED</li> <li>• 0: software control the ICC LED</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.6 int config\_getModelNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getModelNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Model Number

**Parameters**

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.7 int config\_getModelNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Model Number

## Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.8 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.9 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.10 int config\_setBeeperController ( int *firmwareControlBeeper* )

Set the Beeper Controller Set the Beeper controlled by software or firmware

## Parameters

<i>firmwareControlBeeper</i>	1 means firmware control the beeper, 0 means software control beeper.
------------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.11 int config\_setEncryptionControl ( int *msr*, int *icc* )

Set Encryption Control

Set Encryption Control to switch status between MSR and ICC/EMV function. Following Encryption status supported:

- MSR ON, ICC/EMV ON,

- MSR ON, ICC/EMV OFF,
- MSR OFF, ICC/EMV OFF,

## Parameters

<i>msr</i>	<ul style="list-style-type: none"> <li>• 1: enable MSR with Encryption,</li> <li>• 0: disable MSR with Encryption,</li> </ul>
<i>icc</i>	<ul style="list-style-type: none"> <li>• 1: enable ICC with Encryption,</li> <li>• 0: disable ICC with Encryption,</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.12 int config\_setLEDController ( int *firmwareControlMSRLED*, int *firmwareControlICCLED* )

Set the LED Controller Set the MSR / ICC LED controlled by software or firmware NOTE: The ICC LED always controlled by software.

## Parameters

<i>firmwareControl-MSRLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the MSR LED</li> <li>• 0: software control the MSR LED</li> </ul>
<i>firmwareControl-ICCLED</i>	<ul style="list-style-type: none"> <li>• 1: firmware control the ICC LED</li> <li>• 0: software control the ICC LED</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.13 int device\_close ( )

Close the device

## Returns

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.5.4.14 int device\_controlBeep ( int *index*, int *frequency*, int *duration* )

Control Beep

Controls the Beeper



## Parameters

<i>index</i>	For Augusta, must be set to 1 (only one beeper)
<i>frequency</i>	Frequency, range 1000-20000 (suggest minimum 3000)
<i>duration</i>	Duration, in milliseconds (range 1 - 65525)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.15 int device\_controlLED ( byte *indexLED*, byte *control*, int *intervalOn*, int *intervalOff* )

## Control MSR LED

Controls the LED for the MSR

## Parameters

<i>indexLED</i>	For Augusta, must be set to 1 (MSR LED)
<i>control</i>	LED Status: <ul style="list-style-type: none"><li>• 00: OFF</li><li>• 01: RED Solid</li><li>• 02: RED Blink</li><li>• 11: GREEN Solid</li><li>• 12: GREEN Blink</li><li>• 21: BLUE Solid</li><li>• 22: BLUE Blink</li></ul>
<i>intervalOn</i>	Blink interval ON, in ms (Range 200 - 2000)
<i>intervalOff</i>	Blink interval OFF, in ms (Range 200 - 2000)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.16 int device\_controlLED\_ICC ( int *controlMode*, int *interval* )

## Control ICC LED

Controls the LED for the ICC card slot

## Parameters

<i>controlMode</i>	0 = off, 1 = solid, 2 = blink
<i>interval</i>	Blink interval, in ms (500 = 500 ms)

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.17 int device\_controlLED\_MSR ( byte *control*, int *intervalOn*, int *intervalOff* )

## Control the MSR LED

Controls the MSR / ICC LED This API not recommended to control ICC LED

## Parameters

<i>control</i>	<ul style="list-style-type: none"> <li>• 0x00 = off,</li> <li>• 0x01 = RED Solid,</li> <li>• 0x02 = RED Blink,</li> <li>• 0x11 = GREEN Solid,</li> <li>• 0x12 = GREEN Blink,</li> <li>• 0x21 = BLUE Solid,</li> <li>• 0x22 = BLUE Blink,</li> </ul>
<i>intervalOn</i>	Blink interval on time last, in ms (500 = 500 ms, valid from 200 to 2000)
<i>intervalOff</i>	Blink interval off time last, in ms (500 = 500 ms, valid from 200 to 2000)

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

## 12.5.4.18 int device\_getCurrentDeviceType ( )

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.5.4.19 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.20 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
------------------------	---------------------------------------

<i>firmwareVersion-Len</i>	Length of Firmware Version
----------------------------	----------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.21 int device\_getKeyStatus ( int \* newFormat, BYTE \* status, int \* statusLen )

## Get Key Status

Gets the status of loaded keys

## Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len\_L Len\_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

## Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.5.4.22 void device\_getResponseCodeString ( IN int returnCode, OUT char \* description )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";

- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";

- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";

- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";

- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";



- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";

- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE\_OPEN\_FAILED";
- 0X1003: "FILE OPERATION\_FAILED";
- 0X2001: "MEMORY\_NOT\_ENOUGH";
- 0X3002: "SMARTCARD\_FAIL";
- 0X3003: "SMARTCARD\_INIT\_FAILED";
- 0X3004: "FALLBACK\_SITUATION";
- 0X3005: "SMARTCARD\_ABSENT";
- 0X3006: "SMARTCARD\_TIMEOUT";
- 0X3012: "EMV\_RESULT\_CODE\_MSR\_CARD\_ERROR\_FALLBACK";
- 0X5001: "EMV\_PARSING\_TAGS\_FAILED";
- 0X5002: "EMV\_DUPLICATE\_CARD\_DATA\_ELEMENT";
- 0X5003: "EMV\_DATA\_FORMAT\_INCORRECT";
- 0X5004: "EMV\_NO\_TERM\_APP";
- 0X5005: "EMV\_NO\_MATCHING\_APP";
- 0X5006: "EMV\_MISSING\_MANDATORY\_OBJECT";
- 0X5007: "EMV\_APP\_SELECTION\_RETRY";
- 0X5008: "EMV\_GET\_AMOUNT\_ERROR";
- 0X5009: "EMV\_CARD\_REJECTED";
- 0X5010: "EMV\_AIP\_NOT\_RECEIVED";
- 0X5011: "EMV\_AFL\_NOT\_RECEIVED";
- 0X5012: "EMV\_AFL\_LEN\_OUT\_OF\_RANGE";
- 0X5013: "EMV\_SFI\_OUT\_OF\_RANGE";
- 0X5014: "EMV\_AFL\_INCORRECT";
- 0X5015: "EMV\_EXP\_DATE\_INCORRECT";
- 0X5016: "EMV\_EFF\_DATE\_INCORRECT";
- 0X5017: "EMV\_ISS\_COD\_TBL\_OUT\_OF\_RANGE";
- 0X5018: "EMV\_CRYPTOGAM\_TYPE\_INCORRECT";
- 0X5019: "EMV\_PSE\_NOT\_SUPPORTED\_BY\_CARD";
- 0X5020: "EMV\_USER\_SELECTED\_LANGUAGE";
- 0X5021: "EMV\_SERVICE\_NOT\_ALLOWED";
- 0X5022: "EMV\_NO\_TAG\_FOUND";
- 0X5023: "EMV\_CARD\_BLOCKED";
- 0X5024: "EMV\_LEN\_INCORRECT";
- 0X5025: "CARD\_COM\_ERROR";

- 0X5026: "EMV\_TSC\_NOT\_INCREASED";
- 0X5027: "EMV\_HASH\_INCORRECT";
- 0X5028: "EMV\_NO\_ARC";
- 0X5029: "EMV\_INVALID\_ARC";
- 0X5030: "EMV\_NO\_ONLINE\_COMM";
- 0X5031: "TRAN\_TYPE\_INCORRECT";
- 0X5032: "EMV\_APP\_NO\_SUPPORT";
- 0X5033: "EMV\_APP\_NOT\_SELECT";
- 0X5034: "EMV\_LANG\_NOT\_SELECT";
- 0X5035: "EMV\_NO\_TERM\_DATA";
- 0X5039: "EMV\_PIN\_ENTRY\_TIMEOUT";
- 0X6001: "CVM\_TYPE\_UNKNOWN";
- 0X6002: "CVM\_AIP\_NOT\_SUPPORTED";
- 0X6003: "CVM\_TAG\_8E\_MISSING";
- 0X6004: "CVM\_TAG\_8E\_FORMAT\_ERROR";
- 0X6005: "CVM\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6006: "CVM\_COND\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6007: "NO\_MORE\_CVM";
- 0X6008: "PIN\_BYPASSED\_BEFORE";
- 0X7001: "PK\_BUFFER\_SIZE\_TOO\_BIG";
- 0X7002: "PK\_FILE\_WRITE\_ERROR";
- 0X7003: "PK\_HASH\_ERROR";
- 0X8001: "NO\_CARD\_HOLDER\_CONFIRMATION";
- 0X8002: "GET\_ONLINE\_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";

- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBE C: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XB BEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";

- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

#### 12.5.4.23 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

##### Returns

SDK wait time in seconds

#### 12.5.4.24 int device\_getThreadStackSize ( )

##### Get Thread Stack Size

Get the stack size setting for newly created threads

##### Returns

Thread Stack Size

#### 12.5.4.25 int device\_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.26 int device\_isAttached ( int *deviceType* )

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

##### Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

##### Returns

1 if the device is attached, or 0 if the device is not attached

#### 12.5.4.27 int device\_isConnected ( )

Check the device connected status

##### Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

#### 12.5.4.28 int device\_rebootDevice ( )

Reboot Device Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.29 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.5.4.30 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.5.4.31 int device\_SendDataCommand ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

#### Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.32 int device\_setCurrentDevice ( int deviceType )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VEND1,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

## 12.5.4.33 void device\_setSDKWaitTime ( int waitTime )

## Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

## 12.5.4.34 void device\_setThreadStackSize ( int threadSize )

## Set Thread Stack Size

Set the stack size setting for newly created threads

## 12.5.4.35 int device\_updateFirmware ( IN BYTE \* firmwareData, IN int firmwareDataLen, IN char \* firmwareName, IN int encryptionType, IN BYTE \* keyBlob, IN int keyBlobLen )

Update Firmware Updates the firmware of Augusta.



## Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"><li>• For example "Augusta_S_TTK_V1.00.002.fm"</li></ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"><li>• 0 : Plaintext</li><li>• 1 : TDES ECB, PKCS#5 padding</li><li>• 2 : TDES CBC, PKCS#5, IV is all 0</li></ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of keyBlob

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

Firmware update status is returned in the callback with the following values: sender = AUGUSTA state = Device-State.FirmwareUpdate data = File Progress. Two bytes, with `byte[0]` = current block, and `byte[1]` = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

#### 12.5.4.36 int emv\_activateTransaction ( IN int timeout, IN BYTE \* tags, IN int tagsLen, IN int forceOnline )

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.5.4.37 void emv\_allowFallback ( IN int *allow* )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.5.4.38 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.39 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>

<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.5.4.40 int emv\_callbackResponseLCD ( IN int *type*, byte *selection* )

**Callback Response LCD Display**

Provides menu selection responses to the kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV\_CALLBACK\_TYPE.EMV\_CALLBACK\_TYPE\_LCD, and lcd\_displayMode = EMV\_LCD\_DISPLAY\_MODE\_MENU, EMV\_LCD\_DISPLAY\_MODE\_PROMPT, or EMV\_LCD\_DISPLAY\_MODE\_LANGUAGE\_SELECT

**Parameters**

<i>type</i>	If Cancel key pressed during menu selection, then value is EMV_LCD_DISPLAY_MODE_CANCEL. Otherwise, value can be EMV_LCD_DISPLAY_MODE_MENU, EMV_LCD_DISPLAY_MODE_PROMPT, or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT
<i>selection</i>	If type = EMV_LCD_DISPLAY_MODE_MENU or EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT, provide the selection ID line number. Otherwise, if type = EMV_LCD_DISPLAY_MODE_PROMPT supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.41 int emv\_callbackResponseMSR ( IN BYTE \* *MSR*, IN\_OUT int *MSRLen* )

**Callback Response MSR Entry**

Provides MSR information to kernel after a callback was received with DeviceState.EMVCallback, and callbackType = EMV\_CALLBACK\_MSR

**Parameters**

<i>MSR</i>	Swiped track data
<i>MSRLen</i>	the length of Swiped track data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.42 int emv\_cancelTransaction ( )

**Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.5.4.43** `int emv_completeTransaction ( IN int commError, IN BYTE * authCode, IN int authCodeLen, IN BYTE * iad, IN int iadLen, IN BYTE * tlvScripts, IN int tlvScriptsLen, IN BYTE * tlv, IN int tlvLen )`

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

#### Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, <i>authCode</i> , <i>iad</i> , <i>tlvScripts</i> can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of <i>authCode</i>
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of <i>iad</i>
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of <i>tlvScripts</i>
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of <i>tlv</i>

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.5.4.44** `int emv_getAutoAuthenticateTransaction ( )`

Gets auto authenticate value for EMV transactions.

#### Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

**12.5.4.45** `int emv_getAutoCompleteTransaction ( )`

Gets auto complete value for EMV transactions.

#### Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

**12.5.4.46** `int emv_getEMVConfigurationCheckValue ( OUT BYTE * checkValue, IN_OUT int * checkValueLen )`

Get EMV Kernel configuration check value info

#### Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of <i>checkValue</i>

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.5.4.47** `int emv_getEMVKernelCheckValue ( OUT BYTE * checkValue, IN_OUT int * checkValueLen )`

Get EMV Kernel check value info

## Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.5.4.48 `int emv_getEMVKernelVersion ( OUT char * version )`

DEPRECATED : please use `emv_getEMVKernelVersion_Len(OUT char* version, IN_OUT int *versionLen)`

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.5.4.49 `int emv_getEMVKernelVersion_Len ( OUT char * version, IN_OUT int * versionLen )`

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.5.4.50 `void emv_registerCallBk ( pEMV_callBack pEMVf )`

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.5.4.51 `int emv_removeAllApplicationData ( )`

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.5.4.52 `int emv_removeAllCAPK ( )`

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

#### 12.5.4.53 int emv\_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.54 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

##### Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.55 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

##### Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.56 int emv\_removeCRL ( IN BYTE \* list, IN int lssLen )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

##### Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.5.4.57 int emv\_removeTerminalData ( )**

Remove Terminal Data

Removes the Terminal Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.5.4.58 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )**

Retrieve AID list

Returns all the AID names installed on the terminal.

**Parameters**

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.5.4.59 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )**

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

**Parameters**

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.



## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.60 `int emv_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

## Retrieve Certificate Authority Public Key

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.61 `int emv_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

## Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.62 `int emv_retrieveCRL ( OUT BYTE * list, IN_OUT int * lssLen )`

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.63 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

## Retrieve Terminal Data

Retrieves the Terminal Data.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.64 int emv\_retrieveTerminalID ( OUT char \* *terminalID* )

DEPRECATED : please use [emv\\_retrieveTerminalID\\_Len](#)(OUT char\* *terminalID*, IN\_OUT int \**terminalIDLen*)

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.65 int emv\_retrieveTerminalID\_Len ( OUT char \* *terminalID*, IN\_OUT int \* *terminalIDLen* )

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.66 int emv\_retrieveTransactionResult ( IN BYTE \* *tags*, IN int *tagsLen*, IDTTransactionData \* *cardData* )

## Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

## Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDT-TransactionData object

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.67 `int emv_setApplicationData ( IN BYTE * name, IN int nameLen, IN BYTE * tlv, IN int tlvLen )`

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.68 `void emv_setAutoAuthenticateTransaction ( IN int authenticate )`

Enables authenticate for EMV transactions. If a `emv_startTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

12.5.4.69 `void emv_setAutoCompleteTransaction ( IN int complete )`

Enables complete for EMV transactions. If a `emv_authenticateTransaction` results in code 0x0004 (go online), then `emv_completeTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

12.5.4.70 `int emv_setCAPK ( IN BYTE * capk, IN int capkLen )`

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.71 int emv\_setCRL ( IN BYTE \* *list*, IN int *lsLen* )

Set Certificate Revocation List

Sets the CRL

## Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.72 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>device_getResponseCodeString()</code>
--------------------	---

12.5.4.73 `int emv_setTerminalID ( IN char * terminalID )`

Sets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.5.4.74 `int emv_startTransaction ( IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.5.4.75 `int icc_disable ( )`

ICC Function enable/disable Disable ICC function

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.5.4.76 int `icc_enable` ( IN int *withNotification* )

ICC Function enable/disable Enable ICC function with or without seated notification

## Parameters

<i>withNotification</i>	<ul style="list-style-type: none"> <li>• 1: with notification when ICC seated status changed,</li> <li>• 0: without notification.</li> </ul>
-------------------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.77 int icc\_exchangeAPDU ( IN BYTE \* *c\_APDU*, IN int *cLen*, OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.5.4.78 int icc\_exchangeEncryptedAPDU ( IN BYTE \* *c\_APDU*, IN int *cLen*, OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )

Exchange APDU with encrypted data For SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	KSN + encrypted APDU data packet, or no KSN (use last known KSN) + encrypted APDU data packet With KSN: [0A][KSN][Encrypted C-APDU] Without KSN: [00][Encrypted C-APDU]
---------------	---

The format of Raw C-APDU Data Structure of [m-bytes Encrypted C-APDU] is below:

- m = 2 bytes Valid C-APDU Length + x bytes Valid C-APDU + y bytes Padding (0x00) Note: For TDES mode: 2+x should be multiple of 8. If it was not multiple of 8, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 8). For AES mode: 2+x should be multiple of 16. If it was not multiple of 16, unit should padded y bytes 0x00 automatically (2+x+y should be multiple of 16).

## Parameters

<i>cLen</i>	data packet length
<i>reData</i>	response encrypted APDU response. Can be three options:

[00] + [Plaintext R-APDU]

- [01] + [0A] + [KSN] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]
- [01] + [00] + [n bytes Encrypted R-APDU without Status Bytes] + [2 bytes Status Bytes]

The KSN, when provided, will be 10 bytes. The KSN will only be provided when it has changed since the last provided KSN. Each card Power-On generates a new KSN. During a sequence of commands where the KSN



is identical, the first response will have a KSN length set to [0x0A] followed by the KSN, while subsequent commands with the same KSN value will have a KSN length of [0x00] followed by the Encrypted R-APDU without Status Bytes.

**Parameters**

<i>reLen</i>	encrypted APDU response data length
--------------	-------------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.79 int icc\_getAPDU\_KSN ( OUT BYTE \* KSN, IN\_OUT int \* inLen )

**Get APDU KSN**

Retrieves the KSN used in ICC Encrypted APDU usage

**Parameters**

<i>KSN</i>	Returns the encrypted APDU packet KSN
<i>inLen</i>	KSN data length

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.80 int icc\_getFunctionStatus ( OUT int \* enabled, OUT int \* withNotification )

Get ICC Function status Get ICC Function status about enable/disable and with or without seated notification

**Parameters**

<i>enabled</i>	<ul style="list-style-type: none"> <li>• 1: ICC Function enabled,</li> <li>• 0: means disabled.</li> </ul>
<i>withNotification</i>	1 means with notification when ICC seated status changed. 0 means without notification.

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.81 int icc\_getICCRReaderStatus ( OUT BYTE \* status )

**Get Reader Status**

Returns the reader status

**Parameters**

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.5.4.82 int `icc_getKeyFormatForICCDUKPT` ( OUT BYTE \* *format* )

Get Key Format For DUKPT

Specifies how data will be encrypted with Data Key or PIN key (if DUKPT key loaded). This applies to both MSR and ICC

## Parameters

<i>format</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'TDES': Encrypted card data with TDES if DUKPT Key had been loaded.(default)</li> <li>• 'AES': Encrypted card data with AES if DUKPT Key had been loaded.</li> <li>• 'NONE': No Encryption.</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.5.4.83 int `icc_getKeyTypeForICCDUKPT ( OUT BYTE * type )`

Get Key Type for DUKPT

Specifies the key type used for DUKPT encryption This applies to both MSR and ICC

## Parameters

<i>type</i>	Response returned from method: <ul style="list-style-type: none"> <li>• 'DATA': Encrypted card data with Data Key DUKPT Key had been loaded.(default)</li> <li>• 'PIN': Encrypted card data with PIN Key if DUKPT Key had been loaded.</li> </ul>
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.5.4.84 int `icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

12.5.4.85 int `icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

## Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.5.4.86 void msr\_registerCallBk ( pMSR\_callBack pMSRf )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.5.4.87 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.5.4.88 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.5.4.89 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.5.4.90 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.5.4.91 int rs232\_device\_init ( int deviceType, int port\_number, int brate )

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

#### Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

#### Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

#### Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.5.4.92 char\* SDK\_Version ( )

To Get SDK version

##### Returns

return the SDK version string

#### 12.5.4.93 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

##### Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.6 Source\_C/libIDT\_NEO2.h File Reference

NEO2 API.

```
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callBack )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pFW_callBack )(int, int, int, int, int)`
- `typedef void(* pMSR_callBack )(int, IDTMSRData)`
- `typedef void(* pMSR_callBackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callBack )(int, IDTPINData *)`
- `typedef void(* pCMR_callBack )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack )(BYTE status)`
- `typedef void(* pLCD_callBack )(int, IDTLCDItem *)`
- `typedef void(* ftpComm_callBack )(int, int, int)`
- `typedef void(* httpComm_callBack )(BYTE *, int)`
- `typedef void(* v4Comm_callBack )(BYTE, BYTE, BYTE *, int)`

## Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, [pReadDataLog](#) pFRead)
- void [device\\_registerFWCallBk](#) (pFW\_callBack pFWf)
- void [device\\_registerCameraCallBk](#) (pCMR\_callBack pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callBack pCSFSf)
- void [emv\\_registerCallBk](#) (pEMV\_callBack pEMVf)
- void [loyalty\\_registerCallBk](#) (pEMV\_callBack pEMVf)
- void [msr\\_registerCallBk](#) (pMSR\_callBack pMSRf)
- void [msr\\_registerCallBkp](#) (pMSR\_callBackp pMSRf)
- void [ctls\\_registerCallBk](#) (pMSR\_callBack pCTLSf)
- void [ctls\\_registerCallBkp](#) (pMSR\_callBackp pCTLSf)
- void [pin\\_registerCallBk](#) (pPIN\_callBack pPINf)
- void [lcd\\_registerCallBk](#) (pLCD\_callBack pLCDf)
- void [comm\\_registerHTTPCallback](#) (httpComm\_callBack cBack)
- void [comm\\_registerV4Callback](#) (v4Comm\_callBack cBack)
- char \* [SDK\\_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char \*absoluteLibraryPath)
- int [device\\_setConfigPath](#) (const char \*path)
- int [device\\_setNEO2DevicesConfigs](#) (IN const char \*configs, IN int len)
- int [device\\_init](#) ()
- int [rs232\\_device\\_init](#) (int deviceType, int port\_number, int brate)
- int [device\\_setCurrentDevice](#) (int deviceType)
- int [device\\_isAttached](#) (int deviceType)
- int [device\\_close](#) ()
- void [device\\_getIDGStatusCodeString](#) (IN int returnCode, OUT char \*despcrition)
- int [device\\_isConnected](#) ()
- int [device\\_getFirmwareVersion](#) (OUT char \*firmwareVersion)
- int [device\\_getFirmwareVersion\\_Len](#) (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int [device\\_pingDevice](#) ()
- int [device\\_controlUserInterface](#) (IN BYTE \*values)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- void [device\\_toSDCard](#) (int forSDCard)
- int [ctls\\_displayOnlineAuthResult](#) (IN int statusCode, IN BYTE \*TLV, IN int TLVLen)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_enableL100PassThrough](#) (int enableL100PassThrough)
- int [device\\_getL100PassThroughMode](#) ()
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_pollForToken](#) (IN int timeout, OUT BYTE \*respData, IN\_OUT int \*respDataLen)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [device\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [loyalty\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen, IN const int cardType, IN const int iccReadType)

- void [device\\_setTransactionExponent](#) (int exponent)
- int [device\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [device\\_cancelTransaction](#) ()
- int [loyalty\\_cancelTransaction](#) ()
- int [device\\_setCancelTransactionMode](#) (int mode)
- int [device\\_cancelTransactionSilent](#) (int enable)
- int [loyalty\\_cancelTransactionSilent](#) (int enable)
- int [device\\_configureButtons](#) (IN BYTE done, IN BYTE swipe, IN BYTE delay)
- int [device\\_getButtonConfiguration](#) (OUT BYTE \*done, OUT BYTE \*swipe, OUT BYTE \*delay)
- int [device\\_disableBlueLED](#) ()
- int [device\\_enableBlueLED](#) (IN BYTE \*data, IN int dataLen)
- int [device\\_lcdDisplayClear](#) ()
- int [device\\_enableExternalLCDMessages](#) (IN int enableExtLCDMsg)
- int [device\\_enableRFAntenna](#) (IN int enableAntenna)
- int [device\\_turnOffYellowLED](#) ()
- int [device\\_turnOnYellowLED](#) ()
- int [device\\_buzzerOnOff](#) ()
- int [device\\_lcdDisplayLine1Message](#) (IN BYTE \*message, IN int messageLen)
- int [device\\_lcdDisplayLine2Message](#) (IN BYTE \*message, IN int messageLen)
- int [device\\_getKeyStatus](#) (int \*newFormat, BYTE \*status, int \*statusLen)
- int [device\\_updateFirmware](#) (IN BYTE \*firmwareData, IN int firmwareDataLen, IN char \*firmwareName, IN int encryptionType, IN BYTE \*keyBlob, IN int keyBlobLen)
- int [device\\_transferFile](#) (IN char \*fileName, IN int fileNameLen, IN BYTE \*file, IN int fileLen)
- int [device\\_deleteFile](#) (IN char \*fileName, IN int fileNameLen)
- int [device\\_queryFile](#) (IN char \*directoryName, IN int directoryNameLen, IN char \*fileName, IN int fileNameLen, OUT int \*isExist, OUT BYTE \*timeStamp, IN\_OUT int \*timeStampLen, OUT char \*fileSize, IN\_OUT int \*fileSizeLen)
- int [device\\_startListenNotifications](#) ()
- int [device\\_stopListenNotifications](#) ()
- int [device\\_startQRCodeScan](#) (IN int \_timeout)
- int [device\\_stopQRCodeScan](#) ()
- int [device\\_startTakingPhoto](#) (IN int \_timeout)
- int [device\\_stopTakingPhoto](#) ()
- void [device\\_getResponseCodeString](#) (IN int returnCode, OUT char \*despcrition)
- int [device\\_listDirectory](#) (IN char \*directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char \*directory, IN\_OUT int \*directoryLen)
- int [device\\_deleteDirectory](#) (IN char \*dirName, IN int dirNameLen)
- int [device\\_getDeviceMemoryUsagelInfo](#) (OUT int \*freeHeapSize, OUT int \*notFreedBlockCnt, OUT int \*min-EverFreeHeapSize)
- int [felica\\_authentication](#) (IN BYTE \*key, IN int keyLen)
- int [felica\\_readWithMac](#) (IN int blockCnt, IN BYTE \*blockList, IN int blockListLen, OUT BYTE \*blockData, OUT int \*blockDataLen)
- int [felica\\_writeWithMac](#) (IN BYTE blockNum, IN BYTE \*blockData, IN int blockDataLen)
- int [felica\\_read](#) (IN BYTE \*serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE \*blockList, IN int blockListLen, OUT BYTE \*blockData, OUT int \*blockDataLen)
- int [felica\\_write](#) (IN BYTE \*serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE \*blockList, IN int blockListLen, IN BYTE \*blockData, IN int blockDataLen, OUT BYTE \*statusFlag, OUT int \*statusFlagLen)
- int [felica\\_poll](#) (IN BYTE \*systemCode, IN int systemCodeLen, OUT BYTE \*respData, OUT int \*respDataLen)
- int [felica\\_SendCommand](#) (IN BYTE \*command, IN int commandLen, OUT BYTE \*respData, OUT int \*respDataLen)
- int [felica\\_requestService](#) (IN BYTE \*nodeCode, IN int nodeCodeLen, OUT BYTE \*respData, OUT int \*respDataLen)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_getModelNumber](#) (OUT char \*sNumber)



- int [config\\_getModelNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [config\\_setCmdTimeOutDuration](#) (IN int millisecond)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()
- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- int [emv\\_getEMVKernelVersion](#) (OUT char \*version)
- int [emv\\_getEMVKernelVersion\\_Len](#) (OUT char \*version, IN\_OUT int \*versionLen)
- int [emv\\_getEMVKernelCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- int [emv\\_getEMVConfigurationCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- void [emv\\_allowFallback](#) (IN int allow)
- void [emv\\_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv\\_setAutoCompleteTransaction](#) (IN int complete)
- int [emv\\_getAutoAuthenticateTransaction](#) ()
- int [emv\\_getAutoCompleteTransaction](#) ()
- void [emv\\_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen)
- int [emv\\_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveTransactionResult](#) (IN BYTE \*tags, IN int tagsLen, OUT IDTTransactionData \*cardData)
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()

- `int emv_retrieveCAPKList (OUT BYTE *keys, IN_OUT int *keysLen)`
- `int emv_retrieveCRL (OUT BYTE *list, IN_OUT int *lssLen)`
- `int emv_setCRL (IN BYTE *list, IN int lsLen)`
- `int emv_removeCRL (IN BYTE *list, IN int lsLen)`
- `int emv_removeAllCRL ()`
- `int icc_getICCReaderStatus (OUT BYTE *status)`
- `int icc_powerOnICC (OUT BYTE *ATR, IN_OUT int *inLen)`
- `int icc_powerOffICC ()`
- `int icc_exchangeAPDU (IN BYTE *c_APDU, IN int cLen, OUT BYTE *reData, IN_OUT int *reLen)`
- `int lcd_createScreen (IN char *screenName, IN int screenNameLen, OUT int *ScreenID)`
- `int lcd_destroyScreen (IN char *screenName, IN int screenNameLen)`
- `int lcd_getActiveScreen (OUT char *screenName, IN_OUT int *screenNameLen)`
- `int lcd_showScreen (IN char *screenName, IN int screenNameLen)`
- `int lcd_getButtonEvent (OUT int *screenID, OUT int *objectID, OUT char *screenName, IN_OUT int *screenNameLen, OUT char *objectName, IN_OUT int *objectNameLen, OUT int *isLongPress)`
- `int lcd_addButton (IN char *screenName, IN int screenNameLen, IN char *buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addEthernet (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem)`
- `int lcd_addLED (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem *returnItem, IN BYTE *LED, IN int LEDLen)`
- `int lcd_addText (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE *color, IN int colorLen, IN char *label, IN int labelLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addImage (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)`
- `int lcd_addVideo (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char *filename, IN int filenameLen, OUT IDTLCDItem *returnItem)`
- `int lcd_cloneScreen (IN char *screenName, IN int screenNameLen, IN char *cloneName, IN int cloneNameLen, OUT int *cloneID)`
- `int lcd_updateLabel (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN char *label, IN int labelLen)`
- `int lcd_updateColor (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE *color, IN int colorLen)`
- `int lcd_updatePosition (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord)`
- `int lcd_removeItem (IN char *screenName, IN int screenNameLen, IN char *objectName, IN int objectNameLen)`
- `int lcd_storeScreenInfo ()`
- `int lcd_loadScreenInfo ()`
- `int lcd_clearScreenInfo ()`
- `int lcd_getAllScreens (IN_OUT int *screenNumbers, OUT IDTScreenInfo *screenInfo)`
- `int lcd_getAllObjects (IN char *screenName, IN int screenNameLen, IN_OUT int *objectNumbers, OUT IDTObjectInfo *objectInfo)`
- `int lcd_queryScreenbyName (IN char *screenName, IN int screenNameLen, OUT int *result)`
- `int lcd_queryObjectbyName (IN char *objectName, IN int objectNameLen, IN_OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)`
- `int lcd_queryScreenbyID (IN int screenID, OUT int *result, OUT int *screenName, IN_OUT int *screenNameLen)`
- `int lcd_queryObjectbyID (IN int objectID, OUT int *objectNumbers, OUT IDTScreenInfo *screenInfo)`
- `int lcd_setBacklight (IN BYTE backlightVal)`
- `int msr_cancelMSRSwipe ()`
- `int msr_startMSRSwipe (IN int _timeout)`

- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)
- int [pin\\_capturePin](#) (IN int timeout, IN int type, IN char \*PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char \*message, IN int messageLen)
- int [pin\\_capturePinExt](#) (IN int type, IN char \*PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char \*message, IN int messageLen, IN char \*verify, IN int verifyLen)
- int [pin\\_promptForNumericKeyWithSwipe](#) (IN int timeout, IN BYTE function, IN int minLen, IN int maxLen, IN char \*line1, IN int line1Len, IN char \*line2, IN int line2Len, BYTE \*signature, IN int signatureLen)
- int [pin\\_promptForNumericKey](#) (IN int timeout, IN int maskInput, IN int minLen, IN int maxLen, IN char \*message, IN int messageLen, BYTE \*signature, IN int signatureLen)
- int [pin\\_inputFromPrompt](#) (BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char \*lang, BYTE promptID, char \*defaultResponse, int defaultResponseLen, int timeout)
- int [pin\\_getPanEntry](#) (IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly)
- int [pin\\_cancelPINEntry](#) ()
- int [pin\\_setKeyValues](#) (int mode)

### 12.6.1 Detailed Description

NEO2 API. NEO2 Global API methods.

### 12.6.2 Macro Definition Documentation

#### 12.6.2.1 #define IN

INPUT parameter.

#### 12.6.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.6.2.3 #define OUT

OUTPUT parameter.

### 12.6.3 Typedef Documentation

#### 12.6.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.6.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

**12.6.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)**

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

**12.6.3.4 typedef void(\* pCSFS\_callback)(BYTE status)**

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

**12.6.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)**

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk

**12.6.3.6 typedef void(\* pFW\_callback)(int, int, int, int, int)**

Define the firmware update callback function to get the firmware update status

It should be registered using the device\_registerFWCallBk

**12.6.3.7 typedef void(\* pLCD\_callback)(int, IDTLCDItem \*)**

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd\_registerCallBk,

**12.6.3.8 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.6.3.9 typedef void(\* pMSR\_callback)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.6.3.10 typedef void(\* pMSR\_callbackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callback

**12.6.3.11 typedef void(\* pPIN\_callback)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.6.3.12 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk

**12.6.3.13 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk

**12.6.3.14 typedef void(\* v4Comm\_callBack)(BYTE, BYTE, BYTE \*, int)**

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

**12.6.4 Function Documentation****12.6.4.1 void comm\_registerHTTPCallback ( httpComm\_callBack cBack )**

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

**12.6.4.2 void comm\_registerV4Callback ( v4Comm\_callBack cBack )**

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

**12.6.4.3 int config\_getModelNumber ( OUT char \* sNumber )**

DEPRECATED : please use [config\\_getModelNumber\\_Len\(OUT char\\* sNumber, IN\\_OUT int \\*sNumberLen\)](#)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

**12.6.4.4 int config\_getModelNumber\_Len ( OUT char \* sNumber, IN\_OUT int \* sNumberLen )**

Polls device for Model Number

## Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.6.4.5 `int config_getSerialNumber ( OUT char * sNumber )`

DEPRECATED : please use `config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)`

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.6.4.6 `int config_getSerialNumber_Len ( OUT char * sNumber, IN_OUT int * sNumberLen )`

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.6.4.7 `int config_setCmdTimeOutDuration ( IN int millisecond )`

Set the timeout duration for regular commands The new timeout value will affect all the functions actually send (sync) commands that doesn't need to wait for a callback function, such as `device_getFirmwareVersion()`, `device_pingDevice()`, `device_SendDataCommandNEO()`, `device_enablePassThrough()`, `device_setBurstMode()`, `device_setPollMode()`, `device_updateFirmware()` ... etc.

## Parameters

<i>millisecond</i>	timeout value in milliseconds
--------------------	-------------------------------

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

12.6.4.8 `int ctls_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

#### 12.6.4.9 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.10 int ctls\_displayOnlineAuthResult ( IN int *statusCode*, IN BYTE \* *TLV*, IN int *TLVLen* )

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

Display Online Authorization Result Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

##### Parameters

<i>statusCode</i>	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
<i>TLV</i>	Optional TLV for AOSA
<i>TLVLen</i>	TLV Length

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.11 int ctls\_getAllConfigurationGroups ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

##### Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.12 int ctls\_getConfigurationGroup ( IN int *group*, OUT BYTE \* *tlv*, OUT int \* *tlvLen* )

Get Configuration Group

Retrieves the Configuration for the specified Group.

##### Parameters



<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.13 void ctls\_registerCallBk ( pMSR\_callBack pCTLSf )

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.6.4.14 void ctls\_registerCallBkp ( pMSR\_callBackp pCTLSf )

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.6.4.15 int ctls\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.16 int ctls\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.17 int ctls\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.18 int ctls\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.19 int ctls\_removeConfigurationGroup ( int group )

## Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

## Parameters

<i>group</i>	Configuration Group
--------------	---------------------

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

## 12.6.4.20 int ctls\_retrieveAIDList ( OUT BYTE \* AIDList, IN\_OUT int \* AIDListLen )

## Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.21 int ctls\_retrieveApplicationData ( IN BYTE \* AID, IN int AIDLen, OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

## Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.22 int ctls\_retrieveCAPK ( IN BYTE \* capk, IN int capkLen, OUT BYTE \* key, IN\_OUT int \* keyLen )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.23 int ctls\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.24 int ctls\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.25 int ctls\_setApplicationData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

## Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.26 int ctls\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

## Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.27 int ctls\_setConfigurationGroup ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (DFEE2D). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.28 int ctls\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.6.4.29 int ctls\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *\_timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)  • SEE IMPORTANT NOTE BELOW
---------------	---

<i>amtOther</i>	Other amount value, if any (tag value 9F03)  • SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)

- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.6.4.30 int device\_activateTransaction ( IN const int \_timeout, IN BYTE \* tags, IN int tagsLen )

Activate Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

##### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 Be sure to include 9F02 (amount) and 9C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of device\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DFO10101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal

- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.6.4.31 int device\_buzzerOnOff ( )

Use this function to make the buzzer beep once

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.32 int device\_cancelTransaction ( )

Cancel Transaction request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.33 int device\_cancelTransactionSilent ( int *enable* )

Cancel Transaction Silent

Cancel transaction with or without showing the LCD message

##### Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

##### Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

#### 12.6.4.34 int device\_close ( )

Close the device

##### Returns

RETURN\_CODE: 0: success, 0x0A: failed



12.6.4.35 int device\_configureButtons ( IN BYTE *done*, IN BYTE *swipe*, IN BYTE *delay* )

Configures the buttons on the ViVOpay Vendi reader

## Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value (<= 30) in seconds

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.36 int device\_controlUserInterface ( IN BYTE \* *values* )

##### Control User Interface

Controls the User Interface: Display, Beep, LED

## Parameters

<i>values</i>	<p>Four bytes to control the user interface Byte[0] = LCD Message Messages 00-07 are normally controlled by the reader.</p> <ul style="list-style-type: none"> <li>• 00h: Idle Message (Welcome)</li> <li>• 01h: Present card (Please Present Card)</li> <li>• 02h: Time Out or Transaction cancel (No Card)</li> <li>• 03h: Transaction between reader and card is in the middle (Processing...)</li> <li>• 04h: Transaction Pass (Thank You)</li> <li>• 05h: Transaction Fail (Fail)</li> <li>• 06h: Amount (Amount \$ 0.00 Tap Card)</li> <li>• 07h: Balance or Offline Available funds (Balance \$ 0.00) Messages 08-0B are controlled by the terminal</li> <li>• 08h: Insert or Swipe card (Use Chip &amp; PIN)</li> <li>• 09h: Try Again(Tap Again)</li> <li>• 0Ah: Tells the customer to present only one card (Present 1 card only)</li> <li>• 0Bh: Tells the customer to wait for authentication/authorization (Wait)</li> <li>• FFh: indicates the command is setting the LED/Buzzer only. Byte[1] = Beep Indicator</li> <li>• 00h: No beep</li> <li>• 01h: Single beep</li> <li>• 02h: Double beep</li> <li>• 03h: Three short beeps</li> <li>• 04h: Four short beeps</li> <li>• 05h: One long beep of 200 ms</li> <li>• 06h: One long beep of 400 ms</li> <li>• 07h: One long beep of 600 ms</li> <li>• 08h: One long beep of 800 ms Byte[2] = LED Number</li> <li>• 00h: LED 0 (Power LED) 01h: LED 1</li> <li>• 02h: LED 2</li> <li>• 03h: LED 3</li> <li>• FFh: All LEDs Byte[3] = LED Status</li> <li>• 00h: LED Off</li> <li>• 01h: LED On</li> </ul>
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.37 `int device_deleteDirectory ( IN char * dirName, IN int dirNameLen )`

**Delete Directory** This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

## Parameters

<i>dirName</i>	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
<i>dirNameLen</i>	Directory Name Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.38 int device\_deleteFile ( IN char \* *fileName*, IN int *fileNameLen* )

Delete File This command deletes a file or group of files.

## Parameters

<i>filename</i>	Complete path and file name of the file you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>filenameLen</i>	File Name Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.39 int device\_disableBlueLED ( )

Stops the blue LEDs on the ViVOpay Vendi reader from flashing in left to right sequence and turns the LEDs off, and contactless function is disabled at the same time

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.40 int device\_enableBlueLED ( IN BYTE \* *data*, IN int *dataLen* )

Use this function to control the blue LED behavior on the Vendi reader

## Parameters

<i>data</i>	Sequence data Byte 0 (Cycle): 0 = Cycle once, 1 = Repeat Byte 1 (LEDs): LED State Bitmap Byte 2-3 (Duration): Given in multiples of 10 millisecond Byte 4 (LED): LED State Bitmap Byte 5-6 (Duration): Given in multiples of 10 millisecond Byte 7-24 (Additional LED/Durations): Define up to 8 LED and duration pars
-------------	--

LED State Bitmap: Bit 8: Left blue LED, 0 = off, 1 = on Bit 7: Center Blue LED, 0 = off, 1 = on Bit 6: Right Blue LED 0 = off, 1 = on Bit 5: Yellow LED, 0 = off, 1 = on Bit 4: Reserved for future use Bit 3: Reserved for future use Bit 2: Reserved for future use Bit 1: Reserved for future use

## Parameters

<i>dataLen</i>	Length of the sequence data: 0 or 4 to 25 bytes
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.41 `int device_enableExternalLCDMessages ( IN int enableExtLCDMsg )`

Enable or disable the external LCD message It will turn off the external LCD messages including EMV transactions. (For the users who only need MSR and/or CTLS transactions.) The function only works for VP5300

**Parameters**

<i>enableExtLCD- Msg</i>	1 = ON, 0 = OFF
------------------------------	-----------------

**Returns**

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

**See Also**

`ErrorCode`

**12.6.4.42 int device\_enableL100PassThrough ( int *enableL100PassThrough* )**

Enable L100 Pass Through

Enables Pass Through Mode for direct communication to L100 hook up to NEO II device

**Parameters**

<i>enableL100- PassThrough</i>	1 = pass through ON, 0 = pass through OFF
------------------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

**12.6.4.43 int device\_enablePassThrough ( int *enablePassThrough* )**

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

**Parameters**

<i>enablePass- Through</i>	1 = pass through ON, 0 = pass through OFF
--------------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.44 int device\_enableRFAntenna ( IN int *enableAntenna* )**

Enable or disable the RF Antenna

**Parameters**

<i>enableAntenna</i>	1 = ON, 0 = OFF
----------------------	-----------------

**Returns**

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

**See Also**

`ErrorCode`

12.6.4.45 `int device_getButtonConfiguration ( OUT BYTE * done, OUT BYTE * swipe, OUT BYTE * delay )`

Reads the button configuration from the ViVOpay Vendi reader



## Parameters

<i>done</i>	0x01: the Done switch is enabled 0x00: the Done switch is disabled
<i>swipe</i>	0x01: the Swipe Card switch is enabled 0x00: the Swipe Card switch is disabled
<i>delay</i>	an unsigned delay value in seconds

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.46 int device\_getCurrentDeviceType ( )

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.6.4.47 int device\_getDeviceMemoryUsageInfo ( OUT int \* *freeHeapSize*, OUT int \* *notFreedBlockCnt*, OUT int \* *minEverFreeHeapSize* )

Get Device Memory Usage Information

## Parameters

<i>freeHeapSize</i>	Free Heap Size: Available heap size
<i>notFreedBlockCnt</i>	Memory Not Freed Block Count: Memory in use block count
<i>minEverFreeHeapSize</i>	Minimum Ever Free Heap Size: The lowest ever available heap size

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.48 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.49 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.50 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.6.4.51 int device\_getKeyStatus ( int \* newFormat, BYTE \* status, int \* statusLen )

##### Get Key Status

Gets the status of loaded keys

##### Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len\_L Len\_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

## Parameters

<i>statusLen</i>	the length of status
------------------	----------------------

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.6.4.52 `int device_getL100PassThroughMode ( )`

Get L100 Pass Through Mode

Get current Pass Through Mode for direct communication to L100 hook up to NEO II device

## Returns

RETURN\_CODE: return 1 if L100 Pass Through Mode is TRUE, 0 if L100 Pass Through Mode is FALSE

12.6.4.53 `int device_getMerchantRecord ( IN int index, OUT BYTE * record )`

DEPRECATED : please use `device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)`

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

## Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString()`

## See Also

`ErrorCode`

12.6.4.54 `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

<i>recordLen</i>	Length of record
------------------	------------------

#### Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### See Also

[ErrorCode](#)

#### 12.6.4.55 void device\_getResponseCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

#### Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

#### Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";

- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";
- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";

- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";
- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D2: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";

- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";
- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'";
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";



- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";
- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported, ";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";

- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";
- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";

- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount, Other Amount, Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";
- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE\_OPEN\_FAILED";
- 0X1003: "FILE OPERATION\_FAILED";
- 0X2001: "MEMORY\_NOT\_ENOUGH";
- 0X3002: "SMARTCARD\_FAIL";
- 0X3003: "SMARTCARD\_INIT\_FAILED";
- 0X3004: "FALLBACK\_SITUATION";
- 0X3005: "SMARTCARD\_ABSENT";
- 0X3006: "SMARTCARD\_TIMEOUT";
- 0X3012: "EMV\_RESULT\_CODE\_MSR\_CARD\_ERROR\_FALLBACK";
- 0X5001: "EMV\_PARSING\_TAGS\_FAILED";
- 0X5002: "EMV\_DUPLICATE\_CARD\_DATA\_ELEMENT";
- 0X5003: "EMV\_DATA\_FORMAT\_INCORRECT";
- 0X5004: "EMV\_NO\_TERM\_APP";
- 0X5005: "EMV\_NO\_MATCHING\_APP";
- 0X5006: "EMV\_MISSING\_MANDATORY\_OBJECT";
- 0X5007: "EMV\_APP\_SELECTION\_RETRY";
- 0X5008: "EMV\_GET\_AMOUNT\_ERROR";
- 0X5009: "EMV\_CARD\_REJECTED";
- 0X5010: "EMV\_AIP\_NOT\_RECEIVED";
- 0X5011: "EMV\_AFL\_NOT\_RECEIVED";
- 0X5012: "EMV\_AFL\_LEN\_OUT\_OF\_RANGE";
- 0X5013: "EMV\_SFI\_OUT\_OF\_RANGE";
- 0X5014: "EMV\_AFL\_INCORRECT";
- 0X5015: "EMV\_EXP\_DATE\_INCORRECT";
- 0X5016: "EMV\_EFF\_DATE\_INCORRECT";

- 0X5017: "EMV\_ISS\_COD\_TBL\_OUT\_OF\_RANGE";
- 0X5018: "EMV\_CRYPTOGAM\_TYPE\_INCORRECT";
- 0X5019: "EMV\_PSE\_NOT\_SUPPORTED\_BY\_CARD";
- 0X5020: "EMV\_USER\_SELECTED\_LANGUAGE";
- 0X5021: "EMV\_SERVICE\_NOT\_ALLOWED";
- 0X5022: "EMV\_NO\_TAG\_FOUND";
- 0X5023: "EMV\_CARD\_BLOCKED";
- 0X5024: "EMV\_LEN\_INCORRECT";
- 0X5025: "CARD\_COM\_ERROR";
- 0X5026: "EMV\_TSC\_NOT\_INCREASED";
- 0X5027: "EMV\_HASH\_INCORRECT";
- 0X5028: "EMV\_NO\_ARC";
- 0X5029: "EMV\_INVALID\_ARC";
- 0X5030: "EMV\_NO\_ONLINE\_COMM";
- 0X5031: "TRAN\_TYPE\_INCORRECT";
- 0X5032: "EMV\_APP\_NO\_SUPPORT";
- 0X5033: "EMV\_APP\_NOT\_SELECT";
- 0X5034: "EMV\_LANG\_NOT\_SELECT";
- 0X5035: "EMV\_NO\_TERM\_DATA";
- 0X5039: "EMV\_PIN\_ENTRY\_TIMEOUT";
- 0X6001: "CVM\_TYPE\_UNKNOWN";
- 0X6002: "CVM\_AIP\_NOT\_SUPPORTED";
- 0X6003: "CVM\_TAG\_8E\_MISSING";
- 0X6004: "CVM\_TAG\_8E\_FORMAT\_ERROR";
- 0X6005: "CVM\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6006: "CVM\_COND\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6007: "NO\_MORE\_CVM";
- 0X6008: "PIN\_BYPASSED\_BEFORE";
- 0X7001: "PK\_BUFFER\_SIZE\_TOO\_BIG";
- 0X7002: "PK\_FILE\_WRITE\_ERROR";
- 0X7003: "PK\_HASH\_ERROR";
- 0X8001: "NO\_CARD HOLDER\_CONFIRMATION";
- 0X8002: "GET\_ONLINE\_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";

- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";
- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";

- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0XBBE C: "CM100 Command Unsupported";
- 0XBBED: "CM100 Error In Command Process";
- 0XB BEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";
- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

#### 12.6.4.56 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

##### Returns

SDK wait time in seconds

#### 12.6.4.57 int device\_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

##### Returns

Thread Stack Size

#### 12.6.4.58 int device\_getTransactionResults ( IDTMSRData \* *cardData* )

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

##### Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

##### Returns

success or error code. Values can be parsed with device\_getIDGStatusCodeString

##### See Also

ErrorCode

#### 12.6.4.59 int device\_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.60 int device\_isAttached ( int *deviceType* )

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

## 12.6.4.61 int device\_isConnected ( )

Check the device connected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

## 12.6.4.62 int device\_lcdDisplayClear ( )

Use this function to clear the LCD display

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.63 int device\_lcdDisplayLine1Message ( IN BYTE \* message, IN int messageLen )

Use this function to display text on the LCD display. On the Vendi reader the LCD is a 2-line character display.

## Parameters

<i>message</i>	Valid messages for the first line of text are between 1 and 16 printable characters long. If the text message is greater than 16 bytes but not more than 32 bytes, byte 17 and onward are displayed as a second row of text. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 32 bytes

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.64 int device\_lcdDisplayLine2Message ( IN BYTE \* message, IN int messageLen )

Use this function to display the message on line 2 of the LCD display. On the Vendi reader the LCD is a 2-line character display.

## Parameters

<i>message</i>	Valid messages are between 1 and 16 printable characters long. All messages are left justified on the LCD display.
<i>messageLen</i>	Length of the message: 1 to 16 bytes

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



12.6.4.65 int device\_listDirectory ( IN char \* *directoryName*, IN int *directoryNameLen*, IN int *recursive*, IN int *onSD*, OUT char \* *directory*, IN\_OUT int \* *directoryLen* )

List Directory This command retrieves a directory listing of user accessible files from the reader.

## Parameters

<i>directoryName</i>	Directory Name. If null, root directory is listed
<i>directoryName-Len</i>	Directory Name Length. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	TRUE = use flash storage The returned directory information The returned directory information length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.66 int device\_pingDevice ( )

## Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.67 int device\_pollForToken ( IN int timeout, OUT BYTE \* respData, IN\_OUT int \* respDataLen )

## Poll for Token

## Polls for a PICC

## Parameters

<i>timeout</i>	timeout in milliseconds, must be multiple of 10 milliseconds. 30, 120, 630, or 1150 for example.
<i>respData</i>	Response data will be stored in respData. 1 byte of card type, and the Serial Number (or the UID) of the PICC if available.
<i>respDataLen</i>	Length of systemCode.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.68 int device\_queryFile ( IN char \* directoryName, IN int directoryNameLen, IN char \* fileName, IN int fileNameLen, OUT int \* isExist, OUT BYTE \* timeStamp, IN\_OUT int \* timeStampLen, OUT char \* fileSize, IN\_OUT int \* fileSizeLen )

Query File This command checks if the specified file exists in NAND Flash..

## Parameters

<i>directoryName</i>	Directory name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>directoryName-Len</i>	Directory Name Length.

<i>fileName</i>	File name string. No longer than 32 bytes. ASCII string, terminated by 0x00.
<i>fileNameLen</i>	File Name Length.
<i>isExist</i>	File exists: 1, File not exists 0.
<i>timeStamp</i>	Latest time stamp of the file. 6 bytes BCD code if the file exists.
<i>timeStampLen</i>	Length of timeStamp. 6 if the file exists, 0 if the file does not exist.
<i>fileSize</i>	Zero-terminated ASCII string of the file size.
<i>fileSizeLen</i>	Length of fileSize.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

##### 12.6.4.69 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

##### 12.6.4.70 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

##### 12.6.4.71 void device\_registerFWCallBk ( pFW\_callBack pFWf )

To register the firmware update callback function to get the firmware update processing response. (Pass NULL to disable the callback.)

##### 12.6.4.72 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to NEO device

Sends a command to the NEO device .

#### Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

##### 12.6.4.73 int device\_setBurstMode ( IN BYTE mode )

Send Burst Mode

Sets the burst mode for the device.

## Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

## Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

## See Also

`ErrorCode`

12.6.4.74 `int device_setCancelTransactionMode ( int mode )`

Set Cancel Transaction Mode

Set the cancel transaction mode to be with or without LCD message

## Parameters

<i>mode</i>	0: With LCD message 1: Without LCD message
-------------	--

## Returns

success or error code. 1: Success, 0: Failed

12.6.4.75 `int device_setConfigPath ( const char * path )`

Set the path to the config xml file(s) if any

## Parameters

<i>path</i>	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

12.6.4.76 `int device_setCurrentDevice ( int deviceType )`

Sets the current device to talk to

The connect status can be checked by `device_isConnected()`.

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.6.4.77 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.78 int device\_setNEO2DevicesConfigs ( IN const char \* *configs*, IN int *len* )

Pass the content of the config xml file ("NEO2\_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK It needs to be called before [device\\_init\(\)](#), otherwise the SDK will try to read the config xml file.

## Parameters

<i>configs</i>	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
<i>len</i>	The length of the string configs. The maximum length is 5000 bytes.

#### 12.6.4.79 int device\_setPollMode ( IN BYTE mode )

##### Set Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

##### Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.80 void device\_setSDKWaitTime ( int waitTime )

##### Set SDK Wait Time

Set the SDK wait time for transactions

##### Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

#### 12.6.4.81 void device\_setThreadStackSize ( int threadSize )

##### Set Thread Stack Size

Set the stack size setting for newly created threads

#### 12.6.4.82 void device\_setTransactionExponent ( int exponent )

Sets the transaction exponent to be used with device\_startTransaction. Default value is 2

##### Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

#### 12.6.4.83 int device\_startListenNotifications ( )

Start Listen Notifications This function enables Card Status and Front Switch notifications.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.84 int device\_startQRCodeScan ( IN int \_timeout )

##### Start QR Code Scanning

Enables QR Code scanning, waiting for the QR code.

## Parameters

<i>timeout</i>	QR Code Scan Timeout Value. Between 30 and 65536 seconds.
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.6.4.85 int device\_startTakingPhoto ( IN int *timeout* )

## Start Taking Photo

Enables the camera to take a photo.

## Parameters

<i>timeout</i>	Photo taking Timeout Value. Between 30 and 65536 seconds.
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.6.4.86 int device\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)  • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03)  • SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part

of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101

9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
  - - - 0 = Payment Terminal
  - - - 1 = Transit Terminal
  - - - 2 = Access Terminal
  - - - 3 = Wireless Handoff Terminal
  - - - 4 = App Handoff Terminal
  - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

#### 12.6.4.87 int device\_stopListenNotifications ( )

Stop Listen Notifications This function disables Card Status and Front Switch notifications.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



## 12.6.4.88 int device\_stopQRCodeScan ( )

Stop QR Code Scanning Cancels QR Code scanning request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.89 int device\_stopTakingPhoto ( )

Stop Taking Photo Cancels Photo Taking request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.90 void device\_toSDCard ( int forSDCard )

To SD Card

Set the destination of the file or directory function

## Parameters

<i>forSDCard</i>	0: for internal memory, 1: for SD card
------------------	--

## 12.6.4.91 int device\_transferFile ( IN char \* fileName, IN int fileNameLen, IN BYTE \* file, IN int fileLen )

Transfer File This command transfers a data file to the reader.

## Parameters

<i>fileName</i>	Filename. The data for this command is a ASCII string with the complete path and file name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>fileNameLen</i>	File Name Length.
<i>file</i>	The data file.
<i>fileLen</i>	File Length.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.92 int device\_turnOffYellowLED ( )

Use this function to turn off the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.93 int device\_turnOnYellowLED ( )

Use this function to turn on the ViVOpay Vendi reader yellow LED. This LED is located below the three blue LEDs

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.94 int device\_updateFirmware ( IN BYTE \* *firmwareData*, IN int *firmwareDataLen*, IN char \* *firmwareName*, IN int *encryptionType*, IN BYTE \* *keyBlob*, IN int *keyBlobLen* )

Update Firmware Updates the firmware of NEO 2 devices.

##### Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of <i>firmwareData</i>
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"> <li>• For example "VP5300_v1.00.023.0167.S_Test.fm"</li> </ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"> <li>• 0 : Plaintext</li> <li>• 1 : TDES ECB, PKCS#5 padding</li> <li>• 2 : TDES CBC, PKCS#5, IV is all 0</li> </ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of <i>keyBlob</i>

##### Returns

RETURN\_CODE: Values can be parsed with [errorCode.getErrorString\(\)](#)

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE-\_FIRMWARE\_UPDATE current block total blocks ResultCode:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

#### 12.6.4.95 int emv\_activateTransaction ( IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

Activate EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.96 void emv\_allowFallback ( IN int *allow* )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.6.4.97 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type</li> </ul>
-------------------	--

In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Parameters

<i>updatedTLVLen</i>	
----------------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.98 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.99 int emv\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.100 int emv\_completeTransaction ( IN int *commError*, IN BYTE \* *authCode*, IN int *authCodeLen*, IN BYTE \* *iad*, IN int *iadLen*, IN BYTE \* *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv\_authenticateTransaction

The tags will be returned in the callback routine.

## Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.101 int emv\_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

##### Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

#### 12.6.4.102 int emv\_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

##### Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

#### 12.6.4.103 int emv\_getEMVConfigurationCheckValue ( OUT BYTE \* *checkValue*, IN\_OUT int \* *checkValueLen* )

Get EMV Kernel configuration check value info

##### Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.104 int emv\_getEMVKernelCheckValue ( OUT BYTE \* *checkValue*, IN\_OUT int \* *checkValueLen* )

Get EMV Kernel check value info

##### Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

#### 12.6.4.105 int emv\_getEMVKernelVersion ( OUT char \* *version* )

DEPRECATED : please use [emv\\_getEMVKernelVersion\\_Len\(OUT char\\* version, IN\\_OUT int \\*versionLen\)](#)

Polls device for EMV Kernel Version

##### Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.106 int emv\_getEMVKernelVersion\_Len ( OUT char \* *version*, IN\_OUT int \* *versionLen* )

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.107 void emv\_registerCallBk ( pEMV\_callBack pEMVf )**

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

**12.6.4.108 int emv\_removeAllApplicationData ( )**

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.109 int emv\_removeAllCAPK ( )**

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.110 int emv\_removeAllCRL ( )**

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.111 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
------------	--

<i>AIDLen</i>	the length of AID data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.112 int emv\_removeCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.113 int emv\_removeCRL ( IN BYTE \* *list*, IN int *lsLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

**Parameters**

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.114 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

**Parameters**

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.115 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.



## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.116 int emv\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>

<i>keyLen</i>	the length of key data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.117 int emv\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keyLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keyLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.118 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *listLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

**Parameters**

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>listLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.119 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.120 int emv\_retrieveTransactionResult ( IN BYTE \* *tags*, IN int *tagsLen*, OUT IDTTTransactionData \* *cardData* )

Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

## Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDT-TransactionData object

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.121 int emv\_setApplicationData ( IN BYTE \* name, IN int nameLen, IN BYTE \* tlv, IN int tlvLen )

## Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.122 int emv\_setApplicationDataTLV ( IN BYTE \* tlv, IN int tlvLen )

## Set Application Data by TLV

Sets the Application Data as specified by the TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.123 void emv\_setAutoAuthenticateTransaction ( IN int authenticate )

Enables authenticate for EMV transactions. If a emv\_startTransaction results in code 0x0010 (start transaction success), then emv\_authenticateTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

#### 12.6.4.124 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a *emv\_authenticateTransaction* results in code 0x0004 (go online), then *emv\_completeTransaction* can automatically execute if parameter is set to TRUE

##### Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

#### 12.6.4.125 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

##### Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
-------------	--

<i>capkLen</i>	the length of capk data buffer
----------------	--------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.126 int emv\_setCRL ( IN BYTE \* list, IN int lsLen )****Set Certificate Revocation List**

Sets the CRL

**Parameters**

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.127 int emv\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )****Set Terminal Data**

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

**Parameters**

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

**12.6.4.128 int emv\_setTerminalMajorConfiguration ( IN int configuration )**

Sets the terminal major configuration in ICS .

**Parameters**

<i>configuration</i>	<p>A configuration value, range 1-23</p> <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.129 void emv\_setTransactionParameters ( IN double *amount*, IN double *amtOther*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F02 with amount 0x000000000100 would be "9F0206000000000100" If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

12.6.4.130 int emv\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

## Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable

Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#) >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.6.4.131 int felica\_authentication ( IN BYTE \* *key*, IN int *keyLen* )

FeliCa Authentication Provides a key to be used in a follow up FeliCa Read with MAC (3 blocks max) or Write with MAC (1 block max). This command must be executed before each Read w/MAC or Write w/MAC command

## Parameters

<i>key</i>	16-byte key used for MAC generation of Read or Write with MAC
<i>keyLen</i>	length of key, must be 16 bytes

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.132 `int felica_poll ( IN BYTE * systemCode, IN int systemCodeLen, OUT BYTE * respData, OUT int * respDataLen )`

## FeliCa Poll for Card

Polls for a Felica Card

## Parameters

<i>systemCode</i>	System Code.
<i>systemCodeLen</i>	Length of systemCode. Must be 2 bytes
<i>respData</i>	response data will be stored in respData. Poll response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of systemCode.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.133 `int felica_read ( IN BYTE * serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE * blockList, IN int blockListLen, OUT BYTE * blockData, OUT int * blockDataLen )`

## FeliCa Read

Reads up to 4 blocks.

## Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Maximum 4 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block 16 bytes.
<i>blockDataLen</i>	Length of blockData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.134 `int felica_readWithMac ( IN int blockCnt, IN BYTE * blockList, IN int blockListLen, OUT BYTE * blockData, OUT int * blockDataLen )`

## FeliCa Read with MAC Generation

Reads up to 3 blocks with MAC Generation. FeliCa Authentication must be performed first



## Parameters

<i>blockCnt</i>	Number of blocks in blockList. Maximum 3 block requests
<i>blockList</i>	Block to read. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockList.
<i>blockData</i>	Blocks read will be stored in blockData. Each block is 16 bytes.
<i>blockDataLen</i>	Length of blockData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.135 `int felica_requestService ( IN BYTE * nodeCode, IN int nodeCodeLen, OUT BYTE * respData, OUT int * respDataLen )`

FeliCa Request Service

Request Service for a Felica Card

## Parameters

<i>nodeCode</i>	Node Code List. Each node 2 bytes
<i>nodeCodeLen</i>	Length of nodeCode.
<i>respData</i>	response data will be stored in respData. Response as explained in FeliCA Lite-S User's Manual
<i>respDataLen</i>	Length of respData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.136 `int felica_SendCommand ( IN BYTE * command, IN int commandLen, OUT BYTE * respData, OUT int * respDataLen )`

FeliCa Send Command

Send a Felica Command

## Parameters

<i>command</i>	Command data from settlement center to be sent to felica card
<i>commandLen</i>	Length of command data
<i>respData</i>	Response data from felica card.
<i>respDataLen</i>	Length of respData.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.137 `int felica_write ( IN BYTE * serviceCodeList, IN int serviceCodeListLen, IN int blockCnt, IN BYTE * blockList, IN int blockListLen, IN BYTE * blockData, IN int blockDataLen, OUT BYTE * statusFlag, OUT int * statusFlagLen )`

FeliCa Write

Writes a block

## Parameters

<i>serviceCodeList</i>	Service Code List. Each service code in Service Code List = 2 bytes of data
<i>serviceCodeListLen</i>	Length of serviceCodeList
<i>blockCnt</i>	Number of blocks in blockList. Currently only support 1 block.
<i>blockList</i>	Block list. Each block in blockList = 2 or 3 bytes of data.
<i>blockListLen</i>	Length of blockData.
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.
<i>respData</i>	If successful, the Status Flag (2 bytes) is stored in respData.resData. Status flag response as explained in FeliCA Lite-S User's Manual, Section 4.5

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.138 `int felica_writeWithMac ( IN BYTE blockNum, IN BYTE * blockData, IN int blockDataLen )`

## FeliCa Write with MAC Generation

Writes a block with MAC Generation. FeliCa Authentication must be performed first

## Parameters

<i>blockNum</i>	Number of block
<i>blockData</i>	Block to write.
<i>blockDataLen</i>	Length of blockData. Must be 16 bytes.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.139 `int icc_exchangeAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

## Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.6.4.140 `int icc_getICCRReaderStatus ( OUT BYTE * status )`

## Get Reader Status

Returns the reader status

## Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.6.4.141 int `icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

12.6.4.142 int `icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

## Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

12.6.4.143 int `lcd_addButton ( IN char * screenName, IN int screenNameLen, IN char * buttonName, IN int buttonNameLen, IN BYTE type, IN BYTE alignment, IN int xCord, IN int yCord, IN char * label, IN int labelLen, OUT IDTLCDItem * returnItem )`

Add Button

Adds a button to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add button
<i>screenNameLen</i>	Length of screenName
<i>buttonName</i>	Button name that will be the target of add button
<i>buttonNameLen</i>	Length of buttonName
<i>type</i>	Button Type <ul style="list-style-type: none"> <li>• Large = 0x01</li> <li>• Medium = 0x02</li> <li>• Invisible = 0x03 (70px by 60 px)</li> </ul>
<i>alignment</i>	Position for Button <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x andy</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Button, range 0-271
<i>yCord</i>	y-coordinate for Button, range 0-479
<i>label</i>	Label to show on the button. Required for Large/Medium buttons. Not used for Invisible buttons.
<i>labelLen</i>	Length of label
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created button

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.144 `int lcd_addEthernet ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, OUT IDTLCDItem * returnItem )`

## Add Ethernet Settings

Adds an Ethernet settings to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add widget
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add widget
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for widget <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for widget, range 0-271
<i>yCord</i>	y-coordinate for widget, range 0-479
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.6.4.145 `int lcd_addImage ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char * filename, IN int filenameLen, OUT IDTLCDItem * returnItem )`

## Add Image

Adds a image to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add image
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add image
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Image <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Image, range 0-271
<i>yCord</i>	y-coordinate for Image, range 0-479
<i>filename</i>	Filename of the image. Must be available in device memory.
<i>filenameLen</i>	Length of filename
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created image

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.6.4.146 int lcd\_addLED ( IN char \* *screenName*, IN int *screenNameLen*, IN char \* *objectName*, IN int *objectNameLen*, IN BYTE *alignment*, IN int *xCord*, IN int *yCord*, OUT IDTLCDItem \* *returnItem*, IN BYTE \* *LED*, IN int *LEDLen* )

## Add LED

Adds a LED widget to a selected screen. Must execute lcd\_createScreen first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add LED
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add LED
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for LED <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x andy</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for LED, range 0-271
<i>yCord</i>	y-coordinate for LED, range 0-479
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created widget
<i>LED</i>	Must be 4 bytes, LED 0 = byte 0, LED 1 = byte 1, LED 2 = byte 2, LED 3 = byte 3 <ul style="list-style-type: none"> <li>• Value 0 = LED OFF</li> <li>• Value 1 = LED Green</li> <li>• Value 2 = LED Yellow</li> <li>• Value 3 = LED Red</li> </ul>
<i>LEDLen</i>	Length of LED

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.6.4.147 `int lcd_addText ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN int width, IN int height, IN BYTE fontID, IN BYTE * color, IN int colorLen, IN char * label, IN int labelLen, OUT IDTLCDItem * returnItem )`

## Add text

Adds a text component to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

## Parameters

<i>screenName</i>	Screen name that will be the target of add text
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Object name that will be the target of add text
<i>objectNameLen</i>	Length of objectName
<i>alignment</i>	Position for Text <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x andy</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Text, range 0-271
<i>yCord</i>	y-coordinate for Text, range 0-479
<i>width</i>	Width of text area
<i>height</i>	Height of text area
<i>fontID</i>	Font ID
<i>color</i>	Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000 <ul style="list-style-type: none"> <li>• Byte 0 = B</li> <li>• Byte 1 = G</li> <li>• Byte 2 = R</li> <li>• Byte 3 = Reserved</li> </ul>
<i>colorLen</i>	Length of color
<i>label</i>	Label to show on the text
<i>labelLen</i>	Length of label
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created text area

### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

Font ID	Typography Name	Font	Size
0	RoundBold_12	RoundBold.ttf	12
1	RoundBold_18	RoundBold.ttf	18
2	RoundBold_24	RoundBold.ttf	24
3	RoundBold_36	RoundBold.ttf	36
4	RoundBold_48	RoundBold.ttf	48
5	RoundBold_60	RoundBold.ttf	60
6	RoundBold_72	RoundBold.ttf	72
7	RoundCondensedBold_12	RoundCondensedBold.ttf	12
8	RoundCondensedBold_18	RoundCondensedBold.ttf	18
9	RoundCondensedBold_24	RoundCondensedBold.ttf	24



10	RoundCondensedBold_ - 36	RoundCondensedBold.ttf	36
11	RoundCondensedBold_ - 48	RoundCondensedBold.ttf	48
12	RoundCondensedBold_ - 60	RoundCondensedBold.ttf	60
13	RoundCondensedBold_ - 72	RoundCondensedBold.ttf	72
14	RoundCondensed-Medium_ 12	RoundCondensed-Medium_0.ttf	12
15	RoundCondensed-Medium_ 18	RoundCondensed-Medium_0.ttf	18
16	RoundCondensed-Medium_ 24	RoundCondensed-Medium_0.ttf	24
17	RoundCondensed-Medium_ 36	RoundCondensed-Medium_0.ttf	36
18	RoundCondensed-Medium_ 48	RoundCondensed-Medium_0.ttf	48
19	RoundCondensed-Medium_ 60	RoundCondensed-Medium_0.ttf	60
20	RoundCondensed-Medium_ 72	RoundCondensed-Medium_0.ttf	72
21	RoundCondensed-Semibold_ 12	RoundCondensed-Semibold.ttf	12
22	RoundCondensed-Semibold_ 18	RoundCondensed-Semibold.ttf	18
23	RoundCondensed-Semibold_ 24	RoundCondensed-Semibold.ttf	24
24	RoundCondensed-Semibold_ 36	RoundCondensed-Semibold.ttf	36
25	RoundCondensed-Semibold_ 48	RoundCondensed-Semibold.ttf	48
26	RoundCondensed-Semibold_ 60	RoundCondensed-Semibold.ttf	60
27	RoundCondensed-Semibold_ 72	RoundCondensed-Semibold.ttf	72
28	RoundMedium_ 12	RoundMedium.ttf	12
29	RoundMedium_ 18	RoundMedium.ttf	18
30	RoundMedium_ 24	RoundMedium.ttf	24
31	RoundMedium_ 36	RoundMedium.ttf	36
32	RoundMedium_ 48	RoundMedium.ttf	48
33	RoundMedium_ 60	RoundMedium.ttf	60
34	RoundMedium_ 72	RoundMedium.ttf	72
35	RoundSemibold_ 12	RoundSemibold.ttf	12
36	RoundSemibold_ 18	RoundSemibold.ttf	18
37	RoundSemibold_ 24	RoundSemibold.ttf	24
38	RoundSemibold_ 36	RoundSemibold.ttf	36
39	RoundSemibold_ 48	RoundSemibold.ttf	48
40	RoundSemibold_ 60	RoundSemibold.ttf	60
41	RoundSemibold_ 72	RoundSemibold.ttf	72

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16

Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20

12.6.4.148 `int lcd_addVideo ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int xCord, IN int yCord, IN char * filename, IN int filenameLen, OUT IDTLCDItem * returnItem )`

#### Add Video

Adds a video to a selected screen. Must execute `lcd_createScreen` first to establish a screen to draw on.

#### Parameters

<i>screenName</i>	Screen name that will be the target of add video
<i>screenNameLen</i>	Length of <i>screenName</i>
<i>objectName</i>	Object name that will be the target of add video
<i>objectNameLen</i>	Length of <i>objectName</i>
<i>alignment</i>	Position for Video <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>xCord</i>	x-coordinate for Video, range 0-271
<i>yCord</i>	y-coordinate for Video, range 0-479
<i>filename</i>	Filename of the video. Must be available in the sd card.
<i>filenameLen</i>	Length of filename
<i>returnItem</i>	The item includes screen ID, object ID, top-left x-coordinate, top-left y-coordinate, bottom-right x-coordinate, bottom-left y-coordinate, which are all assigned to the created video

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

Item	Maximum can be created for each screen
Text Area	20
Large Button	8
Medium Button	16
Invisible Button	16
Numeric Entry	1
Ethernet Setting	1
Led widget	1
image	20
video	1

12.6.4.149 `int lcd_clearScreenInfo ( )`

#### Clear Screen Info

Clear all current screen information in RAM and flash. And then show 'power-on screen'

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.150 `int lcd_cloneScreen ( IN char * screenName, IN int screenNameLen, IN char * cloneName, IN int cloneNameLen, OUT int * cloneID )`

## Clone Screen

Clones an existing screen.

## Parameters

<i>screenName</i>	Screen name to clone.
<i>screenNameLen</i>	Length of <i>screenName</i> .
<i>cloneName</i>	Name of the cloned screen.
<i>cloneNameLen</i>	Length of <i>cloneName</i> .
<i>cloneID</i>	Screen ID of the cloned screen

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.6.4.151 `int lcd_createScreen ( IN char * screenName, IN int screenNameLen, OUT int * ScreenID )`

## Create Screen

Creates a new screen.

## Parameters

<i>screenName</i>	Screen name to use.
<i>screenNameLen</i>	Length of <i>screenName</i> .
<i>screenID</i>	Screen ID that was created.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.152 `int lcd_destroyScreen ( IN char * screenName, IN int screenNameLen )`

## Destroy Screen

Destroys a previously created inactive screen. The screen cannot be active

## Parameters

<i>screenName</i>	Screen name to destroy. The screen number is assigned with <code>lcd_createScreen</code> .
<i>screenNameLen</i>	Length of <i>screenName</i> .

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.153 `int lcd_getActiveScreen ( OUT char * screenName, IN_OUT int * screenNameLen )`

## Get Active Screen

Returns the active screen ID.

## Parameters

<i>screenName</i>	Screen name this is active.
<i>screenNameLen</i>	Length of screenName.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.154 `int lcd_getAllObjects ( IN char * screenName, IN int screenNameLen, IN_OUT int * objectNumbers, OUT IDTObjectInfo * objectInfo )`

## Get All Objects on Screen

Get all created objects' name on certain screen

## Parameters

<i>screenName</i>	Screen name to get all objects
<i>screenNameLen</i>	Length of screenName
<i>objectNumbers</i>	Number of created objects
<i>returnObjects</i>	Array of all created objects each element includes -objectID of a created object -objectName of a created object -objectNameLen of objectName

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.155 `int lcd_getAllScreens ( IN_OUT int * screenNumbers, OUT IDTScreenInfo * screenInfo )`

## Get All Screens

Get all created screens' name

## Parameters

<i>screenNumbers</i>	Number of created screens
<i>screenInfo</i>	Array of all created screens each element includes -screenID of a created screen -screen-Name of a created screen -screenNameLen of screenName

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.156 `int lcd_getButtonEvent ( OUT int * screenID, OUT int * objectID, OUT char * screenName, IN_OUT int * screenNameLen, OUT char * objectName, IN_OUT int * objectNameLen, OUT int * isLongPress )`

## Get Button Event

Reports back the ID of the button that encountered a click event after the last Get Button Event.

## Parameters

<i>screenID</i>	Screen ID of the last clicked button
-----------------	--------------------------------------

<i>objectID</i>	Button ID of the last clicked button
<i>screenName</i>	Screen Name of the last clicked button
<i>screenNameLen</i>	Length of screenName
<i>objectName</i>	Button Name of the last clicked button
<i>objectNameLen</i>	Length of objectName
<i>isLongPress</i>	1 = Long Press, 0 = Short Press
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices)

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.157 int lcd\_loadScreenInfo ( )**

Load Screen Info

Load all current screen information from flash to RAM

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.158 int lcd\_queryObjectbyID ( IN int *objectID*, OUT int \* *objectNumbers*, OUT IDTScreenInfo \* *screenInfo* )**

Queery Object by ID

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object ID

**Parameters**

<i>objectID</i>	ID of the checked object
<i>objectNumbers</i>	Number of the checked object
<i>screenInfo</i>	screen names containing the item

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.159 int lcd\_queryObjectbyName ( IN char \* *objectName*, IN int *objectNameLen*, IN\_OUT int \* *objectNumbers*, OUT IDTScreenInfo \* *screenInfo* )**

Queery Object by Name

Check if the given object exists or not. If exists, return all screen names which contains the object of the given object name

**Parameters**

<i>objectName</i>	Name of the checked object
<i>objectNameLen</i>	Length of objectName
<i>objectNumbers</i>	Number of the checked object

<i>screenInfo</i>	Array of all the screen names that contain the object
-------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.160 `int lcd_queryScreenbyID ( IN int screenID, OUT int * result, OUT int * screenName, IN_OUT int * screenNameLen )`

**Queery Screen by ID**

Check if the given screen exists or not

**Parameters**

<i>screenID</i>	ID of the checked screen
<i>result</i>	the checking result
<i>screenName</i>	Name of the checked screen
<i>screenNameLen</i>	Length of screenName
<i>ip</i>	Optional: IP address to execute command on (for IP connected devices)

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.161 `int lcd_queryScreenbyName ( IN char * screenName, IN int screenNameLen, OUT int * result )`

**Queery Screen by Name**

Check if the given screen exists or not

**Parameters**

<i>screenName</i>	Name of the checked screen
<i>screenNameLen</i>	Length of screenName
<i>result</i>	the checking result

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.162 `void lcd_registerCallBk ( pLCD_callBack pLCDf )`

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

12.6.4.163 `int lcd_removeItem ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen )`

**Removed Item**

Removes a component.

**Parameters**

<i>screenName</i>	Screen name where to remove the target from.
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.164 int lcd\_setBacklight ( IN BYTE *backlightVal* )

**Set Backlight**

Set backlight percentage. If the percent > 100, it will be rejected. If 0 < percent < 10, backlight percent will be set to 10. If percent == 0, backlight will be turned off

**Parameters**

<i>backlightVal</i>	Backlight percent value to be sat
---------------------	-----------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.165 int lcd\_showScreen ( IN char \* *screenName*, IN int *screenNameLen* )

**Show Screen**

Displays and makes active a previously created screen.

**Parameters**

<i>screenName</i>	Screen to display. The screen name is defined with <code>lcd_createScreen</code> .
<i>screenNameLen</i>	Length of screenName.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.166 int lcd\_storeScreenInfo ( )

**Store Screen Info**

Store all current screen information from RAM to flash

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.167 int lcd\_updateColor ( IN char \* *screenName*, IN int *screenNameLen*, IN char \* *objectName*, IN int *objectNameLen*, IN BYTE \* *color*, IN int *colorLen* )

**Update Color**

Updates the component color, or updates the LED colors if specifying LCD component

## Parameters

<i>screenName</i>	Screen name that will be the target of update color
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>color</i>	<p>Non LCD Widget: Four bytes for color, example, Blue = 0xFF000000, Black = 0x00000000</p> <ul style="list-style-type: none"> <li>• Byte 0 = B</li> <li>• Byte 1 = G</li> <li>• Byte 2 = R</li> <li>• Byte 3 = Reserved LCD Widget: Four bytes for LED color, byte 0 = LED 0, byte 1 = LED 1, byte 2 = LED2, byte 3 = LED 3</li> <li>• Value 0 = LED OFF</li> <li>• Value 1 = LED Green</li> <li>• Value 2 = LED Yellow</li> <li>• Value 3 = LED Red</li> </ul>
<i>colorLen</i>	Length of color.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.168 `int lcd_updateLabel ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN char * label, IN int labelLen )`

## Update Label

Updates the component label.

## Parameters

<i>screenName</i>	Screen name that will be the target of update label
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>label</i>	Label to show on the component
<i>labelLen</i>	Length of label.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.169 `int lcd_updatePosition ( IN char * screenName, IN int screenNameLen, IN char * objectName, IN int objectNameLen, IN BYTE alignment, IN int new_xCord, IN int new_yCord )`

## Update Position

Updates the component position.



## Parameters

<i>screenName</i>	Screen Name that will be the target of update position
<i>screenNameLen</i>	Length of screenName.
<i>objectName</i>	Identifier of the component
<i>objectNameLen</i>	Length of objectName.
<i>alignment</i>	Alignment for the target <ul style="list-style-type: none"> <li>• 0 = Display object at the horizon center of specified y, while x ignored</li> <li>• 1 = Display object at specified x and y</li> <li>• 2 = Display object at center of screen, x, y are both ignored</li> <li>• 3 = Display object at left of the screen of specified y, while x ignored</li> <li>• 4 = Display object at right of the screen of specified y, while x ignored</li> </ul>
<i>new_xCord</i>	x-coordinate for Text, range 0-271
<i>new_yCord</i>	y-coordinate for Text, range 0-479

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.170 int `loyalty_cancelTransaction ( )`

Cancel Loyalty Transaction Only for VP6800

Cancels the currently executing transaction.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.171 int `loyalty_cancelTransactionSilent ( int enable )`

Cancel Loyalty Transaction Silent Only for VP6800

Cancel transaction with or without showing the LCD message

## Parameters

<i>enable</i>	0: With LCD message 1: Without LCD message
---------------	--

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.6.4.172 void `loyalty_registerCallBk ( pEMV_callBack pEMVf )`

To register the loyalty callback function to get the EMV processing response. (Pass NULL to disable the callback.)  
Only for VP6800

12.6.4.173 int `loyalty_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen, IN const int cardType, IN const int iccReadType )`

Start Loyalty Transaction Request Only for VP6800

Authorizes the transaction for an MSR/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)  • SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03)  • SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F02 with amount 0x000000000100 would be 0x9F0206000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Parameters

<i>cardType</i>	Set available card to accept. 0 = MSR only. 1 = MSR and ICC.
<i>iccReadType</i>	In case of ICC reading, this is how to behave. 0 = Same as device_startTransaction 1 = When reading ICC, read DF4F(JIS2EquivalentData) in ReadRecord.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of device\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1, 2, 3, 4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal

- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.6.4.174 int msr\_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.175 void msr\_registerCallBk ( pMSR\_callBack pMSRf )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.6.4.176 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.6.4.177 int msr\_startMSRSwipe ( IN int \_timeout )

Start MSR Swipe

Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to swipeMSRData()

##### Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

#### 12.6.4.178 void parseMSRData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTMSRData \* cardData )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

## 12.6.4.179 int pin\_cancelPINEntry ( )

## Cancel PIN Entry

Cancels PIN entry request

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.180 int pin\_capturePin ( IN int timeout, IN int type, IN char \* PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char \* message, IN int messageLen )

## Capture PIN

## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> <li>• 00h = MKSK to encrypt PIN, Internal PAN (from MSR)</li> <li>• 01h = DUKPT to encrypt PIN, Internal PAN (from MSR)</li> <li>• 10h = MKSK to encrypt PIN, External Plaintext PAN</li> <li>• 11h = DUKPT to encrypt PIN, External Plaintext PAN</li> <li>• 20h = MKSK to encrypt PIN, External Ciphertext PAN</li> <li>• 21h = DUKPT to encrypt PIN, External Ciphertext PAN</li> </ul>
<i>PAN</i>	Personal Account Number (if internal, value is 0)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message
<i>messageLen</i>	Length of message

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.181 int pin\_capturePinExt ( IN int type, IN char \* PAN, IN int PANLen, IN int minPIN, IN int maxPIN, IN char \* message, IN int messageLen, IN char \* verify, IN int verifyLen )

- Capture PIN Ext

## Parameters

<i>type</i>	PAN and Key Type <ul style="list-style-type: none"> <li>• 00h: MKSK to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 01h: DUKPT to encrypt PIN, Internal PAN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 10h: MKSK to encrypt PIN, External Plaintext PAN</li> <li>• 11h: DUKPT to encrypt PIN, External Plaintext PAN</li> <li>• 20h: MKSK to encrypt PIN, External Ciphertext PAN (for PIN pad only)</li> <li>• 21h: DUKPT to encrypt PIN, External Ciphertext PAN (for PIN pad only)</li> <li>• 80h: MKSK to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 81h: DUKPT to encrypt PIN, Internal PAN, Verify PIN (from MSR or Manual PAN Entry or Contactless EMV Transaction)</li> <li>• 90h: MKSK to encrypt PIN, External Plaintext PAN, Verify PIN</li> <li>• 91h: DUKPT to encrypt PIN, External Plaintext PAN, Verify PIN</li> </ul>
<i>PAN</i>	Personal Account Number (if internal, value is 0)
<i>PANLen</i>	Length of PAN
<i>minPIN</i>	Minimum PIN Length
<i>maxPIN</i>	Maximum PIN Length
<i>message</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>messageLen</i>	Length of 1st scenario LCD message, valid in 00h~21h (0~33).If no LCD message input, length is 00h, and display default msg "PLEASE ENTER PIN"
<i>verify</i>	LCD Message Up to 2 lines of text, each line 1-16, separated by 0x00
<i>verifyLen</i>	Length of 2nd Scenario LCD message.valid in 00h~21h (0~33).This field is present only when PAN and Key Type has Verify PIN.If no LCD message input, length is 00h, and display default msg " ENTER PIN AGAIN"

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.182 `int pin_getPanEntry ( IN int csc, IN int expDate, IN int ADR, IN int ZIP, IN int mod10CK, IN int timeout, IN int encPANOnly )`

## Capture Amount

## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	LCD Message
<i>messageLen</i>	Length of message

<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> <li>1. Calculate 32 bytes Hash for &lt;Display Flag&gt;&lt;Key max="" length&gt;=""&gt;&lt; Key Min Length&gt;&lt;Plaintext display="" message&gt;=""&gt;</li> <li>2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data</li> <li>3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature</li> </ol>
<i>signatureLen</i>	Length of signature

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Get Pan

prompt the user to manually enter a card PAN and Expiry Date (and optionally CSC) from the keypad and return it to the POS.

#### Parameters

<i>csc</i>	Request CSS
<i>expDate</i>	Request Expiration Date
<i>ADR</i>	Request Address
<i>ZIP</i>	Request Zip
<i>mod10CK</i>	Validate entered PAN passes MOD-10 checking before accepting
<i>timeout</i>	Number of seconds that the reader waits for the data entry session to complete, stored as a big-endian number. 0 = no timeout
<i>encPANOnly</i>	Output only encrypted PAN

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.6.4.183** `int pin_inputFromPrompt ( BYTE mask, BYTE preClearText, BYTE postClearText, int minLen, int maxLen, char * lang, BYTE promptID, char * defaultResponse, int defaultResponseLen, int timeout )`

Get PIN Input from Prompt Results returned to PIN Callback. If successful function key capture, data is returned as IDTPINData.keyString.

#### Parameters

<i>mask</i>	0 = no masking, 1 = Display "*" for numeric key according to Pre-Cleartext and Post-Cleartext display
<i>preClearText</i>	Range 0-6 Characters to mask at start of text if masking is on;
<i>postClearText</i>	Range 0-6 Characters to mask at end of text if masking is on;
<i>minLen</i>	Minimum number of digits required to input
<i>maxLen</i>	Maximum number of digits allowed to be input
<i>lang</i>	Valid values; "EN" is English display message "JP" is Japanese display message "ES" is Spanish display message "FR" is French display message "ZH" is Chinese display message "PT" is Portuguese display message
<i>promptID</i>	Valid values: 0x00: Enter Phone Number 0x01: Enter IP Address 0x02: Enter Subnet Mask 0x03: Enter Default Gateway 0x04: Enter Odometer Reading/Mileage 0x05: Enter Employee ID 0x06: Enter Password for Default Factory Setting 0x07: Enter Email Address (Full keyboard)

<i>defaultResponse</i>	Default String on input field
<i>default-ResponseLen</i>	Length of defaultResponse
<i>timeout</i>	Timeout, in seconds

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.184 int pin\_promptForNumericKey ( IN int *timeout*, IN int *maskInput*, IN int *minLen*, IN int *maxLen*, IN char \* *message*, IN int *messageLen*, BYTE \* *signature*, IN int *signatureLen* )

#### Capture Numeric Input

##### Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>maskInput</i>	<ul style="list-style-type: none"> <li>• 0 = Display numeric for numeric key on LCD</li> <li>• 1 = Display * for numeric key on LCD</li> </ul>
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>message</i>	Plaintext Display Message. - 16 chars max
<i>messageLen</i>	Length of message
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> <li>1. Calculate 32 bytes Hash for &lt;Display Flag&gt;&lt;Key max="" length&gt;=""&gt;&lt; Key Min Length&gt;&lt;Plaintext display="" message&gt;=""&gt;</li> <li>2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data</li> <li>3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature</li> </ol>
<i>signatureLen</i>	Length of signature

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.6.4.185 int pin\_promptForNumericKeyWithSwipe ( IN int *timeout*, IN BYTE *function*, IN int *minLen*, IN int *maxLen*, IN char \* *line1*, IN int *line1Len*, IN char \* *line2*, IN int *line2Len*, BYTE \* *signature*, IN int *signatureLen* )

#### Capture Numeric Input



## Parameters

<i>timeout</i>	Timeout, in seconds. Value of 0 will use system timeout, if any
<i>function</i>	<ul style="list-style-type: none"> <li>• 0x00 = Plaintext Input</li> <li>• 0x01 = Masked Input</li> <li>• 0x02 = Delayed Masking Input</li> <li>• 0x10 = Plaintext Input + MSR Active</li> <li>• 0x11 = Masked Input + MSR Active</li> <li>• 0x12 = Delayed Masking Input + MSR Active</li> </ul>
<i>minLen</i>	Minimum input Length
<i>maxLen</i>	Maximum input Length
<i>line1</i>	Line 1 of LCD Message - 16 chars max
<i>line1Len</i>	Length of line1
<i>line2</i>	Line 2 of LCD Message - 16 chars max
<i>line2Len</i>	Length of line2
<i>signature</i>	Display message signed by Numeric Private Key using RSAPSS algorithm: <ol style="list-style-type: none"> <li>1. Calculate 32 bytes Hash for &lt;Display Flag&gt;&lt;Key max="" length&gt;=""&gt;&lt; Key Min Length&gt;&lt;Plaintext display="" message&gt;=""&gt;</li> <li>2. Using RSAPSS algorithm calculate the Hash to be 256 bytes Raw Data</li> <li>3. Using Numeric Private Key to sign the Raw Data to be 256 bytes signature</li> </ol>
<i>signatureLen</i>	Length of signature

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.186 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

## 12.6.4.187 int pin\_setKeyValues ( int mode )

Set Key Values

Set return key values on or off

## Parameters

<i>mode</i>	On: 1, Off: 0
-------------	---------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.6.4.188 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

#### 12.6.4.189 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

#### 12.6.4.190 int rs232\_device\_init ( int deviceType, int port\_number, int brate )

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

##### Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. | | 38 | ttyIDG | n.a. |

##### Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.6.4.191 char\* SDK\_Version ( )

To Get SDK version

##### Returns

return the SDK version string

#### 12.6.4.192 int setAbsoluteLibraryPath ( const char \* absoluteLibraryPath )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

##### Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.7 Source\_C/libIDT\_PipReader.h File Reference

PipReader API.

```
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

### Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callback pEMVf)`
- `void ctls_registerCallBk (pMSR_callback pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callbackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callback pPINf)`
- `void device_registerCameraCallBk (pCMR_callback pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callback pCSFSf)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int rs232_device_init (int deviceType, int port_number, int brate)`
- `int device_setCurrentDevice (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcriton)`
- `int device_isConnected ()`
- `int device_isAttached (int deviceType)`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`
- `int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`
- `int device_pingDevice ()`
- `int device_controlUserInterface (IN BYTE *values)`
- `int device_getCurrentDeviceType ()`
- `int device_SendDataCommandNEO (IN int cmd, IN int subCmd, IN BYTE *data, IN int dataLen, OUT BYTE *response, IN_OUT int *respLen)`

- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()
- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.7.1 Detailed Description

PipReader API. PipReader Global API methods.

### 12.7.2 Macro Definition Documentation

#### 12.7.2.1 #define IN

INPUT parameter.

#### 12.7.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.7.2.3 #define OUT

OUTPUT parameter.

### 12.7.3 Typedef Documentation

#### 12.7.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.7.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

#### 12.7.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

#### 12.7.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

#### 12.7.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

#### 12.7.3.6 typedef void(\* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.7.3.7 typedef void(\* pMSR\_callback)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

#### 12.7.3.8 typedef void(\* pMSR\_callbackp)(int, IDTMSRData \*)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callback

**12.7.3.9** `typedef void(* pPIN_callBack)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

**12.7.3.10** `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

**12.7.3.11** `typedef void(* pSendDataLog)(unsigned char *, int)`

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

**12.7.3.12** `typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)`

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

## 12.7.4 Function Documentation

**12.7.4.1** `int config_getSerialNumber ( OUT char * sNumber )`

DEPRECATED : please use `config_getSerialNumber_Len(OUT char* sNumber, IN_OUT int *sNumberLen)`

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

**12.7.4.2** `int config_getSerialNumber_Len ( OUT char * sNumber, IN_OUT int * sNumberLen )`

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.7.4.3 int ctls\_activateTransaction ( IN const int *\_timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DFO1 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU



#### 12.7.4.4 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.5 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.6 int ctls\_getConfigurationGroup ( IN int group, OUT BYTE \* tlv, OUT int \* tlvLen )

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.7 void ctls\_registerCallBk ( pMSR\_callBack pCTLSf )

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.7.4.8 void ctls\_registerCallBkp ( pMSR\_callBackp pCTLSf )

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.7.4.9 int ctls\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.10 int ctls\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.11 int ctls\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

##### Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.12 int ctls\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

##### Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.13 int ctls\_removeConfigurationGroup ( int group )

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

##### Parameters

<i>group</i>	Configuration Group
--------------	---------------------

##### Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

#### 12.7.4.14 int ctls\_retrieveAIDList ( OUT BYTE \* AIDList, IN\_OUT int \* AIDListLen )

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.7.4.15 int `ctls_retrieveApplicationData` ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

## Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.7.4.16 int `ctls_retrieveCAPK` ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	the length of key data buffer <ul style="list-style-type: none"> <li>•</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.7.4.17 int ctls\_retrieveCAPKList ( OUT BYTE \* keys, IN\_OUT int \* keysLen )**

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.7.4.18 int ctls\_retrieveTerminalData ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )**

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.7.4.19 int ctls\_setApplicationData ( IN BYTE \* tlv, IN int tlvLen )**

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

**Parameters**

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.7.4.20 int ctls\_setCAPK ( IN BYTE \* capk, IN int capkLen )**

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.7.4.21 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.7.4.22 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.7.4.23 `int ctls_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal

- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.7.4.24 int device\_close ( )

Close the device

##### Returns

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.7.4.25 int device\_controlUserInterface ( IN BYTE \* values )

##### Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
```

```

- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On

```

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.26 int device\_enablePassThrough ( int *enablePassThrough* )

##### Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

##### Enable Pass Through - NEO

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

#### Parameters

<i>enablePass-Through</i>	1 = pass through ON, 0 = pass through OFF
---------------------------	---

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.27 int device\_getCurrentDeviceType ( )

##### Get current active device type

#### Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

#### 12.7.4.28 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

#### Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.7.4.29 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version



## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.7.4.30 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";

- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.7.4.31 int device\_getMerchantRecord ( IN int *index*, OUT BYTE \* *record* )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int *index*, OUT BYTE \* *record*, IN\_OUT int \**recordLen*)

##### Get Merchant Record

Gets the merchant record for the device.

##### Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i> ;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

##### Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

##### See Also

[ErrorCode](#)

#### 12.7.4.32 int device\_getMerchantRecord\_Len ( IN int *index*, OUT BYTE \* *record*, IN\_OUT int \* *recordLen* )

##### Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

[ErrorCode](#)

## 12.7.4.33 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

## Returns

SDK wait time in seconds

12.7.4.34 int device\_getTransactionResults ( IDTMSRData \* *cardData* )

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

## Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

## Returns

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

## See Also

[ErrorCode](#)

## 12.7.4.35 int device\_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.7.4.36 int device\_isAttached ( int *deviceType* )

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

## 12.7.4.37 int device\_isConnected ( )

Check the device connected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

## 12.7.4.38 int device\_pingDevice ( )

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.7.4.39 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.7.4.40 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.7.4.41 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to NEO device

Sends a command to the NEO device .

## Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.7.4.42 int device\_setBurstMode ( IN BYTE *mode* )

Send Burst Mode - NEO

Sets the burst mode for the device.

## Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

## Returns

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

## See Also

`ErrorCode`

12.7.4.43 `int device_setCurrentDevice ( int deviceType )`

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

12.7.4.44 `int device_setMerchantRecord ( int index, int enabled, char * merchantID, char * merchantURL )`

Set Merchant Record - NEO Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.7.4.45 int device\_setPollMode ( IN BYTE mode )

## Set Poll Mode - NEO

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

## Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.7.4.46 void device\_setSDKWaitTime ( int waitTime )

## Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

## 12.7.4.47 void emv\_registerCallBk ( pEMV\_callback pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

## 12.7.4.48 void parseMSRData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTMSRData \* cardData )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

## 12.7.4.49 void pin\_registerCallBk ( pPIN\_callback pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

#### 12.7.4.50 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

#### 12.7.4.51 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

#### 12.7.4.52 int rs232\_device\_init ( int deviceType, int port\_number, int brate )

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

##### Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

##### Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

##### Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.7.4.53 char\* SDK\_Version ( )

To Get SDK version

##### Returns

return the SDK version string

#### 12.7.4.54 int setAbsoluteLibraryPath ( const char \* absoluteLibraryPath )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.



## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.8 Source\_C/libIDT\_SpectrumPro.h File Reference

SpectrumPro API.

```
#include "IDTDef.h"
```

## Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

## Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callBack )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack )(int, IDTMSRData)`
- `typedef void(* pMSR_callBackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callBack )(int, IDTPINData *)`
- `typedef void(* pCMR_callBack )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack )(BYTE status)`
- `typedef void(* ftpComm_callBack )(int, int, int)`
- `typedef void(* httpComm_callBack )(BYTE *, int)`
- `typedef void(* v4Comm_callBack )(BYTE, BYTE, BYTE *, int)`

## Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, pReadDataLog pFRead)
- void [emv\\_registerCallBk](#) (pEMV\_callBack pEMVf)
- void [msr\\_registerCallBk](#) (pMSR\_callBack pMSRf)
- void [msr\\_registerCallBkp](#) (pMSR\_callBackp pMSRf)
- void [pin\\_registerCallBk](#) (pPIN\_callBack pPINf)
- void [device\\_registerCameraCallBk](#) (pCMR\_callBack pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callBack pCSFSf)
- char \* [SDK\\_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char \*absoluteLibraryPath)
- int [device\\_init](#) ()
- int [rs232\\_device\\_init](#) (int deviceType, int port\_number, int brate)
- int [device\\_setCurrentDevice](#) (int deviceType)
- int [device\\_close](#) ()

- void `device_getResponseCodeString` (IN int returnCode, OUT char \*despcriton)
- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_getFirmwareVersion` (OUT char \*firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommand` (IN BYTE \*cmd, IN int cmdLen, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int `device_rebootDevice` ()
- int `device_updateFirmware` (IN BYTE \*firmwareData, IN int firmwareDataLen, IN char \*firmwareName, IN int encryptionType, IN BYTE \*keyBlob, IN int keyBlobLen)
- int `config_getModelNumber` (OUT char \*sNumber)
- int `config_getModelNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int `config_getSerialNumber` (OUT char \*sNumber)
- int `config_getSerialNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int `device_pollCardReader` (OUT BYTE \*status)
- int `device_pollCardReader_Len` (OUT BYTE \*status, IN\_OUT int \*statusLen)
- int `device_getSpectrumProKSN` (IN int type, OUT BYTE \*KSN)
- int `device_getSpectrumProKSN_Len` (IN int type, OUT BYTE \*KSN, IN\_OUT int \*KSNLen)
- int `device_getSDKWaitTime` ()
- void `device_setSDKWaitTime` (int waitTime)
- int `device_getThreadStackSize` ()
- void `device_setThreadStackSize` (int threadSize)
- int `icc_powerOnICC` (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int `icc_powerOffICC` ()
- int `icc_getICCReaderStatus` (OUT BYTE \*status)
- int `emv_getEMVKernelVersion` (OUT char \*version)
- int `emv_getEMVKernelVersion_Len` (OUT char \*version, IN\_OUT int \*versionLen)
- int `emv_getEMVKernelCheckValue` (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- void `emv_setAutoAuthenticateTransaction` (IN int authenticate)
- void `emv_setAutoCompleteTransaction` (IN int complete)
- int `emv_getAutoAuthenticateTransaction` ()
- int `emv_getAutoCompleteTransaction` ()
- int `emv_getEMVConfigurationCheckValue` (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- void `emv_allowFallback` (IN int allow)
- int `emv_startTransaction` (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int `emv_activateTransaction` (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int `emv_authenticateTransaction` (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int `emv_authenticateTransactionWithTimeout` (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int `emv_completeTransaction` (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int `emv_cancelTransaction` ()
- int `emv_retrieveTransactionResult` (IN BYTE \*tags, IN int tagsLen, IDTTransactionData \*cardData)
- int `emv_callbackResponseLCD` (IN int type, byte selection)
- int `emv_callbackResponseMSR` (IN BYTE \*MSR, IN\_OUT int MSRLen)
- int `emv_retrieveApplicationData` (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setApplicationData` (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int `emv_removeApplicationData` (IN BYTE \*AID, IN int AIDLen)
- int `emv_removeAllApplicationData` ()
- int `emv_retrieveAIDList` (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int `emv_retrieveTerminalData` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setTerminalData` (IN BYTE \*tlv, IN int tlvLen)
- int `emv_removeTerminalData` ()
- int `emv_retrieveCAPK` (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)

- int `emv_setCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeAllCAPK` ()
- int `emv_retrieveCAPKList` (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int `emv_retrieveTerminalID` (OUT char \*terminalID)
- int `emv_retrieveTerminalID_Len` (OUT char \*terminalID, IN\_OUT int \*terminalIDLen)
- int `emv_setTerminalID` (IN char \*terminalID)
- int `emv_retrieveCRL` (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int `emv_setCRL` (IN BYTE \*list, IN int lsLen)
- int `emv_removeCRL` (IN BYTE \*list, IN int lsLen)
- int `emv_removeAllCRL` ()
- int `msr_clearMSRData` ()
- int `msr_getMSRData` (OUT BYTE \*reData, IN\_OUT int \*reLen)
- int `msr_cancelMSRSwipe` ()
- int `msr_startMSRSwipe` (IN int \_timeout)
- void `parseMSRData` (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)
- int `pin_getPIN` (IN int mode, IN int PANSource, IN char \*iccPAN, IN int IN iccPANLen, int startTimeout, IN int entryTimeout, IN char \*language, IN int languageLen)
- int `pin_cancelPINEntry` ()
- void `parsePINBlockData` (IN BYTE \*resData, IN int resLen, IN\_OUT IDTPINData \*cardData)
- void `parsePINData` (IN BYTE \*resData, IN int resLen, IN\_OUT IDTPINData \*cardData)

### 12.8.1 Detailed Description

SpectrumPro API. Spectrum Pro Global API methods.

### 12.8.2 Macro Definition Documentation

#### 12.8.2.1 #define IN

INPUT parameter.

#### 12.8.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.8.2.3 #define OUT

OUTPUT parameter.

### 12.8.3 Typedef Documentation

#### 12.8.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.8.3.2 `typedef void(* httpComm_callBack)(BYTE *, int)`

Define the comm callback function to get the async url data

It should be registered using the `comm_registerHTTPCallback`

#### 12.8.3.3 `typedef void(* pCMR_callBack)(int, IDTCMRData *)`

Define the camera callback function to get the image data

It should be registered using the `device_registerCameraCallBk`,

#### 12.8.3.4 `typedef void(* pCSFS_callBack)(BYTE status)`

Define the card status and front switch callback function to get card and front switch status

It should be registered using the `device_registerCardStatusFrontSwitchCallBk`,

#### 12.8.3.5 `typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the `emv_registerCallBk`,

#### 12.8.3.6 `typedef void(* pMessageHotplug)(int, int)`

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.8.3.7 `typedef void(* pMSR_callBack)(int, IDTMSRData)`

Define the MSR callback function to get the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

#### 12.8.3.8 `typedef void(* pMSR_callBackp)(int, IDTMSRData *)`

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

#### 12.8.3.9 `typedef void(* pPIN_callBack)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

#### 12.8.3.10 `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

**12.8.3.11** typedef void(\* pSendDataLog)(unsigned char \*, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

**12.8.3.12** typedef void(\* v4Comm\_callback)(BYTE, BYTE, BYTE \*, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

**12.8.4 Function Documentation****12.8.4.1** int config\_getModelNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getModelNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.8.4.2** int config\_getModelNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.8.4.3** int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.8.4.4** int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.8.4.5 `int device_close ( )`

Close the device

## Returns

RETURN\_CODE: 0: success, 0x0A: failed

12.8.4.6 `int device_getCurrentDeviceType ( )`

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.8.4.7 `int device_getFirmwareVersion ( OUT char * firmwareVersion )`

DEPRECATED : please use `device_getFirmwareVersion_Len(OUT char* firmwareVersion, IN_OUT int *firmwareVersionLen)`

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.8.4.8 `int device_getFirmwareVersion_Len ( OUT char * firmwareVersion, IN_OUT int * firmwareVersionLen )`

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.8.4.9 `void device_getResponseCodeString ( IN int returnCode, OUT char * description )`

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
- 3: "time out for task or CMD";
- 4: "wrong parameter";
- 5: "SDK is doing MSR or ICC task";
- 6: "SDK is doing PINPad task";
- 7: "SDK is doing CTLS task";
- 8: "SDK is doing EMV task";
- 9: "SDK is doing Other task";
- 10: "err response or data";
- 11: "no reader attached";
- 12: "mono audio is enabled";
- 13: "did connection";
- 14: "audio volume is too low";
- 15: "task or CMD be canceled";
- 16: "UF wrong string format";
- 17: "UF file not found";
- 18: "UF wrong file format";
- 19: "Attempt to contact online host failed";
- 20: "Attempt to perform RKI failed";
- 22: "Buffer size is not enough";
- 0X300: "Key Type(TDES) of Session Key is not same as the related Master Key.";
- 0X400: "Related Key was not loaded.";
- 0X500: "Key Same.";
- 0X501: "Key is all zero";
- 0X502: "TR-31 format error";
- 0X702: "PAN is Error Key.";
- 0X705: "No Internal MSR PAN (or Internal MSR PAN is erased timeout)";
- 0X0C01: "Incorrect Frame Tag";

- 0X0C02: "Incorrect Frame Type";
- 0X0C03: "Unknown Frame Type";
- 0X0C04: "Unknown Command";
- 0X0C05: "Unknown Sub-Command";
- 0X0C06: "CRC Error";
- 0X0C07: "Failed";
- 0X0C08: "Timeout";
- 0X0C0A: "Incorrect Parameter";
- 0X0C0B: "Command Not Supported";
- 0X0C0C: "Sub-Command Not Supported";
- 0X0C0D: "Parameter Not Supported / Status Abort Command";
- 0X0C0F: "Sub-Command Not Allowed";
- 0X0D01: "Incorrect Header Tag";
- 0X0D02: "Unknown Command";
- 0X0D03: "Unknown Sub-Command";
- 0X0D04: "CRC Error in Frame";
- 0X0D05: "Incorrect Parameter";
- 0X0D06: "Parameter Not Supported";
- 0X0D07: "Mal-formatted Data";
- 0X0D08: "Timeout";
- 0X0D0A: "Failed / NACK";
- 0X0D0B: "Command not Allowed";
- 0X0D0C: "Sub-Command not Allowed";
- 0X0D0D: "Buffer Overflow (Data Length too large for reader buffer)";
- 0X0D0E: "User Interface Event";
- 0X0D11: "Communication type not supported, VT-1, burst, etc.";
- 0X0D12: "Secure interface is not functional or is in an intermediate state.";
- 0X0D13: "Data field is not mod 8";
- 0X0D14: "Pad - 0X80 not found where expected";
- 0X0D15: "Specified key type is invalid";
- 0X0D1: "Could not retrieve key from the SAM(InitSecureComm)";
- 0X0D17: "Hash code problem";
- 0X0D18: "Could not store the key into the SAM(InstallKey)";
- 0X0D19: "Frame is too large";
- 0X0D1A: "Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 0X0D1B: "The EEPROM may not be initialized because SecCommInterface does not make sense";



- 0X0D1C: "Problem encoding APDU";
- 0X0D20: "Unsupported Index(ILM) SAM Transceiver error - problem communicating with the SAM(Key Mgr)";
- 0X0D22: "Unexpected Sequence Counter in multiple frames for single bitmap(ILM) Length error in data returned from the SAM(Key Mgr)";
- 0X0D22: "Improper bit map(ILM)";
- 0X0D23: "Request Online Authorization";
- 0X0D24: "ViVOCard3 raw data read successful";
- 0X0D25: "Message index not available(ILM) ViVOcomm activate transaction card type(ViVOcomm)";
- 0X0D26: "Version Information Mismatch(ILM)";
- 0X0D27: "Not sending commands in correct index message index(ILM)";
- 0X0D28: "Time out or next expected message not received(ILM)";
- 0X0D29: "ILM languages not available for viewing(ILM)";
- 0X0D2A: "Other language not supported(ILM)";
- 0X0D41: "Unknown Error from SAM";
- 0X0D42: "Invalid data detected by SAM";
- 0X0D43: "Incomplete data detected by SAM";
- 0X0D44: "Reserved";
- 0X0D45: "Invalid key hash algorithm";
- 0X0D46: "Invalid key encryption algorithm";
- 0X0D47: "Invalid modulus length";
- 0X0D48: "Invalid exponent";
- 0X0D49: "Key already exists";
- 0X0D4A: "No space for new RID";
- 0X0D4B: "Key not found";
- 0X0D4C: "Crypto not responding";
- 0X0D4D: "Crypto communication error";
- 0X0D4E: "Module-specific error for Key Manager";
- 0X0D4F: "All key slots are full (maximum number of keys has been installed)";
- 0X0D50: "Auto-Switch OK";
- 0X0D51: "Auto-Switch failed";
- 0X0D90: "Account DUKPT Key not exist";
- 0X0D91: "Account DUKPT Key KSN exhausted";
- 0X0D00: "This Key had been loaded.";
- 0X0E00: "Base Time was loaded.";
- 0X0F00: "Encryption Or Decryption Failed.";
- 0X1000: "Battery Low Warning (It is High Priority Response while Battery is Low.)";

- 0X1800: "Send 'Cancel Command' after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount'; - 0X1900: "Press 'Cancel' key after send 'Get Encrypted PIN' & 'Get Numeric' & 'Get Amount';
- 0X30FF: "Security Chip is not connect";
- 0X3000: "Security Chip is deactivation & Device is In Removal Legally State.";
- 0X3101: "Security Chip is activation & Device is In Removal Legally State.";
- 0X5500: "No Admin DUKPT Key.";
- 0X5501: "Admin DUKPT Key STOP.";
- 0X5502: "Admin DUKPT Key KSN is Error.";
- 0X5503: "Get Authentication Code1 Failed.";
- 0X5504: "Validate Authentication Code Error.";
- 0X5505: "Encrypt or Decrypt data failed.";
- 0X5506: "Not Support the New Key Type.";
- 0X5507: "New Key Index is Error.";
- 0X5508: "Step Error.";
- 0X5509: "KSN Error";
- 0X550A: "MAC Error.";
- 0X550B: "Key Usage Error.";
- 0X550C: "Mode Of Use Error.";
- 0X550F: "Other Error.";
- 0X6000: "Save or Config Failed / Or Read Config Error.";
- 0X6200: "No Serial Number.";
- 0X6900: "Invalid Command - Protocol is right, but task ID is invalid.";
- 0X6A01: "Unsupported Command - Protocol and task ID are right, but command is invalid - In this State";
- 0X6A00: "Unsupported Command - Protocol and task ID are right, but command is invalid.";
- 0X6B00: "Unknown parameter in command - Protocol task ID and command are right, but parameter is invalid.";
- 0X6C00: "Unknown parameter in command - Protocol task ID and command are right, but length is out of the requirement.";
- 0X7200: "Device is suspend (MKSK suspend or press password suspend).";
- 0X7300: "PIN DUKPT is STOP (21 bit 1).";
- 0X7400: "Device is Busy.";
- 0XE100: "Can not enter sleep mode";
- 0XE200: "File has existed";
- 0XE300: "File has not existed";
- 0XE313: "IO line low -- Card error after session start";
- 0XE400: "Open File Error";
- 0XE500: "SmartCard Error";

- 0XE600: "Get MSR Card data is error";
- 0XE700: "Command time out";
- 0XE800: "File read or write is error";
- 0XE900: "Active 1850 error!";
- 0XEA00: "Load bootloader error";
- 0XEF00: "Protocol Error- STX or ETX or check error.";
- 0XEB00: "Picture is not exist";
- 0X2C02: "No Microprocessor ICC seated";
- 0X2C06: "no card seated to request ATR";
- 0X2D01: "Card Not Supported,";
- 0X2D03: "Card Not Supported, wants CRC";
- 0X690D: "Command not supported on reader without ICC support";
- 0X8100: "ICC error time out on power-up";
- 0X8200: "invalid TS character received - Wrong operation step";
- 0X8300: "Decode MSR Error";
- 0X8400: "TriMagII no Response";
- 0X8500: "No Swipe MSR Card";
- 0X8510: "No Financial Card";
- 0X8600: "Unsupported F, D, or combination of F and D";
- 0X8700: "protocol not supported EMV TD1 out of range";
- 0X8800: "power not at proper level";
- 0X8900: "ATR length too long";
- 0X8B01: "EMV invalid TA1 byte value";
- 0X8B02: "EMV TB1 required";
- 0X8B03: "EMV Unsupported TB1 only 00 allowed";
- 0X8B04: "EMV Card Error, invalid BWI or CWI";
- 0X8B06: "EMV TB2 not allowed in ATR";
- 0X8B07: "EMV TC2 out of range";
- 0X8B08: "EMV TC2 out of range";
- 0X8B09: "per EMV96 TA3 must be > - 0XF";
- 0X8B10: "ICC error on power-up";
- 0X8B11: "EMV T=1 then TB3 required";
- 0X8B12: "Card Error, invalid BWI or CWI";
- 0X8B13: "Card Error, invalid BWI or CWI";
- 0X8B17: "EMV TC1/TB3 conflict-";
- 0X8B20: "EMV TD2 out of range must be T=1";

- 0X8C00: "TCK error";
- 0XA304: "connector has no voltage setting";
- 0XA305: "ICC error on power-up invalid (SBLK(IFSD) exchange";
- 0XE301: "ICC error after session start";
- 0XFF00: "Request to go online";
- 0XFF01: "EMV: Accept the offline transaction";
- 0XFF02: "EMV: Decline the offline transaction";
- 0XFF03: "EMV: Accept the online transaction";
- 0XFF04: "EMV: Decline the online transaction";
- 0XFF05: "EMV: Application may fallback to magstripe technology";
- 0XFF06: "EMV: ICC detected tah the conditions of use are not satisfied";
- 0XFF07: "EMV: ICC didn't accept transaction";
- 0XFF08: "EMV: Transaction was cancelled";
- 0XFF09: "EMV: Application was not selected by kernel or ICC format error or ICC missing data error";
- 0XFF0A: "EMV: Transaction is terminated";
- 0XFF0B: "EMV: Other EMV Error";
- 0XFFFF: "NO RESPONSE";
- 0XF002: "ICC communication timeout";
- 0XF003: "ICC communication Error";
- 0XF00F: "ICC Card Seated and Highest Priority, disable MSR work request";
- 0XF200: "AID List / Application Data is not exist";
- 0XF201: "Terminal Data is not exist";
- 0XF202: "TLV format is error";
- 0XF203: "AID List is full";
- 0XF204: "Any CA Key is not exist";
- 0XF205: "CA Key RID is not exist";
- 0XF206: "CA Key Index it not exist";
- 0XF207: "CA Key is full";
- 0XF208: "CA Key Hash Value is Error";
- 0XF209: "Transaction format error";
- 0XF20A: "The command will not be processing";
- 0XF20B: "CRL is not exist";
- 0XF20C: "CRL number exceed max number";
- 0XF20D: "Amount,Other Amount,Trasaction Type are missing";
- 0XF20E: "The Identification of algorithm is mistake";
- 0XF20F: "No Financial Card";

- 0XF210: "In Encrypt Result state, TLV total Length is greater than Max Length";
- 0X1001: "INVALID ARG";
- 0X1002: "FILE\_OPEN\_FAILED";
- 0X1003: "FILE OPERATION\_FAILED";
- 0X2001: "MEMORY\_NOT\_ENOUGH";
- 0X3002: "SMARTCARD\_FAIL";
- 0X3003: "SMARTCARD\_INIT\_FAILED";
- 0X3004: "FALLBACK\_SITUATION";
- 0X3005: "SMARTCARD\_ABSENT";
- 0X3006: "SMARTCARD\_TIMEOUT";
- 0X3012: "EMV\_RESULT\_CODE\_MSR\_CARD\_ERROR\_FALLBACK";
- 0X5001: "EMV\_PARSING\_TAGS\_FAILED";
- 0X5002: "EMV\_DUPLICATE\_CARD\_DATA\_ELEMENT";
- 0X5003: "EMV\_DATA\_FORMAT\_INCORRECT";
- 0X5004: "EMV\_NO\_TERM\_APP";
- 0X5005: "EMV\_NO\_MATCHING\_APP";
- 0X5006: "EMV\_MISSING\_MANDATORY\_OBJECT";
- 0X5007: "EMV\_APP\_SELECTION\_RETRY";
- 0X5008: "EMV\_GET\_AMOUNT\_ERROR";
- 0X5009: "EMV\_CARD\_REJECTED";
- 0X5010: "EMV\_AIP\_NOT\_RECEIVED";
- 0X5011: "EMV\_AFL\_NOT\_RECEIVED";
- 0X5012: "EMV\_AFL\_LEN\_OUT\_OF\_RANGE";
- 0X5013: "EMV\_SFI\_OUT\_OF\_RANGE";
- 0X5014: "EMV\_AFL\_INCORRECT";
- 0X5015: "EMV\_EXP\_DATE\_INCORRECT";
- 0X5016: "EMV\_EFF\_DATE\_INCORRECT";
- 0X5017: "EMV\_ISS\_COD\_TBL\_OUT\_OF\_RANGE";
- 0X5018: "EMV\_CRYPTOGram\_TYPE\_INCORRECT";
- 0X5019: "EMV\_PSE\_NOT\_SUPPORTED\_BY\_CARD";
- 0X5020: "EMV\_USER\_SELECTED\_LANGUAGE";
- 0X5021: "EMV\_SERVICE\_NOT\_ALLOWED";
- 0X5022: "EMV\_NO\_TAG\_FOUND";
- 0X5023: "EMV\_CARD\_BLOCKED";
- 0X5024: "EMV\_LEN\_INCORRECT";
- 0X5025: "CARD\_COM\_ERROR";

- 0X5026: "EMV\_TSC\_NOT\_INCREASED";
- 0X5027: "EMV\_HASH\_INCORRECT";
- 0X5028: "EMV\_NO\_ARC";
- 0X5029: "EMV\_INVALID\_ARC";
- 0X5030: "EMV\_NO\_ONLINE\_COMM";
- 0X5031: "TRAN\_TYPE\_INCORRECT";
- 0X5032: "EMV\_APP\_NO\_SUPPORT";
- 0X5033: "EMV\_APP\_NOT\_SELECT";
- 0X5034: "EMV\_LANG\_NOT\_SELECT";
- 0X5035: "EMV\_NO\_TERM\_DATA";
- 0X5039: "EMV\_PIN\_ENTRY\_TIMEOUT";
- 0X6001: "CVM\_TYPE\_UNKNOWN";
- 0X6002: "CVM\_AIP\_NOT\_SUPPORTED";
- 0X6003: "CVM\_TAG\_8E\_MISSING";
- 0X6004: "CVM\_TAG\_8E\_FORMAT\_ERROR";
- 0X6005: "CVM\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6006: "CVM\_COND\_CODE\_IS\_NOT\_SUPPORTED";
- 0X6007: "NO\_MORE\_CVM";
- 0X6008: "PIN\_BYPASSED\_BEFORE";
- 0X7001: "PK\_BUFFER\_SIZE\_TOO\_BIG";
- 0X7002: "PK\_FILE\_WRITE\_ERROR";
- 0X7003: "PK\_HASH\_ERROR";
- 0X8001: "NO\_CARD\_HOLDER\_CONFIRMATION";
- 0X8002: "GET\_ONLINE\_PIN";
- 0XD000: "Data not exist";
- 0XD001: "Data access error";
- 0XD100: "RID not exist";
- 0XD101: "RID existed";
- 0XD102: "Index not exist";
- 0XD200: "Maximum exceeded";
- 0XD201: "Hash error";
- 0XD205: "System Busy";
- 0X0E01: "Unable to go online";
- 0X0E02: "Technical Issue";
- 0X0E03: "Declined";
- 0X0E04: "Issuer Referral transaction";

- 0X0F01: "Decline the online transaction";
- 0X0F02: "Request to go online";
- 0X0F03: "Transaction is terminated";
- 0X0F05: "Application was not selected by kernel or ICC format error or ICC missing data error";
- 0X0F07: "ICC didn't accept transaction";
- 0X0F0A: "Application may fallback to magstripe technology";
- 0X0F0C: "Transaction was cancelled";
- 0X0F0D: "Timeout";
- 0X0F0F: "Other EMV Error";
- 0X0F10: "Accept the offline transaction";
- 0X0F11: "Decline the offline transaction";
- 0X0F21: "ICC detected tah the conditions of use are not satisfied";
- 0X0F22: "No app were found on card matching terminal configuration";
- 0X0F23: "Terminal file does not exist";
- 0X0F24: "CAPK file does not exist";
- 0X0F25: "CRL Entry does not exist";
- 0X0FFE: "code when blocking is disabled";
- 0X0FFF: "code when command is not applicable on the selected device";
- 0XF005: "ICC Encrypted C-APDU Data Structure Length Error Or Format Error.";
- 0XBBE0: "CM100 Success";
- 0XBBE1: "CM100 Parameter Error";
- 0XBBE2: "CM100 Low Output Buffer";
- 0XBBE3: "CM100 Card Not Found";
- 0XBBE4: "CM100 Collision Card Exists";
- 0XBBE5: "CM100 Too Many Cards Exist";
- 0XBBE6: "CM100 Saved Data Does Not Exist";
- 0XBBE8: "CM100 No Data Available";
- 0XBBE9: "CM100 Invalid CID Returned";
- 0XBBEA: "CM100 Invalid Card Exists";
- 0BBEC: "CM100 Command Unsupported";
- 0BBED: "CM100 Error In Command Process";
- 0BBEE: "CM100 Invalid Command";
- 0X9031: "Unknown command";
- 0X9032: "Wrong parameter (such as the length of the command is incorrect)";
- 0X9038: "Wait (the command couldnt be finished in BWT)";
- 0X9039: "Busy (a previously command has not been finished)";

- 0X903A: "Number of retries over limit";
- 0X9040: "Invalid Manufacturing system data";
- 0X9041: "Not authenticated";
- 0X9042: "Invalid Master DUKPT Key";
- 0X9043: "Invalid MAC Key";
- 0X9044: "Reserved for future use";
- 0X9045: "Reserved for future use";
- 0X9046: "Invalid DATA DUKPT Key";
- 0X9047: "Invalid PIN Pairing DUKPT Key";
- 0X9048: "Invalid DATA Pairing DUKPT Key";
- 0X9049: "No nonce generated";
- 0X9949: "No GUID available. Perform getVersion first.";
- 0X9950: "MAC Calculation unsuccessful. Check BDK value.";
- 0X904A: "Not ready";
- 0X904B: "Not MAC data";
- 0X9050: "Invalid Certificate";
- 0X9051: "Duplicate key detected";
- 0X9052: "AT checks failed";
- 0X9053: "TR34 checks failed";
- 0X9054: "TR31 checks failed";
- 0X9055: "MAC checks failed";
- 0X9056: "Firmware download failed";
- 0X9060: "Log is full";
- 0X9061: "Removal sensor unengaged";
- 0X9062: "Any hardware problems";
- 0X9070: "ICC communication timeout";
- 0X9071: "ICC data error (such check sum error)";
- 0X9072: "Smart Card not powered up";

#### 12.8.4.10 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

#### Returns

SDK wait time in seconds



#### 12.8.4.11 int device\_getSpectrumProKSN ( IN int *type*, OUT BYTE \* *KSN* )

DEPRECATED : please use device\_getSpectrumProKSN\_Len(IN int *type*, OUT BYTE \* *KSN*, IN\_OUT int \**KSN*-Len)

Get DUKPT KSN

Returns the KSN for the provided key index

##### Parameters

<i>type</i>	Key type: <ul style="list-style-type: none"><li>• 0: Key Encryption Key (Master Key or KEK)</li><li>• 2: Data Encryption Key (DEK)</li><li>• 5: MAC Key (MAK)</li><li>• 10: RKL Key Encryption Key (REK)</li><li>• 20: HSM DUKPT Key</li></ul>
<i>KSN</i>	Key Serial Number; needs to have at least 10 bytes of memory

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.8.4.12 int device\_getSpectrumProKSN\_Len ( IN int *type*, OUT BYTE \* *KSN*, IN\_OUT int \* *KSN*Len )

Get DUKPT KSN

Returns the KSN for the provided key index

##### Parameters

<i>type</i>	Key type: <ul style="list-style-type: none"><li>• 0: Key Encryption Key (Master Key or KEK)</li><li>• 2: Data Encryption Key (DEK)</li><li>• 5: MAC Key (MAK)</li><li>• 10: RKL Key Encryption Key (REK)</li><li>• 20: HSM DUKPT Key</li></ul>
-------------	--

<i>KSN</i>	Key Serial Number
<i>KSNLen</i>	Length of KSN

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.13 int device\_getThreadStackSize ( )**

Get Thread Stack Size

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.8.4.14 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.15 int device\_isAttached ( int *deviceType* )**

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

**Parameters**

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

**Returns**

1 if the device is attached, or 0 if the device is not attached

**12.8.4.16 int device\_isConnected ( )**

Check the device connected status

**Returns**

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

**12.8.4.17 int device\_pollCardReader ( OUT BYTE \* *status* )**

DEPRECATED : please use [device\\_pollCardReader\\_Len\(OUT BYTE \\* status, IN\\_OUT int \\*statusLen\)](#)

Poll Card Reader

Provides information about the state of the Card Reader

## Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> <li>• Bit 0: Device Manufacturing CA data valid</li> <li>• Bit 1: Device Manufacturing Secure data valid</li> <li>• Bit 2: HOST_CR_MASTER_DUKPT Key valid</li> <li>• Bit 3: HOST_CR_MAC Keys valid (Authenticated)</li> <li>• Bit 4: RFU</li> <li>• Bit 5: RFU</li> <li>• Bit 6: DATA_DUKPT Key Valid</li> <li>• Bit 7: Key is initialized (MFK and RSA Key pairs)</li> </ul>
---------------	--

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU
- Bit 2: CR\_PINPAD\_MASTER\_DUKPT Key valid
- Bit 3: CR\_PINPAD\_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present

- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.18** `int device_pollCardReader_Len ( OUT BYTE * status, IN_OUT int * statusLen )`

#### Poll Card Reader

Provides information about the state of the Card Reader

#### Parameters

<i>status</i>	<p>Six bytes indicating card reader information Byte 0:</p> <ul style="list-style-type: none"> <li>• Bit 0: Device Manufacturing CA data valid</li> <li>• Bit 1: Device Manufacturing Secure data valid</li> <li>• Bit 2: HOST_CR_MASTER_DUKPT Key valid</li> <li>• Bit 3: HOST_CR_MAC Keys valid (Authenticated)</li> <li>• Bit 4: RFU</li> <li>• Bit 5: RFU</li> <li>• Bit 6: DATA_DUKPT Key Valid</li> <li>• Bit 7: Key is initialized (MFK and RSA Key pairs)</li> </ul>
---------------	--

Byte 1:

- Bit 0: Firmware Key Valid
- Bit 1: RFU

- Bit 2: CR\_PINPAD\_MASTER\_DUKPT Key valid
- Bit 3: CR\_PINPAD\_MAC Keys valid (Authenticated)
- Bit 4: DATA Pairing DUKPT Key valid
- Bit 5: PIN Pairing DUKPT Key Valid
- Bit 6: RFU
- Bit 7: RFU

Byte 2:

- Bit 0: RFU
- Bit 1: Tamper Switch #1 Error
- Bit 2: Battery Backup Error
- Bit 3: Temperature Error
- Bit 4: Voltage Sensor Error
- Bit 5: Firmware Authentication Error
- Bit 6: Tamper Switch #2 Error
- Bit 7: Removal Tamper Error

Byte 3:

- Battery Voltage (example 0x32 = 3.2V, 0x24 = 2.4V)

Byte 4:

- Bit 0: Log is Full
- Bit 1: Mag Data Present
- Bit 2: Card Insert
- Bit 3: Removal Sensor connected
- Bit 4: Card Seated
- Bit 5: Latch Mechanism Active
- Bit 6: Removal Sensor Active
- Bit 7: Tamper Detector Active

Byte 5:

- Bit 0: SAM Available
- Bit 1: Chip Card Reader Available
- Bit 2: Host Connected
- Bit 3: Contactless Available
- Bit 4: PINPAD connected
- Bit 5: MSR Header connected
- Bit 6: RFU
- Bit 7: Production Unit

## Parameters

<i>statusLen</i>	Length of status
------------------	------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.8.4.19 int device\_rebootDevice ( )

Reboot Device Executes a command to restart the device.

- Card data is cleared, resetting card status bits.
- Response data of the previous command is cleared.
- Resetting firmware.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.8.4.20 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.8.4.21 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.8.4.22 int device\_SendDataCommand ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.8.4.23 int device\_setCurrentDevice ( int deviceType )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VEND1,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

## 12.8.4.24 void device\_setSDKWaitTime ( int waitTime )

## Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

## 12.8.4.25 void device\_setThreadStackSize ( int threadSize )

## Set Thread Stack Size

Set the stack size setting for newly created threads

## 12.8.4.26 int device\_updateFirmware ( IN BYTE \* firmwareData, IN int firmwareDataLen, IN char \* firmwareName, IN int encryptionType, IN BYTE \* keyBlob, IN int keyBlobLen )

Update Firmware Updates the firmware of the Spectrum Pro K21 HUB or Maxq1050.

## Parameters

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of firmwareData
<i>firmwareName</i>	Firmware name. Must be one of the following two strings (with appropriate version information) <ul style="list-style-type: none"> <li>• "SP K21 APP Vx.xx.xxx"</li> <li>• "SP MAX APP Vx.xx.xxx"</li> </ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"> <li>• 0 : Plaintext</li> <li>• 1 : TDES ECB, PKCS#5 padding</li> <li>• 2 : TDES CBC, PKCS#5, IV is all 0</li> </ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of keyBlob

## Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

Firmware update status is returned in the callback with the following values: sender = SPECTRUM\_PRO state = DeviceState.FirmwareUpdate data = File Progress. Two bytes, with `byte[0]` = current block, and `byte[1]` = total blocks. 0x0310 = block 3 of 16 transactionResultCode:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

12.8.4.27 `int emv_activateTransaction ( IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F37

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable



12.8.4.28 void emv\_allowFallback ( IN int *allow* )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.8.4.29 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.8.4.30 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.8.4.31 `int emv_callbackResponseLCD ( IN int type, byte selection )`

**Callback Response LCD Display**

Provides menu selection responses to the kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_TYPE.EMV_CALLBACK_TYPE_LCD`, and `lcd_displayMode = EMV_LCD_DISPLAY_MODE_MENU`, `EMV_LCD_DISPLAY_MODE_PROMPT`, or `EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT`

**Parameters**

<i>type</i>	If Cancel key pressed during menu selection, then value is <code>EMV_LCD_DISPLAY_MODE_CANCEL</code> . Otherwise, value can be <code>EMV_LCD_DISPLAY_MODE_MENU</code> , <code>EMV_LCD_DISPLAY_MODE_PROMPT</code> , or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code>
<i>selection</i>	If <code>type = EMV_LCD_DISPLAY_MODE_MENU</code> or <code>EMV_LCD_DISPLAY_MODE_LANGUAGE_SELECT</code> , provide the selection ID line number. Otherwise, if <code>type = EMV_LCD_DISPLAY_MODE_PROMPT</code> supply either 0x43 ('C') for Cancel, or 0x45 ('E') for Enter/accept

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

#### 12.8.4.32 `int emv_callbackResponseMSR ( IN BYTE * MSR, IN_OUT int MSRLen )`

**Callback Response MSR Entry**

Provides MSR information to kernel after a callback was received with `DeviceState.EMVCallback`, and `callbackType = EMV_CALLBACK_MSR`

**Parameters**

<i>MSR</i>	Swiped track data
<i>MSRLen</i>	the length of Swiped track data

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

#### 12.8.4.33 `int emv_cancelTransaction ( )`

**Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

#### 12.8.4.34 `int emv_completeTransaction ( IN int commError, IN BYTE * authCode, IN int authCodeLen, IN BYTE * iad, IN int iadLen, IN BYTE * tlvScripts, IN int tlvScriptsLen, IN BYTE * tlv, IN int tlvLen )`

**Complete EMV Transaction Request**

Completes the EMV transaction for an ICC card when online authorization request is received from emv\_authenticateTransaction

The tags will be returned in the callback routine.

## Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.8.4.35 int emv\_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

## Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

## 12.8.4.36 int emv\_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

## Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

## 12.8.4.37 int emv\_getEMVConfigurationCheckValue ( OUT BYTE \* checkValue, IN\_OUT int \* checkValueLen )

Get EMV Kernel configuration check value info

## Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

## 12.8.4.38 int emv\_getEMVKernelCheckValue ( OUT BYTE \* checkValue, IN\_OUT int \* checkValueLen )

Get EMV Kernel check value info

## Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

12.8.4.39 `int emv_getEMVKernelVersion ( OUT char * version )`

DEPRECATED : please use [emv\\_getEMVKernelVersion\\_Len](#)(OUT char\* *version*, IN\_OUT int \**versionLen*)

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.40 `int emv_getEMVKernelVersion_Len ( OUT char * version, IN_OUT int * versionLen )`

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.41 `void emv_registerCallBk ( pEMV_callBack pEMVf )`

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.8.4.42 `int emv_removeAllApplicationData ( )`

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.43 `int emv_removeAllCAPK ( )`

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.44 int emv\_removeAllCRL ( )**

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.45 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID

Removes the Application Data as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.46 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )**

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.47 int emv\_removeCRL ( IN BYTE \* list, IN int lsLen )**

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

**Parameters**

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.48 int emv\_removeTerminalData ( )**

Remove Terminal Data

Removes the Terminal Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.49 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )**

Retrieve AID list

Returns all the AID names installed on the terminal.

**Parameters**

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.50 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )**

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

**Parameters**

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.51** `int emv_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

**Retrieve Certificate Authority Public Key**

Retrieves the CAPK as specified by the RID/Index passed as a parameter.

**Parameters**

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.52** `int emv_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

**Retrieve the Certificate Authority Public Key list**

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)



12.8.4.53 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *lssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.54 `int emv_retrieveTerminalData ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

## Retrieve Terminal Data

Retrieves the Terminal Data.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.55 `int emv_retrieveTerminalID ( OUT char * terminalID )`

DEPRECATED : please use [emv\\_retrieveTerminalID\\_Len\(OUT char\\* terminalID, IN\\_OUT int \\*terminalIDLen\)](#)

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string; needs to have at least 30 bytes of memory
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.56 `int emv_retrieveTerminalID_Len ( OUT char * terminalID, IN_OUT int * terminalIDLen )`

Gets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID string
<i>terminalIDLen</i>	Length of terminalID

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.57 `int emv_retrieveTransactionResult ( IN BYTE * tags, IN int tagsLen, IDTTransactionData * cardData )`

## Retrieve Transaction Results

Retrieves specified EMV tags from the currently executing transaction.

## Parameters

<i>tags</i>	Tags to be retrieved. Example 0x9F028A will retrieve tags 9F02 and 8A
<i>tagsLen</i>	Length of tag list
<i>cardData</i>	All requested tags returned as unencrypted, encrypted and masked TLV data in IDT-TransactionData object

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.58** int emv\_setApplicationData ( IN BYTE \* *name*, IN int *nameLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.59** void emv\_setAutoAuthenticateTransaction ( IN int *authenticate* )

Enables authenticate for EMV transactions. If a emv\_startTransaction results in code 0x0010 (start transaction success), then emv\_authenticateTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

**12.8.4.60** void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

**12.8.4.61** int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.8.4.62 int emv\_setCRL ( IN BYTE \* *list*, IN int *lsLen* )

Set Certificate Revocation List

Sets the CRL

## Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.8.4.63 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Terminal Data

Sets the Terminal Data as specified by the TerminalData structure passed as a parameter

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Return codes listed as typedef enum in IDTCommon:RETURN_CODE. Values can be parsed with <code>device_getResponseCodeString()</code>
--------------------	---

12.8.4.64 int `emv_setTerminalID ( IN char * terminalID )`

Sets the terminal ID as printable characters .

## Parameters

<i>terminalID</i>	Terminal ID to set
-------------------	--------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.65 int `emv_startTransaction ( IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.8.4.66 int `icc_getICCReaderStatus ( OUT BYTE * status )`

Get Reader Status Returns the reader status

**Parameters**

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.8.4.67 int icc\_powerOffICC ( )**

Power Off ICC

Powers down the ICC

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

If Success, empty If Failure, ASCII encoded data of error string

**12.8.4.68 int icc\_powerOnICC ( OUT BYTE \* ATR, IN\_OUT int \* inLen )**

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

**Parameters**

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

**12.8.4.69 int msr\_cancelMSRSwipe ( )**

Disable MSR Swipe Cancels MSR swipe request.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

**12.8.4.70 int msr\_clearMSRData ( )**

Clear MSR Data Clears the MSR Data buffer

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

**12.8.4.71 int msr\_getMSRData ( OUT BYTE \* reData, IN\_OUT int \* reLen )**

Get MSR Data Reads the MSR Data buffer

## Parameters

<i>reData</i>	Card data
<i>reLen</i>	Card data length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.72 void msr\_registerCallBk ( pMSR\_callback pMSRf )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.8.4.73 void msr\_registerCallBkp ( pMSR\_callbackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.8.4.74 int msr\_startMSRSwipe ( IN int \_timeout )

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.8.4.75 void parseMSRData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTMSRData \* cardData )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

12.8.4.76 void parsePINBlockData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTPINData \* cardData )

Parse the PIN block data from the buffer into IDTPINData structure

## Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

12.8.4.77 void parsePINData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTPINData \* cardData )

Parse the PIN data from the buffer into IDTPINData structure

## Parameters

<i>resData</i>	PIN card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTPINData structure

## 12.8.4.78 int pin\_cancelPINEntry ( )

Cancel PIN Entry

Cancels PIN entry request

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.8.4.79 int pin\_getPIN ( IN int mode, IN int PANSource, IN char \* iccPAN, IN int IN iccPANLen, int startTimeout, IN int entryTimeout, IN char \* language, IN int languageLen )

Get Encrypted PIN

Requests PIN Entry

## Parameters

<i>mode</i>	<ul style="list-style-type: none"> <li>• 0x00- Cancel: Cancels PIN entry = also can execute <a href="#">pin_cancelPINEntry()</a>. All other parameters for this method will be ignored</li> <li>• 0x01- Online PIN DUKPT</li> <li>• 0x02- Online PIN MKSK</li> <li>• 0x03- Offline PIN (No need to define PAN Source or ICC PAN)</li> </ul>
<i>PANSource</i>	<ul style="list-style-type: none"> <li>• 0x00- ICC: PAN Captured from ICC and must be provided in iccPAN parameter</li> <li>• 0x01- MSR: PAN Captured from MSR swipe and will be inserted by Spectrum Pro. No need to provide iccPAN parameter.</li> </ul>
<i>iccPAN</i>	PAN captured from ICC. When PAN captured from MSR, this parameter will be ignored
<i>iccPANLen</i>	the length of iccPAN
<i>startTimeout</i>	The amount of time allowed to start PIN entry before timeout
<i>entryTimeout</i>	The amount of time to enter the PIN after first digit selected before timeout
<i>language</i>	Valid values "EN" for English, "ES" for Spanish, "ZH" for Chinese, "FR" for French
<i>languageLen</i>	the length of language

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.8.4.80 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)



**12.8.4.81 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )**

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

**12.8.4.82 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )**

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

**12.8.4.83 int rs232\_device\_init ( int deviceType, int port\_number, int brate )**

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

**Parameters**

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

**Port nr. | Linux | Windows**

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

**Parameters**

<i>brate</i>	Bitrate of the device
--------------	-----------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.8.4.84 char\* SDK\_Version ( )**

To Get SDK version

**Returns**

return the SDK version string

**12.8.4.85 int setAbsoluteLibraryPath ( const char \* absoluteLibraryPath )**

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.9 Source\_C/libIDT\_SREDKey2.h File Reference

SREDKey2 API.

```
#include "IDTDef.h"
```

## Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

## Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pFW_callback )(int, int, int, int, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* pLCD_callback )(int, IDTLCDItem *)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

## Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, pReadDataLog pFRead)
- void [device\\_registerFWCallBk](#) (pFW\_callback pFWf)
- void [device\\_registerCameraCallBk](#) (pCMR\_callback pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callback pCSFSf)
- void [emv\\_registerCallBk](#) (pEMV\_callback pEMVf)
- void [msr\\_registerCallBk](#) (pMSR\_callback pMSRf)
- void [msr\\_registerCallBkp](#) (pMSR\_callbackp pMSRf)
- void [ctls\\_registerCallBk](#) (pMSR\_callback pCTLSf)
- void [ctls\\_registerCallBkp](#) (pMSR\_callbackp pCTLSf)
- void [pin\\_registerCallBk](#) (pPIN\_callback pPINf)
- void [lcd\\_registerCallBk](#) (pLCD\_callback pLCDf)

- void `comm_registerHTTPCallback` (`httpComm_callback` cBack)
- void `comm_registerV4Callback` (`v4Comm_callback` cBack)
- char \* `SDK_Version` ()
- int `setAbsoluteLibraryPath` (const char \*absoluteLibraryPath)
- int `device_setConfigPath` (const char \*path)
- int `device_setNEO2DevicesConfigs` (IN const char \*configs, IN int len)
- int `device_init` ()
- int `rs232_device_init` (int deviceType, int port\_number, int brate)
- int `device_setCurrentDevice` (int deviceType)
- int `device_isAttached` (int deviceType)
- int `device_close` ()
- void `device_getIDGStatusCodeString` (IN int returnCode, OUT char \*despcriton)
- int `device_isConnected` ()
- int `device_getFirmwareVersion` (OUT char \*firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int `device_pingDevice` ()
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommandNEO` (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int `device_SendDataCommand` (IN BYTE \*cmd, IN int cmdLen, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int `device_rebootDevice` ()
- int `device_SendDataCommandITP` (IN BYTE \*cmd, IN int cmdLen, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- void `device_setTransactionExponent` (int exponent)
- int `device_getKeyStatus` (int \*newFormat, BYTE \*status, int \*statusLen)
- int `device_updateFirmware` (IN BYTE \*firmwareData, IN int firmwareDataLen, IN char \*firmwareName, IN int encryptionType, IN BYTE \*keyBlob, IN int keyBlobLen)
- int `config_getModelNumber` (OUT char \*sNumber)
- int `config_getModelNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int `config_getSerialNumber` (OUT char \*sNumber)
- int `config_getSerialNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int `device_setSystemLanguage` (char \*language)
- int `msr_setExpirationMask` (int mask)
- int `msr_getExpirationMask` (BYTE \*value)
- int `msr_setClearPANID` (BYTE val)
- int `msr_getClearPANID` (BYTE \*value)
- int `msr_setSwipeForcedEncryptionOption` (int track1, int track2, int track3, int track3card0)
- int `msr_getSwipeForcedEncryptionOption` (BYTE \*option)
- int `msr_setSwipeMaskOption` (int track1, int track2, int track3)
- int `msr_getSwipeMaskOption` (BYTE \*option)
- int `msr_getFunctionStatus` (int \*enable, int \*isBufferMode, int \*withNotification)
- int `msr_disable` ()

### 12.9.1 Detailed Description

SREDKey2 API. SREDKey2 Global API methods.

### 12.9.2 Macro Definition Documentation

#### 12.9.2.1 #define IN

INPUT parameter.

### 12.9.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

### 12.9.2.3 #define OUT

OUTPUT parameter.

## 12.9.3 Typedef Documentation

### 12.9.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

### 12.9.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

### 12.9.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

### 12.9.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

### 12.9.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk

### 12.9.3.6 typedef void(\* pFW\_callback)(int, int, int, int, int)

Define the firmware update callback function to get the firmware update status

It should be registered using the device\_registerFWCallBk

### 12.9.3.7 typedef void(\* pLCD\_callback)(int, IDTLCDItem \*)

Define the LCD callback function to get the input LCDItem

It should be registered using the lcd\_registerCallBk,

**12.9.3.8 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.9.3.9 typedef void(\* pMSR\_callBack)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.9.3.10 typedef void(\* pMSR\_callBackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callBack

**12.9.3.11 typedef void(\* pPIN\_callBack)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.9.3.12 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk

**12.9.3.13 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk

**12.9.3.14 typedef void(\* v4Comm\_callBack)(BYTE, BYTE, BYTE \*, int)**

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

**12.9.4 Function Documentation****12.9.4.1 void comm\_registerHTTPCallback ( httpComm\_callBack cBack )**

Register Comm HTTP Async Callback

**Parameters**

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

#### 12.9.4.2 void comm\_registerV4Callback ( v4Comm\_callBack *cBack* )

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

#### 12.9.4.3 int config\_getModelNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getModelNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number; needs to have at least 64 bytes of memory
----------------	---

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.9.4.4 int config\_getModelNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Model Number

Parameters

<i>sNumber</i>	Returns Model Number
<i>sNumber</i>	length of Model Number

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.9.4.5 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.9.4.6 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.9.4.7 void `ctls_registerCallBk ( pMSR_callBack pCTLSf )`

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.9.4.8 void `ctls_registerCallBkp ( pMSR_callBackp pCTLSf )`

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.9.4.9 int `device_close ( )`

Close the device

## Returns

RETURN\_CODE: 0: success, 0x0A: failed

12.9.4.10 int `device_getCurrentDeviceType ( )`

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.9.4.11 int `device_getFirmwareVersion ( OUT char * firmwareVersion )`

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* firmwareVersion, IN\_OUT int \*firmwareVersionLen)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.9.4.12 int `device_getFirmwareVersion_Len ( OUT char * firmwareVersion, IN_OUT int * firmwareVersionLen )`

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.9.4.13 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";



- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.9.4.14 int device\_getKeyStatus ( int \* newFormat, BYTE \* status, int \* statusLen )

##### Get Key Status

Gets the status of loaded keys

##### Parameters

<i>status</i>	newFormat for Augusta and miniSmartII only 1: new format of key status 0: reserved format for support previous device
<i>status</i>	For L100, Augusta and miniSmartII: When the newFormat is 0, data format as follows. For Augusta and miniSmartII: byte 0: PIN DUKPT Key, Does not support, always 0 byte 1: PIN Master Key, Does not support, always 0 byte 2: PIN Session Key, Does not support, always 0 byte 3: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 4: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 5: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP For L100: byte 0: PIN DUKPT Key byte 1: PIN Master Key byte 2: Standard PIN Session Key byte 3: Desjardins PIN Session Key byte 4: Account/MSR DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 5: Account/ICC DUKPT Key, 1 Exists, 0 None, 0xFF STOP, Does not support, always 0 byte 6: Admin DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 7: Data DUKPT Key, 1 Exists, 0 None, 0xFF STOP byte 8: MAC DUKPT Key, 1 Exists, 0 None, 0xFF STOP

when the newFormat is 1, data format as follows. [Block Length] [KeyStatusBlock1] [KeyStatusBlock2]...[KeyStatusBlockN] Where: [Block Length] is 2 bytes, format is Len\_L Len\_H, is KeyStatusBlock Number [KeyStatusBlockX> is 4 bytes, format is [Key Index and Key Name] [key slot] [key status]: [Key Index and Key Name] is 1 byte. Please refer to following table 0x14 LCL-KEK to Encrypt Other Keys 0x02 Data encryption Key to Encrypt ICC/MSR 0x05 MAC DUKPT Key for Host-Device - MAC Verification 0x05 MTK DUKPT Key for TTK Self-Test 0x0C RKI-KEK for Remote Key Injection [key slot] is 2 bytes. Range is 0 - 9999 the MTK DUKPT Key slot is 16, the others are all 0 [key status] is 1 byte. 0 - Not Exist 1 - Exist 0xFF - (Stop. Only Valid for DUKPT Key) For NEO2 and SREDKey2: Each unit of three bytes represents one key's parameters (index and slot). Key Name Index (1 byte): 0x14 - LCL-KEK 0x01 - Pin encryption Key (NEO2 only) 0x02 - Data encryption Key 0x05 - MAC DUKPT Key 0x0A - PCI Pairing Key (NEO2 only) Key Slot (2 bytes): Indicate different slots of a certain Key Name Example: slot =5 (0x00 0x05), slot=300 (0x01 0x2C) For BTPay380, slot is always 0 For example, 0x14 0x00 0x00 0x02 0x00 0x00 0x0A 0x00 0x00 will represent [KeyNameIndex=0x14,KeySlot=0x0000], [KeyNameIndex=0x02,KeySlot=0x0000] and [KeyNameIndex=0x0A,KeySlot=0x0000]

**Parameters**

<i>statusLen</i>	the length of status
------------------	----------------------

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.9.4.15 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by `device_isConnected()`.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

**12.9.4.16 int device\_isAttached ( int *deviceType* )**

Check if the device is attached to the USB port The function `device_init()` must be called before this function.

**Parameters**

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

**Returns**

1 if the device is attached, or 0 if the device is not attached

**12.9.4.17 int device\_isConnected ( )**

Check the device connected status

**Returns**

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

**12.9.4.18 int device\_pingDevice ( )**

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

**12.9.4.19 int device\_rebootDevice ( )**

Reboot Device - NGA Executes a command to restart the device.

- Card data is cleared, resetting card status bits.

- Response data of the previous command is cleared.
- Resetting firmware.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.9.4.20 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

#### 12.9.4.21 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

#### 12.9.4.22 void device\_registerFWCallBk ( pFW\_callBack pFWf )

To register the firmware update callback function to get the firmware update processing response. (Pass NULL to disable the callback.)

#### 12.9.4.23 int device\_SendDataCommand ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to NGA device

Sends a command to the device .

#### Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.9.4.24 int device\_SendDataCommandITP ( IN BYTE \* cmd, IN int cmdLen, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to ITP device

Sends a command to the device .

#### Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of ITP command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.9.4.25 `int device_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE * data, IN int dataLen, OUT BYTE * response, IN_OUT int * respLen )`

Send a Command to NEO device

Sends a command to the NEO device .

## Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.9.4.26 `int device_setConfigPath ( const char * path )`

Set the path to the config xml file(s) if any

## Parameters

<i>path</i>	The path to the config xml files (such as "NEO2_Devices.xml" which contains the information of NEO2 devices). Only need to specify the path to the folder which contains the config files. File names are not needed. The maximum length of path is 200 characters including the '\0' at the end.
-------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.9.4.27 int device\_setCurrentDevice ( int *deviceType* )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to
	<pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 }; </pre>

## Returns

RETURN\_CODE: 1: success, 0: failed

12.9.4.28 int device\_setNEO2DevicesConfigs ( IN const char \* *configs*, IN int *len* )

Pass the content of the config xml file ("NEO2\_Devices.xml") as a string to the SDK instead of reading the config xml file by the SDK It needs to be called before [device\\_init\(\)](#), otherwise the SDK will try to read the config xml file.

## Parameters

<i>configs</i>	The content read from the config xml file ("NEO2_Devices.xml" which contains the information of NEO2 devices).
<i>len</i>	The length of the string configs. The maximum length is 5000 bytes.

12.9.4.29 int device\_setSystemLanguage ( char \* *language* )

Set System Language Sets the language for the message displayed in the LCD screen

**Parameters**

<i>language</i>	2-byte ASCII code, can be "EN" or "JP"
-----------------	--

**Returns**

success or error code. Values can be parsed with `device_getIDGStatusCodeString`

**See Also**

`ErrorCode`

**12.9.4.30 void device\_setTransactionExponent ( int *exponent* )**

Sets the transaction exponent to be used with `device_startTransaction`. Default value is 2

**Parameters**

<i>exponent, The</i>	exponent to use when calling <code>device_startTransaction</code>
----------------------	---

**12.9.4.31 int device\_updateFirmware ( IN BYTE \* *firmwareData*, IN int *firmwareDataLen*, IN char \* *firmwareName*, IN int *encryptionType*, IN BYTE \* *keyBlob*, IN int *keyBlobLen* )**

Update Firmware Updates the firmware of NEO 2 devices.

**Parameters**

<i>firmwareData</i>	Signed binary data of a firmware file provided by IDTech
<i>firmwareDataLen</i>	Length of <i>firmwareData</i>
<i>firmwareName</i>	Firmware name. <ul style="list-style-type: none"> <li>• For example "VP5300_v1.00.023.0167.S_Test.fm"</li> </ul>
<i>encryptionType</i>	Encryption type <ul style="list-style-type: none"> <li>• 0 : Plaintext</li> <li>• 1 : TDES ECB, PKCS#5 padding</li> <li>• 2 : TDES CBC, PKCS#5, IV is all 0</li> </ul>
<i>keyBlob</i>	Encrypted firmware session key blob, TR-31 Rev B, wrapped by FW Key (Optional, none if firmware is plaintext)
<i>keyBlobLen</i>	Length of <i>keyBlob</i>

**Returns**

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

Firmware update status is returned in the callback with the following values: sender = device type state = DEVICE-\_FIRMWARE\_UPDATE current block total blocks ResultCode:

- RETURN\_CODE\_DO\_SUCCESS = Firmware Update Completed Successfully
- RETURN\_CODE\_BLOCK\_TRANSFER\_SUCCESS = Current block transferred successfully
- Any other return code represents an error condition

**12.9.4.32 void emv\_registerCallBk ( pEMV\_callBack pEMVf )**

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

**12.9.4.33 void lcd\_registerCallBk ( pLCD\_callBack pLCDf )**

To register the lcd callback function to get the LCDItem. (Pass NULL to disable the callback.)

**12.9.4.34 int msr\_disable ( )**

Disable MSR Disable MSR functions.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.9.4.35 int msr\_getClearPANID ( BYTE \* value )**

Get Clear PAN ID.

Returns the number of digits that begin the PAN that will be in the clear

**Parameters**

<i>value</i>	4901 <Setting value>="">. setting Value: Number of digits in clear. Values are char '0' - '6'
--------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

**12.9.4.36 int msr\_getExpirationMask ( BYTE \* value )**

Get MSR expiration date mask.

**Parameters**

<i>value</i>	5001 <Setting value>="">. setting Value: '0' = masked, '1' = not-masked
--------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

**12.9.4.37 int msr\_getFunctionStatus ( int \* enable, int \* isBufferMode, int \* withNotification )**

Get MSR Function Status.

Gets the MSR function status

**Parameters**

<i>enable</i>	1 = MSR enabled, 0 = MSR disabled
<i>isBufferMode</i>	1 = buffer mode, 0 = auto mode
<i>withNotification</i>	1 = with notification, 0 = without notification

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

12.9.4.38 int msr\_getSwipeForcedEncryptionOption ( BYTE \* *option* )

Get MSR Swipe Forced Encryption Option.



## Parameters

<i>option</i>	8401 <Setting value>="". Setting Value Byte using lower four bits as flags. 0 = Force Encryption Off, 1 = Force Encryption On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 bit4 = Track 3 Card Option 0
---------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.9.4.39 int msr\_getSwipeMaskOption ( BYTE \* *option* )

Get MSR Swipe Mask Option.

Gets the swipe mask/clear data sending option

## Parameters

<i>option</i>	8601 <Setting value>="". Setting Value Byte using lower three bits as flags. 0 = Mask Option Off, 1 = Mask Option On bit0 = Track 1 bit1 = Track 2 bit2 = Track 3 Example: Response 0x03 = Track1/Track2 Masked Option ON, Track3 Masked Option Off
---------------	---

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.9.4.40 void msr\_registerCallBk ( pMSR\_callBack *pMSRf* )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.9.4.41 void msr\_registerCallBkp ( pMSR\_callBackp *pMSRf* )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.9.4.42 int msr\_setClearPANID ( BYTE *val* )

Set Clear PAN ID.

## Parameters

<i>val</i>	Set Clear PAN ID to value: Number of digits to show in clear. Range 0-6.
------------	--

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.9.4.43 int msr\_setExpirationMask ( int *mask* )

Set Expiration Masking

Sets the flag to mask the expiration date

## Parameters

<i>mask</i>	TRUE = mask expiration
-------------	------------------------

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.9.4.44 int msr\_setSwipeForcedEncryptionOption ( int *track1*, int *track2*, int *track3*, int *track3card0* )

Set MSR Swipe Forced Encryption Option.

## Parameters

<i>track1</i>	Set track1 encryption to true or false.
<i>track2</i>	Set track2 encryption to true or false.
<i>track3</i>	Set track3 encryption to true or false.
<i>track3card0</i>	Set track3 card0 encryption to true or false.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.9.4.45 int msr\_setSwipeMaskOption ( int *track1*, int *track2*, int *track3* )

Set MSR Swipe Mask Option.

Sets the swipe mask/clear data sending option

## Parameters

<i>track1</i>	Set track1 mask to true or false.
<i>track2</i>	Set track2 mask to true or false.
<i>track3</i>	Set track3 mask to true or false.

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.9.4.46 void pin\_registerCallBk ( pPIN\_callBack *pPINf* )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

#### 12.9.4.47 void registerHotplugCallBk ( pMessageHotplug *pMsgHotplug* )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

#### 12.9.4.48 void registerLogCallBk ( pSendDataLog *pFSend*, pReadDataLog *pFRead* )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.9.4.49 int rs232\_device\_init ( int *deviceType*, int *port\_number*, int *brate* )

Initial the device by RS232

It will try to connect to the device with provided deviceType, port\_number, and brate.

## Parameters

<i>deviceType</i>	Device to connect to
<i>port_number</i>	Port number of the device

## Port nr. | Linux | Windows

| 0 | ttyS0 | COM1 | | 1 | ttyS1 | COM2 | | 2 | ttyS2 | COM3 | | 3 | ttyS3 | COM4 | | 4 | ttyS4 | COM5 | | 5 | ttyS5 | COM6 | | 6 | ttyS6 | COM7 | | 7 | ttyS7 | COM8 | | 8 | ttyS8 | COM9 | | 9 | ttyS9 | COM10 | | 10 | ttyS10 | COM11 | | 11 | ttyS11 | COM12 | | 12 | ttyS12 | COM13 | | 13 | ttyS13 | COM14 | | 14 | ttyS14 | COM15 | | 15 | ttyS15 | COM16 | | 16 | ttyUSB0 | n.a. | | 17 | ttyUSB1 | n.a. | | 18 | ttyUSB2 | n.a. | | 19 | ttyUSB3 | n.a. | | 20 | ttyUSB4 | n.a. | | 21 | ttyUSB5 | n.a. | | 22 | ttyAMA0 | n.a. | | 23 | ttyAMA1 | n.a. | | 24 | ttyACM0 | n.a. | | 25 | ttyACM1 | n.a. | | 26 | rfcomm0 | n.a. | | 27 | rfcomm1 | n.a. | | 28 | ircomm0 | n.a. | | 29 | ircomm1 | n.a. | | 30 | cuau0 | n.a. | | 31 | cuau1 | n.a. | | 32 | cuau2 | n.a. | | 33 | cuau3 | n.a. | | 34 | cuaU0 | n.a. | | 35 | cuaU1 | n.a. | | 36 | cuaU2 | n.a. | | 37 | cuaU3 | n.a. |

## Parameters

<i>brate</i>	Bitrate of the device
--------------	-----------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.9.4.50 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.9.4.51 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.10 Source\_C/libIDT\_UniPayI\_V.h File Reference

UniPay 1.5 API.

```
#include "IDTDef.h"
```

## Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

## Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

## Functions

- void `registerHotplugCallBk` (pMessageHotplug pMsgHotplug)
- void `registerLogCallBk` (pSendDataLog pFSend, pReadDataLog pFRead)
- void `emv_registerCallBk` (pEMV\_callback pEMVf)
- void `msr_registerCallBk` (pMSR\_callback pMSRf)
- void `msr_registerCallBkp` (pMSR\_callbackp pMSRf)
- void `pin_registerCallBk` (pPIN\_callback pPINf)
- void `device_registerCameraCallBk` (pCMR\_callback pCMRf)
- void `device_registerCardStatusFrontSwitchCallBk` (pCSFS\_callback pCSFSf)
- void `comm_registerHTTPCallback` (httpComm\_callback cBack)
- void `comm_registerV4Callback` (v4Comm\_callback cBack)
- char \* `SDK_Version` ()
- int `setAbsoluteLibraryPath` (const char \*absoluteLibraryPath)
- int `device_init` ()
- int `device_setCurrentDevice` (int deviceType)
- int `device_close` ()
- void `device_getIDGStatusCodeString` (IN int returnCode, OUT char \*despcrition)
- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_getFirmwareVersion` (OUT char \*firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int `device_pingDevice` ()
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommandNEO` (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int `device_enablePassThrough` (int enablePassThrough)
- int `device_setMerchantRecord` (int index, int enabled, char \*merchantID, char \*merchantURL)
- int `device_getMerchantRecord` (IN int index, OUT BYTE \*record)
- int `device_getMerchantRecord_Len` (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int `device_getSDKWaitTime` ()
- void `device_setSDKWaitTime` (int waitTime)
- int `device_getThreadStackSize` ()

- void `device_setThreadStackSize` (int threadSize)
- int `config_getSerialNumber` (OUT char \*sNumber)
- int `config_getSerialNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- void `emv_allowFallback` (IN int allow)
- void `emv_setAutoAuthenticateTransaction` (IN int authenticate)
- void `emv_setAutoCompleteTransaction` (IN int complete)
- int `emv_getAutoAuthenticateTransaction` ()
- int `emv_getAutoCompleteTransaction` ()
- int `emv_startTransaction` (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int `emv_activateTransaction` (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int `emv_authenticateTransaction` (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int `emv_authenticateTransactionWithTimeout` (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int `emv_completeTransaction` (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int `emv_cancelTransaction` ()
- int `emv_retrieveApplicationData` (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setApplicationData` (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int `emv_setApplicationDataTLV` (IN BYTE \*tlv, IN int tlvLen)
- int `emv_removeApplicationData` (IN BYTE \*AID, IN int AIDLen)
- int `emv_removeAllApplicationData` ()
- int `emv_retrieveAIDList` (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int `emv_retrieveTerminalData` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `emv_setTerminalData` (IN BYTE \*tlv, IN int tlvLen)
- int `emv_setTerminalMajorConfiguration` (IN int configuration)
- int `emv_retrieveCAPK` (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int `emv_setCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeCAPK` (IN BYTE \*capk, IN int capkLen)
- int `emv_removeAllCAPK` ()
- int `emv_retrieveCAPKList` (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int `emv_retrieveCRL` (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int `emv_setCRL` (IN BYTE \*list, IN int lssLen)
- int `emv_removeCRL` (IN BYTE \*list, IN int lssLen)
- int `emv_removeAllCRL` ()
- int `icc_getICCRReaderStatus` (OUT BYTE \*status)
- int `icc_powerOnICC` (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int `icc_powerOffICC` ()
- int `icc_exchangeAPDU` (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int `msr_cancelMSRSwipe` ()
- int `msr_startMSRSwipe` (IN int \_timeout)
- void `parseMSRData` (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.10.1 Detailed Description

UniPay 1.5 API. UniPay 1.5 Global API methods.

### 12.10.2 Macro Definition Documentation

#### 12.10.2.1 #define IN

INPUT parameter.

### 12.10.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

### 12.10.2.3 #define OUT

OUTPUT parameter.

## 12.10.3 Typedef Documentation

### 12.10.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

### 12.10.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

### 12.10.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

### 12.10.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

### 12.10.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

### 12.10.3.6 typedef void(\* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

### 12.10.3.7 typedef void(\* pMSR\_callback)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

#### 12.10.3.8 typedef void(\* pMSR\_callBack)(int, IDTMSRData \*)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callBack

#### 12.10.3.9 typedef void(\* pPIN\_callBack)(int, IDTPINData \*)

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

#### 12.10.3.10 typedef void(\* pReadDataLog)(unsigned char \*, int)

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

#### 12.10.3.11 typedef void(\* pSendDataLog)(unsigned char \*, int)

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

#### 12.10.3.12 typedef void(\* v4Comm\_callBack)(BYTE, BYTE, BYTE \*, int)

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

### 12.10.4 Function Documentation

#### 12.10.4.1 void comm\_registerHTTPCallback ( httpComm\_callBack cBack )

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

#### 12.10.4.2 void comm\_registerV4Callback ( v4Comm\_callBack cBack )

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

#### 12.10.4.3 int config\_getSerialNumber ( OUT char \* sNumber )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* sNumber, IN\_OUT int \*sNumberLen)

Polls device for Serial Number



## Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.10.4.4 `int config_getSerialNumber_Len ( OUT char * sNumber, IN_OUT int * sNumberLen )`

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.10.4.5 `int device_close ( )`

Close the device

## Returns

RETURN\_CODE: 0: success, 0x0A: failed

12.10.4.6 `int device_enablePassThrough ( int enablePassThrough )`

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

## Parameters

<i>enablePass-Through</i>	1 = pass through ON, 0 = pass through OFF
---------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.10.4.7 `int device_getCurrentDeviceType ( )`

Get current active device type

## Returns

: return the device type defined as `DEVICE_TYPE` in the `IDTDef.h`

#### 12.10.4.8 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

##### Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)()

#### 12.10.4.9 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

##### Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)()

#### 12.10.4.10 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcrition* )

Review the return code description.

##### Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

##### Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";

- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVocomm activate transaction card type (ViVocomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.10.4.11 int device\_getMerchantRecord ( IN int *index*, OUT BYTE \* *record* )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int index, OUT BYTE \* record, IN\_OUT int \*recordLen)

#### Get Merchant Record

Gets the merchant record for the device.

#### Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i> ;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**See Also**

[ErrorCode](#)

**12.10.4.12** `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

**Get Merchant Record**

Gets the merchant record for the device.

**Parameters**

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**See Also**

[ErrorCode](#)

**12.10.4.13** `int device_getSDKWaitTime ( )`

**Get SDK Wait Time**

Get the SDK wait time for transactions

**Returns**

SDK wait time in seconds

**12.10.4.14** `int device_getThreadStackSize ( )`

**Get Thread Stack Size**

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.10.4.15** `int device_init ( )`

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.10.4.16 `int device_isAttached ( int deviceType )`

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

## 12.10.4.17 int device\_isConnected ( )

Check the device conntected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

## 12.10.4.18 int device\_pingDevice ( )

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.10.4.19 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.10.4.20 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.10.4.21 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

## Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.10.4.22 int device\_setCurrentDevice ( int *deviceType* )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

12.10.4.23 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.10.4.24 void device\_setSDKWaitTime ( int waitTime )

## Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

## 12.10.4.25 void device\_setThreadStackSize ( int threadSize )

## Set Thread Stack Size

Set the stack size setting for newly created threads

## 12.10.4.26 int emv\_activateTransaction ( IN int timeout, IN BYTE \* tags, IN int tagsLen, IN int forceOnline )

## Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## 12.10.4.27 void emv\_allowFallback ( IN int allow )

Allow fallback for EMV transactions. Default is TRUE



## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.10.4.28 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.10.4.29 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

#### 12.10.4.30 int emv\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.10.4.31 int emv\_completeTransaction ( IN int *commError*, IN BYTE \* *authCode*, IN int *authCodeLen*, IN BYTE \* *iad*, IN int *iadLen*, IN BYTE \* *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from emv\_authenticateTransaction

The tags will be returned in the callback routine.

##### Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

#### 12.10.4.32 int emv\_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

##### Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

#### 12.10.4.33 int emv\_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

##### Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

#### 12.10.4.34 void emv\_registerCallBk ( pEMV\_callback pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

**12.10.4.35 int emv\_removeAllApplicationData ( )**

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.10.4.36 int emv\_removeAllCAPK ( )**

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.10.4.37 int emv\_removeAllCRL ( )**

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.10.4.38 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.10.4.39 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )**

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.10.4.40 int emv\_removeCRL ( IN BYTE \* *list*, IN int *lsLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

##### Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.10.4.41 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

##### Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.10.4.42 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

##### Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.10.4.43 int emv\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.10.4.44 int emv\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.10.4.45 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *lssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.10.4.46** `int emv_retrieveTerminalData ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

**Retrieve Terminal Data**

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.10.4.47** `int emv_setApplicationData ( IN BYTE * name, IN int nameLen, IN BYTE * tlv, IN int tlvLen )`

**Set Application Data by AID**

Sets the Application Data as specified by the application name and TLV data

**Parameters**

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.10.4.48** `int emv_setApplicationDataTLV ( IN BYTE * tlv, IN int tlvLen )`

**Set Application Data by TLV**

Sets the Application Data as specified by the TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.10.4.49** `void emv_setAutoAuthenticateTransaction ( IN int authenticate )`

Enables authenticate for EMV transactions. If a `emv_startTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

12.10.4.50 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

12.10.4.51 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
-------------	--

<i>capkLen</i>	the length of capk data buffer
----------------	--------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.10.4.52 int emv\_setCRL ( IN BYTE \* list, IN int lsLen )****Set Certificate Revocation List**

Sets the CRL

**Parameters**

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.10.4.53 int emv\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )****Set Terminal Data**

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

**Parameters**

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

**12.10.4.54 int emv\_setTerminalMajorConfiguration ( IN int configuration )**

Sets the terminal major configuration in ICS .

**Parameters**

<i>configuration</i>	<p>A configuration value, range 1-23</p> <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	---



## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.10.4.55** `int emv_startTransaction ( IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

## Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#) >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**12.10.4.56** `int icc_exchangeAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

## Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

## Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

**12.10.4.57** `int icc_getICCRReaderStatus ( OUT BYTE * status )`

## Get Reader Status

Returns the reader status

## Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.10.4.58 `int icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

12.10.4.59 `int icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

## Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.10.4.60 `int msr_cancelMSRSwipe ( )`

Disable MSR Swipe Cancels MSR swipe request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.10.4.61 `void msr_registerCallBk ( pMSR_callBack pMSRf )`

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.10.4.62 `void msr_registerCallBkp ( pMSR_callBackp pMSRf )`

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.10.4.63 `int msr_startMSRSwipe ( IN int _timeout )`

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `deviceDelegate::swipeMSRData:()`

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.10.4.64 void parseMSRData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTMSRData \* *cardData* )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of <i>resData</i>
<i>cardData</i>	the parser result with IDTMSTData structure

12.10.4.65 void pin\_registerCallBk ( pPIN\_callBack *pPINf* )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.10.4.66 void registerHotplugCallBk ( pMessageHotplug *pMsgHotplug* )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.10.4.67 void registerLogCallBk ( pSendDataLog *pFSend*, pReadDataLog *pFRead* )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.10.4.68 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.10.4.69 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.11 Source\_C/libIDT\_Vendi.h File Reference

Vendi API.

```
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

### Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callback pEMVf)`
- `void msr_registerCallBk (pMSR_callback pMSRf)`
- `void msr_registerCallBkp (pMSR_callbackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callback pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callbackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callback pPINf)`
- `void device_registerCameraCallBk (pCMR_callback pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callback pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callback cBack)`
- `void comm_registerV4Callback (v4Comm_callback cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int device_setCurrentDevice (int deviceType)`
- `int device_isAttached (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcriton)`
- `int device_isConnected ()`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`
- `int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`
- `int device_pingDevice ()`

- int [device\\_controlUserInterface](#) (IN BYTE \*values)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()
- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- int [msr\\_cancelMSRSwipe](#) ()
- int [msr\\_startMSRSwipe](#) (IN int \_timeout)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.11.1 Detailed Description

Vendi API. Vendi Global API methods.

### 12.11.2 Macro Definition Documentation

#### 12.11.2.1 #define IN

INPUT parameter.

#### 12.11.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

### 12.11.2.3 #define OUT

OUTPUT parameter.

## 12.11.3 Typedef Documentation

### 12.11.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

### 12.11.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

### 12.11.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

### 12.11.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

### 12.11.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

### 12.11.3.6 typedef void(\* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

### 12.11.3.7 typedef void(\* pMSR\_callback)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

### 12.11.3.8 typedef void(\* pMSR\_callback)(int, IDTMSRData \*)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

**12.11.3.9** `typedef void(* pPIN_callBack)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

**12.11.3.10** `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

**12.11.3.11** `typedef void(* pSendDataLog)(unsigned char *, int)`

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

**12.11.3.12** `typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)`

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

## 12.11.4 Function Documentation

**12.11.4.1** `void comm_registerHTTPCallback ( httpComm_callBack cBack )`

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	HTTP Comm callback
--------------	--------------------

**12.11.4.2** `void comm_registerV4Callback ( v4Comm_callBack cBack )`

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	V4 Protocol Comm callback
--------------	---------------------------

**12.11.4.3** `int config_getSerialNumber ( OUT char * sNumber )`

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* sNumber, IN\_OUT int \*sNumberLen)

Polls device for Serial Number

**Parameters**

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

#### 12.11.4.4 `int config_getSerialNumber_Len ( OUT char * sNumber, IN_OUT int * sNumberLen )`

Polls device for Serial Number

**Parameters**

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

**Returns**

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

#### 12.11.4.5 `int ctls_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU



- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.11.4.6 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.7 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.8 int ctls\_getConfigurationGroup ( IN int group, OUT BYTE \* tlv, OUT int \* tlvLen )

Get Configuration Group

Retrieves the Configuration for the specified Group.

## Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.9 void ctls\_registerCallBk ( pMSR\_callBack pCTLSf )

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.11.4.10 void ctls\_registerCallBkp ( pMSR\_callBackp pCTLSf )

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.11.4.11 int ctls\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.12 int ctls\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.13 int ctls\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.14 int ctls\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.11.4.15 int ctls\_removeConfigurationGroup ( int group )

## Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

## Parameters

<i>group</i>	Configuration Group
--------------	---------------------

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

## 12.11.4.16 int ctls\_retrieveAIDList ( OUT BYTE \* AIDList, IN\_OUT int \* AIDListLen )

## Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.11.4.17 int ctls\_retrieveApplicationData ( IN BYTE \* AID, IN int AIDLen, OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

## Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.11.4.18 int ctls\_retrieveCAPK ( IN BYTE \* capk, IN int capkLen, OUT BYTE \* key, IN\_OUT int \* keyLen )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.11.4.19 int `ctls_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.11.4.20 int `ctls_retrieveTerminalData ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.11.4.21 int ctls\_setApplicationData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

## Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.11.4.22 int ctls\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

## Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.23 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.24 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

#### 12.11.4.25 int ctls\_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \* tags, IN int tagsLen )

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DFO1 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

**12.11.4.26 int device\_close ( )**

Close the device

**Returns**

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.11.4.27 int device\_controlUserInterface ( IN BYTE \* *values* )

##### Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.11.4.28 int device\_enablePassThrough ( int *enablePassThrough* )

##### Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

```
@return RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString
```

##### Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

##### Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



## 12.11.4.29 int device\_getCurrentDeviceType ( )

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.11.4.30 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)()

12.11.4.31 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)()

12.11.4.32 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcrition* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";

- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCARD3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.11.4.33 int device\_getMerchantRecord ( IN int index, OUT BYTE \* record )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int index, OUT BYTE \* record, IN\_OUT int \*recordLen)

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

**12.11.4.34** `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

## Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

**12.11.4.35** `int device_getSDKWaitTime ( )`

## Get SDK Wait Time

Get the SDK wait time for transactions

## Returns

SDK wait time in seconds

**12.11.4.36** `int device_getThreadStackSize ( )`

## Get Thread Stack Size

Get the stack size setting for newly created threads

## Returns

Thread Stack Size

**12.11.4.37** `int device_getTransactionResults ( IDTMSRData * cardData )`

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

**Parameters**

<i>cardData</i>	The transaction results
-----------------	-------------------------

**Returns**

success or error code. Values can be parsed with `device_getResponseCodeString`

**See Also**

`ErrorCode`

**12.11.4.38 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by `device_isConnected()`.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

**12.11.4.39 int device\_isAttached ( int deviceType )**

Check if the device is attached to the USB port The function `device_init()` must be called before this function.

**Parameters**

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

**Returns**

1 if the device is attached, or 0 if the device is not attached

**12.11.4.40 int device\_isConnected ( )**

Check the device connected status

**Returns**

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

**12.11.4.41 int device\_pingDevice ( )**

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

12.11.4.42 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

12.11.4.43 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

12.11.4.44 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.11.4.45 int device\_setBurstMode ( IN BYTE mode )

Send Burst Mode

Sets the burst mode for the device.

Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

[ErrorCode](#)

#### 12.11.4.46 int device\_setCurrentDevice ( int *deviceType* )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to
	<pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 }; </pre>

## Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.11.4.47 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.11.4.48 int device\_setPollMode ( IN BYTE mode )****Set Poll Mode**

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

**Parameters**

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.11.4.49 void device\_setSDKWaitTime ( int waitTime )****Set SDK Wait Time**

Set the SDK wait time for transactions

**Parameters**

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

**12.11.4.50 void device\_setThreadStackSize ( int threadSize )****Set Thread Stack Size**

Set the stack size setting for newly created threads

**12.11.4.51 void emv\_registerCallBk ( pEMV\_callback pEMVf )**

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

**12.11.4.52 int msr\_cancelMSRSwipe ( )**

Disable MSR Swipe Cancels MSR swipe request.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.11.4.53 void msr\_registerCallBk ( pMSR\_callback pMSRf )**

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

**12.11.4.54 void msr\_registerCallBkp ( pMSR\_callbackp pMSRf )**

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

**12.11.4.55 int msr\_startMSRSwipe ( IN int\_timeout )**

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.11.4.56 void parseMSRData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTMSRData \* *cardData* )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of <i>resData</i>
<i>cardData</i>	the parser result with IDTMSTData structure

12.11.4.57 void pin\_registerCallBk ( pPIN\_callBack *pPINf* )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.11.4.58 void registerHotplugCallBk ( pMessageHotplug *pMsgHotplug* )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.11.4.59 void registerLogCallBk ( pSendDataLog *pFSend*, pReadDataLog *pFRead* )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.11.4.60 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.11.4.61 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



## 12.12 Source\_C/libIDT\_VP3300\_AJ.h File Reference

VP3300 AJ API.

```
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`

### Functions

- void `registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- void `registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- void `device_registerRKICallBk (pRKI_callback pRKIf)`
- void `emv_registerCallBk (pEMV_callback pEMVf)`
- void `msr_registerCallBk (pMSR_callback pMSRf)`
- void `msr_registerCallBkp (pMSR_callbackp pMSRf)`
- void `ctls_registerCallBk (pMSR_callback pCTLSf)`
- void `ctls_registerCallBkp (pMSR_callbackp pCTLSf)`
- void `pin_registerCallBk (pPIN_callback pPINf)`
- void `device_registerCameraCallBk (pCMR_callback pCMRf)`
- void `device_registerCardStatusFrontSwitchCallBk (pCSFS_callback pCSFSf)`
- int `device_getSDKWaitTime ()`
- void `device_setSDKWaitTime (int waitTime)`
- int `device_getThreadStackSize ()`
- void `device_setThreadStackSize (int threadSize)`
- void `comm_registerHTTPCallback (httpComm_callback cBack)`
- void `comm_registerV4Callback (v4Comm_callback cBack)`
- char \* `SDK_Version ()`
- int `setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- int `device_init ()`
- int `device_setCurrentDevice (int deviceType)`
- int `device_close ()`
- void `device_getIDGStatusCodeString (IN int returnCode, OUT char *despcriton)`

- int `device_isConnected` ()
- int `device_isAttached` (int deviceType)
- int `device_getFirmwareVersion` (OUT char \*firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int `device_pingDevice` ()
- int `device_controlUserInterface` (IN BYTE \*values)
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommandNEO` (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int `device_enablePassThrough` (int enablePassThrough)
- int `device_setBurstMode` (IN BYTE mode)
- int `device_setPollMode` (IN BYTE mode)
- int `device_setMerchantRecord` (int index, int enabled, char \*merchantID, char \*merchantURL)
- int `device_getMerchantRecord` (IN int index, OUT BYTE \*record)
- int `device_getMerchantRecord_Len` (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int `device_getTransactionResults` (IDTMSRData \*cardData)
- int `device_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- void `device_setTransactionExponent` (int exponent)
- int `device_activateTransaction` (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `device_cancelTransaction` ()
- int `device_getRTCDDateTime` (IN BYTE \*dateTime, IN\_OUT int \*dateTimeLen)
- int `device_setRTCDDateTime` (IN BYTE \*dateTime, IN int dateTimeLen)
- int `device_startRKI` (const char \*caPath)
- int `config_getSerialNumber` (OUT char \*sNumber)
- int `config_getSerialNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int `ctls_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `ctls_activateTransaction` (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `ctls_cancelTransaction` ()
- int `ctls_retrieveApplicationData` (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `ctls_setApplicationData` (IN BYTE \*tlv, IN int tlvLen)
- int `ctls_removeApplicationData` (IN BYTE \*AID, IN int AIDLen)
- int `ctls_removeAllApplicationData` ()
- int `ctls_retrieveAIDList` (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int `ctls_retrieveTerminalData` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `ctls_setTerminalData` (IN BYTE \*tlv, IN int tlvLen)
- int `ctls_retrieveCAPK` (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int `ctls_setCAPK` (IN BYTE \*capk, IN int capkLen)
- int `ctls_removeCAPK` (IN BYTE \*capk, IN int capkLen)
- int `ctls_removeAllCAPK` ()
- int `ctls_retrieveCAPKList` (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int `ctls_setConfigurationGroup` (IN BYTE \*tlv, IN int tlvLen)
- int `ctls_getConfigurationGroup` (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int `ctls_getAllConfigurationGroups` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `ctls_removeConfigurationGroup` (int group)
- void `emv_allowFallback` (IN int allow)
- void `emv_setAutoAuthenticateTransaction` (IN int authenticate)
- void `emv_setAutoCompleteTransaction` (IN int complete)
- int `emv_getAutoAuthenticateTransaction` ()
- int `emv_getAutoCompleteTransaction` ()
- void `emv_setTransactionParameters` (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen)
- int `emv_startTransaction` (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)

- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()
- int [emv\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [emv\\_retrieveCRL](#) (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int [emv\\_setCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeAllCRL](#) ()
- int [icc\\_getICReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [msr\\_cancelMSRSwipe](#) ()
- int [msr\\_startMSRSwipe](#) (IN int \_timeout)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.12.1 Detailed Description

VP3300 AJ API. VP3300 AJ Global API methods.

### 12.12.2 Macro Definition Documentation

#### 12.12.2.1 #define IN

INPUT parameter.

#### 12.12.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.12.2.3 #define OUT

OUTPUT parameter.

### 12.12.3 Typedef Documentation

#### 12.12.3.1 `typedef void(* ftpComm_callBack)(int, int, int)`

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.12.3.2 `typedef void(* httpComm_callBack)(BYTE *, int)`

Define the comm callback function to get the async url data

It should be registered using the `comm_registerHTTPCallback`

#### 12.12.3.3 `typedef void(* pCMR_callBack)(int, IDTCMRData *)`

Define the camera callback function to get the image data

It should be registered using the `device_registerCameraCallBk`,

#### 12.12.3.4 `typedef void(* pCSFS_callBack)(BYTE status)`

Define the card status and front switch callback function to get card and front switch status

It should be registered using the `device_registerCardStatusFrontSwitchCallBk`,

#### 12.12.3.5 `typedef void(* pEMV_callBack)(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the `emv_registerCallBk`,

#### 12.12.3.6 `typedef void(* pMessageHotplug)(int, int)`

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the `registerHotplugCallBk`, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.12.3.7 `typedef void(* pMSR_callBack)(int, IDTMSRData)`

Define the MSR callback function to get the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is for backward compatibility

#### 12.12.3.8 `typedef void(* pMSR_callBackp)(int, IDTMSRData *)`

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the `msr_registerCallBk`, this callback function is recommended instead of `pMSR_callBack`

**12.12.3.9** `typedef void(* pPIN_callBack)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

**12.12.3.10** `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

**12.12.3.11** `typedef void(* pSendDataLog)(unsigned char *, int)`

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

**12.12.3.12** `typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)`

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

## 12.12.4 Function Documentation

**12.12.4.1** `void comm_registerHTTPCallback ( httpComm_callBack cBack )`

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

**12.12.4.2** `void comm_registerV4Callback ( v4Comm_callBack cBack )`

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

**12.12.4.3** `int config_getSerialNumber ( OUT char * sNumber )`

DEPRECATED : please use [config\\_getSerialNumber\\_Len\(OUT char\\* sNumber, IN\\_OUT int \\*sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.12.4.4 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.12.4.5 `int ctls_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()` Note: if auto poll is on, it will return the error `IDG_P2_STATUS_CODE_COMMAND_NOT_ALLOWED`

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal

- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.12.4.6 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.7 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.8 int ctls\_getConfigurationGroup ( IN int group, OUT BYTE \* tlv, OUT int \* tlvLen )

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



**12.12.4.9 void ctls\_registerCallBk ( pMSR\_callBack pCTLSf )**

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

**12.12.4.10 void ctls\_registerCallBkp ( pMSR\_callBackp pCTLSf )**

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

**12.12.4.11 int ctls\_removeAllApplicationData ( )**

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.12 int ctls\_removeAllCAPK ( )**

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.13 int ctls\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.14 int ctls\_removeCAPK ( IN BYTE \* capk, IN int capkLen )**

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.15 int ctls\_removeConfigurationGroup ( int *group* )

##### Remove Configuration Group

Removes the Configuration as specified by the Group. Must not be group 0

##### Parameters

<i>group</i>	Configuration Group
--------------	---------------------

##### Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

#### 12.12.4.16 int ctls\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

##### Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

##### Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.17 int ctls\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

##### Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

##### Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.18 int ctls\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

##### Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

##### Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---

<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.19 int ctls\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.20 int ctls\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.21 int ctls\_setApplicationData ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

##### Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

##### Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.22 int ctls\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

##### Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
-------------	--

<i>capkLen</i>	the length of capk data buffer
----------------	--------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.23 int ctls\_setConfigurationGroup ( IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Configuration Group**

Sets the Configuration Group for CTLS as specified by the TLV data

**Parameters**

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.24 int ctls\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Terminal Data**

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

**Parameters**

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

#### 12.12.4.25 int ctls\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

**Start CTLS Transaction Request**

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DFO1 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

12.12.4.26 `int device_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9-C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of device\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

**12.12.4.27 int device\_cancelTransaction ( )**

Disable Transaction Cancel Transaction request.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.28 int device\_close ( )**

Close the device

**Returns**

RETURN\_CODE: 0: success, 0x0A: failed

**12.12.4.29 int device\_controlUserInterface ( IN BYTE \* values )**

Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



12.12.4.30 int device\_enablePassThrough ( int *enablePassThrough* )

Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.12.4.31 int device\_getCurrentDeviceType ( )

Get current active device type

Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.12.4.32 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.33 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersionLen</i>	Length of Firmware Version

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.34 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcrition* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";
  - 19: " Frame is too large";
  - 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
  - 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
  - 1C: " Problem encoding APDU Module-Specific Status Codes ";
  - 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
  - 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";

- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.12.4.35 int device\_getMerchantRecord ( IN int *index*, OUT BYTE \* *record* )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int index, OUT BYTE \* record, IN\_OUT int \*recordLen)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

See Also

[ErrorCode](#)

#### 12.12.4.36 int device\_getMerchantRecord\_Len ( IN int *index*, OUT BYTE \* *record*, IN\_OUT int \* *recordLen* )

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

See Also

[ErrorCode](#)

#### 12.12.4.37 int device\_getRTCDateTime ( IN BYTE \* *dateTime*, IN\_OUT int \* *dateTimeLen* )

get RTC date and time of the device

**Parameters**

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

**Returns**

success or error code. Values can be parsed with `device_getResponseCodeString`

**See Also**

`ErrorCode`

**12.12.4.38 int device\_getSDKWaitTime ( )**

Get SDK Wait Time

Get the SDK wait time for transactions

**Returns**

SDK wait time in seconds

**12.12.4.39 int device\_getThreadStackSize ( )**

Get Thread Stack Size

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.12.4.40 int device\_getTransactionResults ( IDTMSRData \* *cardData* )**

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

**Parameters**

<i>cardData</i>	The transaction results
-----------------	-------------------------

**Returns**

success or error code. Values can be parsed with `device_getResponseCodeString`

**See Also**

`ErrorCode`

**12.12.4.41 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by `device_isConnected()`.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.42 int device\_isAttached ( int *deviceType* )**

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

**Parameters**

<i>deviceType</i> , the	device type of the USB device
-------------------------	-------------------------------

**Returns**

1 if the device is attached, or 0 if the device is not attached

**12.12.4.43 int device\_isConnected ( )**

Check the device connected status

**Returns**

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

**12.12.4.44 int device\_pingDevice ( )**

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.45 void device\_registerCameraCallBk ( pCMR\_callBack *pCMRf* )**

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

**12.12.4.46 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack *pCSFSf* )**

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

**12.12.4.47 void device\_registerRKICallBk ( pRKI\_callBack *pRKIf* )**

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

**12.12.4.48 int device\_SendDataCommandNEO ( IN int *cmd*, IN int *subCmd*, IN BYTE \* *data*, IN int *dataLen*, OUT BYTE \* *response*, IN\_OUT int \* *respLen* )**

Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

## Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.12.4.49 int device\_setBurstMode ( IN BYTE mode )

Send Burst Mode

Sets the burst mode for the device.

## Parameters

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

ErrorCode

## 12.12.4.50 int device\_setCurrentDevice ( int deviceType )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.12.4.51 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.52 int device\_setPollMode ( IN BYTE *mode* )

Set Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

## Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.53 `int device_setRTCDateTime ( IN BYTE * dateTime, IN int dateTimeLen )`

set RTC date and time of the device

## Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

## Returns

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

## See Also

[ErrorCode](#)

12.12.4.54 `void device_setSDKWaitTime ( int waitTime )`

Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

12.12.4.55 `void device_setThreadStackSize ( int threadSize )`

Set Thread Stack Size

Set the stack size setting for newly created threads

12.12.4.56 `void device_setTransactionExponent ( int exponent )`

Sets the transaction exponent to be used with [device\\_startTransaction](#). Default value is 2

## Parameters

<i>exponent, The</i>	exponent to use when calling <a href="#">device_startTransaction</a>
----------------------	--

12.12.4.57 `int device_startRKI ( const char * caPath )`

Start remote key injection.



## Parameters

<i>caPath</i>	The path to ca-certificates.crt
---------------	---------------------------------

## Returns

success or error code.

## See Also

ErrorCode

**12.12.4.58** `int device_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

## Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4

- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

12.12.4.59 `int emv_activateTransaction ( IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.12.4.60 `void emv_allowFallback ( IN int allow )`

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.12.4.61 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.12.4.62 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>

<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.12.4.63 int emv\_cancelTransaction ( )****Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.12.4.64 int emv\_completeTransaction ( IN int commError, IN BYTE \* authCode, IN int authCodeLen, IN BYTE \* iad, IN int iadLen, IN BYTE \* tlvScripts, IN int tlvScriptsLen, IN BYTE \* tlv, IN int tlvLen )****Complete EMV Transaction Request**

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

**Parameters**

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.12.4.65 int emv\_getAutoAuthenticateTransaction ( )**

Gets auto authenticate value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

**12.12.4.66 int emv\_getAutoCompleteTransaction ( )**

Gets auto complete value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

12.12.4.67 void emv\_registerCallBk ( pEMV\_callBack pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.12.4.68 int emv\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.69 int emv\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.70 int emv\_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.12.4.71 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.72 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.73 `int emv_removeCRL ( IN BYTE * list, IN int lsLen )`

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.12.4.74 `int emv_retrieveAIDList ( OUT BYTE * AIDList, IN_OUT int * AIDListLen )`

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.75 `int emv_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.76 int emv\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.77 int emv\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.12.4.78 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *lssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters



<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>IssLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.12.4.79 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

**Retrieve Terminal Data**

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.80 int emv\_setApplicationData ( IN BYTE \* *name*, IN int *nameLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Application Data by AID**

Sets the Application Data as specified by the application name and TLV data

**Parameters**

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.12.4.81 int emv\_setApplicationDataTLV ( IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Application Data by TLV**

Sets the Application Data as specified by the TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.12.4.82 void emv\_setAutoAuthenticateTransaction ( IN int *authenticate* )

Enables authenticate for EMV transactions. If a emv\_startTransaction results in code 0x0010 (start transaction success), then emv\_authenticateTransaction can automatically execute if parameter is set to TRUE

##### Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

#### 12.12.4.83 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

##### Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

#### 12.12.4.84 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

##### Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
-------------	--

<i>capkLen</i>	the length of capk data buffer
----------------	--------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.12.4.85 int emv\_setCRL ( IN BYTE \* list, IN int lsLen )****Set Certificate Revocation List**

Sets the CRL

**Parameters**

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.12.4.86 int emv\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )****Set Terminal Data**

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

**Parameters**

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

**12.12.4.87 int emv\_setTerminalMajorConfiguration ( IN int configuration )**

Sets the terminal major configuration in ICS .

**Parameters**

<i>configuration</i>	<p>A configuration value, range 1-23</p> <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.12.4.88** void emv\_setTransactionParameters ( IN double *amount*, IN double *amtOther*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

**12.12.4.89** int emv\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

## Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#) >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.12.4.90 `int icc_exchangeAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

#### Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

#### Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.12.4.91 `int icc_getICCRaderStatus ( OUT BYTE * status )`

Get Reader Status

Returns the reader status

#### Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

#### Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.12.4.92 `int icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

12.12.4.93 `int icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

#### Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.94 int msr\_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.12.4.95 void msr\_registerCallBk ( pMSR\_callBack pMSRf )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.12.4.96 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.12.4.97 int msr\_startMSRSwipe ( IN int \_timeout )

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

##### Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

#### 12.12.4.98 void parseMSRData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTMSRData \* cardData )

Parser the MSR data from the buffer into IDTMSTData structure

##### Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

#### 12.12.4.99 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

#### 12.12.4.100 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

#### 12.12.4.101 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

## 12.12.4.102 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.12.4.103 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.13 Source\_C/libIDT\_VP3300\_BT.h File Reference

VP3300 BT API.

```
#include "IDTDef.h"
```

## Macros

- #define [IN](#)
- #define [OUT](#)
- #define [IN\\_OUT](#)

## Typedefs

- typedef void(\* [pMessageHotplug](#) )(int, int)
- typedef void(\* [pSendDataLog](#) )(unsigned char \*, int)
- typedef void(\* [pReadDataLog](#) )(unsigned char \*, int)
- typedef void(\* [pEMV\\_callBack](#) )(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)
- typedef void(\* [pMSR\\_callBack](#) )(int, IDTMSRData)
- typedef void(\* [pMSR\\_callBackp](#) )(int, IDTMSRData \*)
- typedef void(\* [pPIN\\_callBack](#) )(int, IDTPINData \*)
- typedef void(\* [pCMR\\_callBack](#) )(int, IDTCMRData \*)
- typedef void(\* [pCSFS\\_callBack](#) )(BYTE status)
- typedef void(\* [ftpComm\\_callBack](#) )(int, int, int)
- typedef void(\* [httpComm\\_callBack](#) )(BYTE \*, int)
- typedef void(\* [v4Comm\\_callBack](#) )(BYTE, BYTE, BYTE \*, int)

## Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, [pReadDataLog](#) pFRead)
- void [device\\_registerRKICallBk](#) (pRKI\_callBack pRKIf)
- void [emv\\_registerCallBk](#) (pEMV\_callBack pEMVf)
- void [msr\\_registerCallBk](#) (pMSR\_callBack pMSRf)
- void [msr\\_registerCallBkp](#) (pMSR\_callBack pMSRf)
- void [ctls\\_registerCallBk](#) (pMSR\_callBack pCTLSf)
- void [ctls\\_registerCallBkp](#) (pMSR\_callBack pCTLSf)
- void [pin\\_registerCallBk](#) (pPIN\_callBack pPINf)
- void [device\\_registerCameraCallBk](#) (pCMR\_callBack pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callBack pCSFSf)
- void [comm\\_registerHTTPCallback](#) (httpComm\_callBack cBack)
- void [comm\\_registerV4Callback](#) (v4Comm\_callBack cBack)
- char \* [SDK\\_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char \*absoluteLibraryPath)
- int [device\\_init](#) ()
- int [device\\_setCurrentDevice](#) (int deviceType)
- int [device\\_close](#) ()
- void [device\\_getIDGStatusCodeString](#) (IN int returnCode, OUT char \*despcrition)
- int [device\\_isConnected](#) ()
- int [device\\_isAttached](#) (int deviceType)
- int [device\\_getFirmwareVersion](#) (OUT char \*firmwareVersion)
- int [device\\_getFirmwareVersion\\_Len](#) (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int [device\\_pingDevice](#) ()
- int [device\\_controlUserInterface](#) (IN BYTE \*values)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [device\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- void [device\\_setTransactionExponent](#) (int exponent)
- int [device\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [device\\_cancelTransaction](#) ()
- int [device\\_getRTCDateTime](#) (IN BYTE \*dateTime, IN\_OUT int \*dateTimeLen)
- int [device\\_setRTCDateTime](#) (IN BYTE \*dateTime, IN int dateTimeLen)
- int [device\\_startRKI](#) (const char \*caPath)
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()



- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- void [emv\\_allowFallback](#) (IN int allow)
- void [emv\\_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv\\_setAutoCompleteTransaction](#) (IN int complete)
- int [emv\\_getAutoAuthenticateTransaction](#) ()
- int [emv\\_getAutoCompleteTransaction](#) ()
- void [emv\\_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen)
- int [emv\\_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()
- int [emv\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [emv\\_retrieveCRL](#) (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int [emv\\_setCRL](#) (IN BYTE \*list, IN int lsLen)
- int [emv\\_removeCRL](#) (IN BYTE \*list, IN int lsLen)
- int [emv\\_removeAllCRL](#) ()
- int [icc\\_getICCRReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [msr\\_cancelMSRSwipe](#) ()
- int [msr\\_startMSRSwipe](#) (IN int \_timeout)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.13.1 Detailed Description

VP3300 BT API. VP3300 BT Global API methods.

### 12.13.2 Macro Definition Documentation

#### 12.13.2.1 #define IN

INPUT parameter.

#### 12.13.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.13.2.3 #define OUT

OUTPUT parameter.

### 12.13.3 Typedef Documentation

#### 12.13.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.13.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

#### 12.13.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

#### 12.13.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

#### 12.13.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

**12.13.3.6 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.13.3.7 typedef void(\* pMSR\_callBack)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.13.3.8 typedef void(\* pMSR\_callBackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callBack

**12.13.3.9 typedef void(\* pPIN\_callBack)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.13.3.10 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

**12.13.3.11 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

**12.13.3.12 typedef void(\* v4Comm\_callBack)(BYTE, BYTE, BYTE \*, int)**

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

**12.13.4 Function Documentation****12.13.4.1 void comm\_registerHTTPCallback ( httpComm\_callBack cBack )**

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

12.13.4.2 void comm\_registerV4Callback ( v4Comm\_callBack *cBack* )

Register External V4 Protocol commands Callback

## Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

12.13.4.3 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.13.4.4 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.13.4.5 int ctls\_activateTransaction ( IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using [device\\_setMerchantRecord](#), then container tag FFEE06 must be sent as part of the additional tags parameter of [ctls\\_startTransaction](#). Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

#### 12.13.4.6 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.7 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

## Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.8 `int ctls_getConfigurationGroup ( IN int group, OUT BYTE * tlv, OUT int * tlvLen )`

## Get Configuration Group

Retrieves the Configuration for the specified Group.

## Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.9 `void ctls_registerCallBk ( pMSR_callBack pCTLSf )`

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.13.4.10 `void ctls_registerCallBkp ( pMSR_callBackp pCTLSf )`

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.13.4.11 `int ctls_removeAllApplicationData ( )`

## Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.12 `int ctls_removeAllCAPK ( )`

## Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.13 `int ctls_removeApplicationData ( IN BYTE * AID, IN int AIDLen )`

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.14 `int ctls_removeCAPK ( IN BYTE * capk, IN int capkLen )`

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.15 `int ctls_removeConfigurationGroup ( int group )`

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

## Parameters

<i>group</i>	Configuration Group
--------------	---------------------

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.13.4.16 `int ctls_retrieveAIDList ( OUT BYTE * AIDList, IN_OUT int * AIDListLen )`

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.17 `int ctls_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.



## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.18 int ctls\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	the length of key data buffer <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.19 int ctls\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

## Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.20 `int ctls_retrieveTerminalData ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

## Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.21 `int ctls_setApplicationData ( IN BYTE * tlv, IN int tlvLen )`

## Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

## Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.22 `int ctls_setCAPK ( IN BYTE * capk, IN int capkLen )`

## Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.13.4.23 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.13.4.24 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.13.4.25 `int ctls_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100. If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal

- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

12.13.4.26 `int device_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9-C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU

- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.13.4.27 int device\_cancelTransaction ( )

Disable Transaction Cancel Transaction request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.28 int device\_close ( )

Close the device

##### Returns

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.13.4.29 int device\_controlUserInterface ( IN BYTE \* values )

Control User Interface

Controls the User Interface: Display, Beep, LED

```

@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On

```

### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.30 int device\_enablePassThrough ( int *enablePassThrough* )

##### Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

##### Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

##### Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.31 int device\_getCurrentDeviceType ( )

##### Get current active device type

**Returns**

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.13.4.32 `int device_getFirmwareVersion ( OUT char * firmwareVersion )`

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

**Parameters**

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.33 `int device_getFirmwareVersion_Len ( OUT char * firmwareVersion, IN_OUT int * firmwareVersionLen )`

Polls device for Firmware Version

**Parameters**

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.34 `void device_getIDGStatusCodeString ( IN int returnCode, OUT char * despcrition )`

Review the return code description.

**Parameters**

<i>returnCode</i>	the response result.
<i>description</i>	

**Return values**

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";



- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.13.4.35 int device\_getMerchantRecord ( IN int *index*, OUT BYTE \* *record* )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int index, OUT BYTE \* record, IN\_OUT int \*recordLen)

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

12.13.4.36 `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

## Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

12.13.4.37 `int device_getRTCDateTime ( IN BYTE * dateTime, IN_OUT int * dateTimeLen )`

get RTC date and time of the device

## Parameters

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

## Returns

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

## See Also

ErrorCode

**12.13.4.38 int device\_getSDKWaitTime ( )**

Get SDK Wait Time

Get the SDK wait time for transactions

**Returns**

SDK wait time in seconds

**12.13.4.39 int device\_getThreadStackSize ( )**

Get Thread Stack Size

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.13.4.40 int device\_getTransactionResults ( IDTMSRData \* *cardData* )**

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

**Parameters**

<i>cardData</i>	The transaction results
-----------------	-------------------------

**Returns**

success or error code. Values can be parsed with device\_getResponseCodeString

**See Also**

ErrorCode

**12.13.4.41 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.13.4.42 int device\_isAttached ( int *deviceType* )**

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

## 12.13.4.43 int device\_isConnected ( )

Check the device connected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

## 12.13.4.44 int device\_pingDevice ( )

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.13.4.45 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.13.4.46 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.13.4.47 void device\_registerRKICallBk ( pRKI\_callback pRKIf )

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

## 12.13.4.48 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

**Parameters**

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.13.4.49 int device\_setBurstMode ( IN BYTE mode )****Send Burst Mode**

Sets the burst mode for the device.

**Parameters**

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

**See Also**

[ErrorCode](#)

**12.13.4.50 int device\_setCurrentDevice ( int deviceType )**

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.13.4.51 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.52 int device\_setPollMode ( IN BYTE *mode* )

Set Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

## Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.53 `int device_setRTCDateTime ( IN BYTE * dateTime, IN int dateTimeLen )`

set RTC date and time of the device

## Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

## Returns

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

## See Also

[ErrorCode](#)

12.13.4.54 `void device_setSDKWaitTime ( int waitTime )`

Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

12.13.4.55 `void device_setThreadStackSize ( int threadSize )`

Set Thread Stack Size

Set the stack size setting for newly created threads

12.13.4.56 `void device_setTransactionExponent ( int exponent )`

Sets the transaction exponent to be used with [device\\_startTransaction](#). Default value is 2

## Parameters

<i>exponent, The</i>	exponent to use when calling <a href="#">device_startTransaction</a>
----------------------	--

12.13.4.57 `int device_startRKI ( const char * caPath )`

Start remote key injection.



## Parameters

<i>caPath</i>	The path to ca-certificates.crt
---------------	---------------------------------

## Returns

success or error code.

## See Also

ErrorCode

**12.13.4.58** `int device_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

## Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4

- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

12.13.4.59 `int emv_activateTransaction ( IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.13.4.60 `void emv_allowFallback ( IN int allow )`

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

12.13.4.61 int emv\_authenticateTransaction ( IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.13.4.62 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

## Authenticate EMV Transaction Request with Timeout

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>

<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.13.4.63 int emv\_cancelTransaction ( )****Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.13.4.64 int emv\_completeTransaction ( IN int commError, IN BYTE \* authCode, IN int authCodeLen, IN BYTE \* iad, IN int iadLen, IN BYTE \* tlvScripts, IN int tlvScriptsLen, IN BYTE \* tlv, IN int tlvLen )****Complete EMV Transaction Request**

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

**Parameters**

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.13.4.65 int emv\_getAutoAuthenticateTransaction ( )**

Gets auto authenticate value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

**12.13.4.66 int emv\_getAutoCompleteTransaction ( )**

Gets auto complete value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

12.13.4.67 void emv\_registerCallBk ( pEMV\_callBack pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.13.4.68 int emv\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.69 int emv\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.70 int emv\_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.13.4.71 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.72 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.73 `int emv_removeCRL ( IN BYTE * list, IN int lsLen )`

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.13.4.74 `int emv_retrieveAIDList ( OUT BYTE * AIDList, IN_OUT int * AIDListLen )`

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.75 `int emv_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.76 int emv\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.77 `int emv_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.13.4.78 `int emv_retrieveCRL ( OUT BYTE * list, IN_OUT int * lssLen )`

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters



<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>IssLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.13.4.79 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

**Retrieve Terminal Data**

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.80 int emv\_setApplicationData ( IN BYTE \* *name*, IN int *nameLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Application Data by AID**

Sets the Application Data as specified by the application name and TLV data

**Parameters**

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.13.4.81 int emv\_setApplicationDataTLV ( IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Application Data by TLV**

Sets the Application Data as specified by the TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.13.4.82 void emv\_setAutoAuthenticateTransaction ( IN int *authenticate* )

Enables authenticate for EMV transactions. If a emv\_startTransaction results in code 0x0010 (start transaction success), then emv\_authenticateTransaction can automatically execute if parameter is set to TRUE

##### Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

#### 12.13.4.83 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

##### Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

#### 12.13.4.84 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

##### Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
-------------	--

<i>capkLen</i>	the length of capk data buffer
----------------	--------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.13.4.85 int emv\_setCRL ( IN BYTE \* list, IN int lsLen )****Set Certificate Revocation List**

Sets the CRL

**Parameters**

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.13.4.86 int emv\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )****Set Terminal Data**

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

**Parameters**

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

**Return values**

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

**12.13.4.87 int emv\_setTerminalMajorConfiguration ( IN int configuration )**

Sets the terminal major configuration in ICS .

**Parameters**

<i>configuration</i>	<p>A configuration value, range 1-23</p> <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.13.4.88** void emv\_setTransactionParameters ( IN double *amount*, IN double *amtOther*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37
<i>tagsLen</i>	the length of tags

**12.13.4.89** int emv\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

## Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#) >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.13.4.90 `int icc_exchangeAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

#### Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

#### Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.13.4.91 `int icc_getICCRaderStatus ( OUT BYTE * status )`

Get Reader Status

Returns the reader status

#### Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

#### Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.13.4.92 `int icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

12.13.4.93 `int icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

#### Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.94 int msr\_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.13.4.95 void msr\_registerCallBk ( pMSR\_callBack pMSRf )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

#### 12.13.4.96 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

#### 12.13.4.97 int msr\_startMSRSwipe ( IN int \_timeout )

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

##### Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

#### 12.13.4.98 void parseMSRData ( IN BYTE \* resData, IN int resLen, IN\_OUT IDTMSRData \* cardData )

Parser the MSR data from the buffer into IDTMSTData structure

##### Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of resData
<i>cardData</i>	the parser result with IDTMSTData structure

#### 12.13.4.99 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

#### 12.13.4.100 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

#### 12.13.4.101 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

## 12.13.4.102 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.13.4.103 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibraryPath</i>	The absolute path to ID TECH's libraries.
----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.14 Source\_C/libIDT\_VP3300\_COM.h File Reference

VP3300 COM API.

```
#include "IDTDef.h"
```

### Macros

- #define [IN](#)
- #define [OUT](#)
- #define [IN\\_OUT](#)

### Typedefs

- typedef void(\* [pMessageHotplug](#) )(int, int)
- typedef void(\* [pSendDataLog](#) )(unsigned char \*, int)
- typedef void(\* [pReadDataLog](#) )(unsigned char \*, int)
- typedef void(\* [pEMV\\_callBack](#) )(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)
- typedef void(\* [pMSR\\_callBack](#) )(int, IDTMSRData)
- typedef void(\* [pMSR\\_callBackp](#) )(int, IDTMSRData \*)
- typedef void(\* [pPIN\\_callBack](#) )(int, IDTPINData \*)
- typedef void(\* [pCMR\\_callBack](#) )(int, IDTCMRData \*)
- typedef void(\* [pCSFS\\_callBack](#) )(BYTE status)
- typedef void(\* [ftpComm\\_callBack](#) )(int, int, int)
- typedef void(\* [httpComm\\_callBack](#) )(BYTE \*, int)
- typedef void(\* [v4Comm\\_callBack](#) )(BYTE, BYTE, BYTE \*, int)

## Functions

- void [registerHotplugCallBk](#) (pMessageHotplug pMsgHotplug)
- void [registerLogCallBk](#) (pSendDataLog pFSend, [pReadDataLog](#) pFRead)
- void [device\\_registerRKICallBk](#) (pRKI\_callBack pRKIf)
- void [emv\\_registerCallBk](#) (pEMV\_callBack pEMVf)
- void [msr\\_registerCallBk](#) (pMSR\_callBack pMSRf)
- void [msr\\_registerCallBkp](#) (pMSR\_callBack pMSRf)
- void [ctls\\_registerCallBk](#) (pMSR\_callBack pCTLSf)
- void [ctls\\_registerCallBkp](#) (pMSR\_callBack pCTLSf)
- void [pin\\_registerCallBk](#) (pPIN\_callBack pPINf)
- void [device\\_registerCameraCallBk](#) (pCMR\_callBack pCMRf)
- void [device\\_registerCardStatusFrontSwitchCallBk](#) (pCSFS\_callBack pCSFSf)
- void [comm\\_registerHTTPCallback](#) (httpComm\_callBack cBack)
- void [comm\\_registerV4Callback](#) (v4Comm\_callBack cBack)
- char \* [SDK\\_Version](#) ()
- int [setAbsoluteLibraryPath](#) (const char \*absoluteLibraryPath)
- int [device\\_init](#) ()
- int [device\\_setCurrentDevice](#) (int deviceType)
- int [device\\_isAttached](#) (int deviceType)
- int [device\\_close](#) ()
- void [device\\_getIDGStatusCodeString](#) (IN int returnCode, OUT char \*despcrition)
- int [device\\_isConnected](#) ()
- int [device\\_getFirmwareVersion](#) (OUT char \*firmwareVersion)
- int [device\\_getFirmwareVersion\\_Len](#) (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int [device\\_pingDevice](#) ()
- int [device\\_controlUserInterface](#) (IN BYTE \*values)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [device\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- void [device\\_setTransactionExponent](#) (int exponent)
- int [device\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [device\\_cancelTransaction](#) ()
- int [device\\_getRTCDateTime](#) (IN BYTE \*dateTime, IN\_OUT int \*dateTimeLen)
- int [device\\_setRTCDateTime](#) (IN BYTE \*dateTime, IN int dateTimeLen)
- int [device\\_startRKI](#) ()
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()



- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- void [emv\\_allowFallback](#) (IN int allow)
- void [emv\\_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv\\_setAutoCompleteTransaction](#) (IN int complete)
- int [emv\\_getAutoAuthenticateTransaction](#) ()
- int [emv\\_getAutoCompleteTransaction](#) ()
- void [emv\\_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen)
- int [emv\\_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()
- int [emv\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [emv\\_retrieveCRL](#) (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int [emv\\_setCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeAllCRL](#) ()
- int [icc\\_getICCRReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [msr\\_cancelMSRSwipe](#) ()
- int [msr\\_startMSRSwipe](#) (IN int \_timeout)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.14.1 Detailed Description

VP3300 COM API. VP3300 COM Global API methods.

### 12.14.2 Macro Definition Documentation

#### 12.14.2.1 #define IN

INPUT parameter.

#### 12.14.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.14.2.3 #define OUT

OUTPUT parameter.

### 12.14.3 Typedef Documentation

#### 12.14.3.1 typedef void(\* ftpComm\_callback)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.14.3.2 typedef void(\* httpComm\_callback)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

#### 12.14.3.3 typedef void(\* pCMR\_callback)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

#### 12.14.3.4 typedef void(\* pCSFS\_callback)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

#### 12.14.3.5 typedef void(\* pEMV\_callback)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

**12.14.3.6 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.14.3.7 typedef void(\* pMSR\_callBack)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.14.3.8 typedef void(\* pMSR\_callBackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callBack

**12.14.3.9 typedef void(\* pPIN\_callBack)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.14.3.10 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

**12.14.3.11 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

**12.14.3.12 typedef void(\* v4Comm\_callBack)(BYTE, BYTE, BYTE \*, int)**

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

**12.14.4 Function Documentation****12.14.4.1 void comm\_registerHTTPCallback ( httpComm\_callBack cBack )**

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

12.14.4.2 void comm\_registerV4Callback ( v4Comm\_callBack *cBack* )

Register External V4 Protocol commands Callback

## Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

12.14.4.3 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.14.4.4 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

12.14.4.5 int ctls\_activateTransaction ( IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using [device\\_setMerchantRecord](#), then container tag FFEE06 must be sent as part of the additional tags parameter of [ctls\\_startTransaction](#). Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

#### 12.14.4.6 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.7 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

## Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.8 `int ctls_getConfigurationGroup ( IN int group, OUT BYTE * tlv, OUT int * tlvLen )`

## Get Configuration Group

Retrieves the Configuration for the specified Group.

## Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.9 `void ctls_registerCallBk ( pMSR_callBack pCTLSf )`

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.14.4.10 `void ctls_registerCallBkp ( pMSR_callBackp pCTLSf )`

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.14.4.11 `int ctls_removeAllApplicationData ( )`

## Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.12 `int ctls_removeAllCAPK ( )`

## Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.13 `int ctls_removeApplicationData ( IN BYTE * AID, IN int AIDLen )`

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.14 `int ctls_removeCAPK ( IN BYTE * capk, IN int capkLen )`

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.15 `int ctls_removeConfigurationGroup ( int group )`

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

## Parameters

<i>group</i>	Configuration Group
--------------	---------------------

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.14.4.16 `int ctls_retrieveAIDList ( OUT BYTE * AIDList, IN_OUT int * AIDListLen )`

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.17 `int ctls_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.



## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.18 int ctls\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	the length of key data buffer <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.19 int ctls\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

## Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.20 int ctls\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

## Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.21 int ctls\_setApplicationData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

## Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.22 int ctls\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

## Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.14.4.23 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.14.4.24 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.14.4.25 `int ctls_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal

- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

12.14.4.26 `int device_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9-C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU

- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.14.4.27 int device\_cancelTransaction ( )

Disable Transaction Cancel Transaction request.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.28 int device\_close ( )

Close the device

##### Returns

RETURN\_CODE: 0: success, 0x0A: failed

#### 12.14.4.29 int device\_controlUserInterface ( IN BYTE \* values )

Control User Interface

Controls the User Interface: Display, Beep, LED

```

@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On

```

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

### 12.14.4.30 int device\_enablePassThrough ( int *enablePassThrough* )

#### Start Remote Key Injection

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

#### Enable Pass Through

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

#### Parameters

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

### 12.14.4.31 int device\_getCurrentDeviceType ( )

#### Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.14.4.32 `int device_getFirmwareVersion ( OUT char * firmwareVersion )`

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)()

12.14.4.33 `int device_getFirmwareVersion_Len ( OUT char * firmwareVersion, IN_OUT int * firmwareVersionLen )`

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)()

12.14.4.34 `void device_getIDGStatusCodeString ( IN int returnCode, OUT char * despcrition )`

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";



- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

#### 12.14.4.35 int device\_getMerchantRecord ( IN int *index*, OUT BYTE \* *record* )

DEPRECATED : please use device\_getMerchantRecord\_Len(IN int index, OUT BYTE \* record, IN\_OUT int \*recordLen)

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

12.14.4.36 `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

## Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

12.14.4.37 `int device_getRTCDateTime ( IN BYTE * dateTime, IN_OUT int * dateTimeLen )`

get RTC date and time of the device

## Parameters

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

## Returns

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

## See Also

ErrorCode

#### 12.14.4.38 int device\_getSDKWaitTime ( )

Get SDK Wait Time

Get the SDK wait time for transactions

##### Returns

SDK wait time in seconds

#### 12.14.4.39 int device\_getThreadStackSize ( )

Get Thread Stack Size

Get the stack size setting for newly created threads

##### Returns

Thread Stack Size

#### 12.14.4.40 int device\_getTransactionResults ( IDTMSRData \* *cardData* )

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

##### Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

##### Returns

success or error code. Values can be parsed with device\_getResponseCodeString

##### See Also

ErrorCode

#### 12.14.4.41 int device\_init ( )

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.42 int device\_isAttached ( int *deviceType* )

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

## 12.14.4.43 int device\_isConnected ( )

Check the device connected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

## 12.14.4.44 int device\_pingDevice ( )

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.14.4.45 void device\_registerCameraCallBk ( pCMR\_callback pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.14.4.46 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callback pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.14.4.47 void device\_registerRKICallBk ( pRKI\_callback pRKIf )

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

## 12.14.4.48 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

**Parameters**

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.14.4.49 int device\_setBurstMode ( IN BYTE mode )****Send Burst Mode**

Sets the burst mode for the device.

**Parameters**

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

**See Also**

[ErrorCode](#)

**12.14.4.50 int device\_setCurrentDevice ( int deviceType )**

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to  <pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };           </pre>
-------------------	--

## Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.14.4.51 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.52 int device\_setPollMode ( IN BYTE *mode* )

Set Poll Mode

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

## Parameters

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.53 int device\_setRTCDateTime ( IN BYTE \* *dateTime*, IN int *dateTimeLen* )

set RTC date and time of the device

## Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

## Returns

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

## See Also

[ErrorCode](#)

12.14.4.54 void device\_setSDKWaitTime ( int *waitTime* )

Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

12.14.4.55 void device\_setThreadStackSize ( int *threadSize* )

Set Thread Stack Size

Set the stack size setting for newly created threads

12.14.4.56 void device\_setTransactionExponent ( int *exponent* )

Sets the transaction exponent to be used with [device\\_startTransaction](#). Default value is 2

## Parameters

<i>exponent, The</i>	exponent to use when calling <a href="#">device_startTransaction</a>
----------------------	--

## 12.14.4.57 int device\_startRKI ( )

Start remote key injection.



**Returns**

success or error code.

**See Also**

ErrorCode

**12.14.4.58** int device\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *\_timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

**Start device Transaction Request**

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

**Returns**

success or error code. Values can be parsed with device\_getResponseCodeString

**See Also**

ErrorCode Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

**12.14.4.59** int emv\_activateTransaction ( IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

**Start EMV Transaction Request**

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

**Parameters**

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**12.14.4.60 void emv\_allowFallback ( IN int allow )**

Allow fallback for EMV transactions. Default is TRUE

**Parameters**

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

**12.14.4.61 int emv\_authenticateTransaction ( IN BYTE \* updatedTLV, IN int updatedTLVLen )****Authenticate EMV Transaction Request**

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

**Parameters**

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

**12.14.4.62 int emv\_authenticateTransactionWithTimeout ( IN int timeout, IN BYTE \* updatedTLV, IN int updatedTLVLen )****Authenticate EMV Transaction Request with Timeout**

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.14.4.63 `int emv_cancelTransaction ( )`

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.14.4.64 `int emv_completeTransaction ( IN int commError, IN BYTE * authCode, IN int authCodeLen, IN BYTE * iad, IN int iadLen, IN BYTE * tlvScripts, IN int tlvScriptsLen, IN BYTE * tlv, IN int tlvLen )`

Complete EMV Transaction Request

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

## Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

**12.14.4.65 int emv\_getAutoAuthenticateTransaction ( )**

Gets auto authenticate value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

**12.14.4.66 int emv\_getAutoCompleteTransaction ( )**

Gets auto complete value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

**12.14.4.67 void emv\_registerCallBk ( pEMV\_callback pEMVf )**

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

**12.14.4.68 int emv\_removeAllApplicationData ( )**

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.14.4.69 int emv\_removeAllCAPK ( )**

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.14.4.70 int emv\_removeAllCRL ( )**

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.14.4.71 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

## Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.72 `int emv_removeCAPK ( IN BYTE * capk, IN int capkLen )`

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

## Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.73 `int emv_removeCRL ( IN BYTE * list, IN int lsLen )`

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.14.4.74 `int emv_retrieveAIDList ( OUT BYTE * AIDList, IN_OUT int * AIDListLen )`

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.75 `int emv_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.76 `int emv_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>

<i>keyLen</i>	the length of key data buffer
	•

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.77 int emv\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keyLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keyLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.14.4.78 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *listLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

**Parameters**

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>listLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.14.4.79 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.80 `int emv_setApplicationData ( IN BYTE * name, IN int nameLen, IN BYTE * tlv, IN int tlvLen )`

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data



## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.14.4.81 int emv\_setApplicationDataTLV ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.14.4.82 void emv\_setAutoAuthenticateTransaction ( IN int *authenticate* )

Enables authenticate for EMV transactions. If a emv\_startTransaction results in code 0x0010 (start transaction success), then emv\_authenticateTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

12.14.4.83 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

12.14.4.84 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.85 int emv\_setCRL ( IN BYTE \* *list*, IN int *IsLen* )

## Set Certificate Revocation List

Sets the CRL

## Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>IsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.14.4.86 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

## 12.14.4.87 int emv\_setTerminalMajorConfiguration ( IN int configuration )

Sets the terminal major configuration in ICS .

## Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.14.4.88 void emv\_setTransactionParameters ( IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \* tags, IN int tagsLen )

## Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

<i>tagsLen</i>	the length of tags
----------------	--------------------

**12.14.4.89** `int emv_startTransaction ( IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

#### Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString` >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**12.14.4.90** `int icc_exchangeAPDU ( IN BYTE * c_APDU, IN int cLen, OUT BYTE * reData, IN_OUT int * reLen )`

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

#### Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

#### Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

**12.14.4.91** `int icc_getICCRReaderStatus ( OUT BYTE * status )`

Get Reader Status

Returns the reader status

## Parameters

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.14.4.92 int `icc_powerOffICC ( )`

Power Off ICC

Powers down the ICC

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

12.14.4.93 int `icc_powerOnICC ( OUT BYTE * ATR, IN_OUT int * inLen )`

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

## Parameters

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.94 int `msr_cancelMSRSwipe ( )`

Disable MSR Swipe Cancels MSR swipe request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.14.4.95 void `msr_registerCallBk ( pMSR_callBack pMSRf )`

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.14.4.96 void `msr_registerCallBkp ( pMSR_callBackp pMSRf )`

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

12.14.4.97 int `msr_startMSRSwipe ( IN int _timeout )`

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `deviceDelegate::swipeMSRData:()`

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.14.4.98 void parseMSRData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTMSRData \* *cardData* )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of <i>resData</i>
<i>cardData</i>	the parser result with IDTMSTData structure

12.14.4.99 void pin\_registerCallBk ( pPIN\_callBack *pPINf* )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.14.4.100 void registerHotplugCallBk ( pMessageHotplug *pMsgHotplug* )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.14.4.101 void registerLogCallBk ( pSendDataLog *pFSend*, pReadDataLog *pFRead* )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.14.4.102 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.14.4.103 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.15 Source\_C/libIDT\_VP3300\_USB.h File Reference

VP3300 USB API.

```
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callBack )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callBack )(int, IDTMSRData)`
- `typedef void(* pMSR_callBackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callBack )(int, IDTPINData *)`
- `typedef void(* pCMR_callBack )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callBack )(BYTE status)`
- `typedef void(* ftpComm_callBack )(int, int, int)`
- `typedef void(* httpComm_callBack )(BYTE *, int)`
- `typedef void(* v4Comm_callBack )(BYTE, BYTE, BYTE *, int)`

### Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void device_registerRKICallBk (pRKI_callBack pRKIf)`
- `void emv_registerCallBk (pEMV_callBack pEMVf)`
- `void msr_registerCallBk (pMSR_callBack pMSRf)`
- `void msr_registerCallBkp (pMSR_callBackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callBack pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callBackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callBack pPINf)`
- `void device_registerCameraCallBk (pCMR_callBack pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callBack pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callBack cBack)`
- `void comm_registerV4Callback (v4Comm_callBack cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int device_setCurrentDevice (int deviceType)`
- `int device_isAttached (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)`
- `int device_isConnected ()`
- `int device_getFirmwareVersion (OUT char *firmwareVersion)`
- `int device_getFirmwareVersion_Len (OUT char *firmwareVersion, IN_OUT int *firmwareVersionLen)`

- int [device\\_pingDevice](#) ()
- int [device\\_controlUserInterface](#) (IN BYTE \*values)
- int [device\\_getCurrentDeviceType](#) ()
- int [device\\_SendDataCommandNEO](#) (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int [device\\_enablePassThrough](#) (int enablePassThrough)
- int [device\\_setBurstMode](#) (IN BYTE mode)
- int [device\\_setPollMode](#) (IN BYTE mode)
- int [device\\_setMerchantRecord](#) (int index, int enabled, char \*merchantID, char \*merchantURL)
- int [device\\_getMerchantRecord](#) (IN int index, OUT BYTE \*record)
- int [device\\_getMerchantRecord\\_Len](#) (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int [device\\_getTransactionResults](#) (IDTMSRData \*cardData)
- int [device\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- void [device\\_setTransactionExponent](#) (int exponent)
- int [device\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [device\\_cancelTransaction](#) ()
- int [device\\_getRTCDateTime](#) (IN BYTE \*dateTime, IN\_OUT int \*dateTimeLen)
- int [device\\_setRTCDateTime](#) (IN BYTE \*dateTime, IN int dateTimeLen)
- int [device\\_startRKI](#) ()
- int [device\\_getSDKWaitTime](#) ()
- void [device\\_setSDKWaitTime](#) (int waitTime)
- int [device\\_getThreadStackSize](#) ()
- void [device\\_setThreadStackSize](#) (int threadSize)
- int [config\\_getSerialNumber](#) (OUT char \*sNumber)
- int [config\\_getSerialNumber\\_Len](#) (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int [ctls\\_startTransaction](#) (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_activateTransaction](#) (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int [ctls\\_cancelTransaction](#) ()
- int [ctls\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setApplicationData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [ctls\\_removeAllApplicationData](#) ()
- int [ctls\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [ctls\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [ctls\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [ctls\\_removeAllCAPK](#) ()
- int [ctls\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [ctls\\_setConfigurationGroup](#) (IN BYTE \*tlv, IN int tlvLen)
- int [ctls\\_getConfigurationGroup](#) (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int [ctls\\_getAllConfigurationGroups](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [ctls\\_removeConfigurationGroup](#) (int group)
- void [emv\\_allowFallback](#) (IN int allow)
- void [emv\\_setAutoAuthenticateTransaction](#) (IN int authenticate)
- void [emv\\_setAutoCompleteTransaction](#) (IN int complete)
- int [emv\\_getAutoAuthenticateTransaction](#) ()
- int [emv\\_getAutoCompleteTransaction](#) ()
- void [emv\\_setTransactionParameters](#) (IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen)
- int [emv\\_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)



- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setTerminalMajorConfiguration](#) (IN int configuration)
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()
- int [emv\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [emv\\_retrieveCRL](#) (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int [emv\\_setCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeCRL](#) (IN BYTE \*list, IN int lssLen)
- int [emv\\_removeAllCRL](#) ()
- int [icc\\_getICReaderStatus](#) (OUT BYTE \*status)
- int [icc\\_powerOnICC](#) (OUT BYTE \*ATR, IN\_OUT int \*inLen)
- int [icc\\_powerOffICC](#) ()
- int [icc\\_exchangeAPDU](#) (IN BYTE \*c\_APDU, IN int cLen, OUT BYTE \*reData, IN\_OUT int \*reLen)
- int [msr\\_cancelMSRSwipe](#) ()
- int [msr\\_startMSRSwipe](#) (IN int \_timeout)
- void [parseMSRData](#) (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)

### 12.15.1 Detailed Description

VP3300 USB API. VP3300 USB Global API methods.

### 12.15.2 Macro Definition Documentation

#### 12.15.2.1 #define IN

INPUT parameter.

#### 12.15.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

#### 12.15.2.3 #define OUT

OUTPUT parameter.

### 12.15.3 Typedef Documentation

#### 12.15.3.1 typedef void(\* ftpComm\_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

#### 12.15.3.2 typedef void(\* httpComm\_callBack)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

#### 12.15.3.3 typedef void(\* pCMR\_callBack)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

#### 12.15.3.4 typedef void(\* pCSFS\_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

#### 12.15.3.5 typedef void(\* pEMV\_callBack)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

#### 12.15.3.6 typedef void(\* pMessageHotplug)(int, int)

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

#### 12.15.3.7 typedef void(\* pMSR\_callBack)(int, IDTMSRData)

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

#### 12.15.3.8 typedef void(\* pMSR\_callBackp)(int, IDTMSRData \*)

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callBack

**12.15.3.9** `typedef void(* pPIN_callBack)(int, IDTPINData *)`

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the `pin_registerCallBk`,

**12.15.3.10** `typedef void(* pReadDataLog)(unsigned char *, int)`

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the `registerLogCallBk`,

**12.15.3.11** `typedef void(* pSendDataLog)(unsigned char *, int)`

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the `registerLogCallBk`,

**12.15.3.12** `typedef void(* v4Comm_callBack)(BYTE, BYTE, BYTE *, int)`

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the `comm_registerV4Callback`, Data callback will contain command, sub-command, and data from V4 packet

## 12.15.4 Function Documentation

**12.15.4.1** `void comm_registerHTTPCallback ( httpComm_callBack cBack )`

Register Comm HTTP Async Callback

Parameters

<i>cBack</i>	- HTTP Comm callback
--------------	----------------------

**12.15.4.2** `void comm_registerV4Callback ( v4Comm_callBack cBack )`

Register External V4 Protocol commands Callback

Parameters

<i>cBack</i>	- V4 Protocol Comm callback
--------------	-----------------------------

**12.15.4.3** `int config_getSerialNumber ( OUT char * sNumber )`

DEPRECATED : please use [config\\_getSerialNumber\\_Len\(OUT char\\* sNumber, IN\\_OUT int \\*sNumberLen\)](#)

Polls device for Serial Number

Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.15.4.4 `int config_getSerialNumber_Len ( OUT char * sNumber, IN_OUT int * sNumberLen )`

Polls device for Serial Number

## Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

12.15.4.5 `int ctls_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal

- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.15.4.6 int ctls\_cancelTransaction ( )

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.7 int ctls\_getAllConfigurationGroups ( OUT BYTE \* tlv, IN\_OUT int \* tlvLen )

Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.8 int ctls\_getConfigurationGroup ( IN int group, OUT BYTE \* tlv, OUT int \* tlvLen )

Get Configuration Group

Retrieves the Configuration for the specified Group.

Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.9 void ctls\_registerCallBk ( pMSR\_callBack pCTLSf )**

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

**12.15.4.10 void ctls\_registerCallBkp ( pMSR\_callBackp pCTLSf )**

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

**12.15.4.11 int ctls\_removeAllApplicationData ( )**

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.12 int ctls\_removeAllCAPK ( )**

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.13 int ctls\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.14 int ctls\_removeCAPK ( IN BYTE \* capk, IN int capkLen )**

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.15 int ctls\_removeConfigurationGroup ( int *group* )

##### Remove Configuration Group

Removes the Configuration as specified by the Group. Must not be group 0

##### Parameters

<i>group</i>	Configuration Group
--------------	---------------------

##### Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

#### 12.15.4.16 int ctls\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

##### Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

##### Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.17 int ctls\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

##### Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

##### Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.18 int ctls\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

##### Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

##### Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
-------------	---



<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.19 int ctls\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.20 int ctls\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.21 int ctls\_setApplicationData ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

Parameters

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.22 int ctls\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.23 int ctls\_setConfigurationGroup ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (FFE4). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.24 int ctls\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.15.4.25 int ctls\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using [device\\_setMerchantRecord](#), then container tag FFEE06 must be sent as part of the additional tags parameter of [ctls\\_startTransaction](#). Tag FFEE06 must contain tag 9F26 and 9F22, and can

optionanally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Applicaiton Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
  - - - 0 = Payment Terminal
  - - - 1 = Transit Terminal
  - - - 2 = Access Terminal
  - - - 3 = Wireless Handoff Terminal
  - - - 4 = App Handoff Terminal
  - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

12.15.4.26 `int device_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount)and9-C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
    - - - 0 = Payment Terminal
    - - - 1 = Transit Terminal
    - - - 2 = Access Terminal
    - - - 3 = Wireless Handoff Terminal
    - - - 4 = App Handoff Terminal
    - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

**12.15.4.27 int device\_cancelTransaction ( )**

Disable Transaction Cancel Transaction request.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.28 int device\_close ( )**

Close the device

**Returns**

RETURN\_CODE: 0: success, 0x0A: failed

**12.15.4.29 int device\_controlUserInterface ( IN BYTE \* values )****Control User Interface**

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms
Byte[2] = LED Number
- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs
Byte[3] = LED Status
- 00h: LED Off
- 01h: LED On
```

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.30 int device\_enablePassThrough ( int enablePassThrough )****Start Remote Key Injection**

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

```
@return RETURN_CODE: Values can be parsed with device_getIDGStatusCodeString
```

**Enable Pass Through**

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

## Parameters

<i>enablePass-Through</i>	1 = pass through ON, 0 = pass through OFF
---------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.15.4.31 int device\_getCurrentDeviceType ( )

Get current active device type

## Returns

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.15.4.32 int device\_getFirmwareVersion ( OUT char \* *firmwareVersion* )

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.33 int device\_getFirmwareVersion\_Len ( OUT char \* *firmwareVersion*, IN\_OUT int \* *firmwareVersionLen* )

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.34 void device\_getIDGStatusCodeString ( IN int *returnCode*, OUT char \* *despcriton* )

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";
  - 02: " Unknown Command";
  - 03: " Unknown Sub-Command";
  - 04: " CRC Error in Frame";
  - 05: " Incorrect Parameter";
  - 06: " Parameter Not Supported";
  - 07: " Mal-formatted Data";
  - 08: " Timeout";
  - 0A: " Failed / NACK";
  - 0B: " Command not Allowed";
  - 0C: " Sub-Command not Allowed";
  - 0D: " Buffer Overflow (Data Length too large for reader buffer)";
  - 0E: " User Interface Event";
  - 10: " Need clear firmware(apply in boot loader only)";
  - 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
  - 12: " Secure interface is not functional or is in an intermediate state.";
  - 13: " Data field is not mod 8";
  - 14: " Pad 0x80 not found where expected";
  - 15: " Specified key type is invalid";
  - 16: " Could not retrieve key from the SAM (InitSecureComm)";
  - 17: " Hash code problem";
  - 18: " Could not store the key into the SAM (InstallKey)";
  - 19: " Frame is too large";
  - 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
  - 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
  - 1C: " Problem encoding APDU Module-Specific Status Codes ";
  - 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
  - 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
  - 50: " Auto-Switch OK";
  - 51: " Auto-Switch failed";
  - 70: " Antenna Error 80h Use another card";
  - 81: " Insert or swipe card";
  - 90: " Data encryption Key does not exist";
  - 91: " Data encryption Key KSN exhausted";



12.15.4.35 `int device_getMerchantRecord ( IN int index, OUT BYTE * record )`

DEPRECATED : please use `device_getMerchantRecord_Len`(IN int *index*, OUT BYTE \* *record*, IN\_OUT int \**recordLen*)

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i> ;	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

See Also

[ErrorCode](#)

12.15.4.36 `int device_getMerchantRecord_Len ( IN int index, OUT BYTE * record, IN_OUT int * recordLen )`

Get Merchant Record

Gets the merchant record for the device.

Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

See Also

[ErrorCode](#)

12.15.4.37 `int device_getRTCDateTime ( IN BYTE * dateTime, IN_OUT int * dateTimeLen )`

get RTC date and time of the device

Parameters

<i>dateTime</i>	<dateTime data>=""> is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	return 6 bytes if successful

Returns

success or error code. Values can be parsed with `device_getResponseCodeString`

See Also

[ErrorCode](#)

**12.15.4.38 int device\_getSDKWaitTime ( )**

Get SDK Wait Time

Get the SDK wait time for transactions

**Returns**

SDK wait time in seconds

**12.15.4.39 int device\_getThreadStackSize ( )**

Get Thread Stack Size

Get the stack size setting for newly created threads

**Returns**

Thread Stack Size

**12.15.4.40 int device\_getTransactionResults ( IDTMSRData \* *cardData* )**

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

**Parameters**

<i>cardData</i>	The transaction results
-----------------	-------------------------

**Returns**

success or error code. Values can be parsed with `device_getResponseCodeString`

**See Also**

`ErrorCode`

**12.15.4.41 int device\_init ( )**

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by [device\\_isConnected\(\)](#).

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.42 int device\_isAttached ( int *deviceType* )**

Check if the device is attached to the USB port The function [device\\_init\(\)](#) must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

## 12.15.4.43 int device\_isConnected ( )

Check the device connected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

## 12.15.4.44 int device\_pingDevice ( )

Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.15.4.45 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.15.4.46 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.15.4.47 void device\_registerRKICallBk ( pRKI\_callBack pRKIf )

To register the RKI callback function to get the RKI status. (Pass NULL to disable the callback.)

## 12.15.4.48 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.

<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

**Parameters**

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.49 int device\_setBurstMode ( IN BYTE mode )

Send Burst Mode

Sets the burst mode for the device.

**Parameters**

<i>mode</i>	0 = OFF, 1 = Always On, 2 = Auto Exit
-------------	---------------------------------------

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

## See Also

[ErrorCode](#)

12.15.4.50 int device\_setCurrentDevice ( int *deviceType* )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

## Parameters

<i>deviceType</i>	Device to connect to
	<pre> enum DEVICE_TYPE {     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 }; </pre>

## Returns

RETURN\_CODE: 1: success, 0: failed

12.15.4.51 int device\_setMerchantRecord ( int *index*, int *enabled*, char \* *merchantID*, char \* *merchantURL* )

Set Merchant Record Sets the merchant record for ApplePay VAS

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.52 int device\_setPollMode ( IN BYTE mode )****Set Poll Mode**

Sets the poll mode for the device. Auto Poll keeps reader active, Poll On Demand only polls when requested by terminal

**Parameters**

<i>mode</i>	0 = Auto Poll, 1 = Poll On Demand
-------------	-----------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.53 int device\_setRTCDateTime ( IN BYTE \* dateTime, IN int dateTimeLen )**

set RTC date and time of the device

**Parameters**

<i>dateTime</i>	<dateTime data>="" is: 6 byte data for YYMMDDHHMMSS in hex. For example 0x171003102547 stands for 2017 Oct 3rd 10:25:47
<i>dateTimeLen</i>	should be always 6 bytes

**Returns**

success or error code. Values can be parsed with [device\\_getResponseCodeString](#)

**See Also**

[ErrorCode](#)

**12.15.4.54 void device\_setSDKWaitTime ( int waitTime )****Set SDK Wait Time**

Set the SDK wait time for transactions

**Parameters**

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

**12.15.4.55 void device\_setThreadStackSize ( int threadSize )****Set Thread Stack Size**

Set the stack size setting for newly created threads

**12.15.4.56 void device\_setTransactionExponent ( int exponent )**

Sets the transaction exponent to be used with [device\\_startTransaction](#). Default value is 2

## Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

## 12.15.4.57 int device\_startRKI ( )

Start remote key injection.

## Returns

success or error code.

## See Also

ErrorCode

12.15.4.58 int device\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *\_timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

Start device Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

success or error code. Values can be parsed with device\_getResponseCodeString

## See Also

ErrorCode Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

12.15.4.59 int emv\_activateTransaction ( IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x0000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F369F9F37

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## 12.15.4.60 void emv\_allowFallback ( IN int allow )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

## 12.15.4.61 int emv\_authenticateTransaction ( IN BYTE \* updatedTLV, IN int updatedTLVLen )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37</li> </ul>
-------------------	---



<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.15.4.62 int emv\_authenticateTransactionWithTimeout ( IN int *timeout*, IN BYTE \* *updatedTLV*, IN int *updatedTLVLen* )

**Authenticate EMV Transaction Request with Timeout**

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

**Parameters**

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

**Returns**

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString

12.15.4.63 int emv\_cancelTransaction ( )

**Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.15.4.64 int emv\_completeTransaction ( IN int *commError*, IN BYTE \* *authCode*, IN int *authCodeLen*, IN BYTE \* *iad*, IN int *iadLen*, IN BYTE \* *tlvScripts*, IN int *tlvScriptsLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

**Complete EMV Transaction Request**

Completes the EMV transaction for an ICC card when online authorization request is received from emv\_authenticateTransaction

The tags will be returned in the callback routine.

**Parameters**

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

#### 12.15.4.65 int emv\_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

#### 12.15.4.66 int emv\_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

**Returns**

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

#### 12.15.4.67 void emv\_registerCallBk ( pEMV\_callBack pEMVf )

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

#### 12.15.4.68 int emv\_removeAllApplicationData ( )

Remove All Application Data

Removes all the Application Data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.69 int emv\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.70 int emv\_removeAllCRL ( )

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.15.4.71 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.72 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.15.4.73 int emv\_removeCRL ( IN BYTE \* list, IN int lssLen )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

#### 12.15.4.74 int emv\_retrieveAIDList ( OUT BYTE \* AIDList, IN\_OUT int \* AIDListLen )

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.75 `int emv_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

## Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.76 `int emv_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	<p>the length of key data buffer</p> <ul style="list-style-type: none"> <li>•</li> </ul>

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.77 int emv\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keysLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

## Parameters

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.78 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *lssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>lssLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.15.4.79 int emv\_retrieveTerminalData ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by [ctls\\_getConfigurationGroup\(0\)](#).

## Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.80 int emv\_setApplicationData ( IN BYTE \* *name*, IN int *nameLen*, IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.15.4.81 int emv\_setApplicationDataTLV ( IN BYTE \* *tlv*, IN int *tlvLen* )

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.15.4.82 void emv\_setAutoAuthenticateTransaction ( IN int *authenticate* )

Enables authenticate for EMV transactions. If a emv\_startTransaction results in code 0x0010 (start transaction success), then emv\_authenticateTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

12.15.4.83 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

12.15.4.84 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.15.4.85 int emv\_setCRL ( IN BYTE \* *list*, IN int *IsLen* )

## Set Certificate Revocation List

Sets the CRL

## Parameters

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>IsLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.15.4.86 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

## 12.15.4.87 int emv\_setTerminalMajorConfiguration ( IN int configuration )

Sets the terminal major configuration in ICS .

## Parameters

<i>configuration</i>	A configuration value, range 1-23 <ul style="list-style-type: none"> <li>• 1 = 1C</li> <li>• 2 = 2C</li> <li>• 3 = 3C</li> <li>• 4 = 4C</li> <li>• 5 = 5C ...</li> <li>• 23 = 23C</li> </ul>
----------------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.15.4.88 void emv\_setTransactionParameters ( IN double amount, IN double amtOther, IN int type, IN int timeout, IN BYTE \* tags, IN int tagsLen )

## Set EMV Transaction Parameters

Set the parameters to be used on EMV transactions for an ICC card when Auto Poll is on

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02)
<i>amtOther</i>	Other amount value, if any (tag value 9F03)
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request (Maximum Length = 500 bytes). Passed as a string. Example, tag 9F0C with amount 0x000000000100 would be "9F0C06000000000100" If tags 9F02 (amount),9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method. Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37



<i>tagsLen</i>	the length of tags
----------------	--------------------

**12.15.4.89** int emv\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *exponent*, IN int *type*, IN int *timeout*, IN BYTE \* *tags*, IN int *tagsLen*, IN int *forceOnline* )

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

#### Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**12.15.4.90** int icc\_exchangeAPDU ( IN BYTE \* *c\_APDU*, IN int *cLen*, OUT BYTE \* *reData*, IN\_OUT int \* *reLen* )

Exchange APDU with plain text For Non-SRED Augusta Only

Sends an APDU packet to the ICC. If successful, response is the APDU data in response parameter.

#### Parameters

<i>c_APDU</i>	APDU data packet
<i>cLen</i>	APDU data packet length
<i>reData</i>	Unencrypted APDU response
<i>reLen</i>	Unencrypted APDU response data length

#### Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

**12.15.4.91** int icc\_getICCRReaderStatus ( OUT BYTE \* *status* )

Get Reader Status

Returns the reader status

**Parameters**

<i>status</i>	Pointer that will return with the ICCReaderStatus results. bit 0: 0 = ICC Power Not Ready, 1 = ICC Powered bit 1: 0 = Card not seated, 1 = card seated
---------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString`

**12.15.4.92 int icc\_powerOffICC ( )**

Power Off ICC

Powers down the ICC

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

If Success, empty If Failure, ASCII encoded data of error string

**12.15.4.93 int icc\_powerOnICC ( OUT BYTE \* ATR, IN\_OUT int \* inLen )**

Power On ICC

Power up the currently selected microprocessor card in the ICC reader

**Parameters**

<i>ATR,the</i>	ATR data response when succeeded power on ICC,
<i>inLen,the</i>	length of ATR data

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.94 int msr\_cancelMSRSwipe ( )**

Disable MSR Swipe Cancels MSR swipe request.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.15.4.95 void msr\_registerCallBk ( pMSR\_callBack pMSRf )**

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

**12.15.4.96 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )**

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

**12.15.4.97 int msr\_startMSRSwipe ( IN int \_timeout )**

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to `deviceDelegate::swipeMSRData:()`

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.15.4.98 void parseMSRData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTMSRData \* *cardData* )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of <i>resData</i>
<i>cardData</i>	the parser result with IDTMSTData structure

12.15.4.99 void pin\_registerCallBk ( pPIN\_callBack *pPINf* )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

12.15.4.100 void registerHotplugCallBk ( pMessageHotplug *pMsgHotplug* )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

12.15.4.101 void registerLogCallBk ( pSendDataLog *pFSend*, pReadDataLog *pFRead* )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

12.15.4.102 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

12.15.4.103 int setAbsoluteLibraryPath ( const char \* *absoluteLibraryPath* )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16 Source\_C/libIDT\_VP8800.h File Reference

VP8800 API.

```
#include <stdarg.h>
#include "IDTDef.h"
```

### Macros

- `#define IN`
- `#define OUT`
- `#define IN_OUT`

### Typedefs

- `typedef void(* pMessageHotplug )(int, int)`
- `typedef void(* pSendDataLog )(unsigned char *, int)`
- `typedef void(* pReadDataLog )(unsigned char *, int)`
- `typedef void(* pEMV_callback )(int, int, unsigned char *, int, IDTTransactionData *, EMV_Callback *, int)`
- `typedef void(* pMSR_callback )(int, IDTMSRData)`
- `typedef void(* pMSR_callbackp )(int, IDTMSRData *)`
- `typedef void(* pPIN_callback )(int, IDTPINData *)`
- `typedef void(* pCMR_callback )(int, IDTCMRData *)`
- `typedef void(* pCSFS_callback )(BYTE status)`
- `typedef void(* ftpComm_callback )(int, int, int)`
- `typedef void(* httpComm_callback )(BYTE *, int)`
- `typedef void(* v4Comm_callback )(BYTE, BYTE, BYTE *, int)`
- `typedef void(* pLog_callback )(BYTE, char *)`

### Functions

- `void registerHotplugCallBk (pMessageHotplug pMsgHotplug)`
- `void registerLogCallBk (pSendDataLog pFSend, pReadDataLog pFRead)`
- `void emv_registerCallBk (pEMV_callback pEMVf)`
- `void msr_registerCallBk (pMSR_callback pMSRf)`
- `void msr_registerCallBkp (pMSR_callbackp pMSRf)`
- `void ctls_registerCallBk (pMSR_callback pCTLSf)`
- `void ctls_registerCallBkp (pMSR_callbackp pCTLSf)`
- `void pin_registerCallBk (pPIN_callback pPINf)`
- `void device_registerCameraCallBk (pCMR_callback pCMRf)`
- `void device_registerCardStatusFrontSwitchCallBk (pCSFS_callback pCSFSf)`
- `void comm_registerHTTPCallback (httpComm_callback cBack)`
- `void comm_registerV4Callback (v4Comm_callback cBack)`
- `char * SDK_Version ()`
- `int setAbsoluteLibraryPath (const char *absoluteLibraryPath)`
- `int device_init ()`
- `int device_setCurrentDevice (int deviceType)`
- `int device_close ()`
- `void device_getIDGStatusCodeString (IN int returnCode, OUT char *despcrition)`
- `int device_isConnected ()`
- `int device_isAttached (int deviceType)`
- `int device_startTransaction (IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE *tags, IN int tagsLen)`

- void `device_setTransactionExponent` (int exponent)
- int `device_activateTransaction` (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `device_cancelTransaction` ()
- int `device_getDriveFreeSpace` (OUT int \*free, OUT int \*used)
- int `device_listDirectory` (IN char \*directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char \*directory, IN\_OUT int \*directoryLen)
- int `device_createDirectory` (IN char \*directoryName, IN int directoryNameLen)
- int `device_deleteDirectory` (IN char \*dirName, IN int dirNameLen)
- int `device_transferFile` (IN char \*fileName, IN int fileNameLen, IN BYTE \*file, IN int fileLen)
- int `device_deleteFile` (IN char \*fileName, IN int fileNameLen)
- int `device_getFirmwareVersion` (OUT char \*firmwareVersion)
- int `device_getFirmwareVersion_Len` (OUT char \*firmwareVersion, IN\_OUT int \*firmwareVersionLen)
- int `device_pingDevice` ()
- int `device_controlUserInterface` (IN BYTE \*values)
- int `device_controlIndicator` (IN int indicator, IN int enable)
- int `device_getCurrentDeviceType` ()
- int `device_SendDataCommandNEO` (IN int cmd, IN int subCmd, IN BYTE \*data, IN int dataLen, OUT BYTE \*response, IN\_OUT int \*respLen)
- int `device_enablePassThrough` (int enablePassThrough)
- int `device_enhancedPassthrough` (IN BYTE \*data, IN int dataLen)
- int `device_setMerchantRecord` (int index, int enabled, char \*merchantID, char \*merchantURL)
- int `device_getMerchantRecord` (IN int index, OUT BYTE \*record)
- int `device_getMerchantRecord_Len` (IN int index, OUT BYTE \*record, IN\_OUT int \*recordLen)
- int `device_getTransactionResults` (IDTMSRData \*cardData)
- int `device_calibrateParameters` (BYTE delta)
- int `config_getSerialNumber` (OUT char \*sNumber)
- int `config_getSerialNumber_Len` (OUT char \*sNumber, IN\_OUT int \*sNumberLen)
- int `device_getSDKWaitTime` ()
- void `device_setSDKWaitTime` (int waitTime)
- int `device_getThreadStackSize` ()
- void `device_setThreadStackSize` (int threadSize)
- int `ctls_startTransaction` (IN double amount, IN double amtOther, IN int type, IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `ctls_activateTransaction` (IN const int \_timeout, IN BYTE \*tags, IN int tagsLen)
- int `ctls_cancelTransaction` ()
- int `ctls_retrieveApplicationData` (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `ctls_setApplicationData` (IN BYTE \*tlv, IN int tlvLen)
- int `ctls_removeApplicationData` (IN BYTE \*AID, IN int AIDLen)
- int `ctls_removeAllApplicationData` ()
- int `ctls_retrieveAIDList` (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int `ctls_retrieveTerminalData` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `ctls_setTerminalData` (IN BYTE \*tlv, IN int tlvLen)
- int `ctls_retrieveCAPK` (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int `ctls_setCAPK` (IN BYTE \*capk, IN int capkLen)
- int `ctls_removeCAPK` (IN BYTE \*capk, IN int capkLen)
- int `ctls_removeAllCAPK` ()
- int `ctls_retrieveCAPKList` (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int `ctls_setConfigurationGroup` (IN BYTE \*tlv, IN int tlvLen)
- int `ctls_getConfigurationGroup` (IN int group, OUT BYTE \*tlv, OUT int \*tlvLen)
- int `ctls_getAllConfigurationGroups` (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int `ctls_removeConfigurationGroup` (int group)
- int `ctls_displayOnlineAuthResult` (IN int statusCode, IN BYTE \*TLV, IN int TLVLen)
- void `emv_allowFallback` (IN int allow)
- void `emv_setAutoAuthenticateTransaction` (IN int authenticate)
- void `emv_setAutoCompleteTransaction` (IN int complete)

- int [emv\\_getAutoAuthenticateTransaction](#) ()
- int [emv\\_getAutoCompleteTransaction](#) ()
- int [emv\\_startTransaction](#) (IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_activateTransaction](#) (IN int timeout, IN BYTE \*tags, IN int tagsLen, IN int forceOnline)
- int [emv\\_authenticateTransaction](#) (IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_authenticateTransactionWithTimeout](#) (IN int timeout, IN BYTE \*updatedTLV, IN int updatedTLVLen)
- int [emv\\_completeTransaction](#) (IN int commError, IN BYTE \*authCode, IN int authCodeLen, IN BYTE \*iad, IN int iadLen, IN BYTE \*tlvScripts, IN int tlvScriptsLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_cancelTransaction](#) ()
- int [emv\\_retrieveApplicationData](#) (IN BYTE \*AID, IN int AIDLen, OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setApplicationData](#) (IN BYTE \*name, IN int nameLen, IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_setApplicationDataTLV](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_removeApplicationData](#) (IN BYTE \*AID, IN int AIDLen)
- int [emv\\_removeAllApplicationData](#) ()
- int [emv\\_retrieveAIDList](#) (OUT BYTE \*AIDList, IN\_OUT int \*AIDListLen)
- int [emv\\_retrieveTerminalData](#) (OUT BYTE \*tlv, IN\_OUT int \*tlvLen)
- int [emv\\_setTerminalData](#) (IN BYTE \*tlv, IN int tlvLen)
- int [emv\\_retrieveCAPK](#) (IN BYTE \*capk, IN int capkLen, OUT BYTE \*key, IN\_OUT int \*keyLen)
- int [emv\\_setCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeCAPK](#) (IN BYTE \*capk, IN int capkLen)
- int [emv\\_removeAllCAPK](#) ()
- int [emv\\_retrieveCAPKList](#) (OUT BYTE \*keys, IN\_OUT int \*keysLen)
- int [emv\\_retrieveExceptionList](#) (OUT BYTE \*exceptionList, IN\_OUT int \*exceptionListLen)
- int [emv\\_setException](#) (IN BYTE \*exception, IN int exceptionLen)
- int [emv\\_removeException](#) (IN BYTE \*exception, IN int exceptionLen)
- int [emv\\_removeAllExceptions](#) ()
- int [emv\\_retrieveExceptionLogStatus](#) (OUT BYTE \*exceptionLogStatus, IN\_OUT int \*exceptionLogStatusLen)
- int [emv\\_removeTransactionLog](#) ()
- int [emv\\_retrieveTransactionLogStatus](#) (OUT BYTE \*transactionLogStatus, IN\_OUT int \*transactionLogStatusLen)
- int [emv\\_retrieveTransactionLog](#) (OUT BYTE \*transactionLog, IN\_OUT int \*transactionLogLen, IN\_OUT int \*remainingTransactionLogLen)
- int [emv\\_getEMVKernelVersion](#) (OUT char \*version)
- int [emv\\_getEMVKernelVersion\\_Len](#) (OUT char \*version, IN\_OUT int \*versionLen)
- int [emv\\_getEMVKernelCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- int [emv\\_getEMVConfigurationCheckValue](#) (OUT BYTE \*checkValue, IN\_OUT int \*checkValueLen)
- int [emv\\_retrieveCRL](#) (OUT BYTE \*list, IN\_OUT int \*lssLen)
- int [emv\\_setCRL](#) (IN BYTE \*list, IN int lsLen)
- int [emv\\_removeCRL](#) (IN BYTE \*list, IN int lsLen)
- int [emv\\_removeAllCRL](#) ()
- int [lcd\\_resetInitialState](#) ()
- int [lcd\\_customDisplayMode](#) (IN int enable)
- int [lcd\\_setForeBackColor](#) (IN BYTE \*foreRGB, IN int foreRGBLen, IN BYTE \*backRGB, IN int backRGBLen)
- int [lcd\\_clearDisplay](#) (IN BYTE control)
- int [lcd\\_captureSignature](#) (IN int timeout)
- int [lcd\\_startSlideShow](#) (IN char \*files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen)
- int [lcd\\_cancelSlideShow](#) (OUT BYTE \*statusCode, IN\_OUT int \*statusCodeLen)
- int [lcd\\_setDisplayImage](#) (IN char \*file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int clearScreen)
- int [lcd\\_setBackgroundImage](#) (IN char \*file, IN int fileLen, IN int enable)
- int [lcd\\_displayText](#) (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char \*displayText, OUT BYTE \*graphicsID)

- int `lcd_displayText_Len` (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char \*displayText, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)
- int `lcd_displayParagraph` (IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char \*displayText)
- int `lcd_displayButton` (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char \*buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE \*graphicsID)
- int `lcd_displayButton_Len` (IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char \*buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)
- int `lcd_createList` (IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \*graphicsID)
- int `lcd_createList_Len` (IN int posX, IN int posY, IN int numColumns, IN int numRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderedScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)
- int `lcd_addItemToList` (IN BYTE \*listGraphicsID, IN char \*itemName, IN char \*itemID, IN int selected)
- int `lcd_getSelectedItem` (IN BYTE \*listGraphicsID, OUT char \*itemID)
- int `lcd_getSelectedItem_Len` (IN BYTE \*listGraphicsID, OUT char \*itemID, IN\_OUT int \*itemIDLen)
- int `lcd_clearEventQueue` ()
- int `lcd_getInputEvent` (IN int timeout, OUT int \*dataReceived, OUT BYTE \*eventType, OUT BYTE \*graphicsID, OUT BYTE \*eventData)
- int `lcd_getInputEvent_Len` (IN int timeout, OUT int \*dataReceived, OUT BYTE \*eventType, IN\_OUT int \*eventTypeLen, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen, OUT BYTE \*eventData, IN\_OUT int \*eventDataLen)
- int `lcd_createInputField` (IN BYTE \*specs, IN int specsLen, OUT BYTE \*graphicId)
- int `lcd_createInputField_Len` (IN BYTE \*specs, IN int specsLen, OUT BYTE \*graphicId, IN\_OUT int \*graphicIdLen)
- int `lcd_getInputFieldValue` (IN BYTE \*graphicId, OUT BYTE \*retData, IN\_OUT int \*retDataLen)
- int `msr_cancelMSRSwipe` ()
- int `msr_startMSRSwipe` (IN int \_timeout)
- int `msr_flushTrackData` ()
- void `parseMSRData` (IN BYTE \*resData, IN int resLen, IN\_OUT IDTMSRData \*cardData)
- int `pin_getEncryptedOnlinePIN` (IN int keyType, IN int timeout)
- int `pin_getPAN` (IN int getCSC, IN int timeout)
- int `pin_promptCreditDebit` (IN char \*currencySymbol, IN int currencySymbolLen, IN char \*displayAmount, IN int displayAmountLen, IN int timeout, OUT BYTE \*retData, IN\_OUT int \*retDataLen)
- int `ws_requestCSR` (OUT RequestCSR \*csr)
- int `ws_loadSSLCert` (IN char \*name, IN int nameLen, IN char \*dataDER, IN int dataDERLen)
- int `ws_revokeSSLCert` (IN char \*name, IN int nameLen)
- int `ws_deleteSSLCert` (IN char \*name, IN int nameLen)
- int `ws_getCertChainType` (OUT int \*type)
- int `ws_updateRootCertificate` (IN char \*name, IN int nameLen, IN char \*dataDER, IN int dataDERLen, IN char \*signature, IN int signatureLen)

### 12.16.1 Detailed Description

VP8800 API. VP8800 Global API methods.

## 12.16.2 Macro Definition Documentation

### 12.16.2.1 #define IN

INPUT parameter.

### 12.16.2.2 #define IN\_OUT

INPUT / OUTPUT PARAMETER.

### 12.16.2.3 #define OUT

OUTPUT parameter.

## 12.16.3 Typedef Documentation

### 12.16.3.1 typedef void(\* ftpComm\_callBack)(int, int, int)

Define the comm callback function to get FTP file transfer status

It should be passed as a parameter in a FTP request, Signature (int, int, int) = response code, current block, total blocks  
 RESPONSE CODES: 100 = FILE DOWNLOAD STARTED 101 = FILE BLOCK XX OF XX RECEIVED 102 = FILE DOWNLOAD COMPLETED 103 = FILE DOWNLOAD TERMINATED PREMATURELY

### 12.16.3.2 typedef void(\* httpComm\_callBack)(BYTE \*, int)

Define the comm callback function to get the async url data

It should be registered using the comm\_registerHTTPCallback

### 12.16.3.3 typedef void(\* pCMR\_callBack)(int, IDTCMRData \*)

Define the camera callback function to get the image data

It should be registered using the device\_registerCameraCallBk,

### 12.16.3.4 typedef void(\* pCSFS\_callBack)(BYTE status)

Define the card status and front switch callback function to get card and front switch status

It should be registered using the device\_registerCardStatusFrontSwitchCallBk,

### 12.16.3.5 typedef void(\* pEMV\_callBack)(int, int, unsigned char \*, int, IDTTransactionData \*, EMV\_Callback \*, int)

Define the EMV callback function to get the transaction message/data/result.

It should be registered using the emv\_registerCallBk,

### 12.16.3.6 typedef void(\* pLog\_callback)(BYTE, char \*)

Define the log callback function to receive log messages.



**12.16.3.7 typedef void(\* pMessageHotplug)(int, int)**

Define the USB hot-plug callback function to monitor the info when plug in/out the reader.

It should be registered using the registerHotplugCallBk, The first integer parameter is device type, and the second integer parameter is either 0: Device Plugged Out or 1: Device Plugged In

**12.16.3.8 typedef void(\* pMSR\_callBack)(int, IDTMSRData)**

Define the MSR callback function to get the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is for backward compatibility

**12.16.3.9 typedef void(\* pMSR\_callBackp)(int, IDTMSRData \*)**

Define the MSR callback function to get pointer to the MSR card data

It should be registered using the msr\_registerCallBk, this callback function is recommended instead of pMSR\_callBack

**12.16.3.10 typedef void(\* pPIN\_callBack)(int, IDTPINData \*)**

Define the PINPad callback function to get the input PIN Pad data

It should be registered using the pin\_registerCallBk,

**12.16.3.11 typedef void(\* pReadDataLog)(unsigned char \*, int)**

Define the read response callback function to monitor the reading response from the reader.

It should be registered using the registerLogCallBk,

**12.16.3.12 typedef void(\* pSendDataLog)(unsigned char \*, int)**

Define the send command callback function to monitor the sending command into the reader.

It should be registered using the registerLogCallBk,

**12.16.3.13 typedef void(\* v4Comm\_callBack)(BYTE, BYTE, BYTE \*, int)**

Define the comm callback function to receive the V4 Protocol packets received by the device from an external source (IP/USB/RS-232) It should be registered using the comm\_registerV4Callback, Data callback will contain command, sub-command, and data from V4 packet

**12.16.4 Function Documentation****12.16.4.1 void comm\_registerHTTPCallback ( httpComm\_callBack cBack )**

Register Comm HTTP Async Callback

**Parameters**

<i>cBack</i>	? HTTP Comm callback
--------------	----------------------

#### 12.16.4.2 void comm\_registerV4Callback ( v4Comm\_callBack *cBack* )

Register External V4 Protocol commands Callback

##### Parameters

<i>cBack</i>	? V4 Protocol Comm callback
--------------	-----------------------------

#### 12.16.4.3 int config\_getSerialNumber ( OUT char \* *sNumber* )

DEPRECATED : please use [config\\_getSerialNumber\\_Len](#)(OUT char\* *sNumber*, IN\_OUT int \**sNumberLen*)

Polls device for Serial Number

##### Parameters

<i>sNumber</i>	Returns Serial Number; needs to have at least 64 bytes of memory
----------------	--

##### Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

#### 12.16.4.4 int config\_getSerialNumber\_Len ( OUT char \* *sNumber*, IN\_OUT int \* *sNumberLen* )

Polls device for Serial Number

##### Parameters

<i>sNumber</i>	Returns Serial Number
<i>sNumberLen</i>	Length of Serial Number

##### Returns

RETURN\_CODE: Values can be parsed with device\_getIDGStatusCodeString

#### 12.16.4.5 int ctls\_activateTransaction ( IN const int \_timeout, IN BYTE \* *tags*, IN int *tagsLen* )

Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

##### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
  - - Bit 8 = VAS Support (1=on, 0 = off)
  - - Bit 7 = Touch ID Required (1=on, 0 = off)
  - - Bit 6 = RFU
  - - Bit 5 = RFU
  - - Bit 1,2,3,4
  - - - 0 = Payment Terminal
  - - - 1 = Transit Terminal
  - - - 2 = Access Terminal
  - - - 3 = Wireless Handoff Terminal
  - - - 4 = App Handoff Terminal
  - - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
  - - 0 = ApplePay VAS OR ApplePay
  - - 1 = ApplePay VAS AND ApplePay
  - - 2 = ApplePay VAS ONLY
  - - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DFO1 = 1 byte = ApplePay VAS Protocol. (optional)
  - - Bit 1 : 1 = URL VAS, 0 = Full VAS
  - - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
  - - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
  - - Bit 4-8 : RFU

**12.16.4.6 int ctls\_cancelTransaction ( )**

Cancel EMV Transaction

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.7 int `ctls_displayOnlineAuthResult` ( IN int *statusCode*, IN BYTE \* *TLV*, IN int *TLVLen* )

**Display Online Authorization Result** Use this command to display the status of an online authorization request on the reader's display (OK or NOT OK). Use this command after the reader sends an online request to the issuer.

## Parameters

<i>statusCode</i>	1 = OK, 0 = NOT OK, 2 = ARC response 89 for Interac
<i>TLV</i>	Optional TLV for AOSA
<i>TLVLen</i>	TLV Length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.8 int ctls\_getAllConfigurationGroups ( OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

## Retrieve All Configuration Groups

Returns all the Configuration Groups installed on the terminal for CTLS

## Parameters

<i>tlv</i>	The TLV elements data
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.9 int ctls\_getConfigurationGroup ( IN int *group*, OUT BYTE \* *tlv*, OUT int \* *tlvLen* )

## Get Configuration Group

Retrieves the Configuration for the specified Group.

## Parameters

<i>group</i>	Configuration Group
<i>tlv</i>	return data
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.10 void ctls\_registerCallBk ( pMSR\_callBack *pCTLSf* )

To register the ctls callback function to get the MSR card data. (Pass NULL to disable the callback.)

12.16.4.11 void ctls\_registerCallBkp ( pMSR\_callBackp *pCTLSf* )

To register the ctls callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

## 12.16.4.12 int ctls\_removeAllApplicationData ( )

## Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.13 int ctls\_removeAllCAPK ( )

Remove All Certificate Authority Public Key

Removes all the CAPK

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.14 int ctls\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

##### Parameters

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.15 int ctls\_removeCAPK ( IN BYTE \* capk, IN int capkLen )

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

##### Parameters

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.16 int ctls\_removeConfigurationGroup ( int group )

Remove Configuration Group

Removes the Configuration as specified by the Group. Must not by group 0

##### Parameters

<i>group</i>	Configuration Group
--------------	---------------------

##### Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

#### 12.16.4.17 int ctls\_retrieveAIDList ( OUT BYTE \* AIDList, IN\_OUT int \* AIDListLen )

Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.18 `int ctls_retrieveApplicationData ( IN BYTE * AID, IN int AIDLen, OUT BYTE * tlv, IN_OUT int * tlvLen )`

## Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.19 `int ctls_retrieveCAPK ( IN BYTE * capk, IN int capkLen, OUT BYTE * key, IN_OUT int * keyLen )`

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>keyLen</i>	the length of key data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.20** `int ctls_retrieveCAPKList ( OUT BYTE * keys, IN_OUT int * keysLen )`

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keysLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.21** `int ctls_retrieveTerminalData ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

**Parameters**

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.22** `int ctls_setApplicationData ( IN BYTE * tlv, IN int tlvLen )`

Set Application Data by AID

Sets the Application Data for CTLS as specified by TLV data

**Parameters**

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (FFE4). The second tag of the TLV data must be the AID (9F06)
------------	---

Example valid TLV, for Group #2, AID a0000000035010: "ffe401029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"

**Parameters**

<i>tlvLen</i>	the length of tlv data buffer
---------------	-------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.23** `int ctls_setCAPK ( IN BYTE * capk, IN int capkLen )`

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure



## Parameters

<i>capk</i>	CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where: <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
<i>capkLen</i>	the length of capk data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.24 int ctls\_setConfigurationGroup ( IN BYTE \* tlv, IN int tlvLen )

## Set Configuration Group

Sets the Configuration Group for CTLS as specified by the TLV data

## Parameters

<i>tlv</i>	Configuration Group Data in TLV format The first tag of the TLV data must be the group number (DFEE2D). A second tag must exist
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.25 int ctls\_setTerminalData ( IN BYTE \* tlv, IN int tlvLen )

## Set Terminal Data

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [ctls\\_getConfigurationGroup\(int group\)](#), and deleted with [ctls\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

## Parameters

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.16.4.26 `int ctls_startTransaction ( IN double amount, IN double amtOther, IN int type, IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

## Start CTLS Transaction Request

Authorizes the CTLS transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `ctls_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal

- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

12.16.4.27 `int device_activateTransaction ( IN const int _timeout, IN BYTE * tags, IN int tagsLen )`

Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

#### Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	The tags to be included in the request. Passed as a TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 Be sure to include 9F02 (amount) and 9-C (transaction type).
<i>tagsLen</i>	The length of tags data buffer.

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using `device_setMerchantRecord`, then container tag FFEE06 must be sent as part of the additional tags parameter of `device_startTransaction`. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DF01. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)
- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU

- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

#### 12.16.4.28 int device\_calibrateParameters ( BYTE *delta* )

Calibrate reference parameters

Parameters

<i>delta</i>	Delta value (0x02 standard default value)
--------------	---

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.29 int device\_cancelTransaction ( )

Cancel Transaction

Cancels the currently executing transaction.

Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.30 int device\_close ( )

Close the device

## Returns

RETURN\_CODE: 0: success, 0x0A: failed

12.16.4.31 int device\_controllIndicator ( IN int *indicator*, IN int *enable* )

## Control Indicators

Control the reader. If connected, returns success. Otherwise, returns timeout.

## Parameters

<i>indicator</i>	description as follows: <ul style="list-style-type: none"> <li>• 00h: ICC LED</li> <li>• 01h: Blue MSR</li> <li>• 02h: Red MSR</li> <li>• 03h: Green MSR</li> </ul>
<i>enable</i>	TRUE = ON, FALSE = OFF

## Returns

success or error code. Values can be parsed with device\_getResponseCodeString

## See Also

ErrorCode

12.16.4.32 int device\_controlUserInterface ( IN BYTE \* *values* )

## Control User Interface

Controls the User Interface: Display, Beep, LED

```
@param values Four bytes to control the user interface
Byte[0] = LCD Message
Messages 00-07 are normally controlled by the reader.
- 00h: Idle Message (Welcome)
- 01h: Present card (Please Present Card)
- 02h: Time Out or Transaction cancel (No Card)
- 03h: Transaction between reader and card is in the middle (Processing...)
- 04h: Transaction Pass (Thank You)
- 05h: Transaction Fail (Fail)
- 06h: Amount (Amount $ 0.00 Tap Card)
- 07h: Balance or Offline Available funds (Balance $ 0.00) Messages 08-0B are controlled by the terminal
- 08h: Insert or Swipe card (Use Chip & PIN)
- 09h: Try Again(Tap Again)
- 0Ah: Tells the customer to present only one card (Present 1 card only)
- 0Bh: Tells the customer to wait for authentication/authorization (Wait)
- FFh: indicates the command is setting the LED/Buzzer only.
Byte[1] = Beep Indicator
- 00h: No beep
- 01h: Single beep
- 02h: Double beep
- 03h: Three short beeps
```

- 04h: Four short beeps
- 05h: One long beep of 200 ms
- 06h: One long beep of 400 ms
- 07h: One long beep of 600 ms
- 08h: One long beep of 800 ms

Byte[2] = LED Number

- 00h: LED 0 (Power LED) 01h: LED 1
- 02h: LED 2
- 03h: LED 3
- FFh: All LEDs

Byte[3] = LED Status

- 00h: LED Off
- 01h: LED On

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.33 int device\_createDirectory ( IN char \* *directoryName*, IN int *directoryNameLen* )

Create Directory This command adds a subdirectory to the indicated path.

##### Parameters

<i>directoryName</i>	Directory Name. The data for this command is a ASCII string with the complete path and directory name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>directoryNameLen</i>	Directory Name Length.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.34 int device\_deleteDirectory ( IN char \* *dirName*, IN int *dirNameLen* )

Delete Directory This command deletes an empty directory. For NEO 2 devices, it will delete the directory even the directory is not empty.

##### Parameters

<i>dirName</i>	Complete path of the directory you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/). For NEO 2 devices, to delete the root directory, simply pass "" with 0 for dirNameLen.
<i>dirNameLen</i>	Directory Name Length.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.35 int device\_deleteFile ( IN char \* *fileName*, IN int *fileNameLen* )

Delete File This command deletes a file or group of files.

**Parameters**

<i>filename</i>	Complete path and file name of the file you want to delete. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>filenameLen</i>	File Name Length.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.36 int device\_enablePassThrough ( int enablePassThrough )****Start Remote Key Injection**

Starts a remote key injection request with IDTech RKI servers. This function is reserved and not implemented.

@return RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString](#)

**Enable Pass Through**

Enables Pass Through Mode for direct communication with L1 interface (power on icc, send apdu, etc).

**Parameters**

<i>enablePassThrough</i>	1 = pass through ON, 0 = pass through OFF
--------------------------	---

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.37 int device\_enhancedPassthrough ( IN BYTE \* data, IN int dataLen )**

Enables pass through mode for ICC. Required when direct ICC commands are required (power on/off ICC, exchange APDU)

**Parameters**

<i>data</i>	The data includes Poll Timeout, Flags, Contact Interface to Use, Beep Indicator, LED Status, and Display Strings.
<i>dataLen</i>	length of data

**Returns**

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString](#)

**See Also**

[ErrorCode](#)

**12.16.4.38 int device\_getCurrentDeviceType ( )**

Get current active device type

**Returns**

: return the device type defined as DEVICE\_TYPE in the IDTDef.h

12.16.4.39 `int device_getDriveFreeSpace ( OUT int * free, OUT int * used )`

Drive Free Space This command returns the free and used disk space on the flash drive.



## Parameters

<i>free</i>	Free bytes available on device
<i>used</i>	Used bytes on on device

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.40 `int device_getFirmwareVersion ( OUT char * firmwareVersion )`

DEPRECATED : please use [device\\_getFirmwareVersion\\_Len](#)(OUT char\* *firmwareVersion*, IN\_OUT int \**firmwareVersionLen*)

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version; needs to have at least 128 bytes of memory
------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.41 `int device_getFirmwareVersion_Len ( OUT char * firmwareVersion, IN_OUT int * firmwareVersionLen )`

Polls device for Firmware Version

## Parameters

<i>firmwareVersion</i>	Response returned of Firmware Version
<i>firmwareVersion-Len</i>	Length of Firmware Version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.42 `void device_getIDGStatusCodeString ( IN int returnCode, OUT char * despcrition )`

Review the return code description.

## Parameters

<i>returnCode</i>	the response result.
<i>description</i>	

## Return values

<i>the</i>	string for description of response result
------------	---

- 0: "no error, beginning task";
- 1: "no response from reader";
- 2: "invalid response data";
  - 01: " Incorrect Header Tag";

- 02: " Unknown Command";
- 03: " Unknown Sub-Command";
- 04: " CRC Error in Frame";
- 05: " Incorrect Parameter";
- 06: " Parameter Not Supported";
- 07: " Mal-formatted Data";
- 08: " Timeout";
- 0A: " Failed / NACK";
- 0B: " Command not Allowed";
- 0C: " Sub-Command not Allowed";
- 0D: " Buffer Overflow (Data Length too large for reader buffer)";
- 0E: " User Interface Event";
- 10: " Need clear firmware(apply in boot loader only)";
- 11: " Communication type not supported, VT-1, burst, etc. Need encrypted firmware (apply in boot loader only)";
- 12: " Secure interface is not functional or is in an intermediate state.";
- 13: " Data field is not mod 8";
- 14: " Pad 0x80 not found where expected";
- 15: " Specified key type is invalid";
- 16: " Could not retrieve key from the SAM (InitSecureComm)";
- 17: " Hash code problem";
- 18: " Could not store the key into the SAM (InstallKey)";
- 19: " Frame is too large";
- 1A: " Unit powered up in authentication state but POS must resend the InitSecureComm command";
- 1B: " The EEPROM may not be initialized because SecCommInterface does not make sense";
- 1C: " Problem encoding APDU Module-Specific Status Codes ";
- 20: " Unsupported Index (ILM) SAM Transceiver error - problem communicating with the SAM (Key Mgr)";
- 21: " Unexpected Sequence Counter in multiple frames for single bitmap (ILM)Length error in data returned from the SAM (Key Mgr) - 22: " Improper bit map (ILM)"; - 23: " Request Online Authorization"; - 24: " ViVOCard3 raw data read successful"; - 25: " Message index not available (ILM) ViVOcomm activate transaction card type (ViVOcomm)"; - 26: " Version Information Mismatch (ILM)"; - 27: " Not sending commands in correct index message index (ILM)"; - 28: " Time out or next expected message not received (ILM)"; - 29: " ILM languages not available for viewing (ILM)"; - 2A: " Other language not supported (ILM)"; - 41: " from 41 to 4F, Module-specific errors for Key Manager";
- 50: " Auto-Switch OK";
- 51: " Auto-Switch failed";
- 70: " Antenna Error 80h Use another card";
- 81: " Insert or swipe card";
- 90: " Data encryption Key does not exist";
- 91: " Data encryption Key KSN exhausted";

**12.16.4.43** `int device_getMerchantRecord ( IN int index, OUT BYTE * record )`

DEPRECATED : please use `device_getMerchantRecord_Len(IN int index, OUT BYTE * record, IN_OUT int *recordLen)`

Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record;</i>	needs to have at least 99 bytes of memory response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

**12.16.4.44** int device\_getMerchantRecord\_Len ( IN int *index*, OUT BYTE \* *record*, IN\_OUT int \* *recordLen* )

## Get Merchant Record

Gets the merchant record for the device.

## Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>record</i>	response data from reader. Merchant Record Index: 1 byte enabled: 1 byte Merchant ID: 32 bytes Length of Merchant URL: 1 byte Merchant URL: 64 bytes
<i>recordLen</i>	Length of record

## Returns

success or error code. Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## See Also

ErrorCode

**12.16.4.45** int device\_getSDKWaitTime ( )

## Get SDK Wait Time

Get the SDK wait time for transactions

## Returns

SDK wait time in seconds

**12.16.4.46** int device\_getThreadStackSize ( )

## Get Thread Stack Size

Get the stack size setting for newly created threads

## Returns

Thread Stack Size

**12.16.4.47** int device\_getTransactionResults ( IDTMSRData \* *cardData* )

Get Transaction Results Gets the transaction results when the reader is functioning in "Auto Poll" mode

## Parameters

<i>cardData</i>	The transaction results
-----------------	-------------------------

## Returns

success or error code. Values can be parsed with `device_getResponseCodeString`

## See Also

`ErrorCode`

12.16.4.48 `int device_init ( )`

Initial the device by USB

It will detect the device and trying connect.

The connect status can be checked by `device_isConnected()`.

## Returns

RETURN\_CODE: Values can be parsed with `device_getIDGStatusCodeString()`

12.16.4.49 `int device_isAttached ( int deviceType )`

Check if the device is attached to the USB port The function `device_init()` must be called before this function.

## Parameters

<i>deviceType,the</i>	device type of the USB device
-----------------------	-------------------------------

## Returns

1 if the device is attached, or 0 if the device is not attached

12.16.4.50 `int device_isConnected ( )`

Check the device connected status

## Returns

DEVICE\_DISCONNECT=0, or DEVICE\_CONNECTED = 1

12.16.4.51 `int device_listDirectory ( IN char * directoryName, IN int directoryNameLen, IN int recursive, IN int onSD, OUT char * directory, IN_OUT int * directoryLen )`

List Directory This command retrieves a directory listing of user accessible files from the reader.

## Parameters

<i>directoryName</i>	Directory Name. If null, root directory is listed
<i>directoryName-Len</i>	Directory Name Length. If null, root directory is listed
<i>recursive</i>	Included sub-directories
<i>onSD</i>	TRUE = use flash storage The returned directory information The returned directory information length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.52 int device\_pingDevice ( )

## Ping Device

Pings the reader. If connected, returns success. Otherwise, returns timeout.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.53 void device\_registerCameraCallBk ( pCMR\_callBack pCMRf )

To register the camera callback function to get the image data. (Pass NULL to disable the callback.)

## 12.16.4.54 void device\_registerCardStatusFrontSwitchCallBk ( pCSFS\_callBack pCSFSf )

To register the card status and front switch callback function to get status. (Pass NULL to disable the callback.)

## 12.16.4.55 int device\_SendDataCommandNEO ( IN int cmd, IN int subCmd, IN BYTE \* data, IN int dataLen, OUT BYTE \* response, IN\_OUT int \* respLen )

## Send a Command to device

Sends a command to the device .

## Parameters

<i>cmd</i>	buffer of command to execute.
<i>cmdLen,the</i>	length of the buffer cmd.
<i>data</i>	buffer of IDG command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Send a Command to NEO device

Sends a command to the NEO device .

## Parameters

<i>cmd</i>	command to execute.
<i>subCmd,sub</i>	command to execute.
<i>data</i>	buffer of NEO command data.
<i>dataLen,the</i>	length of the buffer data.
<i>response</i>	Response data
<i>respLen,the</i>	length of Response data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.56 int device\_setCurrentDevice ( int deviceType )

Sets the current device to talk to

The connect status can be checked by [device\\_isConnected\(\)](#).

##### Parameters

<i>deviceType</i>	Device to connect to
	<pre>enum DEVICE_TYPE{     IDT_DEVICE_UNKNOWN=0,     IDT_DEVICE_AUGUSTA_HID,     IDT_DEVICE_AUGUSTA_KB,     IDT_DEVICE_AUGUSTA_S_HID,     IDT_DEVICE_AUGUSTA_S_KB,     IDT_DEVICE_AUGUSTA_S_TTK_HID,     IDT_DEVICE_SPECTRUM_PRO,     IDT_DEVICE_MINISMART_II,     IDT_DEVICE_L100,     IDT_DEVICE_UNIPAY,     IDT_DEVICE_UNIPAY_I_V,     IDT_DEVICE_VP3300_AJ,     IDT_DEVICE_KIOSK_III,     IDT_DEVICE_KIOSK_III_S,     IDT_DEVICE_PIP_READER,     IDT_DEVICE_VENDI,     IDT_DEVICE_VP3300_USB,     IDT_DEVICE_UNIPAY_I_V_TTK,     IDT_DEVICE_VP3300_BT,     IDT_DEVICE_VP8800,     IDT_DEVICE_SREDKEY2_HID,     IDT_DEVICE_SREDKEY2_KB,     IDT_DEVICE_NEO2,     IDT_DEVICE_MINISMART_II_COM = IDT_DEVICE_NEO2+5,     IDT_DEVICE_SPECTRUM_PRO_COM,     IDT_DEVICE_KIOSK_III_COM,     IDT_DEVICE_KIOSK_III_S_COM,     IDT_DEVICE_VP3300_COM,     IDT_DEVICE_NEO2_COM,     IDT_DEVICE_MAX_DEVICES = IDT_DEVICE_NEO2_COM+5 };</pre>

##### Returns

RETURN\_CODE: 1: success, 0: failed

#### 12.16.4.57 int device\_setMerchantRecord ( int index, int enabled, char \* merchantID, char \* merchantURL )

Set Merchant Record Sets the merchant record for ApplePay VAS

##### Parameters

<i>index</i>	Merchant Record index, valid values 1-6
<i>enabled</i>	Merchant Enabled/Valid flag
<i>merchantID</i>	Merchant unique identifier registered with Apple. Example com.idtechproducts.applePay
<i>merchantURL</i>	Merchant URL, when applicable

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

#### 12.16.4.58 void device\_setSDKWaitTime ( int waitTime )

Set SDK Wait Time

Set the SDK wait time for transactions

## Parameters

<i>waitTime</i>	The SDK wait time for transaction in seconds
-----------------	--

12.16.4.59 void device\_setThreadStackSize ( int *threadSize* )

## Set Thread Stack Size

Set the stack size setting for newly created threads

12.16.4.60 void device\_setTransactionExponent ( int *exponent* )

Sets the transaction exponent to be used with device\_startTransaction. Default value is 2

## Parameters

<i>exponent, The</i>	exponent to use when calling device_startTransaction
----------------------	--

12.16.4.61 int device\_startTransaction ( IN double *amount*, IN double *amtOther*, IN int *type*, IN const int *timeout*, IN BYTE \* *tags*, IN int *tagsLen* )

## Start Transaction Request

Authorizes the transaction for an MSR/CTLS/ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as TLV. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags data buffer.

>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll is on, it will return the error IDG\_P2\_STATUS\_CODE\_COMMAND\_NOT\_ALLOWED

NOTE ON APPLEPAY VAS: To enable ApplePay VAS, first a merchant record must be defined in one of the six available index positions (1-6) using device\_setMerchantRecord, then container tag FFEE06 must be sent as part of the additional tags parameter of device\_startTransaction. Tag FFEE06 must contain tag 9F26 and 9F22, and can optionally contain tags 9F2B and DFO1. Example FFEE06189F220201009F2604000000009F2B050100000000-DF010101 9F22 = two bytes = ApplePay Terminal Application Version Number. Hard defined as 0100 for now. (required) 9F26 = four bytes = ApplePay Terminal Capabilities Information (required)

- Byte 1 = RFU
- Byte 2 = Terminal Type
- - Bit 8 = VAS Support (1=on, 0 = off)

- - Bit 7 = Touch ID Required (1=on, 0 = off)
- - Bit 6 = RFU
- - Bit 5 = RFU
- - Bit 1,2,3,4
- - - 0 = Payment Terminal
- - - 1 = Transit Terminal
- - - 2 = Access Terminal
- - - 3 = Wireless Handoff Terminal
- - - 4 = App Handoff Terminal
- - - 15 = Other Terminal
- Byte 3 = RFU
- Byte 4 = Terminal Mode
- - 0 = ApplePay VAS OR ApplePay
- - 1 = ApplePay VAS AND ApplePay
- - 2 = ApplePay VAS ONLY
- - 3 = ApplePay ONLY 9F2B = 5 bytes = ApplePay VAS Filter. Each byte filters for that specific merchant index (optional) DF01 = 1 byte = ApplePay VAS Protocol. (optional)
- - Bit 1 : 1 = URL VAS, 0 = Full VAS
- - Bit 2 : 1 = VAS Beeps, 0 = No VAS Beeps
- - Bit 3 : 1 = Silent Comm Error, 2 = EMEA Comm Error
- - Bit 4-8 : RFU

**12.16.4.62** `int device_transferFile ( IN char * fileName, IN int fileNameLen, IN BYTE * file, IN int fileLen )`

Transfer File This command transfers a data file to the reader.

#### Parameters

<i>fileName</i>	Filename. The data for this command is a ASCII string with the complete path and file name you want to create. You do not need to specify the root directory. Indicate subdirectories with a forward slash (/).
<i>filenameLen</i>	File Name Length.
<i>file</i>	The data file.
<i>fileLen</i>	File Length.

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.63** `int emv_activateTransaction ( IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.



## Parameters

<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100
<i>tagsLen</i>	Length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with device\_getResponseCodeString >>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

## 12.16.4.64 void emv\_allowFallback ( IN int allow )

Allow fallback for EMV transactions. Default is TRUE

## Parameters

<i>allow</i>	TRUE = allow fallback, FALSE = don't allow fallback
--------------	---

## 12.16.4.65 int emv\_authenticateTransaction ( IN BYTE \* updatedTLV, IN int updatedTLVLen )

## Authenticate EMV Transaction Request

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to emv\_startTransaction

The tags will be returned in the callback routine.

## Parameters

<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37</li> </ul>
-------------------	---

<i>updatedTLVLen</i>	
----------------------	--

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.16.4.66 `int emv_authenticateTransactionWithTimeout ( IN int timeout, IN BYTE * updatedTLV, IN int updatedTLVLen )`

**Authenticate EMV Transaction Request with Timeout**

Authenticates the EMV transaction for an ICC card. Execute this after receiving response with result code 0x10 to `emv_startTransaction`

The tags will be returned in the callback routine.

**Parameters**

<i>timeout</i>	Timeout value in seconds.
<i>updatedTLV</i>	TLV stream that can be used to update the following values: <ul style="list-style-type: none"> <li>• 9F02: Amount</li> <li>• 9F03: Other amount</li> <li>• 9C: Transaction type</li> <li>• 5F57: Account type In addition tag DFEE1A can be sent to specify tag list to include in results. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1-A079F029F36959F37</li> </ul>
<i>updatedTLVLen</i>	

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

12.16.4.67 `int emv_cancelTransaction ( )`

**Cancel EMV Transaction**

Cancels the currently executing EMV transaction.

**Returns**

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString()`

12.16.4.68 `int emv_completeTransaction ( IN int commError, IN BYTE * authCode, IN int authCodeLen, IN BYTE * iad, IN int iadLen, IN BYTE * tlvScripts, IN int tlvScriptsLen, IN BYTE * tlv, IN int tlvLen )`

**Complete EMV Transaction Request**

Completes the EMV transaction for an ICC card when online authorization request is received from `emv_authenticateTransaction`

The tags will be returned in the callback routine.

## Parameters

<i>commError</i>	Communication error with host. Set to TRUE(1) if host was unreachable, or FALSE(0) if host response received. If Communication error, authCode, iad, tlvScripts can be null.
<i>authCode</i>	Authorization code from host. Two bytes. Example 0x3030. (Tag value 8A). Required
<i>authCodeLen</i>	the length of authCode
<i>iad</i>	Issuer Authentication Data, if any. Example 0x11223344556677883030 (tag value 91).
<i>iadLen</i>	the length of iadLen
<i>tlvScripts</i>	71/72 scripts, if any
<i>tlvScriptsLen</i>	the length of tlvScriptsLen
<i>tlv</i>	Additional TLV data to return with transaction results (if any)
<i>tlvLen</i>	the length of tlv

## Returns

RETURN\_CODE: Values can be parsed with `device_getResponseCodeString`

## 12.16.4.69 int emv\_getAutoAuthenticateTransaction ( )

Gets auto authenticate value for EMV transactions.

## Returns

RETURN\_CODE: TRUE = auto authenticate, FALSE = manually authenticate

## 12.16.4.70 int emv\_getAutoCompleteTransaction ( )

Gets auto complete value for EMV transactions.

## Returns

RETURN\_CODE: TRUE = auto complete, FALSE = manually complete

## 12.16.4.71 int emv\_getEMVConfigurationCheckValue ( OUT BYTE \* checkValue, IN\_OUT int \* checkValueLen )

Get EMV Kernel configuration check value info

## Parameters

<i>checkValue</i>	Response returned of Kernel configuration check value info
<i>checkValueLen</i>	the length of checkValue

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.72 int emv\_getEMVKernelCheckValue ( OUT BYTE \* checkValue, IN\_OUT int \* checkValueLen )

Get EMV Kernel check value info

## Parameters

<i>checkValue</i>	Response returned of Kernel check value info
<i>checkValueLen</i>	the length of checkValue

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.73 `int emv_getEMVKernelVersion ( OUT char * version )`

DEPRECATED : please use [emv\\_getEMVKernelVersion\\_Len\(OUT char\\* version, IN\\_OUT int \\*versionLen\)](#)

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version; needs to have at least 128 bytes of memory.
----------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.74 `int emv_getEMVKernelVersion_Len ( OUT char * version, IN_OUT int * versionLen )`

Polls device for EMV Kernel Version

## Parameters

<i>version</i>	Response returned of Kernel Version
<i>versionLen</i>	Length of version

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.75 `void emv_registerCallBk ( pEMV_callback pEMVf )`

To register the emv callback function to get the EMV processing response. (Pass NULL to disable the callback.)

12.16.4.76 `int emv_removeAllApplicationData ( )`

Remove All Application Data

Removes all the Application Data

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.77 `int emv_removeAllCAPK ( )`

Remove All Certificate Authority Public Key

Removes all the CAPK

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.78 int emv\_removeAllCRL ( )**

Remove All Certificate Revocation List Entries

Removes all CRLentry entries

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

**12.16.4.79 int emv\_removeAllExceptions ( )**

Remove All EMV Exceptions

Removes all entries from the EMV Exception List

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.80 int emv\_removeApplicationData ( IN BYTE \* AID, IN int AIDLen )**

Remove Application Data by AID Removes the Application Data for CTLS as specified by the AID name passed as a parameter

**Parameters**

<i>AID</i>	Name of ApplicationID Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.81 int emv\_removeCAPK ( IN BYTE \* capk, IN int capkLen )**

Remove Certificate Authority Public Key

Removes the CAPK as specified by the RID/Index

**Parameters**

<i>capk</i>	6 byte CAPK = 5 bytes RID + 1 byte INDEX
<i>capkLen</i>	the length of capk data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

**12.16.4.82 int emv\_removeCRL ( IN BYTE \* list, IN int lsLen )**

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

## Parameters

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>IssLen</i>	the length of list data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.83 int emv\_removeException ( IN BYTE \* *exception*, IN int *exceptionLen* )

## Remove EMV Exception

Removes an entry to the EMV Exception List

## Parameters

<i>exception</i>	EMV Exception entry containing the PAN and Sequence Number where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number] PAN, in compressed numeric, padded with F if required (example 0x5413339000001596FFFF)
<i>exceptionLen</i>	The length of the exception.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.84 int emv\_removeTransactionLog ( )

## Clear Transaction Log

Clears the transaction log.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.85 int emv\_retrieveAIDList ( OUT BYTE \* *AIDList*, IN\_OUT int \* *AIDListLen* )

## Retrieve AID list

Returns all the AID names installed on the terminal for CTLS. .

## Parameters

<i>AIDList</i>	array of AID name byte arrays
<i>AIDListLen</i>	the length of AIDList array buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.86 int emv\_retrieveApplicationData ( IN BYTE \* *AID*, IN int *AIDLen*, OUT BYTE \* *tlv*, IN\_OUT int \* *tlvLen* )

## Retrieve Application Data by AID

Retrieves the Application Data as specified by the AID name passed as a parameter.

## Parameters

<i>AID</i>	Name of ApplicationID. Must be between 5 and 16 bytes
<i>AIDLen</i>	the length of AID data buffer.
<i>tlv</i>	The TLV elements of the requested AID
<i>tlvLen</i>	the length of tlv data buffer.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.87 int emv\_retrieveCAPK ( IN BYTE \* *capk*, IN int *capkLen*, OUT BYTE \* *key*, IN\_OUT int \* *keyLen* )

## Retrieve Certificate Authority Public Key

Retrieves the CAPK for CTLS as specified by the RID/Index passed as a parameter.

## Parameters

<i>capk</i>	6 bytes CAPK = 5 bytes RID + 1 byte Index
<i>capkLen</i>	the length of capk data buffer
<i>key</i>	<p>Response returned as a CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm] [20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>

<i>keyLen</i>	the length of key data buffer •
---------------	------------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.88 int emv\_retrieveCAPKList ( OUT BYTE \* *keys*, IN\_OUT int \* *keyLen* )

Retrieve the Certificate Authority Public Key list

Returns all the CAPK RID and Index installed on the terminal.

**Parameters**

<i>keys</i>	[key1][key2]...[keyn], each key 6 bytes where key = 5 bytes RID + 1 byte index
<i>keyLen</i>	the length of keys data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.89 int emv\_retrieveCRL ( OUT BYTE \* *list*, IN\_OUT int \* *IssLen* )

Retrieve the Certificate Revocation List

Returns the CRL entries on the terminal.

**Parameters**

<i>list</i>	[CRL1][CRL2]...[CRLn], each CRL 9 bytes where CRL = 5 bytes RID + 1 byte index + 3 bytes serial number
<i>IssLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.90 int emv\_retrieveExceptionList ( OUT BYTE \* *exceptionList*, IN\_OUT int \* *exceptionListLen* )

Retrieve the EMV Exception List

Returns the EMV Exception entries on the terminal.

**Parameters**

<i>exceptionList</i>	[Exception1][Exception2]...[Exceptionn], where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number]
<i>exceptionListLen</i>	The length of the exception list.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)



12.16.4.91 `int emv_retrieveExceptionLogStatus ( OUT BYTE * exceptionLogStatus, IN_OUT int * exceptionLogStatusLen )`

#### Get EMV Exception Log Status

This command returns information about the EMV Exception log. The version number, record size, and number of records contained in the file are returned.

##### Parameters

<i>exceptionLog-Status</i>	12 bytes returned <ul style="list-style-type: none"><li>• bytes 0-3 = Version Number</li><li>• bytes 4-7 = Number of records</li><li>• bytes 8-11 = Size of record</li></ul>
<i>exceptionLog-StatusLen</i>	The length of the exception log status.

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.92 `int emv_retrieveTerminalData ( OUT BYTE * tlv, IN_OUT int * tlvLen )`

#### Retrieve Terminal Data

Retrieves the Terminal Data for CTLS. This is configuration group 0 (Tag FFEE - > FFEE0100). The terminal data can also be retrieved by `ctls_getConfigurationGroup(0)`.

##### Parameters

<i>tlv</i>	Response returned as a TLV
<i>tlvLen</i>	the length of tlv data buffer

##### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.93 `int emv_retrieveTransactionLog ( OUT BYTE * transactionLog, IN_OUT int * transactionLogLen, IN_OUT int * remainingTransactionLogLen )`

#### Get Transaction Log Record

Retrieves oldest transaction record on the Transaction Log. At successful completion, the oldest transaction record is deleted from the transaction log

##### Parameters

<i>transactionLog</i>	Transaction Record
<i>transactionLog-Len</i>	The length of the transaction log.
<i>remaining-TransactionLog-Len</i>	Number of records remaining on the transaction log

Length	Description	Type
4	Transaction Log State (TLS)	Enum (4-byte number, LSB first), SENT ONLINE = 0, NOT SENT = 1
4	Transaction Log Content (TLC)	Enum (4-byte number, LSB first), BATCH = 0, OFFLINE ADVICE = 1, ONLINEADVICE = 2, REVERSAL = 3
4	AppExpDate	unsigned char [4]
3	AuthRespCode	unsigned char [3]
3	MerchantCategoryCode	unsigned char [3]
16	MerchantID	unsigned char [16]
2	PosEntryMode	unsigned char [2]
9	TermID	unsigned char [9]
3	AIP	unsigned char [3]
3	ATC	unsigned char [3]
33	IssuerAppData	unsigned char [33]
6	TVR	unsigned char [6]
3	TSI	unsigned char [3]
11	Pan	unsigned char [11]
2	PanSQNCNum	unsigned char [2]
3	TermCountryCode	unsigned char [3]
7	TranAmount	unsigned char [7]
3	TranCurCode	unsigned char [3]
4	TranDate	unsigned char [4]
2	TranType	unsigned char [2]
9	IFDSerialNum	unsigned char [9]
12	AcquirerID	unsigned char [12]
2	CID	unsigned char [2]
9	AppCryptogram	unsigned char [9]
5	UnpNum	unsigned char [5]
7	AmountAuth	unsigned char [7]
4	AppEffDate	unsigned char [4]
4	CVMResults	unsigned char [4]
129	IssScriptResults	unsigned char [129]
4	TermCap	unsigned char [4]
2	TermType	unsigned char [2]
20	Track2	unsigned char [20]
4	TranTime	unsigned char [4]
7	AmountOther	unsigned char [7]
1	Unused	Unsigned char [1]

#### Returns

RETURN\_CODE: Values can be parsed with `errorCode.getErrorString()`

**12.16.4.94** `int emv_retrieveTransactionLogStatus ( OUT BYTE * transactionLogStatus, IN_OUT int * transactionLogStatusLen )`

#### Get Transaction Log Status

This command returns information about the EMV transaction log. The version number, record size, and number of records contained in the file are returned.

## Parameters

<i>transactionLog-Status</i>	12 bytes returned <ul style="list-style-type: none"> <li>• bytes 0-3 = Version Number</li> <li>• bytes 4-7 = Number of records</li> <li>• bytes 8-11 = Size of record</li> </ul>
<i>transactionLog-StatusLen</i>	The length of the transaction log status.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.95 `int emv_setApplicationData ( IN BYTE * name, IN int nameLen, IN BYTE * tlv, IN int tlvLen )`

Set Application Data by AID

Sets the Application Data as specified by the application name and TLV data

## Parameters

<i>name</i>	Application name, 10-32 ASCII hex characters representing 5-16 bytes Example "a0000000031010"
<i>nameLen</i>	the length of name data buffer of Application name,
<i>tlv</i>	Application data in TLV format
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.96 `int emv_setApplicationDataTLV ( IN BYTE * tlv, IN int tlvLen )`

Set Application Data by TLV

Sets the Application Data as specified by the TLV data

## Parameters

<i>tlv</i>	Application data in TLV format The first tag of the TLV data must be the group number (DFEE2D). The second tag of the TLV data must be the AID (9F06) Example valid TLV, for Group #2, AID a0000000035010- : "dfee2d01029f0607a0000000051010ffe10101ffe50110ffe30114ffe20106"
<i>tlvLen</i>	the length of tlv data buffer

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.97 `void emv_setAutoAuthenticateTransaction ( IN int authenticate )`

Enables authenticate for EMV transactions. If a `emv_startTransaction` results in code 0x0010 (start transaction success), then `emv_authenticateTransaction` can automatically execute if parameter is set to TRUE

## Parameters

<i>authenticate</i>	TRUE = auto authenticate, FALSE = manually authenticate
---------------------	---

12.16.4.98 void emv\_setAutoCompleteTransaction ( IN int *complete* )

Enables complete for EMV transactions. If a emv\_authenticateTransaction results in code 0x0004 (go online), then emv\_completeTransaction can automatically execute if parameter is set to TRUE

## Parameters

<i>complete</i>	TRUE = auto complete, FALSE = manually complete
-----------------	---

12.16.4.99 int emv\_setCAPK ( IN BYTE \* *capk*, IN int *capkLen* )

Set Certificate Authority Public Key

Sets the CAPK as specified by the CAKey structure

## Parameters

<i>capk</i>	<p>CAKey format: [5 bytes RID][1 byte Index][1 byte Hash Algorithm][1 byte Encryption Algorithm][20 bytes HashValue][4 bytes Public Key Exponent][2 bytes Modulus Length][Variable bytes Modulus] Where:</p> <ul style="list-style-type: none"> <li>• Hash Algorithm: The only algorithm supported is SHA-1. The value is set to 0x01</li> <li>• Encryption Algorithm: The encryption algorithm in which this key is used. Currently support only one type: RSA. The value is set to 0x01.</li> <li>• HashValue: Which is calculated using SHA-1 over the following fields: RID &amp; Index &amp; Modulus &amp; Exponent</li> <li>• Public Key Exponent: Actually, the real length of the exponent is either one byte or 3 bytes. It can have two values: 3 (Format is 0x00 00 00 03), or 65537 (Format is 0x00 01 00 01)</li> <li>• Modulus Length: LenL LenH Indicated the length of the next field.</li> <li>• Modulus: This is the modulus field of the public key. Its length is specified in the field above.</li> </ul>
-------------	--

<i>capkLen</i>	the length of capk data buffer
----------------	--------------------------------

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.100 int emv\_setCRL ( IN BYTE \* *list*, IN int *lsLen* )

**Set Certificate Revocation List**

Sets the CRL

**Parameters**

<i>list</i>	CRL Entries containing the RID, Index, and serial numbers to set [CRL1][CRL2]...[CRLn] where each [CRL] is 9 bytes: [5 bytes RID][1 byte CAPK Index][3 bytes serial number]
<i>lsLen</i>	the length of list data buffer

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.101 int emv\_setException ( IN BYTE \* *exception*, IN int *exceptionLen* )

**Set EMV Exception**

Adds an entry to the EMV Exception List

**Parameters**

<i>exception</i>	EMV Exception entry containing the PAN and Sequence Number where [Exception] is 12 bytes: [1 byte Len][10 bytes PAN][1 byte Sequence Number] PAN, in compressed numeric, padded with F if required (example 0x5413339000001596FFFF)
<i>exceptionLen</i>	The length of the exception.

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.102 int emv\_setTerminalData ( IN BYTE \* *tlv*, IN int *tlvLen* )

**Set Terminal Data**

Sets the Terminal Data for CTLS as specified by the TLV. The first TLV must be Configuration Group Number (Tag FFE4). The terminal global data is group 0, so the first TLV would be FFE40100. Other groups can be defined using this method (1 or greater), and those can be retrieved with [emv\\_getConfigurationGroup\(int group\)](#), and deleted with [emv\\_removeConfigurationGroup\(int group\)](#). You cannot delete group 0.

**Parameters**

<i>tlv</i>	TerminalData configuration file
<i>tlvLen</i>	the length of tlv data buffer

## Return values

<i>RETURN_CODE</i>	Values can be parsed with <a href="#">device_getIDGStatusCodeString()</a>
--------------------	---

12.16.4.103 `int emv_startTransaction ( IN double amount, IN double amtOther, IN int exponent, IN int type, IN int timeout, IN BYTE * tags, IN int tagsLen, IN int forceOnline )`

## Start EMV Transaction Request

Authorizes the EMV transaction for an ICC card

The tags will be returned in the callback routine.

## Parameters

<i>amount</i>	Transaction amount value (tag value 9F02) - SEE IMPORTANT NOTE BELOW
<i>amtOther</i>	Other amount value, if any (tag value 9F03) - SEE IMPORTANT NOTE BELOW
<i>exponent</i>	Number of characters after decimal point
<i>type</i>	Transaction type (tag value 9C).
<i>timeout</i>	Timeout value in seconds.
<i>tags</i>	Any other tags to be included in the request. Passed as a TLV stream. Example, tag 9F0C with amount 0x000000000100 would be 0x9F0C06000000000100 If tags 9F02 (amount), 9F03 (other amount), or 9C (transaction type) are included, they will take priority over these values supplied as individual parameters to this method.
<i>tagsLen</i>	The length of tags
<i>forceOnline</i>	TRUE = do not allow offline approval, FALSE = allow ICC to approve offline if terminal capable Note: To request tags to be included in default response, use tag DFEE1A, and specify tag list. Example four tags 9F02, 9F36, 95, 9F37 to be included in response = DFEE1A079F029F36959F37

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString](#)

>>>>>IMPORTANT: parameters for amount and amtOther MUST BE PASSED AS A DOUBLE VALUE WITH DECIMAL POINT. Example, do not pass 1, but instead pass 1.0 or 1.00. Otherwise, results will be unpredictable

12.16.4.104 `int lcd_addItemToList ( IN BYTE * listGraphicsID, IN char * itemName, IN char * itemID, IN int selected )`

Adds an item to an existing list.

Custom Display Mode must be enabled for custom text.

## Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemName</i>	Item name (Maximum: 127 characters)
<i>itemID</i>	Identifier for the item (Maximum: 31 characters)
<i>selected</i>	If the item should be selected

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.105 `int lcd_cancelSlideShow ( OUT BYTE * statusCode, IN_OUT int * statusCodeLen )`

Cancel slide show Cancel the slide show currently running

## Parameters

<i>statusCode</i>	If the return code is not Success (0), the kernel may return a four-byte Extended Status Code
<i>statusCodeLen</i>	the length of the Extended Status Code (should be 4 bytes)

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.106 int lcd\_captureSignature ( IN int timeout )

Enables Signature Capture This command executes the signature capture screen. Once a signature is captured, it is sent to the callback with DeviceState.Signature, and the data will contain a .png of the signature

## Parameters

<i>timeout</i>	Timeout waiting for the signature capture
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.107 int lcd\_clearDisplay ( IN BYTE control )

Clear Display Command to clear the display screen on the reader. It returns the display to the currently defined background color and terminates all events

## Parameters

<i>control</i>	for L100 only. 0:First Line 1:Second Line 2:Third Line 3:Fourth Line 0xFF: All Screen
----------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.108 int lcd\_clearEventQueue ( )

Removes all entries from the event queue.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.109 int lcd\_createInputField ( IN BYTE \* specs, IN int specsLen, OUT BYTE \* graphicId )

DEPRECATED : please use [lcd\\_createInputField\\_Len\(IN BYTE \\*specs, IN int specsLen, OUT BYTE \\*graphicId, IN\\_OUT int \\*graphicIdLen\)](#)

Create an input field on the screen.

## Parameters

<i>specs</i>	The specs of the input field:
--------------	-------------------------------

Length (bytes)	Description
2 - 4	X coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Y coordinate in pixels, zero terminated ASCII
-----	-----
2 - 4	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width.
-----	-----
2 - 4	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height.
-----	-----
2	Font designation. Default font = 1, zero terminated ASCII
-----	-----
2 - 3	Zero terminated ASCII Font ID
-----	-----
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
	Bit 2 - 15 Reserved
-----	-----
1 or 9	Foreground color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Background color, zero terminated ASCII hexadecimal
-----	-----
1 or 9	Border color, zero terminated ASCII hexadecimal
-----	-----
1 - 65	Prefill String, zero terminated ASCII
-----	-----
1 - 65	Format String, zero terminated ASCII

## Parameters

<i>specsLen</i>	The length of specs
-----------------	---------------------



<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)
-------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.110 int lcd\_createInputField\_Len ( IN BYTE \* *specs*, IN int *specsLen*, OUT BYTE \* *graphicId*, IN\_OUT int \* *graphicIdLen* )

Create an input field on the screen.

## Parameters

<i>specs</i>	The specs of the input field:
--------------	-------------------------------

Length (bytes)	Description
2 - 4 -----	X coordinate in pixels, zero terminated ASCII -----
2 - 4 -----	Y coordinate in pixels, zero terminated ASCII -----
2 - 4 -----	Width in pixels, zero terminated ASCII. Set to 0 (30h) for calculated width. -----
2 - 4 -----	Height in pixels, zero terminated ASCII. Set to 0 (30h) for calculated height. -----
2 -----	Font designation. Default font = 1, zero terminated ASCII -----
2 - 3 -----	Zero terminated ASCII Font ID -----
3	Zero terminated ASCII hexadecimal display option flag
	Bit 0 0 = No Border
	1 = Show Border
	Bit 1 0 = Characters are first displayed on the leftmost area of the screen.
	1 = The first character entered is displayed on the rightmost area of
	the screen, and, as further digits are entered, characters scroll
	from the right to the left.
	Bit 2 - 15 Reserved
1 or 9 -----	Foreground color, zero terminated ASCII hexadecimal -----

1 or 9 -----	Background color, zero terminated ASCII hexadecimal ----- -----
1 or 9 -----	Border color, zero terminated ASCII hexadecimal ----- -----
1 - 65 -----	Prefill String, zero terminated ASCII ----- -----
1 - 65	Format String, zero terminated ASCII

**Parameters**

<i>specsLen</i>	The length of specs
<i>graphicsID</i>	The graphicID of the event (required to be 4 bytes)
<i>graphicsIDLen</i>	Length of graphicsID

**Returns**

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.111 **int** lcd\_createList ( **IN** int *posX*, **IN** int *posY*, **IN** int *numOfColumns*, **IN** int *numOfRows*, **IN** int *fontDesignation*, **IN** int *fontID*, **IN** int *verticalScrollArrowsVisible*, **IN** int *borderedListItems*, **IN** int *borderdScrollArrows*, **IN** int *touchSensitive*, **IN** int *automaticScrolling*, **OUT** BYTE \* *graphicsID* )

DEPRECATED : please use lcd\_createList\_Len(IN int posX, IN int posY, IN int numOfColumns, IN int numOfRows, IN int fontDesignation, IN int fontID, IN int verticalScrollArrowsVisible, IN int borderedListItems, IN int borderdScrollArrows, IN int touchSensitive, IN int automaticScrolling, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)

Creates a display list.

## Parameters

<i>posX</i>	X coordinate in pixels																																																																																
<i>posY</i>	Y coordinate in pixels																																																																																
<i>numOfColumns</i>	Number of columns to display																																																																																
<i>numOfRows</i>	Number of rows to display																																																																																
<i>fontDesignation</i>	Font Designation 1 - Default font																																																																																
<i>fontID</i>	Font styling <table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															
<i>verticalScroll-ArrowsVisible</i>	Display vertical scroll arrows by default																																																																																
<i>borederedList-Items</i>	Draw border around list items																																																																																
<i>borederedScroll-Arrows</i>	Draw border around scroll arrows (if visible)																																																																																
<i>touchSensitive</i>	List items are touch enabled																																																																																
<i>automatic-Scrolling</i>	Enable automatic scrolling of list when new items exceed display area																																																																																
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory																																																																																

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.112 int lcd\_createList\_Len ( IN int *posX*, IN int *posY*, IN int *numOfColumns*, IN int *numOfRows*, IN int *fontDesignation*, IN int *fontID*, IN int *verticalScrollArrowsVisible*, IN int *borderedListItems*, IN int *borderedScrollArrows*, IN int *touchSensitive*, IN int *automaticScrolling*, OUT BYTE \* *graphicsID*, IN\_OUT int \* *graphicsIDLen* )

Creates a display list.

## Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels

<i>numOfColumns</i>	Number of columns to display																																																																																
<i>numOfRows</i>	Number of rows to display																																																																																
<i>fontDesignation</i>	Font Designation 1 - Default font																																																																																
<i>fontID</i>	Font styling <table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															
<i>verticalScroll-ArrowsVisible</i>	Display vertical scroll arrows by default																																																																																
<i>borederedList-Items</i>	Draw border around list items																																																																																
<i>borederedScroll-Arrows</i>	Draw border around scroll arrows (if visible)																																																																																
<i>touchSensitive</i>	List items are touch enabled																																																																																
<i>automatic-Scrolling</i>	Enable automatic scrolling of list when new items exceed display area																																																																																
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)																																																																																
<i>graphicsIDLen</i>	Length of graphicsID (optional)																																																																																

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.113 int lcd\_customDisplayMode ( IN int enable )

Custom Display Mode Controls the LCD display mode to custom display. Keyboard entry is limited to the Cancel, Clear, Enter and the function keys, if present. PIN entry is not permitted while the reader is in Custom Display Mode

## Parameters

<i>enable</i>	TRUE = enabled, FALSE = disabled
---------------	----------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.114 int lcd\_displayButton ( IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char \* buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE \* graphicsID )

DEPRECATED : please use lcd\_displayButton\_Len(IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char \*buttonLabel, IN int buttonTextColorR, IN int

buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)

Displays an interactive button.

#### Parameters

<i>posX</i>	X coordinate in pixels																																																																																
<i>posY</i>	Y coordinate in pixels																																																																																
<i>buttonWidth</i>	Width of the button																																																																																
<i>buttonHeight</i>	Height of the button																																																																																
<i>fontDesignation</i>	Font designation 1 - Default																																																																																
<i>Font</i>	ID Font styling <table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															
<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap																																																																																
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)																																																																																
<i>buttonTextColor-R</i>	- Red component for foreground color (0 - 255)																																																																																
<i>buttonTextColor-G</i>	- Green component for foreground color (0 - 255)																																																																																
<i>buttonTextColor-B</i>	- Blue component for foreground color (0 - 255)																																																																																
<i>button-Background-ColorR</i>	- Red component for background color (0 - 255)																																																																																

<i>button-Background-ColorG</i>	- Green component for background color (0 - 255)
<i>button-Background-ColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.115 `int lcd_displayButton_Len ( IN int posX, IN int posY, IN int buttonWidth, IN int buttonHeight, IN int fontDesignation, IN int fontID, IN int displayPosition, IN char * buttonLabel, IN int buttonTextColorR, IN int buttonTextColorG, IN int buttonTextColorB, IN int buttonBackgroundColorR, IN int buttonBackgroundColorG, IN int buttonBackgroundColorB, OUT BYTE * graphicsID, IN_OUT int * graphicsIDLen )`

Displays an interactive button.

## Parameters

<i>posX</i>	X coordinate in pixels																																																																																
<i>posY</i>	Y coordinate in pixels																																																																																
<i>buttonWidth</i>	Width of the button																																																																																
<i>buttonHeight</i>	Height of the button																																																																																
<i>fontDesignation</i>	Font designation 1 - Default																																																																																
<i>Font</i>	ID Font styling																																																																																
	<table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															

<i>displayPosition</i>	Button display position 0 - Center on line Y without clearing screen and without word wrap 1 - Center on line Y after clearing screen and without word wrap 2 - Display at (X, Y) without clearing screen and without word wrap 3 - Display at (X, Y) after clearing screen and without word wrap 4 - Center button on screen without clearing screen and without word wrap 5 - Center button on screen after clearing screen and without word wrap 64 - Center on line Y without clearing screen and with word wrap 65 - Center on line Y after clearing the screen and with word wrap 66 - Display at (X, Y) without clearing screen and with word wrap 67 - Display at (X, Y) after clearing screen and with word wrap 68 - Center button on screen without clearing screen and with word wrap 69 - Center button on screen after clearing screen and with word wrap
<i>buttonLabel</i>	Button label text (Maximum: 31 characters)
<i>buttonTextColor-R</i>	- Red component for foreground color (0 - 255)
<i>buttonTextColor-G</i>	- Green component for foreground color (0 - 255)
<i>buttonTextColor-B</i>	- Blue component for foreground color (0 - 255)
<i>button-Background-ColorR</i>	- Red component for background color (0 - 255)
<i>button-Background-ColorG</i>	- Green component for background color (0 - 255)
<i>button-Background-ColorB</i>	- Blue component for background color (0 - 255)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.116 `int lcd_displayParagraph ( IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int displayProperties, IN char * displayText )`

Displays text with scroll feature.

Custom Display Mode must be enabled.

## Parameters

<i>posX</i>	X coordinate in pixels
<i>posY</i>	Y coordinate in pixels
<i>displayWidth</i>	Width of the display area in pixels (Minimum: 40px) 0 or NULL - Use the full width to display text
<i>displayHeight</i>	Height of the display area in pixels (Minimum: 100px) 0 or NULL - Use the full height to display text
<i>fontDesignation</i>	Font designation 1 - Default
<i>fontID</i>	Font styling  Font ID   Height in pixels   Font Properties

| 1 | 13 | Regular | | 2 | 17 | Regular | | 3 | 17 | Bold | | 4 | 22 | Regular | | 5 | 20 | Regular | | 6 | 20 | Bold | | 7 | 29 | Regular | | 8 | 38 | Regular | | 9 | 38 | Bold | | 10 | 58 | Regular | | 11 | 58 | Bold, mono-space | | 12 | 14 | Regular, mono-space, 8 pixels wide | | 13 | 15 | Regular, mono-space, 9 pixels wide | | 14 | 17 | Regular, mono-space, 9 pixels wide | | 15 | 20 | Regular, mono-space, 11 pixels wide | | 16 | 21 | Regular, mono-space, 12 pixels wide | | 17 | 25 | Regular, mono-space, 14 pixels wide | | 18 | 30 | Regular, mono-space, 17 pixels wide |

## Parameters

<i>display-Properties</i>	Display properties for the text 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Center on screen without clearing screen 5 - Center on screen after clearing screen
<i>displayText</i>	Display text (Maximum: 3999 characters plus terminator)

12.16.4.117 int lcd\_displayText ( IN int *posX*, IN int *posY*, IN int *displayWidth*, IN int *displayHeight*, IN int *fontDesignation*, IN int *fontID*, IN int *screenPosition*, IN char \* *displayText*, OUT BYTE \* *graphicsID* )

DEPRECATED : please use lcd\_displayText\_Len(IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char \*displayText, OUT BYTE \*graphicsID, IN\_OUT int \*graphicsIDLen)

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

## Parameters

<i>posX</i>	X coordinate in pixels		
<i>posY</i>	Y coordinate in pixels		
<i>displayWidth</i>	Width of the display area in pixels (optional)		
<i>displayHeight</i>	Height of the display area in pixels (optional)		
<i>fontDesignation</i>	Font designation 1 - Default		
<i>Font</i>	ID Font styling		
	Font ID	Height in pixels	Font Properties
	-----	-----	-----
	1	13	Regular
	2	17	Regular
	3	17	Bold
	4	22	Regular
	5	20	Regular
	6	20	Bold
	7	29	Regular
	8	38	Regular
	9	38	Bold
	10	58	Regular
	11	58	Bold, mono-space
	12	14	Regular, mono-space, 8 pixels wide
	13	15	Regular, mono-space, 9 pixels wide
	14	17	Regular, mono-space, 9 pixels wide
	15	20	Regular, mono-space, 11 pixels wide
	16	21	Regular, mono-space, 12 pixels wide
	17	25	Regular, mono-space, 14 pixels wide
	18	30	Regular, mono-space, 17 pixels wide



<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
<i>displayText</i>	Display text (Maximum: 1900 characters)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional) if graphicsID is NULL, the SDK will not return graphicsID if graphicsID is not NULL, the SDK will return graphicsID, but it will need 4 bytes of memory

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.118 `int lcd_displayText_Len ( IN int posX, IN int posY, IN int displayWidth, IN int displayHeight, IN int fontDesignation, IN int fontID, IN int screenPosition, IN char * displayText, OUT BYTE * graphicsID, IN_OUT int * graphicsIDLen )`

Displays text.

Custom Display Mode must be enabled for custom text. PIN pad entry is not allowed in Custom Display Mode but the Cancel, OK, and Clear keys remain active.

## Parameters

<i>posX</i>	X coordinate in pixels																																																																																
<i>posY</i>	Y coordinate in pixels																																																																																
<i>displayWidth</i>	Width of the display area in pixels (optional)																																																																																
<i>displayHeight</i>	Height of the display area in pixels (optional)																																																																																
<i>fontDesignation</i>	Font designation 1 - Default																																																																																
<i>Font</i>	ID Font styling																																																																																
	<table><tr><td>  Font ID</td><td>  Height in pixels</td><td>  Font Properties</td><td> </td></tr><tr><td>  -----</td><td>  -----</td><td>  -----</td><td> </td></tr><tr><td>  1</td><td>  13</td><td>  Regular</td><td> </td></tr><tr><td>  2</td><td>  17</td><td>  Regular</td><td> </td></tr><tr><td>  3</td><td>  17</td><td>  Bold</td><td> </td></tr><tr><td>  4</td><td>  22</td><td>  Regular</td><td> </td></tr><tr><td>  5</td><td>  20</td><td>  Regular</td><td> </td></tr><tr><td>  6</td><td>  20</td><td>  Bold</td><td> </td></tr><tr><td>  7</td><td>  29</td><td>  Regular</td><td> </td></tr><tr><td>  8</td><td>  38</td><td>  Regular</td><td> </td></tr><tr><td>  9</td><td>  38</td><td>  Bold</td><td> </td></tr><tr><td>  10</td><td>  58</td><td>  Regular</td><td> </td></tr><tr><td>  11</td><td>  58</td><td>  Bold, mono-space</td><td> </td></tr><tr><td>  12</td><td>  14</td><td>  Regular, mono-space, 8 pixels wide</td><td> </td></tr><tr><td>  13</td><td>  15</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  14</td><td>  17</td><td>  Regular, mono-space, 9 pixels wide</td><td> </td></tr><tr><td>  15</td><td>  20</td><td>  Regular, mono-space, 11 pixels wide</td><td> </td></tr><tr><td>  16</td><td>  21</td><td>  Regular, mono-space, 12 pixels wide</td><td> </td></tr><tr><td>  17</td><td>  25</td><td>  Regular, mono-space, 14 pixels wide</td><td> </td></tr><tr><td>  18</td><td>  30</td><td>  Regular, mono-space, 17 pixels wide</td><td> </td></tr></table>	Font ID	Height in pixels	Font Properties		-----	-----	-----		1	13	Regular		2	17	Regular		3	17	Bold		4	22	Regular		5	20	Regular		6	20	Bold		7	29	Regular		8	38	Regular		9	38	Bold		10	58	Regular		11	58	Bold, mono-space		12	14	Regular, mono-space, 8 pixels wide		13	15	Regular, mono-space, 9 pixels wide		14	17	Regular, mono-space, 9 pixels wide		15	20	Regular, mono-space, 11 pixels wide		16	21	Regular, mono-space, 12 pixels wide		17	25	Regular, mono-space, 14 pixels wide		18	30	Regular, mono-space, 17 pixels wide	
Font ID	Height in pixels	Font Properties																																																																															
-----	-----	-----																																																																															
1	13	Regular																																																																															
2	17	Regular																																																																															
3	17	Bold																																																																															
4	22	Regular																																																																															
5	20	Regular																																																																															
6	20	Bold																																																																															
7	29	Regular																																																																															
8	38	Regular																																																																															
9	38	Bold																																																																															
10	58	Regular																																																																															
11	58	Bold, mono-space																																																																															
12	14	Regular, mono-space, 8 pixels wide																																																																															
13	15	Regular, mono-space, 9 pixels wide																																																																															
14	17	Regular, mono-space, 9 pixels wide																																																																															
15	20	Regular, mono-space, 11 pixels wide																																																																															
16	21	Regular, mono-space, 12 pixels wide																																																																															
17	25	Regular, mono-space, 14 pixels wide																																																																															
18	30	Regular, mono-space, 17 pixels wide																																																																															

<i>screenPosition</i>	Display position 0 - Center on line Y without clearing screen 1 - Center on line Y after clearing screen 2 - Display at (X, Y) without clearing screen 3 - Display at (X, Y) after clearing screen 4 - Display at center of screen without clearing screen 5 - Display at center of screen after clearing screen 6 - Display text right-justified without clearing screen 7 - Display text right-justified after clearing screen
<i>displayText</i>	Display text (Maximum: 1900 characters)
<i>graphicsID</i>	A four byte array containing the ID of the created element (optional)
<i>graphicsIDLen</i>	Length of graphicsID (optional)

#### Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.119 `int lcd_getInputEvent ( IN int timeout, OUT int * dataReceived, OUT BYTE * eventType, OUT BYTE * graphicsID, OUT BYTE * eventData )`

DEPRECATED : please use `lcd_getInputEvent_Len(IN int timeout, OUT int *dataReceived, OUT BYTE *eventType, IN_OUT int *eventTypeLen, OUT BYTE *graphicsID, IN_OUT int *graphicsIDLen, OUT BYTE *eventData, IN_OUT int *eventDataLen)`

Requests input from the reader.

#### Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable Byte 1: State (1 = Pressed, other values RFU) Byte 2 - n: Null terminated caption
-----	-----	-----
Checkbox Event	00030001h	Length = 1 byte Byte 1: State (1 = Checked, 0 = Unchecked)
-----	-----	-----
Line Item Event	00030002h	Length = 5 bytes Byte 1: State (1 = Item Selected, other values RFU) Byte 2 - n: Caption of the selected item
-----	-----	-----
Keypad Event	00030003h	Length - 3 bytes Byte 1: State (1 = key pressed, 2 = key released, other values RFU) Byte 2 - 3: Key pressed and Key release

		0030h - KEYPAD_KEY_0
		0031h - KEYPAD_KEY_1
		0032h - KEYPAD_KEY_2
		0033h - KEYPAD_KEY_3
		0034h - KEYPAD_KEY_4
		0035h - KEYPAD_KEY_5
		0036h - KEYPAD_KEY_6
		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
-----	-----	-----
-----	-----	-----
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
-----	-----	-----
-----	-----	-----
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
-----	-----	-----
-----	-----	-----
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
-----	-----	-----
-----	-----	-----
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name

		Byte 34 - 65: Radio button caption
--	--	------------------------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.120 `int lcd_getInputEvent_Len ( IN int timeout, OUT int * dataReceived, OUT BYTE * eventType, IN_OUT int * eventTypeLen, OUT BYTE * graphicsID, IN_OUT int * graphicsIDLen, OUT BYTE * eventData, IN_OUT int * eventDataLen )`

Requests input from the reader.

## Parameters

<i>timeout</i>	Timeout amount in seconds 0 - No timeout
<i>dataReceived</i>	Indicates if an event occurred and data was received 0 - No data received 1 - Data received
<i>eventType</i>	The event type (required to be at least 4 bytes), see table below
<i>eventTypeLen</i>	Length of eventType
<i>graphicsID</i>	The graphicID of the event (required to be at least 4 bytes)
<i>graphicsIDLen</i>	length of graphicID
<i>eventData</i>	The event data, see table below (required to be at least 73 bytes)

Event Type	Value (4 bytes)	Event Specific Data
Button Event	00030000h	Length = Variable Byte 1: State (1 = Pressed, other values RFU) Byte 2 - n: Null terminated caption ----- -----
Checkbox Event	00030001h	Length = 1 byte Byte 1: State (1 = Checked, 0 = Unchecked) ----- -----
Line Item Event	00030002h	Length = 5 bytes Byte 1: State (1 = Item Selected, other values RFU) Byte 2 - n: Caption of the selected item ----- -----
Keypad Event	00030003h	Length - 3 bytes Byte 1: State (1 = key pressed, 2 = key released, other values RFU) Byte 2 - 3: Key pressed and Key release 0030h - KEYPAD_KEY_0 0031h - KEYPAD_KEY_1 0032h - KEYPAD_KEY_2 0033h - KEYPAD_KEY_3 0034h - KEYPAD_KEY_4 0035h - KEYPAD_KEY_5 0036h - KEYPAD_KEY_6

		0037h - KEYPAD_KEY_7
		0038h - KEYPAD_KEY_8
		0039h - KEYPAD_KEY_9
		Byte 2 - 3: Only Key pressed
		000Dh - KEYPAD_KEY_ENTER
		0008h - KEYPAD_KEY_CLEAR
		001Bh - KEYPAD_KEY_CANCEL
		0070h - FUNC_KEY_F1 (Vend III)
		0071h - FUNC_KEY_F2 (Vend III)
		0072h - FUNC_KEY_F3 (Vend III)
		0073h - FUNC_KEY_F4 (Vend III)
-----	-----	-----
-----	-----	-----
Touchscreen Event	00030004h	Length = 1 - 33 bytes
		Byte 1: State (not used)
		Byte 2 - 33: Image name (zero terminated)
-----	-----	-----
-----	-----	-----
Slideshow Event	00030005h	Length = 1 byte
		Byte 1: State (not used)
-----	-----	-----
-----	-----	-----
Transaction Event	00030006h	Length = 9 bytes
		Byte 1: State (not used)
		Byte 2 - 5: Card type (0 = unknown)
		Byte 6 - 9: Status - A four byte, big-endian field
		Byte 9 is used to store the 1-byte status code
		00 - SUCCESS
		08 - TIMEOUT
		0A - FAILED
		This is not related to the extended status codes
-----	-----	-----
-----	-----	-----
Radio Button Event	00030007h	Length = 73 bytes
		Byte 1: State (1 = Change ins selected button, other values RFU)
		Byte 2 - 33: Null terminated group name
		Byte 34 - 65: Radio button caption

## Parameters

<i>eventDataLen</i>	Length of eventData
---------------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.121 int lcd\_getInputFieldValue ( IN BYTE \* *graphicId*, OUT BYTE \* *retData*, IN\_OUT int \* *retDataLen* )

Get the keypad data that was entered into the specified Input Field.

## Parameters

<i>graphicsID</i>	The graphicID of the input field (required to be 4 bytes)
<i>retData</i>	return keypad data
<i>retDataLen</i>	The length of retData

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.122 `int lcd_getSelectedListItem ( IN BYTE * listGraphicsID, OUT char * itemID )`

DEPRECATED : please use [lcd\\_getSelectedListItem\\_Len\(IN BYTE \\*listGraphicsID, OUT char \\*itemID, IN\\_OUT int \\*itemIDLen\)](#)

Retrieves the selected item's ID.

## Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'

12.16.4.123 `int lcd_getSelectedListItem_Len ( IN BYTE * listGraphicsID, OUT char * itemID, IN_OUT int * itemIDLen )`

Retrieves the selected item's ID.

## Parameters

<i>listGraphicsID</i>	Existing list's graphics ID (4 byte array) that is provided during creation
<i>itemID</i>	The selected item's ID (Maximum: 32 characters) Need 33 bytes of memory including '\0'
<i>itemIDLen</i>	Length of itemID

12.16.4.124 `int lcd_resetInitialState ( )`

Reset to Initial State This command places the reader UI into the idle state and displays the appropriate idle display.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.125 `int lcd_setBackgroundImage ( IN char * file, IN int fileLen, IN int enable )`

Set Background Image You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

## Parameters

<i>file</i>	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
<i>fileLen</i>	Length of files
<i>enable</i>	TRUE = Use Background Image, FALSE = Use Background Color

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.126 `int lcd_setDisplayImage ( IN char * file, IN int fileLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int clearScreen )`

Set Display Image You must send images to the reader's memory and send a Start Custom Mode command to the reader before it will respond to Image commands. Image files must be in .bmp or .png format.

## Parameters

<i>file</i>	Complete path and file name of the file you want to use. Example "file.png" will put in root directory, while "ss/file.png" will put in ss directory (which must exist)
<i>fileLen</i>	Length of files
<i>posX</i>	X coordinate in pixels, Range 0-271
<i>posY</i>	Y coordinate in pixels, Range 0-479
<i>posMode</i>	Position Mode <ul style="list-style-type: none"> <li>• 0 = Center on Line Y</li> <li>• 1 = Display at (X,Y)</li> <li>• 2 - Center on screen</li> </ul>
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>clearScreen</i>	TRUE = Clear screen

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.127 `int lcd_setForeBackColor ( IN BYTE * foreRGB, IN int foreRGBLen, IN BYTE * backRGB, IN int backRGBLen )`

Set Foreground and Background Color This command sets the foreground and background colors of the LCD.

## Parameters

<i>foreRGB</i>	Foreground RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>Length</i>	of foreRGB. Must be 3.
<i>backRGB</i>	Background RGB. 000000 = black, FF0000 = red, 00FF00 = green, 0000FF = blue, FFFFFFFF = white
<i>Length</i>	of backRGB. Must be 3.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.128 `int lcd_startSlideShow ( IN char * files, IN int filesLen, IN int posX, IN int posY, IN int posMode, IN int touchEnable, IN int recursion, IN int touchTerminate, IN int delay, IN int loops, IN int clearScreen )`

Start slide show You must send images to the reader??s memory and send a Start Custom Mode command to the reader before it will respond to this commands. Image files must be in .bmp or .png format.



## Parameters

<i>files</i>	Complete paths and file names of the files you want to use, separated by commas. If a directory is specified, all files in the directory are displayed
<i>filesLen</i>	Length of files
<i>posX</i>	X coordinate in pixels, Range 0-271
<i>posY</i>	Y coordinate in pixels, Range 0-479
<i>posMode</i>	Position Mode <ul style="list-style-type: none"> <li>• 0 = Center on Line Y</li> <li>• 1 = Display at (X,Y)</li> <li>• 2 - Center on screen</li> </ul>
<i>touchEnable</i>	TRUE = Image is touch sensitive
<i>recursion</i>	TRUE = Recursively follow directorys in list
<i>touchTerminate</i>	TRUE = Terminate slideshow on touch (if touch enabled)
<i>delay</i>	Number of seconds between image displays
<i>loops</i>	Number of display loops. A zero indicates continuous display
<i>clearScreen</i>	TRUE = Clear screen

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.129 int msr\_cancelMSRSwipe ( )

Disable MSR Swipe Cancels MSR swipe request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.130 int msr\_flushTrackData ( )

Flush Track Data Clears any track data being retained in memory by future PIN Block request.

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.131 void msr\_registerCallBk ( pMSR\_callBack pMSRf )

To register the msr callback function to get the MSR card data. (Pass NULL to disable the callback.)

## 12.16.4.132 void msr\_registerCallBkp ( pMSR\_callBackp pMSRf )

To register the msr callback function to get the MSR card data pointer. (Pass NULL to disable the callback.)

## 12.16.4.133 int msr\_startMSRSwipe ( IN int \_timeout )

Start MSR Swipe Enables MSR, waiting for swipe to occur. Allows track selection. Returns IDTMSRData instance to deviceDelegate::swipeMSRData:()

## Parameters

<i>timeout</i>	Swipe Timeout Value
----------------	---------------------

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#) Note: if auto poll mode is on, it will return command not allowed error

12.16.4.134 void parseMSRData ( IN BYTE \* *resData*, IN int *resLen*, IN\_OUT IDTMSRData \* *cardData* )

Parser the MSR data from the buffer into IDTMSTData structure

## Parameters

<i>resData</i>	MSR card data buffer
<i>resLen</i>	the length of <i>resData</i>
<i>cardData</i>	the parser result with IDTMSTData structure

12.16.4.135 int pin\_getEncryptedOnlinePIN ( IN int *keyType*, IN int *timeout* )

Get Encrypted DUKPT PIN

Requests PIN Entry for online authorization. PIN block and KSN returned in callback function DeviceState.- TransactionData with *cardData.pin\_pinblock*. A swipe must be captured first before this function can execute

## Parameters

<i>keyType</i>	PIN block key type. Valid values 0,3 for TDES, 4 for AES
<i>timeout</i>	PIN entry timeout

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.136 int pin\_getPAN ( IN int *getCSC*, IN int *timeout* )

Get PAN

Requests PAN Entry on pinpad

## Parameters

<i>getCSC</i>	Include Customer Service Code (also known as CVV, CVC)
<i>timeout</i>	PAN entry timeout

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

12.16.4.137 int pin\_promptCreditDebit ( IN char \* *currencySymbol*, IN int *currencySymbolLen*, IN char \* *displayAmount*, IN int *displayAmountLen*, IN int *timeout*, OUT BYTE \* *retData*, IN\_OUT int \* *retDataLen* )

Prompt for Credit or Debit

Requests prompt for Credit or Debit. Response returned in callback function as DeviceState.MenuItem with data MENU\_SELECTION\_CREDIT = 0, MENU\_SELECTION\_DEBIT = 1

## Parameters

<i>currencySymbol</i>	Allowed values are \$ (0x24), ???(0xA5), ???(0xA3), ???(0xA4), or NULL
<i>currencySymbol-Len</i>	length of currencySymbol
<i>displayAmount</i>	Amount to display (can be NULL)
<i>displayAmount-Len</i>	length of displayAmount
<i>timeout</i>	Menu entry timeout. Valid values 2-20 seconds

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getResponseCodeString\(\)](#)

## 12.16.4.138 void pin\_registerCallBk ( pPIN\_callBack pPINf )

To register the pin callback function to get the PINPad data. (Pass NULL to disable the callback.)

## 12.16.4.139 void registerHotplugCallBk ( pMessageHotplug pMsgHotplug )

To register the USB HID hot-plug callback function which implemented in the application to monitor the hotplug message from the SDK.

## 12.16.4.140 void registerLogCallBk ( pSendDataLog pFSend, pReadDataLog pFRead )

To register the log callback function which implemented in the application to monitor sending/reading data between application and reader.

## 12.16.4.141 char\* SDK\_Version ( )

To Get SDK version

## Returns

return the SDK version string

## 12.16.4.142 int setAbsoluteLibraryPath ( const char \* absoluteLibraryPath )

Set the path to use when searching for ID TECH's libraries. If this is not set, the libraries will be searched for with the system's default procedures.

## Parameters

<i>absoluteLibrary-Path</i>	The absolute path to ID TECH's libraries.
-----------------------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

## 12.16.4.143 int ws\_deleteSSLCert ( IN char \* name, IN int nameLen )

Delete SSL Certificate Deletes a SSL Certificate by name

## Parameters

<i>name</i>	Name of certificate to delete
<i>nameLen</i>	Certificate Name Length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.144 `int ws_getCertChainType ( OUT int * type )`

Get Certificate Chain Type Returns indicator for using test/production certificate chain

## Parameters

<i>type</i>	0 = test certificate chain, 1 = production certificate chain
-------------	--

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.145 `int ws_loadSSLCert ( IN char * name, IN int nameLen, IN char * dataDER, IN int dataDERLen )`

Load SSL Certificate Loads a SSL certificate

## Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.146 `int ws_requestCSR ( OUT RequestCSR * csr )`

Request CSR Requests 3 sets of public keys: encrypting Keys, signing/validating keys, signing/validating 3rd party apps

## Parameters

<i>csr</i>	RequestCSR structure to return the data
------------	---

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.147 `int ws_revokeSSLCert ( IN char * name, IN int nameLen )`

Revoke SSL Certificate Revokes a SSL Certificate by name

## Parameters

<i>name</i>	Name of certificate to revoke
<i>nameLen</i>	Certificate Name Length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

12.16.4.148 int ws\_updateRootCertificate ( IN char \* *name*, IN int *nameLen*, IN char \* *dataDER*, IN int *dataDERLen*, IN char \* *signature*, IN int *signatureLen* )

Update Root Certificate Updates the root certificate

## Parameters

<i>name</i>	Certificate Name
<i>nameLen</i>	Certificate Name Length
<i>dataDER</i>	DER encoded certificate data
<i>dataDERLen</i>	DER encoded certificate data length
<i>signature</i>	Future Root CA signed (RSASSA PSS SHA256) by current Root CA
<i>signature</i>	length

## Returns

RETURN\_CODE: Values can be parsed with [device\\_getIDGStatusCodeString\(\)](#)

# Index

comm\_registerHTTPCallback  
liblDT\_MiniSmartII.h, [318](#)  
liblDT\_NEO2.h, [367](#)  
liblDT\_SREDKey2.h, [519](#)  
liblDT\_UniPayI\_V.h, [538](#)  
liblDT\_Vendi.h, [563](#)  
liblDT\_VP3300\_AJ.h, [585](#)  
liblDT\_VP3300\_BT.h, [624](#)  
liblDT\_VP3300\_COM.h, [660](#)  
liblDT\_VP3300\_USB.h, [697](#)  
liblDT\_VP8800.h, [734](#)  
comm\_registerV4Callback  
liblDT\_MiniSmartII.h, [318](#)  
liblDT\_NEO2.h, [367](#)  
liblDT\_SREDKey2.h, [519](#)  
liblDT\_UniPayI\_V.h, [538](#)  
liblDT\_Vendi.h, [563](#)  
liblDT\_VP3300\_AJ.h, [585](#)  
liblDT\_VP3300\_BT.h, [624](#)  
liblDT\_VP3300\_COM.h, [660](#)  
liblDT\_VP3300\_USB.h, [697](#)  
liblDT\_VP8800.h, [734](#)  
config\_getBeeperController  
liblDT\_Augusta.h, [52](#)  
liblDT\_Device.h, [111](#)  
liblDT\_MiniSmartII.h, [318](#)  
config\_getEncryptionControl  
liblDT\_Augusta.h, [53](#)  
liblDT\_Device.h, [111](#)  
liblDT\_MiniSmartII.h, [318](#)  
config\_getLEDController  
liblDT\_Augusta.h, [53](#)  
liblDT\_Device.h, [111](#)  
liblDT\_MiniSmartII.h, [319](#)  
config\_getModelNumber  
liblDT\_Augusta.h, [54](#)  
liblDT\_Device.h, [112](#)  
liblDT\_L100.h, [287](#)  
liblDT\_MiniSmartII.h, [319](#)  
liblDT\_NEO2.h, [367](#)  
liblDT\_SpectrumPro.h, [479](#)  
liblDT\_SREDKey2.h, [521](#)  
config\_getModelNumber\_Len  
liblDT\_Augusta.h, [54](#)  
liblDT\_Device.h, [112](#)  
liblDT\_L100.h, [287](#)  
liblDT\_MiniSmartII.h, [319](#)  
liblDT\_NEO2.h, [367](#)  
liblDT\_SpectrumPro.h, [479](#)  
liblDT\_SREDKey2.h, [521](#)  
config\_getSerialNumber  
liblDT\_Augusta.h, [54](#)  
liblDT\_Device.h, [112](#)  
liblDT\_KioskIII.h, [265](#)  
liblDT\_L100.h, [287](#)  
liblDT\_MiniSmartII.h, [320](#)  
liblDT\_NEO2.h, [368](#)  
liblDT\_PipReader.h, [456](#)  
liblDT\_SpectrumPro.h, [479](#)  
liblDT\_SREDKey2.h, [521](#)  
liblDT\_UniPayI\_V.h, [540](#)  
liblDT\_Vendi.h, [563](#)  
liblDT\_VP3300\_AJ.h, [587](#)  
liblDT\_VP3300\_BT.h, [624](#)  
liblDT\_VP3300\_COM.h, [660](#)  
liblDT\_VP3300\_USB.h, [697](#)  
liblDT\_VP8800.h, [736](#)  
config\_getSerialNumber\_Len  
liblDT\_Augusta.h, [54](#)  
liblDT\_Device.h, [112](#)  
liblDT\_KioskIII.h, [265](#)  
liblDT\_L100.h, [287](#)  
liblDT\_MiniSmartII.h, [320](#)  
liblDT\_NEO2.h, [368](#)  
liblDT\_PipReader.h, [456](#)  
liblDT\_SpectrumPro.h, [480](#)  
liblDT\_SREDKey2.h, [521](#)  
liblDT\_UniPayI\_V.h, [540](#)  
liblDT\_Vendi.h, [564](#)  
liblDT\_VP3300\_AJ.h, [587](#)  
liblDT\_VP3300\_BT.h, [625](#)  
liblDT\_VP3300\_COM.h, [661](#)  
liblDT\_VP3300\_USB.h, [697](#)  
liblDT\_VP8800.h, [736](#)  
config\_setBeeperController  
liblDT\_Augusta.h, [55](#)  
liblDT\_Device.h, [114](#)  
liblDT\_MiniSmartII.h, [320](#)  
config\_setCmdTimeOutDuration  
liblDT\_Device.h, [114](#)  
liblDT\_NEO2.h, [368](#)  
config\_setEncryptionControl  
liblDT\_Augusta.h, [55](#)  
liblDT\_Device.h, [114](#)  
liblDT\_MiniSmartII.h, [320](#)  
config\_setLEDController  
liblDT\_Augusta.h, [55](#)  
liblDT\_Device.h, [115](#)

- libIDT\_MiniSmartII.h, [322](#)
- ctls\_activateTransaction
  - libIDT\_Device.h, [115](#)
  - libIDT\_KioskIII.h, [265](#)
  - libIDT\_NEO2.h, [368](#)
  - libIDT\_PipReader.h, [456](#)
  - libIDT\_Vendi.h, [564](#)
  - libIDT\_VP3300\_AJ.h, [587](#)
  - libIDT\_VP3300\_BT.h, [625](#)
  - libIDT\_VP3300\_COM.h, [661](#)
  - libIDT\_VP3300\_USB.h, [697](#)
  - libIDT\_VP8800.h, [736](#)
- ctls\_cancelTransaction
  - libIDT\_Device.h, [116](#)
  - libIDT\_KioskIII.h, [267](#)
  - libIDT\_NEO2.h, [369](#)
  - libIDT\_PipReader.h, [458](#)
  - libIDT\_Vendi.h, [565](#)
  - libIDT\_VP3300\_AJ.h, [588](#)
  - libIDT\_VP3300\_BT.h, [626](#)
  - libIDT\_VP3300\_COM.h, [662](#)
  - libIDT\_VP3300\_USB.h, [698](#)
  - libIDT\_VP8800.h, [737](#)
- ctls\_displayOnlineAuthResult
  - libIDT\_Device.h, [116](#)
  - libIDT\_NEO2.h, [370](#)
  - libIDT\_VP8800.h, [737](#)
- ctls\_getAllConfigurationGroups
  - libIDT\_Device.h, [118](#)
  - libIDT\_KioskIII.h, [268](#)
  - libIDT\_NEO2.h, [370](#)
  - libIDT\_PipReader.h, [459](#)
  - libIDT\_Vendi.h, [565](#)
  - libIDT\_VP3300\_AJ.h, [588](#)
  - libIDT\_VP3300\_BT.h, [626](#)
  - libIDT\_VP3300\_COM.h, [662](#)
  - libIDT\_VP3300\_USB.h, [699](#)
  - libIDT\_VP8800.h, [738](#)
- ctls\_getConfigurationGroup
  - libIDT\_Device.h, [118](#)
  - libIDT\_KioskIII.h, [268](#)
  - libIDT\_NEO2.h, [370](#)
  - libIDT\_PipReader.h, [459](#)
  - libIDT\_Vendi.h, [565](#)
  - libIDT\_VP3300\_AJ.h, [589](#)
  - libIDT\_VP3300\_BT.h, [626](#)
  - libIDT\_VP3300\_COM.h, [662](#)
  - libIDT\_VP3300\_USB.h, [699](#)
  - libIDT\_VP8800.h, [738](#)
- ctls\_registerCallBk
  - libIDT\_KioskIII.h, [268](#)
  - libIDT\_NEO2.h, [370](#)
  - libIDT\_PipReader.h, [459](#)
  - libIDT\_SREDKey2.h, [521](#)
  - libIDT\_Vendi.h, [566](#)
  - libIDT\_VP3300\_AJ.h, [589](#)
  - libIDT\_VP3300\_BT.h, [627](#)
  - libIDT\_VP3300\_COM.h, [663](#)
- libIDT\_VP3300\_USB.h, [699](#)
- libIDT\_VP8800.h, [738](#)
- ctls\_registerCallBkp
  - libIDT\_KioskIII.h, [268](#)
  - libIDT\_NEO2.h, [371](#)
  - libIDT\_PipReader.h, [459](#)
  - libIDT\_SREDKey2.h, [522](#)
  - libIDT\_Vendi.h, [566](#)
  - libIDT\_VP3300\_AJ.h, [589](#)
  - libIDT\_VP3300\_BT.h, [627](#)
  - libIDT\_VP3300\_COM.h, [663](#)
  - libIDT\_VP3300\_USB.h, [699](#)
  - libIDT\_VP8800.h, [738](#)
- ctls\_removeAllApplicationData
  - libIDT\_Device.h, [118](#)
  - libIDT\_KioskIII.h, [268](#)
  - libIDT\_NEO2.h, [371](#)
  - libIDT\_PipReader.h, [459](#)
  - libIDT\_Vendi.h, [566](#)
  - libIDT\_VP3300\_AJ.h, [589](#)
  - libIDT\_VP3300\_BT.h, [627](#)
  - libIDT\_VP3300\_COM.h, [663](#)
  - libIDT\_VP3300\_USB.h, [699](#)
  - libIDT\_VP8800.h, [738](#)
- ctls\_removeAllCAPK
  - libIDT\_Device.h, [118](#)
  - libIDT\_KioskIII.h, [268](#)
  - libIDT\_NEO2.h, [371](#)
  - libIDT\_PipReader.h, [459](#)
  - libIDT\_Vendi.h, [566](#)
  - libIDT\_VP3300\_AJ.h, [589](#)
  - libIDT\_VP3300\_BT.h, [627](#)
  - libIDT\_VP3300\_COM.h, [663](#)
  - libIDT\_VP3300\_USB.h, [699](#)
  - libIDT\_VP8800.h, [738](#)
- ctls\_removeApplicationData
  - libIDT\_Device.h, [118](#)
  - libIDT\_KioskIII.h, [269](#)
  - libIDT\_NEO2.h, [371](#)
  - libIDT\_PipReader.h, [460](#)
  - libIDT\_Vendi.h, [566](#)
  - libIDT\_VP3300\_AJ.h, [589](#)
  - libIDT\_VP3300\_BT.h, [627](#)
  - libIDT\_VP3300\_COM.h, [663](#)
  - libIDT\_VP3300\_USB.h, [700](#)
  - libIDT\_VP8800.h, [739](#)
- ctls\_removeCAPK
  - libIDT\_Device.h, [119](#)
  - libIDT\_KioskIII.h, [269](#)
  - libIDT\_NEO2.h, [371](#)
  - libIDT\_PipReader.h, [460](#)
  - libIDT\_Vendi.h, [566](#)
  - libIDT\_VP3300\_AJ.h, [590](#)
  - libIDT\_VP3300\_BT.h, [627](#)
  - libIDT\_VP3300\_COM.h, [663](#)
  - libIDT\_VP3300\_USB.h, [700](#)
  - libIDT\_VP8800.h, [739](#)
- ctls\_removeConfigurationGroup

- libIDT\_Device.h, 119
- libIDT\_KioskIII.h, 269
- libIDT\_NEO2.h, 371
- libIDT\_PipReader.h, 460
- libIDT\_Vendi.h, 567
- libIDT\_VP3300\_AJ.h, 590
- libIDT\_VP3300\_BT.h, 628
- libIDT\_VP3300\_COM.h, 664
- libIDT\_VP3300\_USB.h, 700
- libIDT\_VP8800.h, 739
- ctls\_retrieveAIDList
  - libIDT\_Device.h, 119
  - libIDT\_KioskIII.h, 269
  - libIDT\_NEO2.h, 372
  - libIDT\_PipReader.h, 460
  - libIDT\_Vendi.h, 567
  - libIDT\_VP3300\_AJ.h, 590
  - libIDT\_VP3300\_BT.h, 628
  - libIDT\_VP3300\_COM.h, 664
  - libIDT\_VP3300\_USB.h, 700
  - libIDT\_VP8800.h, 739
- ctls\_retrieveApplicationData
  - libIDT\_Device.h, 119
  - libIDT\_KioskIII.h, 270
  - libIDT\_NEO2.h, 372
  - libIDT\_PipReader.h, 461
  - libIDT\_Vendi.h, 567
  - libIDT\_VP3300\_AJ.h, 590
  - libIDT\_VP3300\_BT.h, 628
  - libIDT\_VP3300\_COM.h, 664
  - libIDT\_VP3300\_USB.h, 701
  - libIDT\_VP8800.h, 740
- ctls\_retrieveCAPK
  - libIDT\_Device.h, 120
  - libIDT\_KioskIII.h, 270
  - libIDT\_NEO2.h, 372
  - libIDT\_PipReader.h, 461
  - libIDT\_Vendi.h, 567
  - libIDT\_VP3300\_AJ.h, 591
  - libIDT\_VP3300\_BT.h, 628
  - libIDT\_VP3300\_COM.h, 664
  - libIDT\_VP3300\_USB.h, 701
  - libIDT\_VP8800.h, 740
- ctls\_retrieveCAPKList
  - libIDT\_Device.h, 121
  - libIDT\_KioskIII.h, 271
  - libIDT\_NEO2.h, 373
  - libIDT\_PipReader.h, 462
  - libIDT\_Vendi.h, 568
  - libIDT\_VP3300\_AJ.h, 591
  - libIDT\_VP3300\_BT.h, 629
  - libIDT\_VP3300\_COM.h, 665
  - libIDT\_VP3300\_USB.h, 702
  - libIDT\_VP8800.h, 741
- ctls\_retrieveTerminalData
  - libIDT\_Device.h, 121
  - libIDT\_KioskIII.h, 271
  - libIDT\_NEO2.h, 373
- libIDT\_PipReader.h, 462
- libIDT\_Vendi.h, 568
- libIDT\_VP3300\_AJ.h, 592
- libIDT\_VP3300\_BT.h, 629
- libIDT\_VP3300\_COM.h, 665
- libIDT\_VP3300\_USB.h, 702
- libIDT\_VP8800.h, 741
- ctls\_setApplicationData
  - libIDT\_Device.h, 121
  - libIDT\_KioskIII.h, 271
  - libIDT\_NEO2.h, 374
  - libIDT\_PipReader.h, 462
  - libIDT\_Vendi.h, 569
  - libIDT\_VP3300\_AJ.h, 592
  - libIDT\_VP3300\_BT.h, 630
  - libIDT\_VP3300\_COM.h, 666
  - libIDT\_VP3300\_USB.h, 702
  - libIDT\_VP8800.h, 741
- ctls\_setCAPK
  - libIDT\_Device.h, 121
  - libIDT\_KioskIII.h, 271
  - libIDT\_NEO2.h, 374
  - libIDT\_PipReader.h, 462
  - libIDT\_Vendi.h, 569
  - libIDT\_VP3300\_AJ.h, 592
  - libIDT\_VP3300\_BT.h, 630
  - libIDT\_VP3300\_COM.h, 666
  - libIDT\_VP3300\_USB.h, 702
  - libIDT\_VP8800.h, 741
- ctls\_setConfigurationGroup
  - libIDT\_Device.h, 123
  - libIDT\_KioskIII.h, 272
  - libIDT\_NEO2.h, 375
  - libIDT\_PipReader.h, 463
  - libIDT\_Vendi.h, 570
  - libIDT\_VP3300\_AJ.h, 593
  - libIDT\_VP3300\_BT.h, 631
  - libIDT\_VP3300\_COM.h, 667
  - libIDT\_VP3300\_USB.h, 703
  - libIDT\_VP8800.h, 742
- ctls\_setTerminalData
  - libIDT\_Device.h, 123
  - libIDT\_KioskIII.h, 272
  - libIDT\_NEO2.h, 375
  - libIDT\_PipReader.h, 463
  - libIDT\_Vendi.h, 570
  - libIDT\_VP3300\_AJ.h, 593
  - libIDT\_VP3300\_BT.h, 631
  - libIDT\_VP3300\_COM.h, 667
  - libIDT\_VP3300\_USB.h, 703
  - libIDT\_VP8800.h, 742
- ctls\_startTransaction
  - libIDT\_Device.h, 124
  - libIDT\_KioskIII.h, 273
  - libIDT\_NEO2.h, 375
  - libIDT\_PipReader.h, 464
  - libIDT\_Vendi.h, 570
  - libIDT\_VP3300\_AJ.h, 594



- libIDT\_VP3300\_BT.h, [631](#)
- libIDT\_VP3300\_COM.h, [667](#)
- libIDT\_VP3300\_USB.h, [704](#)
- libIDT\_VP8800.h, [743](#)
- device\_SendDataCommand
  - libIDT\_Augusta.h, [72](#)
  - libIDT\_Device.h, [162](#)
  - libIDT\_L100.h, [301](#)
  - libIDT\_MiniSmartII.h, [337](#)
  - libIDT\_SpectrumPro.h, [496](#)
  - libIDT\_SREDKey2.h, [526](#)
- device\_SendDataCommandITP
  - libIDT\_Device.h, [162](#)
  - libIDT\_SREDKey2.h, [526](#)
- device\_SendDataCommandNEO
  - libIDT\_Device.h, [163](#)
  - libIDT\_KioskIII.h, [279](#)
  - libIDT\_NEO2.h, [405](#)
  - libIDT\_PipReader.h, [470](#)
  - libIDT\_SREDKey2.h, [526](#)
  - libIDT\_UniPayI\_V.h, [544](#)
  - libIDT\_Vendi.h, [577](#)
  - libIDT\_VP3300\_AJ.h, [602](#)
  - libIDT\_VP3300\_BT.h, [639](#)
  - libIDT\_VP3300\_COM.h, [675](#)
  - libIDT\_VP3300\_USB.h, [712](#)
  - libIDT\_VP8800.h, [755](#)
- device\_activateTransaction
  - libIDT\_Device.h, [125](#)
  - libIDT\_NEO2.h, [377](#)
  - libIDT\_VP3300\_AJ.h, [595](#)
  - libIDT\_VP3300\_BT.h, [632](#)
  - libIDT\_VP3300\_COM.h, [668](#)
  - libIDT\_VP3300\_USB.h, [705](#)
  - libIDT\_VP8800.h, [744](#)
- device\_buzzerOnOff
  - libIDT\_Device.h, [126](#)
  - libIDT\_NEO2.h, [378](#)
- device\_calibrateParameters
  - libIDT\_Device.h, [126](#)
  - libIDT\_VP8800.h, [745](#)
- device\_cancelTransaction
  - libIDT\_Device.h, [126](#)
  - libIDT\_NEO2.h, [378](#)
  - libIDT\_VP3300\_AJ.h, [596](#)
  - libIDT\_VP3300\_BT.h, [633](#)
  - libIDT\_VP3300\_COM.h, [669](#)
  - libIDT\_VP3300\_USB.h, [706](#)
  - libIDT\_VP8800.h, [745](#)
- device\_cancelTransactionSilent
  - libIDT\_Device.h, [127](#)
  - libIDT\_NEO2.h, [378](#)
- device\_close
  - libIDT\_Augusta.h, [56](#)
  - libIDT\_Device.h, [127](#)
  - libIDT\_KioskIII.h, [274](#)
  - libIDT\_L100.h, [288](#)
  - libIDT\_MiniSmartII.h, [322](#)
  - libIDT\_NEO2.h, [378](#)
  - libIDT\_PipReader.h, [465](#)
  - libIDT\_SpectrumPro.h, [480](#)
  - libIDT\_SREDKey2.h, [522](#)
  - libIDT\_UniPayI\_V.h, [540](#)
  - libIDT\_Vendi.h, [571](#)
  - libIDT\_VP3300\_AJ.h, [596](#)
  - libIDT\_VP3300\_BT.h, [634](#)
  - libIDT\_VP3300\_COM.h, [670](#)
  - libIDT\_VP3300\_USB.h, [706](#)
  - libIDT\_VP8800.h, [745](#)
- device\_configureButtons
  - libIDT\_Device.h, [127](#)
  - libIDT\_NEO2.h, [378](#)
- device\_controlBeep
  - libIDT\_Augusta.h, [56](#)
  - libIDT\_Device.h, [127](#)
  - libIDT\_MiniSmartII.h, [322](#)
- device\_controlIndicator
  - libIDT\_Device.h, [128](#)
  - libIDT\_VP8800.h, [745](#)
- device\_controlLED
  - libIDT\_Augusta.h, [57](#)
  - libIDT\_Device.h, [128](#)
  - libIDT\_MiniSmartII.h, [323](#)
- device\_controlLED\_ICC
  - libIDT\_Augusta.h, [57](#)
  - libIDT\_Device.h, [129](#)
  - libIDT\_MiniSmartII.h, [323](#)
- device\_controlLED\_MSR
  - libIDT\_Augusta.h, [57](#)
  - libIDT\_Device.h, [129](#)
  - libIDT\_MiniSmartII.h, [323](#)
- device\_controlUserInterface
  - libIDT\_Device.h, [130](#)
  - libIDT\_KioskIII.h, [274](#)
  - libIDT\_NEO2.h, [380](#)
  - libIDT\_PipReader.h, [465](#)
  - libIDT\_Vendi.h, [571](#)
  - libIDT\_VP3300\_AJ.h, [596](#)
  - libIDT\_VP3300\_BT.h, [634](#)
  - libIDT\_VP3300\_COM.h, [670](#)
  - libIDT\_VP3300\_USB.h, [706](#)
  - libIDT\_VP8800.h, [746](#)
- device\_createDirectory
  - libIDT\_Device.h, [131](#)
  - libIDT\_VP8800.h, [747](#)
- device\_deleteDirectory
  - libIDT\_Device.h, [133](#)
  - libIDT\_NEO2.h, [381](#)
  - libIDT\_VP8800.h, [747](#)
- device\_deleteFile
  - libIDT\_Device.h, [133](#)
  - libIDT\_NEO2.h, [383](#)
  - libIDT\_VP8800.h, [747](#)
- device\_disableBlueLED
  - libIDT\_Device.h, [133](#)
  - libIDT\_NEO2.h, [383](#)

- device\_enableBlueLED
  - libIDT\_Device.h, [133](#)
  - libIDT\_NEO2.h, [383](#)
- device\_enableExternalLCDMessages
  - libIDT\_Device.h, [134](#)
  - libIDT\_NEO2.h, [383](#)
- device\_enableL100PassThrough
  - libIDT\_Device.h, [134](#)
  - libIDT\_NEO2.h, [385](#)
- device\_enablePassThrough
  - libIDT\_Device.h, [134](#)
  - libIDT\_KioskIII.h, [275](#)
  - libIDT\_NEO2.h, [385](#)
  - libIDT\_PipReader.h, [466](#)
  - libIDT\_UniPayI\_V.h, [540](#)
  - libIDT\_Vendi.h, [572](#)
  - libIDT\_VP3300\_AJ.h, [597](#)
  - libIDT\_VP3300\_BT.h, [634](#)
  - libIDT\_VP3300\_COM.h, [670](#)
  - libIDT\_VP3300\_USB.h, [707](#)
  - libIDT\_VP8800.h, [747](#)
- device\_enableRFAntenna
  - libIDT\_Device.h, [136](#)
  - libIDT\_NEO2.h, [385](#)
- device\_enhancedPassthrough
  - libIDT\_Device.h, [136](#)
  - libIDT\_VP8800.h, [749](#)
- device\_enterStopMode
  - libIDT\_Device.h, [136](#)
  - libIDT\_L100.h, [288](#)
- device\_getButtonConfiguration
  - libIDT\_Device.h, [136](#)
  - libIDT\_NEO2.h, [385](#)
- device\_getCurrentDeviceType
  - libIDT\_Augusta.h, [58](#)
  - libIDT\_Device.h, [137](#)
  - libIDT\_KioskIII.h, [275](#)
  - libIDT\_L100.h, [288](#)
  - libIDT\_MiniSmartII.h, [324](#)
  - libIDT\_NEO2.h, [387](#)
  - libIDT\_PipReader.h, [466](#)
  - libIDT\_SpectrumPro.h, [480](#)
  - libIDT\_SREDKey2.h, [522](#)
  - libIDT\_UniPayI\_V.h, [540](#)
  - libIDT\_Vendi.h, [572](#)
  - libIDT\_VP3300\_AJ.h, [597](#)
  - libIDT\_VP3300\_BT.h, [635](#)
  - libIDT\_VP3300\_COM.h, [671](#)
  - libIDT\_VP3300\_USB.h, [707](#)
  - libIDT\_VP8800.h, [749](#)
- device\_getDRS
  - libIDT\_Augusta.h, [58](#)
  - libIDT\_Device.h, [138](#)
- device\_getDateTime
  - libIDT\_Device.h, [137](#)
  - libIDT\_L100.h, [288](#)
- device\_getDateTime\_Len
  - libIDT\_Device.h, [137](#)
- libIDT\_L100.h, [288](#)
- device\_getDeviceMemoryUsageInfo
  - libIDT\_Device.h, [137](#)
  - libIDT\_NEO2.h, [387](#)
- device\_getDriveFreeSpace
  - libIDT\_Device.h, [138](#)
  - libIDT\_VP8800.h, [749](#)
- device\_getFirmwareVersion
  - libIDT\_Augusta.h, [58](#)
  - libIDT\_Device.h, [138](#)
  - libIDT\_KioskIII.h, [275](#)
  - libIDT\_L100.h, [289](#)
  - libIDT\_MiniSmartII.h, [324](#)
  - libIDT\_NEO2.h, [387](#)
  - libIDT\_PipReader.h, [466](#)
  - libIDT\_SpectrumPro.h, [480](#)
  - libIDT\_SREDKey2.h, [522](#)
  - libIDT\_UniPayI\_V.h, [541](#)
  - libIDT\_Vendi.h, [573](#)
  - libIDT\_VP3300\_AJ.h, [598](#)
  - libIDT\_VP3300\_BT.h, [635](#)
  - libIDT\_VP3300\_COM.h, [671](#)
  - libIDT\_VP3300\_USB.h, [708](#)
  - libIDT\_VP8800.h, [749](#)
- device\_getFirmwareVersion\_Len
  - libIDT\_Augusta.h, [60](#)
  - libIDT\_Device.h, [139](#)
  - libIDT\_KioskIII.h, [275](#)
  - libIDT\_L100.h, [289](#)
  - libIDT\_MiniSmartII.h, [324](#)
  - libIDT\_NEO2.h, [387](#)
  - libIDT\_PipReader.h, [466](#)
  - libIDT\_SpectrumPro.h, [480](#)
  - libIDT\_SREDKey2.h, [522](#)
  - libIDT\_UniPayI\_V.h, [541](#)
  - libIDT\_Vendi.h, [573](#)
  - libIDT\_VP3300\_AJ.h, [598](#)
  - libIDT\_VP3300\_BT.h, [635](#)
  - libIDT\_VP3300\_COM.h, [671](#)
  - libIDT\_VP3300\_USB.h, [708](#)
  - libIDT\_VP8800.h, [750](#)
- device\_getIDGStatusCodeString
  - libIDT\_Device.h, [139](#)
  - libIDT\_KioskIII.h, [276](#)
  - libIDT\_NEO2.h, [388](#)
  - libIDT\_PipReader.h, [467](#)
  - libIDT\_SREDKey2.h, [522](#)
  - libIDT\_UniPayI\_V.h, [541](#)
  - libIDT\_Vendi.h, [573](#)
  - libIDT\_VP3300\_AJ.h, [598](#)
  - libIDT\_VP3300\_BT.h, [635](#)
  - libIDT\_VP3300\_COM.h, [671](#)
  - libIDT\_VP3300\_USB.h, [708](#)
  - libIDT\_VP8800.h, [750](#)
- device\_getKeyStatus
  - libIDT\_Augusta.h, [60](#)
  - libIDT\_Device.h, [140](#)
  - libIDT\_L100.h, [289](#)

- libIDT\_MiniSmartII.h, [325](#)
- libIDT\_NEO2.h, [389](#)
- libIDT\_SREDKey2.h, [524](#)
- device\_getL100PassThroughMode
  - libIDT\_Device.h, [141](#)
  - libIDT\_NEO2.h, [390](#)
- device\_getMerchantRecord
  - libIDT\_Device.h, [141](#)
  - libIDT\_KioskIII.h, [277](#)
  - libIDT\_NEO2.h, [390](#)
  - libIDT\_PipReader.h, [468](#)
  - libIDT\_UniPayI\_V.h, [542](#)
  - libIDT\_Vendi.h, [574](#)
  - libIDT\_VP3300\_AJ.h, [599](#)
  - libIDT\_VP3300\_BT.h, [637](#)
  - libIDT\_VP3300\_COM.h, [673](#)
  - libIDT\_VP3300\_USB.h, [709](#)
  - libIDT\_VP8800.h, [751](#)
- device\_getMerchantRecord\_Len
  - libIDT\_Device.h, [142](#)
  - libIDT\_KioskIII.h, [277](#)
  - libIDT\_NEO2.h, [390](#)
  - libIDT\_PipReader.h, [468](#)
  - libIDT\_UniPayI\_V.h, [543](#)
  - libIDT\_Vendi.h, [575](#)
  - libIDT\_VP3300\_AJ.h, [600](#)
  - libIDT\_VP3300\_BT.h, [637](#)
  - libIDT\_VP3300\_COM.h, [673](#)
  - libIDT\_VP3300\_USB.h, [710](#)
  - libIDT\_VP8800.h, [751](#)
- device\_getRTCDateTime
  - libIDT\_Device.h, [152](#)
  - libIDT\_VP3300\_AJ.h, [600](#)
  - libIDT\_VP3300\_BT.h, [637](#)
  - libIDT\_VP3300\_COM.h, [673](#)
  - libIDT\_VP3300\_USB.h, [710](#)
- device\_getResponseCodeString
  - libIDT\_Augusta.h, [61](#)
  - libIDT\_Device.h, [142](#)
  - libIDT\_L100.h, [290](#)
  - libIDT\_MiniSmartII.h, [325](#)
  - libIDT\_NEO2.h, [391](#)
  - libIDT\_SpectrumPro.h, [481](#)
- device\_getSDKWaitTime
  - libIDT\_Augusta.h, [70](#)
  - libIDT\_Device.h, [152](#)
  - libIDT\_KioskIII.h, [278](#)
  - libIDT\_MiniSmartII.h, [335](#)
  - libIDT\_NEO2.h, [400](#)
  - libIDT\_PipReader.h, [469](#)
  - libIDT\_SpectrumPro.h, [490](#)
  - libIDT\_UniPayI\_V.h, [543](#)
  - libIDT\_Vendi.h, [575](#)
  - libIDT\_VP3300\_AJ.h, [600](#)
  - libIDT\_VP3300\_BT.h, [638](#)
  - libIDT\_VP3300\_COM.h, [674](#)
  - libIDT\_VP3300\_USB.h, [710](#)
  - libIDT\_VP8800.h, [753](#)
- device\_getSpectrumProKSN
  - libIDT\_Device.h, [152](#)
  - libIDT\_SpectrumPro.h, [491](#)
- device\_getSpectrumProKSN\_Len
  - libIDT\_Device.h, [153](#)
  - libIDT\_SpectrumPro.h, [491](#)
- device\_getThreadStackSize
  - libIDT\_Augusta.h, [71](#)
  - libIDT\_Device.h, [154](#)
  - libIDT\_KioskIII.h, [278](#)
  - libIDT\_MiniSmartII.h, [335](#)
  - libIDT\_NEO2.h, [401](#)
  - libIDT\_SpectrumPro.h, [492](#)
  - libIDT\_UniPayI\_V.h, [543](#)
  - libIDT\_Vendi.h, [575](#)
  - libIDT\_VP3300\_AJ.h, [601](#)
  - libIDT\_VP3300\_BT.h, [638](#)
  - libIDT\_VP3300\_COM.h, [674](#)
  - libIDT\_VP3300\_USB.h, [711](#)
  - libIDT\_VP8800.h, [753](#)
- device\_getTransactionResults
  - libIDT\_KioskIII.h, [278](#)
  - libIDT\_NEO2.h, [401](#)
  - libIDT\_PipReader.h, [469](#)
  - libIDT\_Vendi.h, [575](#)
  - libIDT\_VP3300\_AJ.h, [601](#)
  - libIDT\_VP3300\_BT.h, [638](#)
  - libIDT\_VP3300\_COM.h, [674](#)
  - libIDT\_VP3300\_USB.h, [711](#)
  - libIDT\_VP8800.h, [753](#)
- device\_init
  - libIDT\_Augusta.h, [71](#)
  - libIDT\_Device.h, [154](#)
  - libIDT\_KioskIII.h, [278](#)
  - libIDT\_L100.h, [300](#)
  - libIDT\_MiniSmartII.h, [336](#)
  - libIDT\_NEO2.h, [401](#)
  - libIDT\_PipReader.h, [469](#)
  - libIDT\_SpectrumPro.h, [492](#)
  - libIDT\_SREDKey2.h, [524](#)
  - libIDT\_UniPayI\_V.h, [543](#)
  - libIDT\_Vendi.h, [576](#)
  - libIDT\_VP3300\_AJ.h, [601](#)
  - libIDT\_VP3300\_BT.h, [638](#)
  - libIDT\_VP3300\_COM.h, [674](#)
  - libIDT\_VP3300\_USB.h, [711](#)
  - libIDT\_VP8800.h, [753](#)
- device\_isAttached
  - libIDT\_Augusta.h, [71](#)
  - libIDT\_Device.h, [154](#)
  - libIDT\_KioskIII.h, [279](#)
  - libIDT\_L100.h, [300](#)
  - libIDT\_MiniSmartII.h, [336](#)
  - libIDT\_NEO2.h, [401](#)
  - libIDT\_PipReader.h, [469](#)
  - libIDT\_SpectrumPro.h, [492](#)
  - libIDT\_SREDKey2.h, [525](#)
  - libIDT\_UniPayI\_V.h, [544](#)

- libIDT\_Vendi.h, 576
- libIDT\_VP3300\_AJ.h, 601
- libIDT\_VP3300\_BT.h, 639
- libIDT\_VP3300\_COM.h, 675
- libIDT\_VP3300\_USB.h, 711
- libIDT\_VP8800.h, 754
- device\_isConnected
  - libIDT\_Augusta.h, 71
  - libIDT\_Device.h, 154
  - libIDT\_KioskIII.h, 279
  - libIDT\_L100.h, 300
  - libIDT\_MiniSmartII.h, 336
  - libIDT\_NEO2.h, 402
  - libIDT\_PipReader.h, 470
  - libIDT\_SpectrumPro.h, 496
  - libIDT\_SREDKey2.h, 525
  - libIDT\_UniPayI\_V.h, 544
  - libIDT\_Vendi.h, 576
  - libIDT\_VP3300\_AJ.h, 602
  - libIDT\_VP3300\_BT.h, 639
  - libIDT\_VP3300\_COM.h, 675
  - libIDT\_VP3300\_USB.h, 712
  - libIDT\_VP8800.h, 754
- device\_lcdDisplayClear
  - libIDT\_Device.h, 154
  - libIDT\_NEO2.h, 402
- device\_lcdDisplayLine1Message
  - libIDT\_Device.h, 154
  - libIDT\_NEO2.h, 402
- device\_lcdDisplayLine2Message
  - libIDT\_Device.h, 156
  - libIDT\_NEO2.h, 402
- device\_listDirectory
  - libIDT\_Device.h, 156
  - libIDT\_NEO2.h, 402
  - libIDT\_VP8800.h, 754
- device\_pingDevice
  - libIDT\_Device.h, 156
  - libIDT\_KioskIII.h, 279
  - libIDT\_NEO2.h, 404
  - libIDT\_PipReader.h, 470
  - libIDT\_SREDKey2.h, 525
  - libIDT\_UniPayI\_V.h, 544
  - libIDT\_Vendi.h, 576
  - libIDT\_VP3300\_AJ.h, 602
  - libIDT\_VP3300\_BT.h, 639
  - libIDT\_VP3300\_COM.h, 675
  - libIDT\_VP3300\_USB.h, 712
  - libIDT\_VP8800.h, 754
- device\_pollCardReader
  - libIDT\_Device.h, 156
  - libIDT\_SpectrumPro.h, 492
- device\_pollCardReader\_Len
  - libIDT\_Device.h, 159
  - libIDT\_SpectrumPro.h, 494
- device\_pollForToken
  - libIDT\_Device.h, 161
  - libIDT\_NEO2.h, 404
- device\_queryFile
  - libIDT\_Device.h, 161
  - libIDT\_NEO2.h, 404
- device\_rebootDevice
  - libIDT\_Augusta.h, 71
  - libIDT\_Device.h, 161
  - libIDT\_L100.h, 300
  - libIDT\_MiniSmartII.h, 336
  - libIDT\_SpectrumPro.h, 496
  - libIDT\_SREDKey2.h, 525
- device\_registerCameraCallBk
  - libIDT\_Augusta.h, 72
  - libIDT\_Device.h, 162
  - libIDT\_KioskIII.h, 279
  - libIDT\_L100.h, 300
  - libIDT\_MiniSmartII.h, 336
  - libIDT\_NEO2.h, 405
  - libIDT\_PipReader.h, 470
  - libIDT\_SpectrumPro.h, 496
  - libIDT\_SREDKey2.h, 525
  - libIDT\_UniPayI\_V.h, 544
  - libIDT\_Vendi.h, 576
  - libIDT\_VP3300\_AJ.h, 602
  - libIDT\_VP3300\_BT.h, 639
  - libIDT\_VP3300\_COM.h, 675
  - libIDT\_VP3300\_USB.h, 712
  - libIDT\_VP8800.h, 754
- device\_registerCardStatusFrontSwitchCallBk
  - libIDT\_Augusta.h, 72
  - libIDT\_Device.h, 162
  - libIDT\_KioskIII.h, 279
  - libIDT\_L100.h, 301
  - libIDT\_MiniSmartII.h, 337
  - libIDT\_NEO2.h, 405
  - libIDT\_PipReader.h, 470
  - libIDT\_SpectrumPro.h, 496
  - libIDT\_SREDKey2.h, 525
  - libIDT\_UniPayI\_V.h, 544
  - libIDT\_Vendi.h, 577
  - libIDT\_VP3300\_AJ.h, 602
  - libIDT\_VP3300\_BT.h, 639
  - libIDT\_VP3300\_COM.h, 675
  - libIDT\_VP3300\_USB.h, 712
  - libIDT\_VP8800.h, 755
- device\_registerFWCallBk
  - libIDT\_Augusta.h, 72
  - libIDT\_Device.h, 162
  - libIDT\_L100.h, 301
  - libIDT\_NEO2.h, 405
  - libIDT\_SREDKey2.h, 525
- device\_registerRKICallBk
  - libIDT\_Device.h, 162
  - libIDT\_VP3300\_AJ.h, 602
  - libIDT\_VP3300\_BT.h, 639
  - libIDT\_VP3300\_COM.h, 675
  - libIDT\_VP3300\_USB.h, 712
- device\_selfCheck
  - libIDT\_Device.h, 162

- device\_setBurstMode
  - libIDT\_Device.h, [163](#)
  - libIDT\_KioskIII.h, [280](#)
  - libIDT\_NEO2.h, [405](#)
  - libIDT\_PipReader.h, [470](#)
  - libIDT\_Vendi.h, [577](#)
  - libIDT\_VP3300\_AJ.h, [604](#)
  - libIDT\_VP3300\_BT.h, [640](#)
  - libIDT\_VP3300\_COM.h, [676](#)
  - libIDT\_VP3300\_USB.h, [713](#)
- device\_setCancelTransactionMode
  - libIDT\_Device.h, [163](#)
  - libIDT\_NEO2.h, [406](#)
- device\_setConfigPath
  - libIDT\_Device.h, [164](#)
  - libIDT\_NEO2.h, [406](#)
  - libIDT\_SREDKey2.h, [527](#)
- device\_setCurrentDevice
  - libIDT\_Augusta.h, [72](#)
  - libIDT\_Device.h, [164](#)
  - libIDT\_KioskIII.h, [280](#)
  - libIDT\_L100.h, [301](#)
  - libIDT\_MiniSmartII.h, [337](#)
  - libIDT\_NEO2.h, [406](#)
  - libIDT\_PipReader.h, [472](#)
  - libIDT\_SpectrumPro.h, [496](#)
  - libIDT\_SREDKey2.h, [527](#)
  - libIDT\_UniPayI\_V.h, [545](#)
  - libIDT\_Vendi.h, [578](#)
  - libIDT\_VP3300\_AJ.h, [604](#)
  - libIDT\_VP3300\_BT.h, [640](#)
  - libIDT\_VP3300\_COM.h, [676](#)
  - libIDT\_VP3300\_USB.h, [714](#)
  - libIDT\_VP8800.h, [756](#)
- device\_setMerchantRecord
  - libIDT\_Device.h, [164](#)
  - libIDT\_KioskIII.h, [281](#)
  - libIDT\_NEO2.h, [407](#)
  - libIDT\_PipReader.h, [473](#)
  - libIDT\_UniPayI\_V.h, [546](#)
  - libIDT\_Vendi.h, [579](#)
  - libIDT\_VP3300\_AJ.h, [605](#)
  - libIDT\_VP3300\_BT.h, [641](#)
  - libIDT\_VP3300\_COM.h, [677](#)
  - libIDT\_VP3300\_USB.h, [714](#)
  - libIDT\_VP8800.h, [756](#)
- device\_setNEO2DevicesConfigs
  - libIDT\_Device.h, [165](#)
  - libIDT\_NEO2.h, [407](#)
  - libIDT\_SREDKey2.h, [527](#)
- device\_setPollMode
  - libIDT\_Device.h, [165](#)
  - libIDT\_KioskIII.h, [281](#)
  - libIDT\_NEO2.h, [408](#)
  - libIDT\_PipReader.h, [473](#)
  - libIDT\_Vendi.h, [579](#)
  - libIDT\_VP3300\_AJ.h, [605](#)
  - libIDT\_VP3300\_BT.h, [641](#)
- libIDT\_VP3300\_COM.h, [677](#)
- libIDT\_VP3300\_USB.h, [714](#)
- device\_setRTCDateTime
  - libIDT\_Device.h, [165](#)
  - libIDT\_VP3300\_AJ.h, [606](#)
  - libIDT\_VP3300\_BT.h, [642](#)
  - libIDT\_VP3300\_COM.h, [678](#)
  - libIDT\_VP3300\_USB.h, [715](#)
- device\_setSDKWaitTime
  - libIDT\_Augusta.h, [73](#)
  - libIDT\_Device.h, [165](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_MiniSmartII.h, [338](#)
  - libIDT\_NEO2.h, [408](#)
  - libIDT\_PipReader.h, [473](#)
  - libIDT\_SpectrumPro.h, [497](#)
  - libIDT\_UniPayI\_V.h, [546](#)
  - libIDT\_Vendi.h, [579](#)
  - libIDT\_VP3300\_AJ.h, [606](#)
  - libIDT\_VP3300\_BT.h, [642](#)
  - libIDT\_VP3300\_COM.h, [678](#)
  - libIDT\_VP3300\_USB.h, [715](#)
  - libIDT\_VP8800.h, [756](#)
- device\_setSleepModeTime
  - libIDT\_Device.h, [166](#)
  - libIDT\_L100.h, [302](#)
- device\_setSystemLanguage
  - libIDT\_Device.h, [166](#)
  - libIDT\_SREDKey2.h, [529](#)
- device\_setThreadStackSize
  - libIDT\_Augusta.h, [73](#)
  - libIDT\_Device.h, [166](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_MiniSmartII.h, [338](#)
  - libIDT\_NEO2.h, [408](#)
  - libIDT\_SpectrumPro.h, [497](#)
  - libIDT\_UniPayI\_V.h, [547](#)
  - libIDT\_Vendi.h, [579](#)
  - libIDT\_VP3300\_AJ.h, [606](#)
  - libIDT\_VP3300\_BT.h, [642](#)
  - libIDT\_VP3300\_COM.h, [678](#)
  - libIDT\_VP3300\_USB.h, [715](#)
  - libIDT\_VP8800.h, [757](#)
- device\_setTransactionExponent
  - libIDT\_Device.h, [166](#)
  - libIDT\_NEO2.h, [408](#)
  - libIDT\_SREDKey2.h, [529](#)
  - libIDT\_VP3300\_AJ.h, [606](#)
  - libIDT\_VP3300\_BT.h, [642](#)
  - libIDT\_VP3300\_COM.h, [678](#)
  - libIDT\_VP3300\_USB.h, [715](#)
  - libIDT\_VP8800.h, [757](#)
- device\_startListenNotifications
  - libIDT\_Device.h, [167](#)
  - libIDT\_NEO2.h, [408](#)
- device\_startQRCodeScan
  - libIDT\_Device.h, [167](#)
  - libIDT\_NEO2.h, [408](#)

- device\_startRKI
  - libIDT\_Device.h, 167
  - libIDT\_VP3300\_AJ.h, 606
  - libIDT\_VP3300\_BT.h, 642
  - libIDT\_VP3300\_COM.h, 678
  - libIDT\_VP3300\_USB.h, 716
- device\_startTakingPhoto
  - libIDT\_Device.h, 167
  - libIDT\_NEO2.h, 409
- device\_startTransaction
  - libIDT\_Device.h, 167
  - libIDT\_NEO2.h, 409
  - libIDT\_VP3300\_AJ.h, 607
  - libIDT\_VP3300\_BT.h, 643
  - libIDT\_VP3300\_COM.h, 679
  - libIDT\_VP3300\_USB.h, 716
  - libIDT\_VP8800.h, 757
- device\_stopListenNotifications
  - libIDT\_Device.h, 169
  - libIDT\_NEO2.h, 410
- device\_stopQRCodeScan
  - libIDT\_Device.h, 169
  - libIDT\_NEO2.h, 410
- device\_stopTakingPhoto
  - libIDT\_Device.h, 169
  - libIDT\_NEO2.h, 411
- device\_toSDCard
  - libIDT\_Device.h, 169
  - libIDT\_NEO2.h, 411
- device\_transferFile
  - libIDT\_Device.h, 170
  - libIDT\_NEO2.h, 411
  - libIDT\_VP8800.h, 758
- device\_turnOffYellowLED
  - libIDT\_Device.h, 170
  - libIDT\_NEO2.h, 411
- device\_turnOnYellowLED
  - libIDT\_Device.h, 170
  - libIDT\_NEO2.h, 411
- device\_updateFirmware
  - libIDT\_Augusta.h, 73
  - libIDT\_Device.h, 171
  - libIDT\_L100.h, 302
  - libIDT\_MiniSmartII.h, 338
  - libIDT\_NEO2.h, 412
  - libIDT\_SpectrumPro.h, 497
  - libIDT\_SREDKey2.h, 529
- device\_verifyBackdoorKey
  - libIDT\_Device.h, 171
- emv\_activateTransaction
  - libIDT\_Augusta.h, 74
  - libIDT\_Device.h, 171
  - libIDT\_MiniSmartII.h, 339
  - libIDT\_NEO2.h, 412
  - libIDT\_SpectrumPro.h, 498
  - libIDT\_UniPayI\_V.h, 547
  - libIDT\_VP3300\_AJ.h, 608
  - libIDT\_VP3300\_BT.h, 644
- libIDT\_VP3300\_COM.h, 679
- libIDT\_VP3300\_USB.h, 716
- libIDT\_VP8800.h, 758
- emv\_allowFallback
  - libIDT\_Augusta.h, 74
  - libIDT\_Device.h, 173
  - libIDT\_MiniSmartII.h, 339
  - libIDT\_NEO2.h, 413
  - libIDT\_SpectrumPro.h, 498
  - libIDT\_UniPayI\_V.h, 547
  - libIDT\_VP3300\_AJ.h, 608
  - libIDT\_VP3300\_BT.h, 644
  - libIDT\_VP3300\_COM.h, 680
  - libIDT\_VP3300\_USB.h, 717
  - libIDT\_VP8800.h, 759
- emv\_authenticateTransaction
  - libIDT\_Augusta.h, 76
  - libIDT\_Device.h, 173
  - libIDT\_MiniSmartII.h, 341
  - libIDT\_NEO2.h, 413
  - libIDT\_SpectrumPro.h, 499
  - libIDT\_UniPayI\_V.h, 547
  - libIDT\_VP3300\_AJ.h, 609
  - libIDT\_VP3300\_BT.h, 645
  - libIDT\_VP3300\_COM.h, 680
  - libIDT\_VP3300\_USB.h, 717
  - libIDT\_VP8800.h, 759
- emv\_authenticateTransactionWithTimeout
  - libIDT\_Augusta.h, 76
  - libIDT\_Device.h, 173
  - libIDT\_MiniSmartII.h, 341
  - libIDT\_NEO2.h, 413
  - libIDT\_SpectrumPro.h, 499
  - libIDT\_UniPayI\_V.h, 548
  - libIDT\_VP3300\_AJ.h, 609
  - libIDT\_VP3300\_BT.h, 645
  - libIDT\_VP3300\_COM.h, 680
  - libIDT\_VP3300\_USB.h, 718
  - libIDT\_VP8800.h, 760
- emv\_callbackResponseLCD
  - libIDT\_Augusta.h, 76
  - libIDT\_Device.h, 175
  - libIDT\_MiniSmartII.h, 342
  - libIDT\_SpectrumPro.h, 500
- emv\_callbackResponseMSR
  - libIDT\_Augusta.h, 77
  - libIDT\_Device.h, 175
  - libIDT\_MiniSmartII.h, 342
  - libIDT\_SpectrumPro.h, 500
- emv\_cancelTransaction
  - libIDT\_Augusta.h, 77
  - libIDT\_Device.h, 175
  - libIDT\_MiniSmartII.h, 342
  - libIDT\_NEO2.h, 415
  - libIDT\_SpectrumPro.h, 500
  - libIDT\_UniPayI\_V.h, 548
  - libIDT\_VP3300\_AJ.h, 609
  - libIDT\_VP3300\_BT.h, 645



- libIDT\_VP3300\_COM.h, [681](#)
- libIDT\_VP3300\_USB.h, [718](#)
- libIDT\_VP8800.h, [760](#)
- emv\_completeTransaction
  - libIDT\_Augusta.h, [77](#)
  - libIDT\_Device.h, [176](#)
  - libIDT\_MiniSmartII.h, [342](#)
  - libIDT\_NEO2.h, [415](#)
  - libIDT\_SpectrumPro.h, [500](#)
  - libIDT\_UniPayI\_V.h, [548](#)
  - libIDT\_VP3300\_AJ.h, [610](#)
  - libIDT\_VP3300\_BT.h, [646](#)
  - libIDT\_VP3300\_COM.h, [681](#)
  - libIDT\_VP3300\_USB.h, [718](#)
  - libIDT\_VP8800.h, [760](#)
- emv\_getAutoAuthenticateTransaction
  - libIDT\_Augusta.h, [78](#)
  - libIDT\_Device.h, [176](#)
  - libIDT\_MiniSmartII.h, [343](#)
  - libIDT\_NEO2.h, [415](#)
  - libIDT\_SpectrumPro.h, [502](#)
  - libIDT\_UniPayI\_V.h, [549](#)
  - libIDT\_VP3300\_AJ.h, [610](#)
  - libIDT\_VP3300\_BT.h, [646](#)
  - libIDT\_VP3300\_COM.h, [681](#)
  - libIDT\_VP3300\_USB.h, [719](#)
  - libIDT\_VP8800.h, [761](#)
- emv\_getAutoCompleteTransaction
  - libIDT\_Augusta.h, [78](#)
  - libIDT\_Device.h, [176](#)
  - libIDT\_MiniSmartII.h, [343](#)
  - libIDT\_NEO2.h, [416](#)
  - libIDT\_SpectrumPro.h, [502](#)
  - libIDT\_UniPayI\_V.h, [549](#)
  - libIDT\_VP3300\_AJ.h, [610](#)
  - libIDT\_VP3300\_BT.h, [646](#)
  - libIDT\_VP3300\_COM.h, [682](#)
  - libIDT\_VP3300\_USB.h, [719](#)
  - libIDT\_VP8800.h, [761](#)
- emv\_getEMVConfigurationCheckValue
  - libIDT\_Augusta.h, [78](#)
  - libIDT\_Device.h, [176](#)
  - libIDT\_MiniSmartII.h, [343](#)
  - libIDT\_NEO2.h, [416](#)
  - libIDT\_SpectrumPro.h, [502](#)
  - libIDT\_VP8800.h, [761](#)
- emv\_getEMVKernelCheckValue
  - libIDT\_Augusta.h, [78](#)
  - libIDT\_Device.h, [177](#)
  - libIDT\_MiniSmartII.h, [343](#)
  - libIDT\_NEO2.h, [416](#)
  - libIDT\_SpectrumPro.h, [502](#)
  - libIDT\_VP8800.h, [761](#)
- emv\_getEMVKernelVersion
  - libIDT\_Augusta.h, [79](#)
  - libIDT\_Device.h, [177](#)
  - libIDT\_MiniSmartII.h, [344](#)
  - libIDT\_NEO2.h, [416](#)
- libIDT\_SpectrumPro.h, [503](#)
- libIDT\_VP8800.h, [762](#)
- emv\_getEMVKernelVersion\_Len
  - libIDT\_Augusta.h, [79](#)
  - libIDT\_Device.h, [177](#)
  - libIDT\_MiniSmartII.h, [344](#)
  - libIDT\_NEO2.h, [416](#)
  - libIDT\_SpectrumPro.h, [503](#)
  - libIDT\_VP8800.h, [762](#)
- emv\_registerCallBk
  - libIDT\_Augusta.h, [79](#)
  - libIDT\_Device.h, [177](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_L100.h, [303](#)
  - libIDT\_MiniSmartII.h, [344](#)
  - libIDT\_NEO2.h, [417](#)
  - libIDT\_PipReader.h, [473](#)
  - libIDT\_SpectrumPro.h, [503](#)
  - libIDT\_SREDKey2.h, [530](#)
  - libIDT\_UniPayI\_V.h, [549](#)
  - libIDT\_Vendi.h, [579](#)
  - libIDT\_VP3300\_AJ.h, [610](#)
  - libIDT\_VP3300\_BT.h, [646](#)
  - libIDT\_VP3300\_COM.h, [682](#)
  - libIDT\_VP3300\_USB.h, [719](#)
  - libIDT\_VP8800.h, [762](#)
- emv\_removeAllApplicationData
  - libIDT\_Augusta.h, [79](#)
  - libIDT\_Device.h, [177](#)
  - libIDT\_MiniSmartII.h, [344](#)
  - libIDT\_NEO2.h, [417](#)
  - libIDT\_SpectrumPro.h, [503](#)
  - libIDT\_UniPayI\_V.h, [549](#)
  - libIDT\_VP3300\_AJ.h, [610](#)
  - libIDT\_VP3300\_BT.h, [646](#)
  - libIDT\_VP3300\_COM.h, [682](#)
  - libIDT\_VP3300\_USB.h, [719](#)
  - libIDT\_VP8800.h, [762](#)
- emv\_removeAllCAPK
  - libIDT\_Augusta.h, [79](#)
  - libIDT\_Device.h, [178](#)
  - libIDT\_MiniSmartII.h, [344](#)
  - libIDT\_NEO2.h, [417](#)
  - libIDT\_SpectrumPro.h, [503](#)
  - libIDT\_UniPayI\_V.h, [549](#)
  - libIDT\_VP3300\_AJ.h, [611](#)
  - libIDT\_VP3300\_BT.h, [647](#)
  - libIDT\_VP3300\_COM.h, [682](#)
  - libIDT\_VP3300\_USB.h, [719](#)
  - libIDT\_VP8800.h, [762](#)
- emv\_removeAllCRL
  - libIDT\_Augusta.h, [79](#)
  - libIDT\_Device.h, [178](#)
  - libIDT\_MiniSmartII.h, [344](#)
  - libIDT\_NEO2.h, [417](#)
  - libIDT\_SpectrumPro.h, [503](#)
  - libIDT\_UniPayI\_V.h, [550](#)
  - libIDT\_VP3300\_AJ.h, [611](#)

- libIDT\_VP3300\_BT.h, [647](#)
- libIDT\_VP3300\_COM.h, [682](#)
- libIDT\_VP3300\_USB.h, [719](#)
- libIDT\_VP8800.h, [762](#)
- emv\_removeAllExceptions
  - libIDT\_VP8800.h, [763](#)
- emv\_removeApplicationData
  - libIDT\_Augusta.h, [80](#)
  - libIDT\_Device.h, [178](#)
  - libIDT\_MiniSmartII.h, [345](#)
  - libIDT\_NEO2.h, [417](#)
  - libIDT\_SpectrumPro.h, [504](#)
  - libIDT\_UniPayI\_V.h, [550](#)
  - libIDT\_VP3300\_AJ.h, [611](#)
  - libIDT\_VP3300\_BT.h, [647](#)
  - libIDT\_VP3300\_COM.h, [682](#)
  - libIDT\_VP3300\_USB.h, [720](#)
  - libIDT\_VP8800.h, [763](#)
- emv\_removeCAPK
  - libIDT\_Augusta.h, [80](#)
  - libIDT\_Device.h, [178](#)
  - libIDT\_MiniSmartII.h, [345](#)
  - libIDT\_NEO2.h, [418](#)
  - libIDT\_SpectrumPro.h, [504](#)
  - libIDT\_UniPayI\_V.h, [550](#)
  - libIDT\_VP3300\_AJ.h, [611](#)
  - libIDT\_VP3300\_BT.h, [647](#)
  - libIDT\_VP3300\_COM.h, [683](#)
  - libIDT\_VP3300\_USB.h, [720](#)
  - libIDT\_VP8800.h, [763](#)
- emv\_removeCRL
  - libIDT\_Augusta.h, [80](#)
  - libIDT\_Device.h, [178](#)
  - libIDT\_MiniSmartII.h, [345](#)
  - libIDT\_NEO2.h, [418](#)
  - libIDT\_SpectrumPro.h, [504](#)
  - libIDT\_UniPayI\_V.h, [550](#)
  - libIDT\_VP3300\_AJ.h, [611](#)
  - libIDT\_VP3300\_BT.h, [647](#)
  - libIDT\_VP3300\_COM.h, [683](#)
  - libIDT\_VP3300\_USB.h, [720](#)
  - libIDT\_VP8800.h, [763](#)
- emv\_removeException
  - libIDT\_VP8800.h, [764](#)
- emv\_removeTerminalData
  - libIDT\_Augusta.h, [80](#)
  - libIDT\_Device.h, [179](#)
  - libIDT\_MiniSmartII.h, [345](#)
  - libIDT\_SpectrumPro.h, [504](#)
- emv\_removeTransactionLog
  - libIDT\_VP8800.h, [764](#)
- emv\_retrieveAIDLList
  - libIDT\_Augusta.h, [81](#)
  - libIDT\_Device.h, [179](#)
  - libIDT\_MiniSmartII.h, [346](#)
  - libIDT\_NEO2.h, [418](#)
  - libIDT\_SpectrumPro.h, [505](#)
  - libIDT\_UniPayI\_V.h, [551](#)
- libIDT\_VP3300\_AJ.h, [612](#)
- libIDT\_VP3300\_BT.h, [648](#)
- libIDT\_VP3300\_COM.h, [683](#)
- libIDT\_VP3300\_USB.h, [720](#)
- libIDT\_VP8800.h, [764](#)
- emv\_retrieveApplicationData
  - libIDT\_Augusta.h, [81](#)
  - libIDT\_Device.h, [179](#)
  - libIDT\_MiniSmartII.h, [346](#)
  - libIDT\_NEO2.h, [418](#)
  - libIDT\_SpectrumPro.h, [505](#)
  - libIDT\_UniPayI\_V.h, [551](#)
  - libIDT\_VP3300\_AJ.h, [612](#)
  - libIDT\_VP3300\_BT.h, [648](#)
  - libIDT\_VP3300\_COM.h, [683](#)
  - libIDT\_VP3300\_USB.h, [721](#)
  - libIDT\_VP8800.h, [764](#)
- emv\_retrieveCAPK
  - libIDT\_Augusta.h, [82](#)
  - libIDT\_Device.h, [179](#)
  - libIDT\_MiniSmartII.h, [347](#)
  - libIDT\_NEO2.h, [419](#)
  - libIDT\_SpectrumPro.h, [506](#)
  - libIDT\_UniPayI\_V.h, [552](#)
  - libIDT\_VP3300\_AJ.h, [612](#)
  - libIDT\_VP3300\_BT.h, [648](#)
  - libIDT\_VP3300\_COM.h, [684](#)
  - libIDT\_VP3300\_USB.h, [721](#)
  - libIDT\_VP8800.h, [765](#)
- emv\_retrieveCAPKList
  - libIDT\_Augusta.h, [82](#)
  - libIDT\_Device.h, [181](#)
  - libIDT\_MiniSmartII.h, [347](#)
  - libIDT\_NEO2.h, [420](#)
  - libIDT\_SpectrumPro.h, [506](#)
  - libIDT\_UniPayI\_V.h, [552](#)
  - libIDT\_VP3300\_AJ.h, [613](#)
  - libIDT\_VP3300\_BT.h, [649](#)
  - libIDT\_VP3300\_COM.h, [684](#)
  - libIDT\_VP3300\_USB.h, [722](#)
  - libIDT\_VP8800.h, [766](#)
- emv\_retrieveCRL
  - libIDT\_Augusta.h, [82](#)
  - libIDT\_Device.h, [181](#)
  - libIDT\_MiniSmartII.h, [347](#)
  - libIDT\_NEO2.h, [420](#)
  - libIDT\_SpectrumPro.h, [506](#)
  - libIDT\_UniPayI\_V.h, [552](#)
  - libIDT\_VP3300\_AJ.h, [613](#)
  - libIDT\_VP3300\_BT.h, [649](#)
  - libIDT\_VP3300\_COM.h, [685](#)
  - libIDT\_VP3300\_USB.h, [722](#)
  - libIDT\_VP8800.h, [766](#)
- emv\_retrieveExceptionList
  - libIDT\_VP8800.h, [766](#)
- emv\_retrieveExceptionLogStatus
  - libIDT\_VP8800.h, [766](#)
- emv\_retrieveTerminalData



- libIDT\_Augusta.h, [84](#)
- libIDT\_Device.h, [181](#)
- libIDT\_MiniSmartII.h, [349](#)
- libIDT\_NEO2.h, [420](#)
- libIDT\_SpectrumPro.h, [508](#)
- libIDT\_UniPayI\_V.h, [554](#)
- libIDT\_VP3300\_AJ.h, [614](#)
- libIDT\_VP3300\_BT.h, [650](#)
- libIDT\_VP3300\_COM.h, [685](#)
- libIDT\_VP3300\_USB.h, [722](#)
- libIDT\_VP8800.h, [767](#)
- emv\_retrieveTerminalID
  - libIDT\_Augusta.h, [84](#)
  - libIDT\_Device.h, [182](#)
  - libIDT\_MiniSmartII.h, [349](#)
  - libIDT\_SpectrumPro.h, [508](#)
- emv\_retrieveTerminalID\_Len
  - libIDT\_Augusta.h, [84](#)
  - libIDT\_Device.h, [182](#)
  - libIDT\_MiniSmartII.h, [349](#)
  - libIDT\_SpectrumPro.h, [508](#)
- emv\_retrieveTransactionLog
  - libIDT\_VP8800.h, [767](#)
- emv\_retrieveTransactionLogStatus
  - libIDT\_VP8800.h, [768](#)
- emv\_retrieveTransactionResult
  - libIDT\_Augusta.h, [84](#)
  - libIDT\_Device.h, [182](#)
  - libIDT\_MiniSmartII.h, [349](#)
  - libIDT\_NEO2.h, [420](#)
  - libIDT\_SpectrumPro.h, [508](#)
- emv\_setApplicationData
  - libIDT\_Augusta.h, [85](#)
  - libIDT\_Device.h, [182](#)
  - libIDT\_MiniSmartII.h, [350](#)
  - libIDT\_NEO2.h, [421](#)
  - libIDT\_SpectrumPro.h, [509](#)
  - libIDT\_UniPayI\_V.h, [554](#)
  - libIDT\_VP3300\_AJ.h, [614](#)
  - libIDT\_VP3300\_BT.h, [650](#)
  - libIDT\_VP3300\_COM.h, [685](#)
  - libIDT\_VP3300\_USB.h, [722](#)
  - libIDT\_VP8800.h, [769](#)
- emv\_setApplicationDataTLV
  - libIDT\_Device.h, [183](#)
  - libIDT\_NEO2.h, [421](#)
  - libIDT\_UniPayI\_V.h, [554](#)
  - libIDT\_VP3300\_AJ.h, [614](#)
  - libIDT\_VP3300\_BT.h, [650](#)
  - libIDT\_VP3300\_COM.h, [685](#)
  - libIDT\_VP3300\_USB.h, [723](#)
  - libIDT\_VP8800.h, [769](#)
- emv\_setAutoAuthenticateTransaction
  - libIDT\_Augusta.h, [85](#)
  - libIDT\_Device.h, [183](#)
  - libIDT\_MiniSmartII.h, [350](#)
  - libIDT\_NEO2.h, [421](#)
  - libIDT\_SpectrumPro.h, [509](#)
  - libIDT\_UniPayI\_V.h, [555](#)
  - libIDT\_VP3300\_AJ.h, [614](#)
  - libIDT\_VP3300\_BT.h, [650](#)
  - libIDT\_VP3300\_COM.h, [686](#)
  - libIDT\_VP3300\_USB.h, [723](#)
  - libIDT\_VP8800.h, [769](#)
- emv\_setAutoCompleteTransaction
  - libIDT\_Augusta.h, [85](#)
  - libIDT\_Device.h, [183](#)
  - libIDT\_MiniSmartII.h, [350](#)
  - libIDT\_NEO2.h, [422](#)
  - libIDT\_SpectrumPro.h, [509](#)
  - libIDT\_UniPayI\_V.h, [555](#)
  - libIDT\_VP3300\_AJ.h, [615](#)
  - libIDT\_VP3300\_BT.h, [651](#)
  - libIDT\_VP3300\_COM.h, [686](#)
  - libIDT\_VP3300\_USB.h, [723](#)
  - libIDT\_VP8800.h, [770](#)
- emv\_setCAPK
  - libIDT\_Augusta.h, [85](#)
  - libIDT\_Device.h, [183](#)
  - libIDT\_MiniSmartII.h, [350](#)
  - libIDT\_NEO2.h, [422](#)
  - libIDT\_SpectrumPro.h, [509](#)
  - libIDT\_UniPayI\_V.h, [555](#)
  - libIDT\_VP3300\_AJ.h, [615](#)
  - libIDT\_VP3300\_BT.h, [651](#)
  - libIDT\_VP3300\_COM.h, [686](#)
  - libIDT\_VP3300\_USB.h, [723](#)
  - libIDT\_VP8800.h, [770](#)
- emv\_setCRL
  - libIDT\_Augusta.h, [86](#)
  - libIDT\_Device.h, [184](#)
  - libIDT\_MiniSmartII.h, [351](#)
  - libIDT\_NEO2.h, [423](#)
  - libIDT\_SpectrumPro.h, [510](#)
  - libIDT\_UniPayI\_V.h, [555](#)
  - libIDT\_VP3300\_AJ.h, [615](#)
  - libIDT\_VP3300\_BT.h, [651](#)
  - libIDT\_VP3300\_COM.h, [687](#)
  - libIDT\_VP3300\_USB.h, [724](#)
  - libIDT\_VP8800.h, [771](#)
- emv\_setException
  - libIDT\_VP8800.h, [771](#)
- emv\_setTerminalData
  - libIDT\_Augusta.h, [86](#)
  - libIDT\_Device.h, [184](#)
  - libIDT\_MiniSmartII.h, [351](#)
  - libIDT\_NEO2.h, [423](#)
  - libIDT\_SpectrumPro.h, [510](#)
  - libIDT\_UniPayI\_V.h, [556](#)
  - libIDT\_VP3300\_AJ.h, [616](#)
  - libIDT\_VP3300\_BT.h, [652](#)
  - libIDT\_VP3300\_COM.h, [687](#)
  - libIDT\_VP3300\_USB.h, [724](#)
  - libIDT\_VP8800.h, [771](#)
- emv\_setTerminalID
  - libIDT\_Augusta.h, [87](#)

- liblDT\_Device.h, 185
- liblDT\_MiniSmartII.h, 352
- liblDT\_SpectrumPro.h, 511
- emv\_setTerminalMajorConfiguration
  - liblDT\_Device.h, 185
  - liblDT\_NEO2.h, 423
  - liblDT\_UniPayI\_V.h, 556
  - liblDT\_VP3300\_AJ.h, 616
  - liblDT\_VP3300\_BT.h, 652
  - liblDT\_VP3300\_COM.h, 688
  - liblDT\_VP3300\_USB.h, 725
- emv\_setTransactionParameters
  - liblDT\_Device.h, 185
  - liblDT\_NEO2.h, 424
  - liblDT\_VP3300\_AJ.h, 616
  - liblDT\_VP3300\_BT.h, 652
  - liblDT\_VP3300\_COM.h, 688
  - liblDT\_VP3300\_USB.h, 725
- emv\_startTransaction
  - liblDT\_Augusta.h, 87
  - liblDT\_Device.h, 186
  - liblDT\_MiniSmartII.h, 352
  - liblDT\_NEO2.h, 424
  - liblDT\_SpectrumPro.h, 511
  - liblDT\_UniPayI\_V.h, 556
  - liblDT\_VP3300\_AJ.h, 617
  - liblDT\_VP3300\_BT.h, 653
  - liblDT\_VP3300\_COM.h, 688
  - liblDT\_VP3300\_USB.h, 726
  - liblDT\_VP8800.h, 772
- felica\_SendCommand
  - liblDT\_Device.h, 188
  - liblDT\_NEO2.h, 427
- felica\_authentication
  - liblDT\_Device.h, 186
  - liblDT\_NEO2.h, 424
- felica\_poll
  - liblDT\_Device.h, 187
  - liblDT\_NEO2.h, 426
- felica\_read
  - liblDT\_Device.h, 187
  - liblDT\_NEO2.h, 426
- felica\_readWithMac
  - liblDT\_Device.h, 187
  - liblDT\_NEO2.h, 426
- felica\_requestService
  - liblDT\_Device.h, 188
  - liblDT\_NEO2.h, 427
- felica\_write
  - liblDT\_Device.h, 188
  - liblDT\_NEO2.h, 427
- felica\_writeWithMac
  - liblDT\_Device.h, 189
  - liblDT\_NEO2.h, 428
- ftpComm\_callBack
  - liblDT\_Augusta.h, 51
  - liblDT\_Device.h, 109
  - liblDT\_KioskIII.h, 264

- liblDT\_L100.h, 285
- liblDT\_MiniSmartII.h, 316
- liblDT\_NEO2.h, 365
- liblDT\_PipReader.h, 455
- liblDT\_SpectrumPro.h, 478
- liblDT\_SREDKey2.h, 518
- liblDT\_UniPayI\_V.h, 537
- liblDT\_Vendi.h, 562
- liblDT\_VP3300\_AJ.h, 584
- liblDT\_VP3300\_BT.h, 623
- liblDT\_VP3300\_COM.h, 659
- liblDT\_VP3300\_USB.h, 695
- liblDT\_VP8800.h, 733
- httpComm\_callBack
  - liblDT\_Augusta.h, 51
  - liblDT\_Device.h, 109
  - liblDT\_KioskIII.h, 264
  - liblDT\_L100.h, 285
  - liblDT\_MiniSmartII.h, 316
  - liblDT\_NEO2.h, 365
  - liblDT\_PipReader.h, 455
  - liblDT\_SpectrumPro.h, 478
  - liblDT\_SREDKey2.h, 518
  - liblDT\_UniPayI\_V.h, 537
  - liblDT\_Vendi.h, 562
  - liblDT\_VP3300\_AJ.h, 584
  - liblDT\_VP3300\_BT.h, 623
  - liblDT\_VP3300\_COM.h, 659
  - liblDT\_VP3300\_USB.h, 695
  - liblDT\_VP8800.h, 733

## IN

- liblDT\_Augusta.h, 51
- liblDT\_Device.h, 109
- liblDT\_KioskIII.h, 263
- liblDT\_L100.h, 285
- liblDT\_MiniSmartII.h, 316
- liblDT\_NEO2.h, 365
- liblDT\_PipReader.h, 454
- liblDT\_SpectrumPro.h, 477
- liblDT\_SREDKey2.h, 517
- liblDT\_UniPayI\_V.h, 537
- liblDT\_Vendi.h, 561
- liblDT\_VP3300\_AJ.h, 584
- liblDT\_VP3300\_BT.h, 622
- liblDT\_VP3300\_COM.h, 658
- liblDT\_VP3300\_USB.h, 695
- liblDT\_VP8800.h, 733

## IN\_OUT

- liblDT\_Augusta.h, 51
- liblDT\_Device.h, 109
- liblDT\_KioskIII.h, 263
- liblDT\_L100.h, 285
- liblDT\_MiniSmartII.h, 316
- liblDT\_NEO2.h, 365
- liblDT\_PipReader.h, 454
- liblDT\_SpectrumPro.h, 477
- liblDT\_SREDKey2.h, 517

- libIDT\_UniPayI\_V.h, 537
- libIDT\_Vendi.h, 561
- libIDT\_VP3300\_AJ.h, 584
- libIDT\_VP3300\_BT.h, 622
- libIDT\_VP3300\_COM.h, 658
- libIDT\_VP3300\_USB.h, 695
- libIDT\_VP8800.h, 733
- icc\_disable
  - libIDT\_Augusta.h, 87
  - libIDT\_Device.h, 189
  - libIDT\_MiniSmartII.h, 352
- icc\_enable
  - libIDT\_Augusta.h, 87
  - libIDT\_Device.h, 189
  - libIDT\_MiniSmartII.h, 352
- icc\_exchangeAPDU
  - libIDT\_Augusta.h, 89
  - libIDT\_Device.h, 189
  - libIDT\_MiniSmartII.h, 354
  - libIDT\_NEO2.h, 428
  - libIDT\_UniPayI\_V.h, 557
  - libIDT\_VP3300\_AJ.h, 617
  - libIDT\_VP3300\_BT.h, 653
  - libIDT\_VP3300\_COM.h, 689
  - libIDT\_VP3300\_USB.h, 726
- icc\_exchangeEncryptedAPDU
  - libIDT\_Augusta.h, 89
  - libIDT\_Device.h, 190
  - libIDT\_MiniSmartII.h, 354
- icc\_getAPDU\_KSN
  - libIDT\_Augusta.h, 91
  - libIDT\_Device.h, 190
  - libIDT\_MiniSmartII.h, 356
- icc\_getFunctionStatus
  - libIDT\_Augusta.h, 91
  - libIDT\_Device.h, 190
  - libIDT\_MiniSmartII.h, 356
- icc\_getICCReaderStatus
  - libIDT\_Augusta.h, 91
  - libIDT\_Device.h, 192
  - libIDT\_MiniSmartII.h, 356
  - libIDT\_NEO2.h, 428
  - libIDT\_SpectrumPro.h, 511
  - libIDT\_UniPayI\_V.h, 557
  - libIDT\_VP3300\_AJ.h, 618
  - libIDT\_VP3300\_BT.h, 654
  - libIDT\_VP3300\_COM.h, 689
  - libIDT\_VP3300\_USB.h, 726
- icc\_getKeyFormatForICCDUKPT
  - libIDT\_Augusta.h, 91
  - libIDT\_Device.h, 192
  - libIDT\_MiniSmartII.h, 356
- icc\_getKeyTypeForICCDUKPT
  - libIDT\_Augusta.h, 93
  - libIDT\_Device.h, 192
  - libIDT\_MiniSmartII.h, 358
- icc\_powerOffICC
  - libIDT\_Augusta.h, 93
- libIDT\_Device.h, 193
- libIDT\_MiniSmartII.h, 358
- libIDT\_NEO2.h, 429
- libIDT\_SpectrumPro.h, 512
- libIDT\_UniPayI\_V.h, 558
- libIDT\_VP3300\_AJ.h, 618
- libIDT\_VP3300\_BT.h, 654
- libIDT\_VP3300\_COM.h, 689
- libIDT\_VP3300\_USB.h, 727
- icc\_powerOnICC
  - libIDT\_Augusta.h, 93
  - libIDT\_Device.h, 193
  - libIDT\_MiniSmartII.h, 358
  - libIDT\_NEO2.h, 429
  - libIDT\_SpectrumPro.h, 512
  - libIDT\_UniPayI\_V.h, 558
  - libIDT\_VP3300\_AJ.h, 618
  - libIDT\_VP3300\_BT.h, 654
  - libIDT\_VP3300\_COM.h, 690
  - libIDT\_VP3300\_USB.h, 727
- icc\_setKeyFormatForICCDUKPT
  - libIDT\_Device.h, 193
- icc\_setKeyTypeForICCDUKPT
  - libIDT\_Device.h, 193
- iso8583\_deserializeFromXML
  - libIDT\_Device.h, 194
- iso8583\_displayMessage
  - libIDT\_Device.h, 194
- iso8583\_freeMessage
  - libIDT\_Device.h, 194
- iso8583\_get1987Handler
  - libIDT\_Device.h, 195
- iso8583\_get1993Handler
  - libIDT\_Device.h, 195
- iso8583\_get2003Handler
  - libIDT\_Device.h, 195
- iso8583\_getField
  - libIDT\_Device.h, 195
- iso8583\_getMessageField
  - libIDT\_Device.h, 195
- iso8583\_initializeMessage
  - libIDT\_Device.h, 197
- iso8583\_packMessage
  - libIDT\_Device.h, 197
- iso8583\_removeMessageField
  - libIDT\_Device.h, 197
- iso8583\_serializeToXML
  - libIDT\_Device.h, 197
- iso8583\_setMessageField
  - libIDT\_Device.h, 198
- iso8583\_unpackMessage
  - libIDT\_Device.h, 198
- lcd\_addButton
  - libIDT\_Device.h, 198
  - libIDT\_NEO2.h, 429
- lcd\_addEthernet
  - libIDT\_Device.h, 199
  - libIDT\_NEO2.h, 430

- lcd\_addImage
  - libIDT\_Device.h, 200
  - libIDT\_NEO2.h, 431
- lcd\_addItemToList
  - libIDT\_Device.h, 201
  - libIDT\_VP8800.h, 772
- lcd\_addLED
  - libIDT\_Device.h, 201
  - libIDT\_NEO2.h, 432
- lcd\_addText
  - libIDT\_Device.h, 203
  - libIDT\_NEO2.h, 433
- lcd\_addVideo
  - libIDT\_Device.h, 206
  - libIDT\_NEO2.h, 436
- lcd\_cancelSlideShow
  - libIDT\_Device.h, 206
  - libIDT\_VP8800.h, 772
- lcd\_captureSignature
  - libIDT\_Device.h, 207
  - libIDT\_VP8800.h, 773
- lcd\_clearDisplay
  - libIDT\_Device.h, 207
  - libIDT\_VP8800.h, 773
- lcd\_clearEventQueue
  - libIDT\_Device.h, 207
  - libIDT\_VP8800.h, 773
- lcd\_clearScreenInfo
  - libIDT\_Device.h, 207
  - libIDT\_NEO2.h, 436
- lcd\_cloneScreen
  - libIDT\_Device.h, 207
  - libIDT\_NEO2.h, 437
- lcd\_createInputField
  - libIDT\_Device.h, 209
  - libIDT\_VP8800.h, 773
- lcd\_createInputField\_Len
  - libIDT\_Device.h, 210
  - libIDT\_VP8800.h, 775
- lcd\_createList
  - libIDT\_Device.h, 211
  - libIDT\_VP8800.h, 776
- lcd\_createList\_Len
  - libIDT\_Device.h, 212
  - libIDT\_VP8800.h, 777
- lcd\_createScreen
  - libIDT\_Device.h, 213
  - libIDT\_NEO2.h, 437
- lcd\_customDisplayMode
  - libIDT\_Device.h, 213
  - libIDT\_VP8800.h, 778
- lcd\_destroyScreen
  - libIDT\_Device.h, 215
  - libIDT\_NEO2.h, 437
- lcd\_displayButton
  - libIDT\_Device.h, 215
  - libIDT\_VP8800.h, 778
- lcd\_displayButton\_Len
  - libIDT\_Device.h, 217
  - libIDT\_VP8800.h, 780
- lcd\_displayMessage
  - libIDT\_Device.h, 218
  - libIDT\_L100.h, 303
- lcd\_displayParagraph
  - libIDT\_Device.h, 218
  - libIDT\_VP8800.h, 781
- lcd\_displayPrompt
  - libIDT\_Device.h, 219
  - libIDT\_L100.h, 303
- lcd\_displayText
  - libIDT\_Device.h, 219
  - libIDT\_VP8800.h, 782
- lcd\_displayText\_Len
  - libIDT\_Device.h, 221
  - libIDT\_VP8800.h, 783
- lcd\_enableBacklight
  - libIDT\_Device.h, 222
  - libIDT\_L100.h, 304
- lcd\_getActiveScreen
  - libIDT\_Device.h, 222
  - libIDT\_NEO2.h, 437
- lcd\_getAllObjects
  - libIDT\_Device.h, 223
  - libIDT\_NEO2.h, 438
- lcd\_getAllScreens
  - libIDT\_Device.h, 223
  - libIDT\_NEO2.h, 438
- lcd\_getBacklightStatus
  - libIDT\_Device.h, 223
  - libIDT\_L100.h, 304
- lcd\_getButtonEvent
  - libIDT\_Device.h, 223
  - libIDT\_NEO2.h, 438
- lcd\_getInputEvent
  - libIDT\_Device.h, 224
  - libIDT\_VP8800.h, 784
- lcd\_getInputEvent\_Len
  - libIDT\_Device.h, 226
  - libIDT\_VP8800.h, 786
- lcd\_getInputFieldValue
  - libIDT\_Device.h, 228
  - libIDT\_VP8800.h, 787
- lcd\_getSelectedItem
  - libIDT\_Device.h, 228
  - libIDT\_VP8800.h, 788
- lcd\_getSelectedItem\_Len
  - libIDT\_Device.h, 228
  - libIDT\_VP8800.h, 788
- lcd\_loadScreenInfo
  - libIDT\_Device.h, 228
  - libIDT\_NEO2.h, 439
- lcd\_queryObjectbyID
  - libIDT\_Device.h, 228
  - libIDT\_NEO2.h, 439
- lcd\_queryObjectbyName
  - libIDT\_Device.h, 229

- libIDT\_NEO2.h, [439](#)
- lcd\_queryScreenbyID
  - libIDT\_Device.h, [229](#)
  - libIDT\_NEO2.h, [440](#)
- lcd\_queryScreenbyName
  - libIDT\_Device.h, [229](#)
  - libIDT\_NEO2.h, [440](#)
- lcd\_registerCallBk
  - libIDT\_Device.h, [230](#)
  - libIDT\_NEO2.h, [440](#)
  - libIDT\_SREDKey2.h, [530](#)
- lcd\_removeItem
  - libIDT\_Device.h, [230](#)
  - libIDT\_NEO2.h, [440](#)
- lcd\_resetInitialState
  - libIDT\_Device.h, [230](#)
  - libIDT\_VP8800.h, [788](#)
- lcd\_savePrompt
  - libIDT\_Device.h, [230](#)
  - libIDT\_L100.h, [304](#)
- lcd\_setBackgroundImage
  - libIDT\_Device.h, [230](#)
  - libIDT\_VP8800.h, [788](#)
- lcd\_setBacklight
  - libIDT\_Device.h, [232](#)
  - libIDT\_NEO2.h, [441](#)
- lcd\_setDisplayImage
  - libIDT\_Device.h, [232](#)
  - libIDT\_VP8800.h, [788](#)
- lcd\_setForeBackColor
  - libIDT\_Device.h, [233](#)
  - libIDT\_VP8800.h, [790](#)
- lcd\_showScreen
  - libIDT\_Device.h, [233](#)
  - libIDT\_NEO2.h, [441](#)
- lcd\_startSlideShow
  - libIDT\_Device.h, [233](#)
  - libIDT\_VP8800.h, [790](#)
- lcd\_storeScreenInfo
  - libIDT\_Device.h, [234](#)
  - libIDT\_NEO2.h, [441](#)
- lcd\_updateColor
  - libIDT\_Device.h, [234](#)
  - libIDT\_NEO2.h, [441](#)
- lcd\_updateLabel
  - libIDT\_Device.h, [235](#)
  - libIDT\_NEO2.h, [442](#)
- lcd\_updatePosition
  - libIDT\_Device.h, [235](#)
  - libIDT\_NEO2.h, [442](#)
- libIDT\_Augusta.h
  - config\_getBeeperController, [52](#)
  - config\_getEncryptionControl, [53](#)
  - config\_getLEDController, [53](#)
  - config\_getModelNumber, [54](#)
  - config\_getModelNumber\_Len, [54](#)
  - config\_getSerialNumber, [54](#)
  - config\_getSerialNumber\_Len, [54](#)
  - config\_setBeeperController, [55](#)
  - config\_setEncryptionControl, [55](#)
  - config\_setLEDController, [55](#)
  - device\_SendDataCommand, [72](#)
  - device\_close, [56](#)
  - device\_controlBeep, [56](#)
  - device\_controlLED, [57](#)
  - device\_controlLED\_ICC, [57](#)
  - device\_controlLED\_MSR, [57](#)
  - device\_getCurrentDeviceType, [58](#)
  - device\_getDRS, [58](#)
  - device\_getFirmwareVersion, [58](#)
  - device\_getFirmwareVersion\_Len, [60](#)
  - device\_getKeyStatus, [60](#)
  - device\_getResponseCodeString, [61](#)
  - device\_getSDKWaitTime, [70](#)
  - device\_getThreadStackSize, [71](#)
  - device\_init, [71](#)
  - device\_isAttached, [71](#)
  - device\_isConnected, [71](#)
  - device\_rebootDevice, [71](#)
  - device\_registerCameraCallBk, [72](#)
  - device\_registerCardStatusFrontSwitchCallBk, [72](#)
  - device\_registerFWCallBk, [72](#)
  - device\_setCurrentDevice, [72](#)
  - device\_setSDKWaitTime, [73](#)
  - device\_setThreadStackSize, [73](#)
  - device\_updateFirmware, [73](#)
  - emv\_activateTransaction, [74](#)
  - emv\_allowFallback, [74](#)
  - emv\_authenticateTransaction, [76](#)
  - emv\_authenticateTransactionWithTimeout, [76](#)
  - emv\_callbackResponseLCD, [76](#)
  - emv\_callbackResponseMSR, [77](#)
  - emv\_cancelTransaction, [77](#)
  - emv\_completeTransaction, [77](#)
  - emv\_getAutoAuthenticateTransaction, [78](#)
  - emv\_getAutoCompleteTransaction, [78](#)
  - emv\_getEMVConfigurationCheckValue, [78](#)
  - emv\_getEMVKernelCheckValue, [78](#)
  - emv\_getEMVKernelVersion, [79](#)
  - emv\_getEMVKernelVersion\_Len, [79](#)
  - emv\_registerCallBk, [79](#)
  - emv\_removeAllApplicationData, [79](#)
  - emv\_removeAllCAPK, [79](#)
  - emv\_removeAllCRL, [79](#)
  - emv\_removeApplicationData, [80](#)
  - emv\_removeCAPK, [80](#)
  - emv\_removeCRL, [80](#)
  - emv\_removeTerminalData, [80](#)
  - emv\_retrieveAIDList, [81](#)
  - emv\_retrieveApplicationData, [81](#)
  - emv\_retrieveCAPK, [82](#)
  - emv\_retrieveCAPKList, [82](#)
  - emv\_retrieveCRL, [82](#)
  - emv\_retrieveTerminalData, [84](#)
  - emv\_retrieveTerminalID, [84](#)
  - emv\_retrieveTerminalID\_Len, [84](#)

- emv\_retrieveTransactionResult, 84
- emv\_setApplicationData, 85
- emv\_setAutoAuthenticateTransaction, 85
- emv\_setAutoCompleteTransaction, 85
- emv\_setCAPK, 85
- emv\_setCRL, 86
- emv\_setTerminalData, 86
- emv\_setTerminalID, 87
- emv\_startTransaction, 87
- ftpComm\_callBack, 51
- httpComm\_callBack, 51
- IN, 51
- IN\_OUT, 51
- icc\_disable, 87
- icc\_enable, 87
- icc\_exchangeAPDU, 89
- icc\_exchangeEncryptedAPDU, 89
- icc\_getAPDU\_KSN, 91
- icc\_getFunctionStatus, 91
- icc\_getICCReaderStatus, 91
- icc\_getKeyFormatForICCDUKPT, 91
- icc\_getKeyTypeForICCDUKPT, 93
- icc\_powerOffICC, 93
- icc\_powerOnICC, 93
- msr\_cancelMSRSwipe, 93
- msr\_captureMode, 94
- msr\_disable, 94
- msr\_getClearPANID, 94
- msr\_getExpirationMask, 94
- msr\_getKeyFormatForICCDUKPT, 95
- msr\_getKeyTypeForICCDUKPT, 95
- msr\_getMSRData, 95
- msr\_getSetting, 96
- msr\_getSwipeForcedEncryptionOption, 96
- msr\_getSwipeMaskOption, 96
- msr\_registerCallBk, 96
- msr\_registerCallBkp, 96
- msr\_setClearPANID, 97
- msr\_setExpirationMask, 97
- msr\_setKeyFormatForICCDUKPT, 97
- msr\_setKeyTypeForICCDUKPT, 97
- msr\_setSetting, 98
- msr\_setSwipeForcedEncryptionOption, 98
- msr\_setSwipeMaskOption, 98
- msr\_startMSRSwipe, 98
- OUT, 51
- pCMR\_callBack, 51
- pCSFS\_callBack, 51
- pEMV\_callBack, 51
- pFW\_callBack, 51
- pMSR\_callBack, 52
- pMSR\_callBackp, 52
- pMessageHotplug, 52
- pPIN\_callBack, 52
- pReadDataLog, 52
- pSendDataLog, 52
- parseMSRData, 100
- pin\_cancelPINEntry, 100
- pin\_registerCallBk, 100
- registerHotplugCallBk, 100
- registerLogCallBk, 100
- SDK\_Version, 100
- setAbsoluteLibraryPath, 100
- v4Comm\_callBack, 52
- libIDT\_Device.h
  - config\_getBeeperController, 111
  - config\_getEncryptionControl, 111
  - config\_getLEDController, 111
  - config\_getModelNumber, 112
  - config\_getModelNumber\_Len, 112
  - config\_getSerialNumber, 112
  - config\_getSerialNumber\_Len, 112
  - config\_setBeeperController, 114
  - config\_setCmdTimeOutDuration, 114
  - config\_setEncryptionControl, 114
  - config\_setLEDController, 115
  - ctls\_activateTransaction, 115
  - ctls\_cancelTransaction, 116
  - ctls\_displayOnlineAuthResult, 116
  - ctls\_getAllConfigurationGroups, 118
  - ctls\_getConfigurationGroup, 118
  - ctls\_removeAllApplicationData, 118
  - ctls\_removeAllCAPK, 118
  - ctls\_removeApplicationData, 118
  - ctls\_removeCAPK, 119
  - ctls\_removeConfigurationGroup, 119
  - ctls\_retrieveAIDList, 119
  - ctls\_retrieveApplicationData, 119
  - ctls\_retrieveCAPK, 120
  - ctls\_retrieveCAPKList, 121
  - ctls\_retrieveTerminalData, 121
  - ctls\_setApplicationData, 121
  - ctls\_setCAPK, 121
  - ctls\_setConfigurationGroup, 123
  - ctls\_setTerminalData, 123
  - ctls\_startTransaction, 124
  - device\_SendDataCommand, 162
  - device\_SendDataCommandITP, 162
  - device\_SendDataCommandNEO, 163
  - device\_activateTransaction, 125
  - device\_buzzerOnOff, 126
  - device\_calibrateParameters, 126
  - device\_cancelTransaction, 126
  - device\_cancelTransactionSilent, 127
  - device\_close, 127
  - device\_configureButtons, 127
  - device\_controlBeep, 127
  - device\_controlIndicator, 128
  - device\_controlLED, 128
  - device\_controlLED\_ICC, 129
  - device\_controlLED\_MSR, 129
  - device\_controlUserInterface, 130
  - device\_createDirectory, 131
  - device\_deleteDirectory, 133
  - device\_deleteFile, 133
  - device\_disableBlueLED, 133



- device\_enableBlueLED, 133
- device\_enableExternalLCDMessages, 134
- device\_enableL100PassThrough, 134
- device\_enablePassThrough, 134
- device\_enableRFAntenna, 136
- device\_enhancedPassthrough, 136
- device\_enterStopMode, 136
- device\_getButtonConfiguration, 136
- device\_getCurrentDeviceType, 137
- device\_getDRS, 138
- device\_getDateTime, 137
- device\_getDateTime\_Len, 137
- device\_getDeviceMemoryUsageInfo, 137
- device\_getDriveFreeSpace, 138
- device\_getFirmwareVersion, 138
- device\_getFirmwareVersion\_Len, 139
- device\_getIDGStatusCodeString, 139
- device\_getKeyStatus, 140
- device\_getL100PassThroughMode, 141
- device\_getMerchantRecord, 141
- device\_getMerchantRecord\_Len, 142
- device\_getRTCDateTime, 152
- device\_getResponseCodeString, 142
- device\_getSDKWaitTime, 152
- device\_getSpectrumProKSN, 152
- device\_getSpectrumProKSN\_Len, 153
- device\_getThreadStackSize, 154
- device\_init, 154
- device\_isAttached, 154
- device\_isConnected, 154
- device\_lcdDisplayClear, 154
- device\_lcdDisplayLine1Message, 154
- device\_lcdDisplayLine2Message, 156
- device\_listDirectory, 156
- device\_pingDevice, 156
- device\_pollCardReader, 156
- device\_pollCardReader\_Len, 159
- device\_pollForToken, 161
- device\_queryFile, 161
- device\_rebootDevice, 161
- device\_registerCameraCallBk, 162
- device\_registerCardStatusFrontSwitchCallBk, 162
- device\_registerFWCallBk, 162
- device\_registerRKICallBk, 162
- device\_selfCheck, 162
- device\_setBurstMode, 163
- device\_setCancelTransactionMode, 163
- device\_setConfigPath, 164
- device\_setCurrentDevice, 164
- device\_setMerchantRecord, 164
- device\_setNEO2DevicesConfigs, 165
- device\_setPollMode, 165
- device\_setRTCDateTime, 165
- device\_setSDKWaitTime, 165
- device\_setSleepModeTime, 166
- device\_setSystemLanguage, 166
- device\_setThreadStackSize, 166
- device\_setTransactionExponent, 166
- device\_startListenNotifications, 167
- device\_startQRCodeScan, 167
- device\_startRKI, 167
- device\_startTakingPhoto, 167
- device\_startTransaction, 167
- device\_stopListenNotifications, 169
- device\_stopQRCodeScan, 169
- device\_stopTakingPhoto, 169
- device\_toSDCard, 169
- device\_transferFile, 170
- device\_turnOffYellowLED, 170
- device\_turnOnYellowLED, 170
- device\_updateFirmware, 171
- device\_verifyBackdoorKey, 171
- emv\_activateTransaction, 171
- emv\_allowFallback, 173
- emv\_authenticateTransaction, 173
- emv\_authenticateTransactionWithTimeout, 173
- emv\_callbackResponseLCD, 175
- emv\_callbackResponseMSR, 175
- emv\_cancelTransaction, 175
- emv\_completeTransaction, 176
- emv\_getAutoAuthenticateTransaction, 176
- emv\_getAutoCompleteTransaction, 176
- emv\_getEMVConfigurationCheckValue, 176
- emv\_getEMVKernelCheckValue, 177
- emv\_getEMVKernelVersion, 177
- emv\_getEMVKernelVersion\_Len, 177
- emv\_registerCallBk, 177
- emv\_removeAllApplicationData, 177
- emv\_removeAllCAPK, 178
- emv\_removeAllCRL, 178
- emv\_removeApplicationData, 178
- emv\_removeCAPK, 178
- emv\_removeCRL, 178
- emv\_removeTerminalData, 179
- emv\_retrieveAIDList, 179
- emv\_retrieveApplicationData, 179
- emv\_retrieveCAPK, 179
- emv\_retrieveCAPKList, 181
- emv\_retrieveCRL, 181
- emv\_retrieveTerminalData, 181
- emv\_retrieveTerminalID, 182
- emv\_retrieveTerminalID\_Len, 182
- emv\_retrieveTransactionResult, 182
- emv\_setApplicationData, 182
- emv\_setApplicationDataTLV, 183
- emv\_setAutoAuthenticateTransaction, 183
- emv\_setAutoCompleteTransaction, 183
- emv\_setCAPK, 183
- emv\_setCRL, 184
- emv\_setTerminalData, 184
- emv\_setTerminalID, 185
- emv\_setTerminalMajorConfiguration, 185
- emv\_setTransactionParameters, 185
- emv\_startTransaction, 186
- felica\_SendCommand, 188
- felica\_authentication, 186

- felica\_poll, 187
- felica\_read, 187
- felica\_readWithMac, 187
- felica\_requestService, 188
- felica\_write, 188
- felica\_writeWithMac, 189
- ftpComm\_callBack, 109
- httpComm\_callBack, 109
- IN, 109
- IN\_OUT, 109
- icc\_disable, 189
- icc\_enable, 189
- icc\_exchangeAPDU, 189
- icc\_exchangeEncryptedAPDU, 190
- icc\_getAPDU\_KSN, 190
- icc\_getFunctionStatus, 190
- icc\_getICCReaderStatus, 192
- icc\_getKeyFormatForICCDUKPT, 192
- icc\_getKeyTypeForICCDUKPT, 192
- icc\_powerOffICC, 193
- icc\_powerOnICC, 193
- icc\_setKeyFormatForICCDUKPT, 193
- icc\_setKeyTypeForICCDUKPT, 193
- iso8583\_deserializeFromXML, 194
- iso8583\_displayMessage, 194
- iso8583\_freeMessage, 194
- iso8583\_get1987Handler, 195
- iso8583\_get1993Handler, 195
- iso8583\_get2003Handler, 195
- iso8583\_getField, 195
- iso8583\_getMessageField, 195
- iso8583\_initializeMessage, 197
- iso8583\_packMessage, 197
- iso8583\_removeMessageField, 197
- iso8583\_serializeToXML, 197
- iso8583\_setMessageField, 198
- iso8583\_unpackMessage, 198
- lcd\_addButton, 198
- lcd\_addEthernet, 199
- lcd\_addImage, 200
- lcd\_addItemToList, 201
- lcd\_addLED, 201
- lcd\_addText, 203
- lcd\_addVideo, 206
- lcd\_cancelSlideShow, 206
- lcd\_captureSignature, 207
- lcd\_clearDisplay, 207
- lcd\_clearEventQueue, 207
- lcd\_clearScreenInfo, 207
- lcd\_cloneScreen, 207
- lcd\_createInputField, 209
- lcd\_createInputField\_Len, 210
- lcd\_createList, 211
- lcd\_createList\_Len, 212
- lcd\_createScreen, 213
- lcd\_customDisplayMode, 213
- lcd\_destroyScreen, 215
- lcd\_displayButton, 215
- lcd\_displayButton\_Len, 217
- lcd\_displayMessage, 218
- lcd\_displayParagraph, 218
- lcd\_displayPrompt, 219
- lcd\_displayText, 219
- lcd\_displayText\_Len, 221
- lcd\_enableBacklight, 222
- lcd\_getActiveScreen, 222
- lcd\_getAllObjects, 223
- lcd\_getAllScreens, 223
- lcd\_getBacklightStatus, 223
- lcd\_getButtonEvent, 223
- lcd\_getInputEvent, 224
- lcd\_getInputEvent\_Len, 226
- lcd\_getInputFieldValue, 228
- lcd\_getSelectedItem, 228
- lcd\_getSelectedItem\_Len, 228
- lcd\_loadScreenInfo, 228
- lcd\_queryObjectbyID, 228
- lcd\_queryObjectbyName, 229
- lcd\_queryScreenbyID, 229
- lcd\_queryScreenbyName, 229
- lcd\_registerCallBk, 230
- lcd\_removeItem, 230
- lcd\_resetInitialState, 230
- lcd\_savePrompt, 230
- lcd\_setBackgroundImage, 230
- lcd\_setBacklight, 232
- lcd\_setDisplayImage, 232
- lcd\_setForeBackColor, 233
- lcd\_showScreen, 233
- lcd\_startSlideShow, 233
- lcd\_storeScreenInfo, 234
- lcd\_updateColor, 234
- lcd\_updateLabel, 235
- lcd\_updatePosition, 235
- loyalty\_cancelTransaction, 236
- loyalty\_cancelTransactionSilent, 236
- loyalty\_registerCallBk, 236
- loyalty\_startTransaction, 236
- msr\_cancelMSRSwipe, 238
- msr\_captureMode, 238
- msr\_clearMSRData, 238
- msr\_disable, 238
- msr\_flushTrackData, 239
- msr\_getClearPANID, 239
- msr\_getExpirationMask, 239
- msr\_getFunctionStatus, 239
- msr\_getKeyFormatForICCDUKPT, 240
- msr\_getKeyTypeForICCDUKPT, 240
- msr\_getMSRData, 240
- msr\_getSwipeForcedEncryptionOption, 241
- msr\_getSwipeMaskOption, 241
- msr\_registerCallBk, 241
- msr\_registerCallBkp, 241
- msr\_setClearPANID, 241
- msr\_setExpirationMask, 241
- msr\_setKeyFormatForICCDUKPT, 242



- msr\_setKeyTypeForICCDUKPT, 242
- msr\_setSwipeForcedEncryptionOption, 242
- msr\_setSwipeMaskOption, 243
- msr\_startMSRSwipe, 243
- OUT, 109
- pCMR\_callback, 109
- pCSFS\_callback, 109
- pEMV\_callback, 109
- pFW\_callback, 109
- pLCD\_callback, 109
- pLog\_callback, 110
- pMSR\_callback, 110
- pMSR\_callbackp, 110
- pMessageHotplug, 110
- pPIN\_callback, 110
- pRKI\_callback, 110
- pReadDataLog, 110
- pSendDataLog, 110
- parseMSRData, 243
- parsePINBlockData, 243
- parsePINData, 244
- pin\_cancelPINEntry, 244
- pin\_capturePin, 244
- pin\_capturePinExt, 244
- pin\_getEncryptedOnlinePIN, 245
- pin\_getEncryptedPIN, 246
- pin\_getFunctionKey, 246
- pin\_getPAN, 246
- pin\_getPIN, 248
- pin\_getPanEntry, 248
- pin\_inputFromPrompt, 249
- pin\_promptCreditDebit, 249
- pin\_promptForAmount, 250
- pin\_promptForAmountInput, 250
- pin\_promptForKeyInput, 253
- pin\_promptForNumericKey, 257
- pin\_promptForNumericKeyWithSwipe, 258
- pin\_registerCallBk, 258
- pin\_sendBeep, 258
- pin\_setKeyValues, 259
- registerHotplugCallBk, 259
- registerLogCallBk, 259
- rs232\_device\_init, 259
- SDK\_Version, 260
- setAbsoluteLibraryPath, 260
- v4Comm\_callback, 110
- ws\_deleteSSLCert, 260
- ws\_getCertChainType, 260
- ws\_loadSSLCert, 260
- ws\_requestCSR, 261
- ws\_revokeSSLCert, 261
- ws\_updateRootCertificate, 261
- libIDT\_KioskIII.h
  - config\_getSerialNumber, 265
  - config\_getSerialNumber\_Len, 265
  - ctls\_activateTransaction, 265
  - ctls\_cancelTransaction, 267
  - ctls\_getAllConfigurationGroups, 268
  - ctls\_getConfigurationGroup, 268
  - ctls\_registerCallBk, 268
  - ctls\_registerCallBkp, 268
  - ctls\_removeAllApplicationData, 268
  - ctls\_removeAllCAPK, 268
  - ctls\_removeApplicationData, 269
  - ctls\_removeCAPK, 269
  - ctls\_removeConfigurationGroup, 269
  - ctls\_retrieveAIDList, 269
  - ctls\_retrieveApplicationData, 270
  - ctls\_retrieveCAPK, 270
  - ctls\_retrieveCAPKList, 271
  - ctls\_retrieveTerminalData, 271
  - ctls\_setApplicationData, 271
  - ctls\_setCAPK, 271
  - ctls\_setConfigurationGroup, 272
  - ctls\_setTerminalData, 272
  - ctls\_startTransaction, 273
  - device\_SendDataCommandNEO, 279
  - device\_close, 274
  - device\_controlUserInterface, 274
  - device\_enablePassThrough, 275
  - device\_getCurrentDeviceType, 275
  - device\_getFirmwareVersion, 275
  - device\_getFirmwareVersion\_Len, 275
  - device\_getIDGStatusCodeString, 276
  - device\_getMerchantRecord, 277
  - device\_getMerchantRecord\_Len, 277
  - device\_getSDKWaitTime, 278
  - device\_getThreadStackSize, 278
  - device\_getTransactionResults, 278
  - device\_init, 278
  - device\_isAttached, 279
  - device\_isConnected, 279
  - device\_pingDevice, 279
  - device\_registerCameraCallBk, 279
  - device\_registerCardStatusFrontSwitchCallBk, 279
  - device\_setBurstMode, 280
  - device\_setCurrentDevice, 280
  - device\_setMerchantRecord, 281
  - device\_setPollMode, 281
  - device\_setSDKWaitTime, 282
  - device\_setThreadStackSize, 282
  - emv\_registerCallBk, 282
  - ftpComm\_callback, 264
  - httpComm\_callback, 264
  - IN, 263
  - IN\_OUT, 263
  - OUT, 263
  - pCMR\_callback, 264
  - pCSFS\_callback, 264
  - pEMV\_callback, 264
  - pMSR\_callback, 264
  - pMSR\_callbackp, 264
  - pMessageHotplug, 264
  - pPIN\_callback, 264
  - pReadDataLog, 265
  - pSendDataLog, 265

- parseMSRData, [282](#)
- pin\_registerCallBk, [282](#)
- registerHotplugCallBk, [282](#)
- registerLogCallBk, [282](#)
- rs232\_device\_init, [282](#)
- SDK\_Version, [283](#)
- setAbsoluteLibraryPath, [283](#)
- v4Comm\_callBack, [265](#)
- libIDT\_L100.h
  - config\_getModelNumber, [287](#)
  - config\_getModelNumber\_Len, [287](#)
  - config\_getSerialNumber, [287](#)
  - config\_getSerialNumber\_Len, [287](#)
  - device\_SendDataCommand, [301](#)
  - device\_close, [288](#)
  - device\_enterStopMode, [288](#)
  - device\_getCurrentDeviceType, [288](#)
  - device\_getDateTime, [288](#)
  - device\_getDateTime\_Len, [288](#)
  - device\_getFirmwareVersion, [289](#)
  - device\_getFirmwareVersion\_Len, [289](#)
  - device\_getKeyStatus, [289](#)
  - device\_getResponseCodeString, [290](#)
  - device\_init, [300](#)
  - device\_isAttached, [300](#)
  - device\_isConnected, [300](#)
  - device\_rebootDevice, [300](#)
  - device\_registerCameraCallBk, [300](#)
  - device\_registerCardStatusFrontSwitchCallBk, [301](#)
  - device\_registerFWCallBk, [301](#)
  - device\_setCurrentDevice, [301](#)
  - device\_setSleepModeTime, [302](#)
  - device\_updateFirmware, [302](#)
  - emv\_registerCallBk, [303](#)
  - ftpComm\_callBack, [285](#)
  - httpComm\_callBack, [285](#)
  - IN, [285](#)
  - IN\_OUT, [285](#)
  - lcd\_displayMessage, [303](#)
  - lcd\_displayPrompt, [303](#)
  - lcd\_enableBacklight, [304](#)
  - lcd\_getBacklightStatus, [304](#)
  - lcd\_savePrompt, [304](#)
  - msr\_registerCallBk, [304](#)
  - msr\_registerCallBkp, [304](#)
  - OUT, [285](#)
  - pCMR\_callBack, [285](#)
  - pCSFS\_callBack, [286](#)
  - pEMV\_callBack, [286](#)
  - pFW\_callBack, [286](#)
  - pMSR\_callBack, [286](#)
  - pMSR\_callBackp, [286](#)
  - pMessageHotplug, [286](#)
  - pPIN\_callBack, [286](#)
  - pReadDataLog, [286](#)
  - pSendDataLog, [286](#)
  - pin\_getEncryptedPIN, [305](#)
  - pin\_getFunctionKey, [305](#)
  - pin\_promptForAmountInput, [306](#)
  - pin\_promptForKeyInput, [309](#)
  - pin\_registerCallBk, [312](#)
  - pin\_sendBeep, [312](#)
  - pin\_setKeyValues, [313](#)
  - registerHotplugCallBk, [313](#)
  - registerLogCallBk, [313](#)
  - SDK\_Version, [313](#)
  - setAbsoluteLibraryPath, [313](#)
  - v4Comm\_callBack, [287](#)
- libIDT\_MiniSmartII.h
  - comm\_registerHTTPCallback, [318](#)
  - comm\_registerV4Callback, [318](#)
  - config\_getBeeperController, [318](#)
  - config\_getEncryptionControl, [318](#)
  - config\_getLEDController, [319](#)
  - config\_getModelNumber, [319](#)
  - config\_getModelNumber\_Len, [319](#)
  - config\_getSerialNumber, [320](#)
  - config\_getSerialNumber\_Len, [320](#)
  - config\_setBeeperController, [320](#)
  - config\_setEncryptionControl, [320](#)
  - config\_setLEDController, [322](#)
  - device\_SendDataCommand, [337](#)
  - device\_close, [322](#)
  - device\_controlBeep, [322](#)
  - device\_controlLED, [323](#)
  - device\_controlLED\_ICC, [323](#)
  - device\_controlLED\_MSR, [323](#)
  - device\_getCurrentDeviceType, [324](#)
  - device\_getFirmwareVersion, [324](#)
  - device\_getFirmwareVersion\_Len, [324](#)
  - device\_getKeyStatus, [325](#)
  - device\_getResponseCodeString, [325](#)
  - device\_getSDKWaitTime, [335](#)
  - device\_getThreadStackSize, [335](#)
  - device\_init, [336](#)
  - device\_isAttached, [336](#)
  - device\_isConnected, [336](#)
  - device\_rebootDevice, [336](#)
  - device\_registerCameraCallBk, [336](#)
  - device\_registerCardStatusFrontSwitchCallBk, [337](#)
  - device\_setCurrentDevice, [337](#)
  - device\_setSDKWaitTime, [338](#)
  - device\_setThreadStackSize, [338](#)
  - device\_updateFirmware, [338](#)
  - emv\_activateTransaction, [339](#)
  - emv\_allowFallback, [339](#)
  - emv\_authenticateTransaction, [341](#)
  - emv\_authenticateTransactionWithTimeout, [341](#)
  - emv\_callbackResponseLCD, [342](#)
  - emv\_callbackResponseMSR, [342](#)
  - emv\_cancelTransaction, [342](#)
  - emv\_completeTransaction, [342](#)
  - emv\_getAutoAuthenticateTransaction, [343](#)
  - emv\_getAutoCompleteTransaction, [343](#)
  - emv\_getEMVConfigurationCheckValue, [343](#)
  - emv\_getEMVKernelCheckValue, [343](#)

- emv\_getEMVKernelVersion, 344
- emv\_getEMVKernelVersion\_Len, 344
- emv\_registerCallBk, 344
- emv\_removeAllApplicationData, 344
- emv\_removeAllCAPK, 344
- emv\_removeAllCRL, 344
- emv\_removeApplicationData, 345
- emv\_removeCAPK, 345
- emv\_removeCRL, 345
- emv\_removeTerminalData, 345
- emv\_retrieveAIDList, 346
- emv\_retrieveApplicationData, 346
- emv\_retrieveCAPK, 347
- emv\_retrieveCAPKList, 347
- emv\_retrieveCRL, 347
- emv\_retrieveTerminalData, 349
- emv\_retrieveTerminalID, 349
- emv\_retrieveTerminalID\_Len, 349
- emv\_retrieveTransactionResult, 349
- emv\_setApplicationData, 350
- emv\_setAutoAuthenticateTransaction, 350
- emv\_setAutoCompleteTransaction, 350
- emv\_setCAPK, 350
- emv\_setCRL, 351
- emv\_setTerminalData, 351
- emv\_setTerminalID, 352
- emv\_startTransaction, 352
- ftpComm\_callBack, 316
- httpComm\_callBack, 316
- IN, 316
- IN\_OUT, 316
- icc\_disable, 352
- icc\_enable, 352
- icc\_exchangeAPDU, 354
- icc\_exchangeEncryptedAPDU, 354
- icc\_getAPDU\_KSN, 356
- icc\_getFunctionStatus, 356
- icc\_getICCRReaderStatus, 356
- icc\_powerOffICC, 358
- icc\_powerOnICC, 358
- msr\_registerCallBk, 358
- msr\_registerCallBkp, 359
- OUT, 316
- pCMR\_callBack, 316
- pCSFS\_callBack, 317
- pEMV\_callBack, 317
- pMSR\_callBack, 317
- pMSR\_callBkp, 317
- pMessageHotplug, 317
- pPIN\_callBack, 317
- pReadDataLog, 317
- pSendDataLog, 317
- pin\_registerCallBk, 359
- registerHotplugCallBk, 359
- registerLogCallBk, 359
- rs232\_device\_init, 359
- SDK\_Version, 359
- setAbsoluteLibraryPath, 360
- v4Comm\_callBack, 317
- libIDT\_NEO2.h
  - comm\_registerHTTPCallback, 367
  - comm\_registerV4Callback, 367
  - config\_getModelNumber, 367
  - config\_getModelNumber\_Len, 367
  - config\_getSerialNumber, 368
  - config\_getSerialNumber\_Len, 368
  - config\_setCmdTimeOutDuration, 368
  - ctls\_activateTransaction, 368
  - ctls\_cancelTransaction, 369
  - ctls\_displayOnlineAuthResult, 370
  - ctls\_getAllConfigurationGroups, 370
  - ctls\_getConfigurationGroup, 370
  - ctls\_registerCallBk, 370
  - ctls\_registerCallBkp, 371
  - ctls\_removeAllApplicationData, 371
  - ctls\_removeAllCAPK, 371
  - ctls\_removeApplicationData, 371
  - ctls\_removeCAPK, 371
  - ctls\_removeConfigurationGroup, 371
  - ctls\_retrieveAIDList, 372
  - ctls\_retrieveApplicationData, 372
  - ctls\_retrieveCAPK, 372
  - ctls\_retrieveCAPKList, 373
  - ctls\_retrieveTerminalData, 373
  - ctls\_setApplicationData, 374
  - ctls\_setCAPK, 374
  - ctls\_setConfigurationGroup, 375
  - ctls\_setTerminalData, 375
  - ctls\_startTransaction, 375
  - device\_SendDataCommandNEO, 405
  - device\_activateTransaction, 377
  - device\_buzzerOnOff, 378
  - device\_cancelTransaction, 378
  - device\_cancelTransactionSilent, 378
  - device\_close, 378
  - device\_configureButtons, 378
  - device\_controlUserInterface, 380
  - device\_deleteDirectory, 381
  - device\_deleteFile, 383
  - device\_disableBlueLED, 383
  - device\_enableBlueLED, 383
  - device\_enableExternalLCDMessages, 383
  - device\_enableL100PassThrough, 385
  - device\_enablePassThrough, 385
  - device\_enableRFAntenna, 385
  - device\_getButtonConfiguration, 385
  - device\_getCurrentDeviceType, 387
  - device\_getDeviceMemoryUsagelInfo, 387
  - device\_getFirmwareVersion, 387
  - device\_getFirmwareVersion\_Len, 387
  - device\_getIDGStatusCodeString, 388
  - device\_getKeyStatus, 389
  - device\_getL100PassThroughMode, 390
  - device\_getMerchantRecord, 390
  - device\_getMerchantRecord\_Len, 390
  - device\_getResponseCodeString, 391

device\_getSDKWaitTime, 400  
 device\_getThreadStackSize, 401  
 device\_getTransactionResults, 401  
 device\_init, 401  
 device\_isAttached, 401  
 device\_isConnected, 402  
 device\_lcdDisplayClear, 402  
 device\_lcdDisplayLine1Message, 402  
 device\_lcdDisplayLine2Message, 402  
 device\_listDirectory, 402  
 device\_pingDevice, 404  
 device\_pollForToken, 404  
 device\_queryFile, 404  
 device\_registerCameraCallBk, 405  
 device\_registerCardStatusFrontSwitchCallBk, 405  
 device\_registerFWCallBk, 405  
 device\_setBurstMode, 405  
 device\_setCancelTransactionMode, 406  
 device\_setConfigPath, 406  
 device\_setCurrentDevice, 406  
 device\_setMerchantRecord, 407  
 device\_setNEO2DevicesConfigs, 407  
 device\_setPollMode, 408  
 device\_setSDKWaitTime, 408  
 device\_setThreadStackSize, 408  
 device\_setTransactionExponent, 408  
 device\_startListenNotifications, 408  
 device\_startQRCodeScan, 408  
 device\_startTakingPhoto, 409  
 device\_startTransaction, 409  
 device\_stopListenNotifications, 410  
 device\_stopQRCodeScan, 410  
 device\_stopTakingPhoto, 411  
 device\_toSDCard, 411  
 device\_transferFile, 411  
 device\_turnOffYellowLED, 411  
 device\_turnOnYellowLED, 411  
 device\_updateFirmware, 412  
 emv\_activateTransaction, 412  
 emv\_allowFallback, 413  
 emv\_authenticateTransaction, 413  
 emv\_authenticateTransactionWithTimeout, 413  
 emv\_cancelTransaction, 415  
 emv\_completeTransaction, 415  
 emv\_getAutoAuthenticateTransaction, 415  
 emv\_getAutoCompleteTransaction, 416  
 emv\_getEMVConfigurationCheckValue, 416  
 emv\_getEMVKernelCheckValue, 416  
 emv\_getEMVKernelVersion, 416  
 emv\_getEMVKernelVersion\_Len, 416  
 emv\_registerCallBk, 417  
 emv\_removeAllApplicationData, 417  
 emv\_removeAllCAPK, 417  
 emv\_removeAllCRL, 417  
 emv\_removeApplicationData, 417  
 emv\_removeCAPK, 418  
 emv\_removeCRL, 418  
 emv\_retrieveAIDList, 418  
 emv\_retrieveApplicationData, 418  
 emv\_retrieveCAPK, 419  
 emv\_retrieveCAPKList, 420  
 emv\_retrieveCRL, 420  
 emv\_retrieveTerminalData, 420  
 emv\_retrieveTransactionResult, 420  
 emv\_setApplicationData, 421  
 emv\_setApplicationDataTLV, 421  
 emv\_setAutoAuthenticateTransaction, 421  
 emv\_setAutoCompleteTransaction, 422  
 emv\_setCAPK, 422  
 emv\_setCRL, 423  
 emv\_setTerminalData, 423  
 emv\_setTerminalMajorConfiguration, 423  
 emv\_setTransactionParameters, 424  
 emv\_startTransaction, 424  
 felica\_SendCommand, 427  
 felica\_authentication, 424  
 felica\_poll, 426  
 felica\_read, 426  
 felica\_readWithMac, 426  
 felica\_requestService, 427  
 felica\_write, 427  
 felica\_writeWithMac, 428  
 ftpComm\_callBack, 365  
 httpComm\_callBack, 365  
 IN, 365  
 IN\_OUT, 365  
 icc\_exchangeAPDU, 428  
 icc\_getICCRReaderStatus, 428  
 icc\_powerOffICC, 429  
 icc\_powerOnICC, 429  
 lcd\_addButton, 429  
 lcd\_addEthernet, 430  
 lcd\_addImage, 431  
 lcd\_addLED, 432  
 lcd\_addText, 433  
 lcd\_addVideo, 436  
 lcd\_clearScreenInfo, 436  
 lcd\_cloneScreen, 437  
 lcd\_createScreen, 437  
 lcd\_destroyScreen, 437  
 lcd\_getActiveScreen, 437  
 lcd\_getAllObjects, 438  
 lcd\_getAllScreens, 438  
 lcd\_getButtonEvent, 438  
 lcd\_loadScreenInfo, 439  
 lcd\_queryObjectbyID, 439  
 lcd\_queryObjectbyName, 439  
 lcd\_queryScreenbyID, 440  
 lcd\_queryScreenbyName, 440  
 lcd\_registerCallBk, 440  
 lcd\_removeItem, 440  
 lcd\_setBacklight, 441  
 lcd\_showScreen, 441  
 lcd\_storeScreenInfo, 441  
 lcd\_updateColor, 441  
 lcd\_updateLabel, 442

- lcd\_updatePosition, 442
- loyalty\_cancelTransaction, 443
- loyalty\_cancelTransactionSilent, 443
- loyalty\_registerCallBk, 443
- loyalty\_startTransaction, 443
- msr\_cancelMSRSwipe, 446
- msr\_registerCallBk, 446
- msr\_registerCallBkp, 446
- msr\_startMSRSwipe, 446
- OUT, 365
- pCMR\_callBack, 365
- pCSFS\_callBack, 366
- pEMV\_callBack, 366
- pFW\_callBack, 366
- pLCD\_callBack, 366
- pMSR\_callBack, 366
- pMSR\_callBackp, 366
- pMessageHotplug, 366
- pPIN\_callBack, 366
- pReadDataLog, 366
- pSendDataLog, 367
- parseMSRData, 446
- pin\_cancelPINEntry, 447
- pin\_capturePin, 447
- pin\_capturePinExt, 447
- pin\_getPanEntry, 448
- pin\_inputFromPrompt, 449
- pin\_promptForNumericKey, 450
- pin\_promptForNumericKeyWithSwipe, 450
- pin\_registerCallBk, 451
- pin\_setKeyValues, 451
- registerHotplugCallBk, 451
- registerLogCallBk, 451
- rs232\_device\_init, 452
- SDK\_Version, 452
- setAbsoluteLibraryPath, 452
- v4Comm\_callBack, 367
- libIDT\_PipReader.h
  - config\_getSerialNumber, 456
  - config\_getSerialNumber\_Len, 456
  - ctls\_activateTransaction, 456
  - ctls\_cancelTransaction, 458
  - ctls\_getAllConfigurationGroups, 459
  - ctls\_getConfigurationGroup, 459
  - ctls\_registerCallBk, 459
  - ctls\_registerCallBkp, 459
  - ctls\_removeAllApplicationData, 459
  - ctls\_removeAllCAPK, 459
  - ctls\_removeApplicationData, 460
  - ctls\_removeCAPK, 460
  - ctls\_removeConfigurationGroup, 460
  - ctls\_retrieveAIDList, 460
  - ctls\_retrieveApplicationData, 461
  - ctls\_retrieveCAPK, 461
  - ctls\_retrieveCAPKList, 462
  - ctls\_retrieveTerminalData, 462
  - ctls\_setApplicationData, 462
  - ctls\_setCAPK, 462
  - ctls\_setConfigurationGroup, 463
  - ctls\_setTerminalData, 463
  - ctls\_startTransaction, 464
  - device\_SendDataCommandNEO, 470
  - device\_close, 465
  - device\_controlUserInterface, 465
  - device\_enablePassThrough, 466
  - device\_getCurrentDeviceType, 466
  - device\_getFirmwareVersion, 466
  - device\_getFirmwareVersion\_Len, 466
  - device\_getIDGStatusCodeString, 467
  - device\_getMerchantRecord, 468
  - device\_getMerchantRecord\_Len, 468
  - device\_getSDKWaitTime, 469
  - device\_getTransactionResults, 469
  - device\_init, 469
  - device\_isAttached, 469
  - device\_isConnected, 470
  - device\_pingDevice, 470
  - device\_registerCameraCallBk, 470
  - device\_registerCardStatusFrontSwitchCallBk, 470
  - device\_setBurstMode, 470
  - device\_setCurrentDevice, 472
  - device\_setMerchantRecord, 473
  - device\_setPollMode, 473
  - device\_setSDKWaitTime, 473
  - emv\_registerCallBk, 473
  - ftpComm\_callBack, 455
  - httpComm\_callBack, 455
  - IN, 454
  - IN\_OUT, 454
  - OUT, 454
  - pCMR\_callBack, 455
  - pCSFS\_callBack, 455
  - pEMV\_callBack, 455
  - pMSR\_callBack, 455
  - pMSR\_callBackp, 455
  - pMessageHotplug, 455
  - pPIN\_callBack, 455
  - pReadDataLog, 456
  - pSendDataLog, 456
  - parseMSRData, 473
  - pin\_registerCallBk, 474
  - registerHotplugCallBk, 474
  - registerLogCallBk, 474
  - rs232\_device\_init, 474
  - SDK\_Version, 474
  - setAbsoluteLibraryPath, 475
  - v4Comm\_callBack, 456
- libIDT\_SREDKey2.h
  - comm\_registerHTTPCallback, 519
  - comm\_registerV4Callback, 519
  - config\_getModelNumber, 521
  - config\_getModelNumber\_Len, 521
  - config\_getSerialNumber, 521
  - config\_getSerialNumber\_Len, 521
  - ctls\_registerCallBk, 521
  - ctls\_registerCallBkp, 522

- device\_SendDataCommand, 526
- device\_SendDataCommandITP, 526
- device\_SendDataCommandNEO, 526
- device\_close, 522
- device\_getCurrentDeviceType, 522
- device\_getFirmwareVersion, 522
- device\_getFirmwareVersion\_Len, 522
- device\_getIDGStatusCodeString, 522
- device\_getKeyStatus, 524
- device\_init, 524
- device\_isAttached, 525
- device\_isConnected, 525
- device\_pingDevice, 525
- device\_rebootDevice, 525
- device\_registerCameraCallBk, 525
- device\_registerCardStatusFrontSwitchCallBk, 525
- device\_registerFWCallBk, 525
- device\_setConfigPath, 527
- device\_setCurrentDevice, 527
- device\_setNEO2DevicesConfigs, 527
- device\_setSystemLanguage, 529
- device\_setTransactionExponent, 529
- device\_updateFirmware, 529
- emv\_registerCallBk, 530
- ftpComm\_callBack, 518
- httpComm\_callBack, 518
- IN, 517
- IN\_OUT, 517
- lcd\_registerCallBk, 530
- msr\_disable, 530
- msr\_getClearPANID, 530
- msr\_getExpirationMask, 530
- msr\_getFunctionStatus, 531
- msr\_getSwipeForcedEncryptionOption, 531
- msr\_getSwipeMaskOption, 531
- msr\_registerCallBk, 531
- msr\_registerCallBkp, 531
- msr\_setClearPANID, 532
- msr\_setExpirationMask, 532
- msr\_setSwipeForcedEncryptionOption, 532
- msr\_setSwipeMaskOption, 532
- OUT, 518
- pCMR\_callBack, 518
- pCSFS\_callBack, 518
- pEMV\_callBack, 518
- pFW\_callBack, 518
- pLCD\_callBack, 518
- pMSR\_callBack, 519
- pMSR\_callBackp, 519
- pMessageHotplug, 518
- pPIN\_callBack, 519
- pReadDataLog, 519
- pSendDataLog, 519
- pin\_registerCallBk, 533
- registerHotplugCallBk, 533
- registerLogCallBk, 533
- rs232\_device\_init, 533
- SDK\_Version, 533
- setAbsoluteLibraryPath, 533
- v4Comm\_callBack, 519
- libIDT\_SpectrumPro.h
  - config\_getModelNumber, 479
  - config\_getModelNumber\_Len, 479
  - config\_getSerialNumber, 479
  - config\_getSerialNumber\_Len, 480
  - device\_SendDataCommand, 496
  - device\_close, 480
  - device\_getCurrentDeviceType, 480
  - device\_getFirmwareVersion, 480
  - device\_getFirmwareVersion\_Len, 480
  - device\_getResponseCodeString, 481
  - device\_getSDKWaitTime, 490
  - device\_getSpectrumProKSN, 491
  - device\_getSpectrumProKSN\_Len, 491
  - device\_getThreadStackSize, 492
  - device\_init, 492
  - device\_isAttached, 492
  - device\_isConnected, 492
  - device\_pollCardReader, 492
  - device\_pollCardReader\_Len, 494
  - device\_rebootDevice, 496
  - device\_registerCameraCallBk, 496
  - device\_registerCardStatusFrontSwitchCallBk, 496
  - device\_setCurrentDevice, 496
  - device\_setSDKWaitTime, 497
  - device\_setThreadStackSize, 497
  - device\_updateFirmware, 497
  - emv\_activateTransaction, 498
  - emv\_allowFallback, 498
  - emv\_authenticateTransaction, 499
  - emv\_authenticateTransactionWithTimeout, 499
  - emv\_callbackResponseLCD, 500
  - emv\_callbackResponseMSR, 500
  - emv\_cancelTransaction, 500
  - emv\_completeTransaction, 500
  - emv\_getAutoAuthenticateTransaction, 502
  - emv\_getAutoCompleteTransaction, 502
  - emv\_getEMVConfigurationCheckValue, 502
  - emv\_getEMVKernelCheckValue, 502
  - emv\_getEMVKernelVersion, 503
  - emv\_getEMVKernelVersion\_Len, 503
  - emv\_registerCallBk, 503
  - emv\_removeAllApplicationData, 503
  - emv\_removeAllCAPK, 503
  - emv\_removeAllCRL, 503
  - emv\_removeApplicationData, 504
  - emv\_removeCAPK, 504
  - emv\_removeCRL, 504
  - emv\_removeTerminalData, 504
  - emv\_retrieveAIDList, 505
  - emv\_retrieveApplicationData, 505
  - emv\_retrieveCAPK, 506
  - emv\_retrieveCAPKList, 506
  - emv\_retrieveCRL, 506
  - emv\_retrieveTerminalData, 508
  - emv\_retrieveTerminalID, 508



- emv\_retrieveTerminalID\_Len, 508
- emv\_retrieveTransactionResult, 508
- emv\_setApplicationData, 509
- emv\_setAutoAuthenticateTransaction, 509
- emv\_setAutoCompleteTransaction, 509
- emv\_setCAPK, 509
- emv\_setCRL, 510
- emv\_setTerminalData, 510
- emv\_setTerminalID, 511
- emv\_startTransaction, 511
- ftpComm\_callBack, 478
- httpComm\_callBack, 478
- IN, 477
- IN\_OUT, 477
- icc\_getICCRReaderStatus, 511
- icc\_powerOffICC, 512
- icc\_powerOnICC, 512
- msr\_cancelMSRSwipe, 512
- msr\_clearMSRData, 512
- msr\_getMSRData, 512
- msr\_registerCallBk, 513
- msr\_registerCallBkp, 513
- msr\_startMSRSwipe, 513
- OUT, 477
- pCMR\_callBack, 478
- pCSFS\_callBack, 478
- pEMV\_callBack, 478
- pMSR\_callBack, 478
- pMSR\_callBackp, 478
- pMessageHotplug, 478
- pPIN\_callBack, 478
- pReadDataLog, 479
- pSendDataLog, 479
- parseMSRData, 513
- parsePINBlockData, 513
- parsePINData, 513
- pin\_cancelPINEntry, 514
- pin\_getPIN, 514
- pin\_registerCallBk, 514
- registerHotplugCallBk, 514
- registerLogCallBk, 515
- rs232\_device\_init, 515
- SDK\_Version, 515
- setAbsoluteLibraryPath, 515
- v4Comm\_callBack, 479
- libIDT\_UniPayI\_V.h
  - comm\_registerHTTPCallback, 538
  - comm\_registerV4Callback, 538
  - config\_getSerialNumber, 540
  - config\_getSerialNumber\_Len, 540
  - device\_SendDataCommandNEO, 544
  - device\_close, 540
  - device\_enablePassThrough, 540
  - device\_getCurrentDeviceType, 540
  - device\_getFirmwareVersion, 541
  - device\_getFirmwareVersion\_Len, 541
  - device\_getIDGStatusCodeString, 541
  - device\_getMerchantRecord, 542
  - device\_getMerchantRecord\_Len, 543
  - device\_getSDKWaitTime, 543
  - device\_getThreadStackSize, 543
  - device\_init, 543
  - device\_isAttached, 544
  - device\_isConnected, 544
  - device\_pingDevice, 544
  - device\_registerCameraCallBk, 544
  - device\_registerCardStatusFrontSwitchCallBk, 544
  - device\_setCurrentDevice, 545
  - device\_setMerchantRecord, 546
  - device\_setSDKWaitTime, 546
  - device\_setThreadStackSize, 547
  - emv\_activateTransaction, 547
  - emv\_allowFallback, 547
  - emv\_authenticateTransaction, 547
  - emv\_authenticateTransactionWithTimeout, 548
  - emv\_cancelTransaction, 548
  - emv\_completeTransaction, 548
  - emv\_getAutoAuthenticateTransaction, 549
  - emv\_getAutoCompleteTransaction, 549
  - emv\_registerCallBk, 549
  - emv\_removeAllApplicationData, 549
  - emv\_removeAllCAPK, 549
  - emv\_removeAllCRL, 550
  - emv\_removeApplicationData, 550
  - emv\_removeCAPK, 550
  - emv\_removeCRL, 550
  - emv\_retrieveAIDList, 551
  - emv\_retrieveApplicationData, 551
  - emv\_retrieveCAPK, 552
  - emv\_retrieveCAPKList, 552
  - emv\_retrieveCRL, 552
  - emv\_retrieveTerminalData, 554
  - emv\_setApplicationData, 554
  - emv\_setApplicationDataTLV, 554
  - emv\_setAutoAuthenticateTransaction, 555
  - emv\_setAutoCompleteTransaction, 555
  - emv\_setCAPK, 555
  - emv\_setCRL, 555
  - emv\_setTerminalData, 556
  - emv\_setTerminalMajorConfiguration, 556
  - emv\_startTransaction, 556
  - ftpComm\_callBack, 537
  - httpComm\_callBack, 537
  - IN, 537
  - IN\_OUT, 537
  - icc\_exchangeAPDU, 557
  - icc\_getICCRReaderStatus, 557
  - icc\_powerOffICC, 558
  - icc\_powerOnICC, 558
  - msr\_cancelMSRSwipe, 558
  - msr\_registerCallBk, 558
  - msr\_registerCallBkp, 558
  - msr\_startMSRSwipe, 558
  - OUT, 537
  - pCMR\_callBack, 537
  - pCSFS\_callBack, 537

- pEMV\_callBack, 537
- pMSR\_callBack, 538
- pMSR\_callBackp, 538
- pMessageHotplug, 537
- pPIN\_callBack, 538
- pReadDataLog, 538
- pSendDataLog, 538
- parseMSRData, 559
- pin\_registerCallBk, 559
- registerHotplugCallBk, 559
- registerLogCallBk, 559
- SDK\_Version, 559
- setAbsoluteLibraryPath, 559
- v4Comm\_callBack, 538
- libIDT\_VP3300\_AJ.h
  - comm\_registerHTTPCallback, 585
  - comm\_registerV4Callback, 585
  - config\_getSerialNumber, 587
  - config\_getSerialNumber\_Len, 587
  - ctls\_activateTransaction, 587
  - ctls\_cancelTransaction, 588
  - ctls\_getAllConfigurationGroups, 588
  - ctls\_getConfigurationGroup, 589
  - ctls\_registerCallBk, 589
  - ctls\_registerCallBkp, 589
  - ctls\_removeAllApplicationData, 589
  - ctls\_removeAllCAPK, 589
  - ctls\_removeApplicationData, 589
  - ctls\_removeCAPK, 590
  - ctls\_removeConfigurationGroup, 590
  - ctls\_retrieveAIDList, 590
  - ctls\_retrieveApplicationData, 590
  - ctls\_retrieveCAPK, 591
  - ctls\_retrieveCAPKList, 591
  - ctls\_retrieveTerminalData, 592
  - ctls\_setApplicationData, 592
  - ctls\_setCAPK, 592
  - ctls\_setConfigurationGroup, 593
  - ctls\_setTerminalData, 593
  - ctls\_startTransaction, 594
  - device\_SendDataCommandNEO, 602
  - device\_activateTransaction, 595
  - device\_cancelTransaction, 596
  - device\_close, 596
  - device\_controlUserInterface, 596
  - device\_enablePassThrough, 597
  - device\_getCurrentDeviceType, 597
  - device\_getFirmwareVersion, 598
  - device\_getFirmwareVersion\_Len, 598
  - device\_getIDGStatusCodeString, 598
  - device\_getMerchantRecord, 599
  - device\_getMerchantRecord\_Len, 600
  - device\_getRTCTime, 600
  - device\_getSDKWaitTime, 600
  - device\_getThreadStackSize, 601
  - device\_getTransactionResults, 601
  - device\_init, 601
  - device\_isAttached, 601
  - device\_isConnected, 602
  - device\_pingDevice, 602
  - device\_registerCameraCallBk, 602
  - device\_registerCardStatusFrontSwitchCallBk, 602
  - device\_registerRKICallBk, 602
  - device\_setBurstMode, 604
  - device\_setCurrentDevice, 604
  - device\_setMerchantRecord, 605
  - device\_setPollMode, 605
  - device\_setRTCTime, 606
  - device\_setSDKWaitTime, 606
  - device\_setThreadStackSize, 606
  - device\_setTransactionExponent, 606
  - device\_startRKI, 606
  - device\_startTransaction, 607
  - emv\_activateTransaction, 608
  - emv\_allowFallback, 608
  - emv\_authenticateTransaction, 609
  - emv\_authenticateTransactionWithTimeout, 609
  - emv\_cancelTransaction, 609
  - emv\_completeTransaction, 610
  - emv\_getAutoAuthenticateTransaction, 610
  - emv\_getAutoCompleteTransaction, 610
  - emv\_registerCallBk, 610
  - emv\_removeAllApplicationData, 610
  - emv\_removeAllCAPK, 611
  - emv\_removeAllCRL, 611
  - emv\_removeApplicationData, 611
  - emv\_removeCAPK, 611
  - emv\_removeCRL, 611
  - emv\_retrieveAIDList, 612
  - emv\_retrieveApplicationData, 612
  - emv\_retrieveCAPK, 612
  - emv\_retrieveCAPKList, 613
  - emv\_retrieveCRL, 613
  - emv\_retrieveTerminalData, 614
  - emv\_setApplicationData, 614
  - emv\_setApplicationDataTLV, 614
  - emv\_setAutoAuthenticateTransaction, 614
  - emv\_setAutoCompleteTransaction, 615
  - emv\_setCAPK, 615
  - emv\_setCRL, 615
  - emv\_setTerminalData, 616
  - emv\_setTerminalMajorConfiguration, 616
  - emv\_setTransactionParameters, 616
  - emv\_startTransaction, 617
  - ftpComm\_callBack, 584
  - httpComm\_callBack, 584
  - IN, 584
  - IN\_OUT, 584
  - icc\_exchangeAPDU, 617
  - icc\_getICReaderStatus, 618
  - icc\_powerOffICC, 618
  - icc\_powerOnICC, 618
  - msr\_cancelMSRSwipe, 618
  - msr\_registerCallBk, 618
  - msr\_registerCallBkp, 618
  - msr\_startMSRSwipe, 619



- OUT, [584](#)
- pCMR\_callback, [584](#)
- pCSFS\_callback, [584](#)
- pEMV\_callback, [584](#)
- pMSR\_callback, [585](#)
- pMSR\_callbackp, [585](#)
- pMessageHotplug, [584](#)
- pPIN\_callback, [585](#)
- pReadDataLog, [585](#)
- pSendDataLog, [585](#)
- parseMSRData, [619](#)
- pin\_registerCallback, [619](#)
- registerHotplugCallback, [619](#)
- registerLogCallback, [619](#)
- SDK\_Version, [619](#)
- setAbsoluteLibraryPath, [619](#)
- v4Comm\_callback, [585](#)
- libIDT\_VP3300\_BT.h
  - comm\_registerHTTPCallback, [624](#)
  - comm\_registerV4Callback, [624](#)
  - config\_getSerialNumber, [624](#)
  - config\_getSerialNumber\_Len, [625](#)
  - ctls\_activateTransaction, [625](#)
  - ctls\_cancelTransaction, [626](#)
  - ctls\_getAllConfigurationGroups, [626](#)
  - ctls\_getConfigurationGroup, [626](#)
  - ctls\_registerCallback, [627](#)
  - ctls\_registerCallbackp, [627](#)
  - ctls\_removeAllApplicationData, [627](#)
  - ctls\_removeAllCAPK, [627](#)
  - ctls\_removeApplicationData, [627](#)
  - ctls\_removeCAPK, [627](#)
  - ctls\_removeConfigurationGroup, [628](#)
  - ctls\_retrieveAIDList, [628](#)
  - ctls\_retrieveApplicationData, [628](#)
  - ctls\_retrieveCAPK, [628](#)
  - ctls\_retrieveCAPKList, [629](#)
  - ctls\_retrieveTerminalData, [629](#)
  - ctls\_setApplicationData, [630](#)
  - ctls\_setCAPK, [630](#)
  - ctls\_setConfigurationGroup, [631](#)
  - ctls\_setTerminalData, [631](#)
  - ctls\_startTransaction, [631](#)
  - device\_SendDataCommandNEO, [639](#)
  - device\_activateTransaction, [632](#)
  - device\_cancelTransaction, [633](#)
  - device\_close, [634](#)
  - device\_controlUserInterface, [634](#)
  - device\_enablePassThrough, [634](#)
  - device\_getCurrentDeviceType, [635](#)
  - device\_getFirmwareVersion, [635](#)
  - device\_getFirmwareVersion\_Len, [635](#)
  - device\_getIDGStatusCodeString, [635](#)
  - device\_getMerchantRecord, [637](#)
  - device\_getMerchantRecord\_Len, [637](#)
  - device\_getRTCTime, [637](#)
  - device\_getSDKWaitTime, [638](#)
  - device\_getThreadStackSize, [638](#)
  - device\_getTransactionResults, [638](#)
  - device\_init, [638](#)
  - device\_isAttached, [639](#)
  - device\_isConnected, [639](#)
  - device\_pingDevice, [639](#)
  - device\_registerCameraCallBk, [639](#)
  - device\_registerCardStatusFrontSwitchCallBk, [639](#)
  - device\_registerRKICallBk, [639](#)
  - device\_setBurstMode, [640](#)
  - device\_setCurrentDevice, [640](#)
  - device\_setMerchantRecord, [641](#)
  - device\_setPollMode, [641](#)
  - device\_setRTCTime, [642](#)
  - device\_setSDKWaitTime, [642](#)
  - device\_setThreadStackSize, [642](#)
  - device\_setTransactionExponent, [642](#)
  - device\_startRKI, [642](#)
  - device\_startTransaction, [643](#)
  - emv\_activateTransaction, [644](#)
  - emv\_allowFallback, [644](#)
  - emv\_authenticateTransaction, [645](#)
  - emv\_authenticateTransactionWithTimeout, [645](#)
  - emv\_cancelTransaction, [645](#)
  - emv\_completeTransaction, [646](#)
  - emv\_getAutoAuthenticateTransaction, [646](#)
  - emv\_getAutoCompleteTransaction, [646](#)
  - emv\_registerCallBk, [646](#)
  - emv\_removeAllApplicationData, [646](#)
  - emv\_removeAllCAPK, [647](#)
  - emv\_removeAllCRL, [647](#)
  - emv\_removeApplicationData, [647](#)
  - emv\_removeCAPK, [647](#)
  - emv\_removeCRL, [647](#)
  - emv\_retrieveAIDList, [648](#)
  - emv\_retrieveApplicationData, [648](#)
  - emv\_retrieveCAPK, [648](#)
  - emv\_retrieveCAPKList, [649](#)
  - emv\_retrieveCRL, [649](#)
  - emv\_retrieveTerminalData, [650](#)
  - emv\_setApplicationData, [650](#)
  - emv\_setApplicationDataTLV, [650](#)
  - emv\_setAutoAuthenticateTransaction, [650](#)
  - emv\_setAutoCompleteTransaction, [651](#)
  - emv\_setCAPK, [651](#)
  - emv\_setCRL, [651](#)
  - emv\_setTerminalData, [652](#)
  - emv\_setTerminalMajorConfiguration, [652](#)
  - emv\_setTransactionParameters, [652](#)
  - emv\_startTransaction, [653](#)
  - ftpComm\_callback, [623](#)
  - httpComm\_callback, [623](#)
  - IN, [622](#)
  - IN\_OUT, [622](#)
  - icc\_exchangeAPDU, [653](#)
  - icc\_getICCRReaderStatus, [654](#)
  - icc\_powerOffICC, [654](#)
  - icc\_powerOnICC, [654](#)
  - msr\_cancelMSRSwipe, [654](#)

- msr\_registerCallBk, [654](#)
- msr\_registerCallBkp, [654](#)
- msr\_startMSRSwipe, [655](#)
- OUT, [622](#)
- pCMR\_callBack, [623](#)
- pCSFS\_callBack, [623](#)
- pEMV\_callBack, [623](#)
- pMSR\_callBack, [623](#)
- pMSR\_callBkp, [623](#)
- pMessageHotplug, [623](#)
- pPIN\_callBack, [624](#)
- pReadDataLog, [624](#)
- pSendDataLog, [624](#)
- parseMSRData, [655](#)
- pin\_registerCallBk, [655](#)
- registerHotplugCallBk, [655](#)
- registerLogCallBk, [655](#)
- SDK\_Version, [655](#)
- setAbsoluteLibraryPath, [655](#)
- v4Comm\_callBack, [624](#)
- libIDT\_VP3300\_COM.h
  - comm\_registerHTTPCallback, [660](#)
  - comm\_registerV4Callback, [660](#)
  - config\_getSerialNumber, [660](#)
  - config\_getSerialNumber\_Len, [661](#)
  - ctls\_activateTransaction, [661](#)
  - ctls\_cancelTransaction, [662](#)
  - ctls\_getAllConfigurationGroups, [662](#)
  - ctls\_getConfigurationGroup, [662](#)
  - ctls\_registerCallBk, [663](#)
  - ctls\_registerCallBkp, [663](#)
  - ctls\_removeAllApplicationData, [663](#)
  - ctls\_removeAllCAPK, [663](#)
  - ctls\_removeApplicationData, [663](#)
  - ctls\_removeCAPK, [663](#)
  - ctls\_removeConfigurationGroup, [664](#)
  - ctls\_retrieveAIDList, [664](#)
  - ctls\_retrieveApplicationData, [664](#)
  - ctls\_retrieveCAPK, [664](#)
  - ctls\_retrieveCAPKList, [665](#)
  - ctls\_retrieveTerminalData, [665](#)
  - ctls\_setApplicationData, [666](#)
  - ctls\_setCAPK, [666](#)
  - ctls\_setConfigurationGroup, [667](#)
  - ctls\_setTerminalData, [667](#)
  - ctls\_startTransaction, [667](#)
  - device\_activateTransaction, [668](#)
  - device\_cancelTransaction, [669](#)
  - device\_close, [670](#)
  - device\_controlUserInterface, [670](#)
  - device\_enablePassThrough, [670](#)
  - device\_getCurrentDeviceType, [671](#)
  - device\_getFirmwareVersion, [671](#)
  - device\_getFirmwareVersion\_Len, [671](#)
  - device\_getMerchantRecord, [673](#)
  - device\_getMerchantRecord\_Len, [673](#)
  - device\_getRTCTime, [673](#)
  - device\_getSDKWaitTime, [674](#)
  - device\_getThreadStackSize, [674](#)
  - device\_getTransactionResults, [674](#)
  - device\_init, [674](#)
  - device\_isAttached, [675](#)
  - device\_isConnected, [675](#)
  - device\_pingDevice, [675](#)
  - device\_registerCameraCallBk, [675](#)
  - device\_registerRKICallBk, [675](#)
  - device\_setBurstMode, [676](#)
  - device\_setCurrentDevice, [676](#)
  - device\_setMerchantRecord, [677](#)
  - device\_setPollMode, [677](#)
  - device\_setRTCTime, [678](#)
  - device\_setSDKWaitTime, [678](#)
  - device\_setThreadStackSize, [678](#)
  - device\_setTransactionExponent, [678](#)
  - device\_startRKI, [678](#)
  - device\_startTransaction, [679](#)
  - emv\_activateTransaction, [679](#)
  - emv\_allowFallback, [680](#)
  - emv\_authenticateTransaction, [680](#)
  - emv\_authenticateTransactionWithTimeout, [680](#)
  - emv\_cancelTransaction, [681](#)
  - emv\_completeTransaction, [681](#)
  - emv\_getAutoAuthenticateTransaction, [681](#)
  - emv\_getAutoCompleteTransaction, [682](#)
  - emv\_registerCallBk, [682](#)
  - emv\_removeAllApplicationData, [682](#)
  - emv\_removeAllCAPK, [682](#)
  - emv\_removeAllCRL, [682](#)
  - emv\_removeApplicationData, [682](#)
  - emv\_removeCAPK, [683](#)
  - emv\_removeCRL, [683](#)
  - emv\_retrieveAIDList, [683](#)
  - emv\_retrieveApplicationData, [683](#)
  - emv\_retrieveCAPK, [684](#)
  - emv\_retrieveCAPKList, [684](#)
  - emv\_retrieveCRL, [685](#)
  - emv\_retrieveTerminalData, [685](#)
  - emv\_setApplicationData, [685](#)
  - emv\_setApplicationDataTLV, [685](#)
  - emv\_setAutoAuthenticateTransaction, [686](#)
  - emv\_setAutoCompleteTransaction, [686](#)
  - emv\_setCAPK, [686](#)
  - emv\_setCRL, [687](#)
  - emv\_setTerminalData, [687](#)
  - emv\_setTerminalMajorConfiguration, [688](#)
  - emv\_setTransactionParameters, [688](#)
  - emv\_startTransaction, [688](#)
  - ftpComm\_callBack, [659](#)
  - httpComm\_callBack, [659](#)
  - IN, [658](#)
  - IN\_OUT, [658](#)
  - icc\_exchangeAPDU, [689](#)
  - icc\_getICCRReaderStatus, [689](#)
  - icc\_powerOffICC, [689](#)
  - icc\_powerOnICC, [690](#)
  - msr\_cancelMSRSwipe, [690](#)

- msr\_registerCallBk, 690
- msr\_registerCallBkp, 690
- msr\_startMSRSwipe, 690
- OUT, 658
- pCMR\_callBack, 659
- pCSFS\_callBack, 659
- pEMV\_callBack, 659
- pMSR\_callBack, 659
- pMSR\_callBackp, 659
- pMessageHotplug, 659
- pPIN\_callBack, 660
- pReadDataLog, 660
- pSendDataLog, 660
- parseMSRData, 690
- pin\_registerCallBk, 692
- registerHotplugCallBk, 692
- registerLogCallBk, 692
- SDK\_Version, 692
- setAbsoluteLibraryPath, 692
- v4Comm\_callBack, 660
- libIDT\_VP3300\_USB.h
  - comm\_registerHTTPCallback, 697
  - comm\_registerV4Callback, 697
  - config\_getSerialNumber, 697
  - config\_getSerialNumber\_Len, 697
  - ctls\_activateTransaction, 697
  - ctls\_cancelTransaction, 698
  - ctls\_getAllConfigurationGroups, 699
  - ctls\_getConfigurationGroup, 699
  - ctls\_registerCallBk, 699
  - ctls\_registerCallBkp, 699
  - ctls\_removeAllApplicationData, 699
  - ctls\_removeAllCAPK, 699
  - ctls\_removeApplicationData, 700
  - ctls\_removeCAPK, 700
  - ctls\_removeConfigurationGroup, 700
  - ctls\_retrieveAIDList, 700
  - ctls\_retrieveApplicationData, 701
  - ctls\_retrieveCAPK, 701
  - ctls\_retrieveCAPKList, 702
  - ctls\_retrieveTerminalData, 702
  - ctls\_setApplicationData, 702
  - ctls\_setCAPK, 702
  - ctls\_setConfigurationGroup, 703
  - ctls\_setTerminalData, 703
  - ctls\_startTransaction, 704
  - device\_activateTransaction, 705
  - device\_cancelTransaction, 706
  - device\_close, 706
  - device\_controlUserInterface, 706
  - device\_enablePassThrough, 707
  - device\_getCurrentDeviceType, 707
  - device\_getFirmwareVersion, 708
  - device\_getFirmwareVersion\_Len, 708
  - device\_getMerchantRecord, 709
  - device\_getMerchantRecord\_Len, 710
  - device\_getRTCTime, 710
  - device\_getSDKWaitTime, 710
  - device\_getThreadStackSize, 711
  - device\_getTransactionResults, 711
  - device\_init, 711
  - device\_isAttached, 711
  - device\_isConnected, 712
  - device\_pingDevice, 712
  - device\_registerCameraCallBk, 712
  - device\_registerRKICallBk, 712
  - device\_setBurstMode, 713
  - device\_setCurrentDevice, 714
  - device\_setMerchantRecord, 714
  - device\_setPollMode, 714
  - device\_setRTCTime, 715
  - device\_setSDKWaitTime, 715
  - device\_setThreadStackSize, 715
  - device\_setTransactionExponent, 715
  - device\_startRKI, 716
  - device\_startTransaction, 716
  - emv\_activateTransaction, 716
  - emv\_allowFallback, 717
  - emv\_authenticateTransaction, 717
  - emv\_authenticateTransactionWithTimeout, 718
  - emv\_cancelTransaction, 718
  - emv\_completeTransaction, 718
  - emv\_getAutoAuthenticateTransaction, 719
  - emv\_getAutoCompleteTransaction, 719
  - emv\_registerCallBk, 719
  - emv\_removeAllApplicationData, 719
  - emv\_removeAllCAPK, 719
  - emv\_removeAllCRL, 719
  - emv\_removeApplicationData, 720
  - emv\_removeCAPK, 720
  - emv\_removeCRL, 720
  - emv\_retrieveAIDList, 720
  - emv\_retrieveApplicationData, 721
  - emv\_retrieveCAPK, 721
  - emv\_retrieveCAPKList, 722
  - emv\_retrieveCRL, 722
  - emv\_retrieveTerminalData, 722
  - emv\_setApplicationData, 722
  - emv\_setApplicationDataTLV, 723
  - emv\_setAutoAuthenticateTransaction, 723
  - emv\_setAutoCompleteTransaction, 723
  - emv\_setCAPK, 723
  - emv\_setCRL, 724
  - emv\_setTerminalData, 724
  - emv\_setTerminalMajorConfiguration, 725
  - emv\_setTransactionParameters, 725
  - emv\_startTransaction, 726
  - ftpComm\_callBack, 695
  - httpComm\_callBack, 695
  - IN, 695
  - IN\_OUT, 695
  - icc\_exchangeAPDU, 726
  - icc\_getICCRReaderStatus, 726
  - icc\_powerOffICC, 727
  - icc\_powerOnICC, 727
  - msr\_cancelMSRSwipe, 727

- msr\_registerCallBk, [727](#)
- msr\_registerCallBkp, [727](#)
- msr\_startMSRSwipe, [727](#)
- OUT, [695](#)
- pCMR\_callBack, [695](#)
- pCSFS\_callBack, [696](#)
- pEMV\_callBack, [696](#)
- pMSR\_callBack, [696](#)
- pMSR\_callBkp, [696](#)
- pMessageHotplug, [696](#)
- pPIN\_callBack, [696](#)
- pReadDataLog, [696](#)
- pSendDataLog, [696](#)
- parseMSRData, [728](#)
- pin\_registerCallBk, [728](#)
- registerHotplugCallBk, [728](#)
- registerLogCallBk, [728](#)
- SDK\_Version, [728](#)
- setAbsoluteLibraryPath, [728](#)
- v4Comm\_callBack, [696](#)
- libIDT\_VP8800.h
  - comm\_registerHTTPCallback, [734](#)
  - comm\_registerV4Callback, [734](#)
  - config\_getSerialNumber, [736](#)
  - config\_getSerialNumber\_Len, [736](#)
  - ctls\_activateTransaction, [736](#)
  - ctls\_cancelTransaction, [737](#)
  - ctls\_displayOnlineAuthResult, [737](#)
  - ctls\_getAllConfigurationGroups, [738](#)
  - ctls\_getConfigurationGroup, [738](#)
  - ctls\_registerCallBk, [738](#)
  - ctls\_registerCallBkp, [738](#)
  - ctls\_removeAllApplicationData, [738](#)
  - ctls\_removeAllCAPK, [738](#)
  - ctls\_removeApplicationData, [739](#)
  - ctls\_removeCAPK, [739](#)
  - ctls\_removeConfigurationGroup, [739](#)
  - ctls\_retrieveAIDList, [739](#)
  - ctls\_retrieveApplicationData, [740](#)
  - ctls\_retrieveCAPK, [740](#)
  - ctls\_retrieveCAPKList, [741](#)
  - ctls\_retrieveTerminalData, [741](#)
  - ctls\_setApplicationData, [741](#)
  - ctls\_setCAPK, [741](#)
  - ctls\_setConfigurationGroup, [742](#)
  - ctls\_setTerminalData, [742](#)
  - ctls\_startTransaction, [743](#)
  - device\_SendDataCommandNEO, [755](#)
  - device\_activateTransaction, [744](#)
  - device\_calibrateParameters, [745](#)
  - device\_cancelTransaction, [745](#)
  - device\_close, [745](#)
  - device\_controlIndicator, [745](#)
  - device\_controlUserInterface, [746](#)
  - device\_createDirectory, [747](#)
  - device\_deleteDirectory, [747](#)
  - device\_deleteFile, [747](#)
  - device\_enablePassThrough, [747](#)
  - device\_enhancedPassthrough, [749](#)
  - device\_getCurrentDeviceType, [749](#)
  - device\_getDriveFreeSpace, [749](#)
  - device\_getFirmwareVersion, [749](#)
  - device\_getFirmwareVersion\_Len, [750](#)
  - device\_getIDGStatusCodeString, [750](#)
  - device\_getMerchantRecord, [751](#)
  - device\_getMerchantRecord\_Len, [751](#)
  - device\_getSDKWaitTime, [753](#)
  - device\_getThreadStackSize, [753](#)
  - device\_getTransactionResults, [753](#)
  - device\_init, [753](#)
  - device\_isAttached, [754](#)
  - device\_isConnected, [754](#)
  - device\_listDirectory, [754](#)
  - device\_pingDevice, [754](#)
  - device\_registerCameraCallBk, [754](#)
  - device\_registerCardStatusFrontSwitchCallBk, [755](#)
  - device\_setCurrentDevice, [756](#)
  - device\_setMerchantRecord, [756](#)
  - device\_setSDKWaitTime, [756](#)
  - device\_setThreadStackSize, [757](#)
  - device\_setTransactionExponent, [757](#)
  - device\_startTransaction, [757](#)
  - device\_transferFile, [758](#)
  - emv\_activateTransaction, [758](#)
  - emv\_allowFallback, [759](#)
  - emv\_authenticateTransaction, [759](#)
  - emv\_authenticateTransactionWithTimeout, [760](#)
  - emv\_cancelTransaction, [760](#)
  - emv\_completeTransaction, [760](#)
  - emv\_getAutoAuthenticateTransaction, [761](#)
  - emv\_getAutoCompleteTransaction, [761](#)
  - emv\_getEMVConfigurationCheckValue, [761](#)
  - emv\_getEMVKernelCheckValue, [761](#)
  - emv\_getEMVKernelVersion, [762](#)
  - emv\_getEMVKernelVersion\_Len, [762](#)
  - emv\_registerCallBk, [762](#)
  - emv\_removeAllApplicationData, [762](#)
  - emv\_removeAllCAPK, [762](#)
  - emv\_removeAllCRL, [762](#)
  - emv\_removeAllExceptions, [763](#)
  - emv\_removeApplicationData, [763](#)
  - emv\_removeCAPK, [763](#)
  - emv\_removeCRL, [763](#)
  - emv\_removeException, [764](#)
  - emv\_removeTransactionLog, [764](#)
  - emv\_retrieveAIDList, [764](#)
  - emv\_retrieveApplicationData, [764](#)
  - emv\_retrieveCAPK, [765](#)
  - emv\_retrieveCAPKList, [766](#)
  - emv\_retrieveCRL, [766](#)
  - emv\_retrieveExceptionList, [766](#)
  - emv\_retrieveExceptionLogStatus, [766](#)
  - emv\_retrieveTerminalData, [767](#)
  - emv\_retrieveTransactionLog, [767](#)
  - emv\_retrieveTransactionLogStatus, [768](#)
  - emv\_setApplicationData, [769](#)

- emv\_setApplicationDataTLV, 769
- emv\_setAutoAuthenticateTransaction, 769
- emv\_setAutoCompleteTransaction, 770
- emv\_setCAPK, 770
- emv\_setCRL, 771
- emv\_setException, 771
- emv\_setTerminalData, 771
- emv\_startTransaction, 772
- ftpComm\_callBack, 733
- httpComm\_callBack, 733
- IN, 733
- IN\_OUT, 733
- lcd\_addItemToList, 772
- lcd\_cancelSlideShow, 772
- lcd\_captureSignature, 773
- lcd\_clearDisplay, 773
- lcd\_clearEventQueue, 773
- lcd\_createInputField, 773
- lcd\_createInputField\_Len, 775
- lcd\_createList, 776
- lcd\_createList\_Len, 777
- lcd\_customDisplayMode, 778
- lcd\_displayButton, 778
- lcd\_displayButton\_Len, 780
- lcd\_displayParagraph, 781
- lcd\_displayText, 782
- lcd\_displayText\_Len, 783
- lcd\_getInputEvent, 784
- lcd\_getInputEvent\_Len, 786
- lcd\_getInputFieldValue, 787
- lcd\_getSelectedItem, 788
- lcd\_getSelectedItem\_Len, 788
- lcd\_resetInitialState, 788
- lcd\_setBackgroundImage, 788
- lcd\_setDisplayImage, 788
- lcd\_setForeBackColor, 790
- lcd\_startSlideShow, 790
- msr\_cancelMSRSwipe, 791
- msr\_flushTrackData, 791
- msr\_registerCallBk, 791
- msr\_registerCallBkp, 791
- msr\_startMSRSwipe, 791
- OUT, 733
- pCMR\_callBack, 733
- pCSFS\_callBack, 733
- pEMV\_callBack, 733
- pLog\_callback, 733
- pMSR\_callBack, 734
- pMSR\_callBackp, 734
- pMessageHotplug, 733
- pPIN\_callBack, 734
- pReadDataLog, 734
- pSendDataLog, 734
- parseMSRData, 792
- pin\_getEncryptedOnlinePIN, 792
- pin\_getPAN, 792
- pin\_promptCreditDebit, 792
- pin\_registerCallBk, 793
- registerHotplugCallBk, 793
- registerLogCallBk, 793
- SDK\_Version, 793
- setAbsoluteLibraryPath, 793
- v4Comm\_callBack, 734
- ws\_deleteSSLCert, 793
- ws\_getCertChainType, 794
- ws\_loadSSLCert, 794
- ws\_requestCSR, 794
- ws\_revokeSSLCert, 794
- ws\_updateRootCertificate, 795
- libIDT\_Vendi.h
  - comm\_registerHTTPCallback, 563
  - comm\_registerV4Callback, 563
  - config\_getSerialNumber, 563
  - config\_getSerialNumber\_Len, 564
  - ctls\_activateTransaction, 564
  - ctls\_cancelTransaction, 565
  - ctls\_getAllConfigurationGroups, 565
  - ctls\_getConfigurationGroup, 565
  - ctls\_registerCallBk, 566
  - ctls\_registerCallBkp, 566
  - ctls\_removeAllApplicationData, 566
  - ctls\_removeAllCAPK, 566
  - ctls\_removeApplicationData, 566
  - ctls\_removeCAPK, 566
  - ctls\_removeConfigurationGroup, 567
  - ctls\_retrieveAIDList, 567
  - ctls\_retrieveApplicationData, 567
  - ctls\_retrieveCAPK, 567
  - ctls\_retrieveCAPKList, 568
  - ctls\_retrieveTerminalData, 568
  - ctls\_setApplicationData, 569
  - ctls\_setCAPK, 569
  - ctls\_setConfigurationGroup, 570
  - ctls\_setTerminalData, 570
  - ctls\_startTransaction, 570
  - device\_SendDataCommandNEO, 577
  - device\_close, 571
  - device\_controlUserInterface, 571
  - device\_enablePassThrough, 572
  - device\_getCurrentDeviceType, 572
  - device\_getFirmwareVersion, 573
  - device\_getFirmwareVersion\_Len, 573
  - device\_getIDGStatusCodeString, 573
  - device\_getMerchantRecord, 574
  - device\_getMerchantRecord\_Len, 575
  - device\_getSDKWaitTime, 575
  - device\_getThreadStackSize, 575
  - device\_getTransactionResults, 575
  - device\_init, 576
  - device\_isAttached, 576
  - device\_isConnected, 576
  - device\_pingDevice, 576
  - device\_registerCameraCallBk, 576
  - device\_registerCardStatusFrontSwitchCallBk, 577
  - device\_setBurstMode, 577
  - device\_setCurrentDevice, 578

- device\_setMerchantRecord, 579
- device\_setPollMode, 579
- device\_setSDKWaitTime, 579
- device\_setThreadStackSize, 579
- emv\_registerCallBk, 579
- ftpComm\_callBack, 562
- httpComm\_callBack, 562
- IN, 561
- IN\_OUT, 561
- msr\_cancelMSRSwipe, 579
- msr\_registerCallBk, 580
- msr\_registerCallBkp, 580
- msr\_startMSRSwipe, 580
- OUT, 561
- pCMR\_callBack, 562
- pCSFS\_callBack, 562
- pEMV\_callBack, 562
- pMSR\_callBack, 562
- pMSR\_callBackp, 562
- pMessageHotplug, 562
- pPIN\_callBack, 563
- pReadDataLog, 563
- pSendDataLog, 563
- parseMSRData, 580
- pin\_registerCallBk, 580
- registerHotplugCallBk, 580
- registerLogCallBk, 580
- SDK\_Version, 580
- setAbsoluteLibraryPath, 581
- v4Comm\_callBack, 563
- loyalty\_cancelTransaction
  - liblDT\_Device.h, 236
  - liblDT\_NEO2.h, 443
- loyalty\_cancelTransactionSilent
  - liblDT\_Device.h, 236
  - liblDT\_NEO2.h, 443
- loyalty\_registerCallBk
  - liblDT\_Device.h, 236
  - liblDT\_NEO2.h, 443
- loyalty\_startTransaction
  - liblDT\_Device.h, 236
  - liblDT\_NEO2.h, 443
- msr\_cancelMSRSwipe
  - liblDT\_Augusta.h, 93
  - liblDT\_Device.h, 238
  - liblDT\_NEO2.h, 446
  - liblDT\_SpectrumPro.h, 512
  - liblDT\_UniPayI\_V.h, 558
  - liblDT\_Vendi.h, 579
  - liblDT\_VP3300\_AJ.h, 618
  - liblDT\_VP3300\_BT.h, 654
  - liblDT\_VP3300\_COM.h, 690
  - liblDT\_VP3300\_USB.h, 727
  - liblDT\_VP8800.h, 791
- msr\_captureMode
  - liblDT\_Augusta.h, 94
  - liblDT\_Device.h, 238
- msr\_clearMSRData
  - liblDT\_Device.h, 238
  - liblDT\_SpectrumPro.h, 512
- msr\_disable
  - liblDT\_Augusta.h, 94
  - liblDT\_Device.h, 238
  - liblDT\_SREDKey2.h, 530
- msr\_flushTrackData
  - liblDT\_Device.h, 239
  - liblDT\_VP8800.h, 791
- msr\_getClearPANID
  - liblDT\_Augusta.h, 94
  - liblDT\_Device.h, 239
  - liblDT\_SREDKey2.h, 530
- msr\_getExpirationMask
  - liblDT\_Augusta.h, 94
  - liblDT\_Device.h, 239
  - liblDT\_SREDKey2.h, 530
- msr\_getFunctionStatus
  - liblDT\_Device.h, 239
  - liblDT\_SREDKey2.h, 531
- msr\_getKeyFormatForICCDUKPT
  - liblDT\_Augusta.h, 95
  - liblDT\_Device.h, 240
- msr\_getKeyTypeForICCDUKPT
  - liblDT\_Augusta.h, 95
  - liblDT\_Device.h, 240
- msr\_getMSRData
  - liblDT\_Augusta.h, 95
  - liblDT\_Device.h, 240
  - liblDT\_SpectrumPro.h, 512
- msr\_getSetting
  - liblDT\_Augusta.h, 96
- msr\_getSwipeForcedEncryptionOption
  - liblDT\_Augusta.h, 96
  - liblDT\_Device.h, 241
  - liblDT\_SREDKey2.h, 531
- msr\_getSwipeMaskOption
  - liblDT\_Augusta.h, 96
  - liblDT\_Device.h, 241
  - liblDT\_SREDKey2.h, 531
- msr\_registerCallBk
  - liblDT\_Augusta.h, 96
  - liblDT\_Device.h, 241
  - liblDT\_L100.h, 304
  - liblDT\_MiniSmartII.h, 358
  - liblDT\_NEO2.h, 446
  - liblDT\_SpectrumPro.h, 513
  - liblDT\_SREDKey2.h, 531
  - liblDT\_UniPayI\_V.h, 558
  - liblDT\_Vendi.h, 580
  - liblDT\_VP3300\_AJ.h, 618
  - liblDT\_VP3300\_BT.h, 654
  - liblDT\_VP3300\_COM.h, 690
  - liblDT\_VP3300\_USB.h, 727
  - liblDT\_VP8800.h, 791
- msr\_registerCallBkp
  - liblDT\_Augusta.h, 96
  - liblDT\_Device.h, 241



- libIDT\_L100.h, [304](#)
- libIDT\_MiniSmartII.h, [359](#)
- libIDT\_NEO2.h, [446](#)
- libIDT\_SpectrumPro.h, [513](#)
- libIDT\_SREDKey2.h, [531](#)
- libIDT\_UniPayI\_V.h, [558](#)
- libIDT\_Vendi.h, [580](#)
- libIDT\_VP3300\_AJ.h, [618](#)
- libIDT\_VP3300\_BT.h, [654](#)
- libIDT\_VP3300\_COM.h, [690](#)
- libIDT\_VP3300\_USB.h, [727](#)
- libIDT\_VP8800.h, [791](#)
- msr\_setClearPANID
  - libIDT\_Augusta.h, [97](#)
  - libIDT\_Device.h, [241](#)
  - libIDT\_SREDKey2.h, [532](#)
- msr\_setExpirationMask
  - libIDT\_Augusta.h, [97](#)
  - libIDT\_Device.h, [241](#)
  - libIDT\_SREDKey2.h, [532](#)
- msr\_setKeyFormatForICCDUKPT
  - libIDT\_Augusta.h, [97](#)
  - libIDT\_Device.h, [242](#)
- msr\_setKeyTypeForICCDUKPT
  - libIDT\_Augusta.h, [97](#)
  - libIDT\_Device.h, [242](#)
- msr\_setSetting
  - libIDT\_Augusta.h, [98](#)
- msr\_setSwipeForcedEncryptionOption
  - libIDT\_Augusta.h, [98](#)
  - libIDT\_Device.h, [242](#)
  - libIDT\_SREDKey2.h, [532](#)
- msr\_setSwipeMaskOption
  - libIDT\_Augusta.h, [98](#)
  - libIDT\_Device.h, [243](#)
  - libIDT\_SREDKey2.h, [532](#)
- msr\_startMSRSwipe
  - libIDT\_Augusta.h, [98](#)
  - libIDT\_Device.h, [243](#)
  - libIDT\_NEO2.h, [446](#)
  - libIDT\_SpectrumPro.h, [513](#)
  - libIDT\_UniPayI\_V.h, [558](#)
  - libIDT\_Vendi.h, [580](#)
  - libIDT\_VP3300\_AJ.h, [619](#)
  - libIDT\_VP3300\_BT.h, [655](#)
  - libIDT\_VP3300\_COM.h, [690](#)
  - libIDT\_VP3300\_USB.h, [727](#)
  - libIDT\_VP8800.h, [791](#)
- OUT
  - libIDT\_Augusta.h, [51](#)
  - libIDT\_Device.h, [109](#)
  - libIDT\_KioskIII.h, [263](#)
  - libIDT\_L100.h, [285](#)
  - libIDT\_MiniSmartII.h, [316](#)
  - libIDT\_NEO2.h, [365](#)
  - libIDT\_PipReader.h, [454](#)
  - libIDT\_SpectrumPro.h, [477](#)
  - libIDT\_SREDKey2.h, [518](#)
  - libIDT\_UniPayI\_V.h, [537](#)
  - libIDT\_Vendi.h, [561](#)
  - libIDT\_VP3300\_AJ.h, [584](#)
  - libIDT\_VP3300\_BT.h, [622](#)
  - libIDT\_VP3300\_COM.h, [658](#)
  - libIDT\_VP3300\_USB.h, [695](#)
  - libIDT\_VP8800.h, [733](#)
- pCMR\_callBack
  - libIDT\_Augusta.h, [51](#)
  - libIDT\_Device.h, [109](#)
  - libIDT\_KioskIII.h, [264](#)
  - libIDT\_L100.h, [285](#)
  - libIDT\_MiniSmartII.h, [316](#)
  - libIDT\_NEO2.h, [365](#)
  - libIDT\_PipReader.h, [455](#)
  - libIDT\_SpectrumPro.h, [478](#)
  - libIDT\_SREDKey2.h, [518](#)
  - libIDT\_UniPayI\_V.h, [537](#)
  - libIDT\_Vendi.h, [562](#)
  - libIDT\_VP3300\_AJ.h, [584](#)
  - libIDT\_VP3300\_BT.h, [623](#)
  - libIDT\_VP3300\_COM.h, [659](#)
  - libIDT\_VP3300\_USB.h, [695](#)
  - libIDT\_VP8800.h, [733](#)
- pCSFS\_callBack
  - libIDT\_Augusta.h, [51](#)
  - libIDT\_Device.h, [109](#)
  - libIDT\_KioskIII.h, [264](#)
  - libIDT\_L100.h, [286](#)
  - libIDT\_MiniSmartII.h, [317](#)
  - libIDT\_NEO2.h, [366](#)
  - libIDT\_PipReader.h, [455](#)
  - libIDT\_SpectrumPro.h, [478](#)
  - libIDT\_SREDKey2.h, [518](#)
  - libIDT\_UniPayI\_V.h, [537](#)
  - libIDT\_Vendi.h, [562](#)
  - libIDT\_VP3300\_AJ.h, [584](#)
  - libIDT\_VP3300\_BT.h, [623](#)
  - libIDT\_VP3300\_COM.h, [659](#)
  - libIDT\_VP3300\_USB.h, [696](#)
  - libIDT\_VP8800.h, [733](#)
- pEMV\_callBack
  - libIDT\_Augusta.h, [51](#)
  - libIDT\_Device.h, [109](#)
  - libIDT\_KioskIII.h, [264](#)
  - libIDT\_L100.h, [286](#)
  - libIDT\_MiniSmartII.h, [317](#)
  - libIDT\_NEO2.h, [366](#)
  - libIDT\_PipReader.h, [455](#)
  - libIDT\_SpectrumPro.h, [478](#)
  - libIDT\_SREDKey2.h, [518](#)
  - libIDT\_UniPayI\_V.h, [537](#)
  - libIDT\_Vendi.h, [562](#)
  - libIDT\_VP3300\_AJ.h, [584](#)
  - libIDT\_VP3300\_BT.h, [623](#)
  - libIDT\_VP3300\_COM.h, [659](#)
  - libIDT\_VP3300\_USB.h, [696](#)
  - libIDT\_VP8800.h, [733](#)

- pFW\_callback
  - libIDT\_Augusta.h, 51
  - libIDT\_Device.h, 109
  - libIDT\_L100.h, 286
  - libIDT\_NEO2.h, 366
  - libIDT\_SREDKey2.h, 518
- pLCD\_callback
  - libIDT\_Device.h, 109
  - libIDT\_NEO2.h, 366
  - libIDT\_SREDKey2.h, 518
- pLog\_callback
  - libIDT\_Device.h, 110
  - libIDT\_VP8800.h, 733
- pMSR\_callback
  - libIDT\_Augusta.h, 52
  - libIDT\_Device.h, 110
  - libIDT\_KioskIII.h, 264
  - libIDT\_L100.h, 286
  - libIDT\_MiniSmartII.h, 317
  - libIDT\_NEO2.h, 366
  - libIDT\_PipReader.h, 455
  - libIDT\_SpectrumPro.h, 478
  - libIDT\_SREDKey2.h, 519
  - libIDT\_UniPayI\_V.h, 538
  - libIDT\_Vendi.h, 562
  - libIDT\_VP3300\_AJ.h, 585
  - libIDT\_VP3300\_BT.h, 623
  - libIDT\_VP3300\_COM.h, 659
  - libIDT\_VP3300\_USB.h, 696
  - libIDT\_VP8800.h, 734
- pMSR\_callbackp
  - libIDT\_Augusta.h, 52
  - libIDT\_Device.h, 110
  - libIDT\_KioskIII.h, 264
  - libIDT\_L100.h, 286
  - libIDT\_MiniSmartII.h, 317
  - libIDT\_NEO2.h, 366
  - libIDT\_PipReader.h, 455
  - libIDT\_SpectrumPro.h, 478
  - libIDT\_SREDKey2.h, 519
  - libIDT\_UniPayI\_V.h, 538
  - libIDT\_Vendi.h, 562
  - libIDT\_VP3300\_AJ.h, 585
  - libIDT\_VP3300\_BT.h, 623
  - libIDT\_VP3300\_COM.h, 659
  - libIDT\_VP3300\_USB.h, 696
  - libIDT\_VP8800.h, 734
- pMessageHotplug
  - libIDT\_Augusta.h, 52
  - libIDT\_Device.h, 110
  - libIDT\_KioskIII.h, 264
  - libIDT\_L100.h, 286
  - libIDT\_MiniSmartII.h, 317
  - libIDT\_NEO2.h, 366
  - libIDT\_PipReader.h, 455
  - libIDT\_SpectrumPro.h, 478
  - libIDT\_SREDKey2.h, 518
  - libIDT\_UniPayI\_V.h, 537
- libIDT\_Vendi.h, 562
- libIDT\_VP3300\_AJ.h, 584
- libIDT\_VP3300\_BT.h, 623
- libIDT\_VP3300\_COM.h, 659
- libIDT\_VP3300\_USB.h, 696
- libIDT\_VP8800.h, 733
- pPIN\_callback
  - libIDT\_Augusta.h, 52
  - libIDT\_Device.h, 110
  - libIDT\_KioskIII.h, 264
  - libIDT\_L100.h, 286
  - libIDT\_MiniSmartII.h, 317
  - libIDT\_NEO2.h, 366
  - libIDT\_PipReader.h, 455
  - libIDT\_SpectrumPro.h, 478
  - libIDT\_SREDKey2.h, 519
  - libIDT\_UniPayI\_V.h, 538
  - libIDT\_Vendi.h, 563
  - libIDT\_VP3300\_AJ.h, 585
  - libIDT\_VP3300\_BT.h, 624
  - libIDT\_VP3300\_COM.h, 660
  - libIDT\_VP3300\_USB.h, 696
  - libIDT\_VP8800.h, 734
- pRKI\_callback
  - libIDT\_Device.h, 110
- pReadDataLog
  - libIDT\_Augusta.h, 52
  - libIDT\_Device.h, 110
  - libIDT\_KioskIII.h, 265
  - libIDT\_L100.h, 286
  - libIDT\_MiniSmartII.h, 317
  - libIDT\_NEO2.h, 366
  - libIDT\_PipReader.h, 456
  - libIDT\_SpectrumPro.h, 479
  - libIDT\_SREDKey2.h, 519
  - libIDT\_UniPayI\_V.h, 538
  - libIDT\_Vendi.h, 563
  - libIDT\_VP3300\_AJ.h, 585
  - libIDT\_VP3300\_BT.h, 624
  - libIDT\_VP3300\_COM.h, 660
  - libIDT\_VP3300\_USB.h, 696
  - libIDT\_VP8800.h, 734
- pSendDataLog
  - libIDT\_Augusta.h, 52
  - libIDT\_Device.h, 110
  - libIDT\_KioskIII.h, 265
  - libIDT\_L100.h, 286
  - libIDT\_MiniSmartII.h, 317
  - libIDT\_NEO2.h, 367
  - libIDT\_PipReader.h, 456
  - libIDT\_SpectrumPro.h, 479
  - libIDT\_SREDKey2.h, 519
  - libIDT\_UniPayI\_V.h, 538
  - libIDT\_Vendi.h, 563
  - libIDT\_VP3300\_AJ.h, 585
  - libIDT\_VP3300\_BT.h, 624
  - libIDT\_VP3300\_COM.h, 660
  - libIDT\_VP3300\_USB.h, 696



- libIDT\_VP8800.h, [734](#)
- parseMSRData
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [243](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_NEO2.h, [446](#)
  - libIDT\_PipReader.h, [473](#)
  - libIDT\_SpectrumPro.h, [513](#)
  - libIDT\_UniPayI\_V.h, [559](#)
  - libIDT\_Vendi.h, [580](#)
  - libIDT\_VP3300\_AJ.h, [619](#)
  - libIDT\_VP3300\_BT.h, [655](#)
  - libIDT\_VP3300\_COM.h, [690](#)
  - libIDT\_VP3300\_USB.h, [728](#)
  - libIDT\_VP8800.h, [792](#)
- parsePINBlockData
  - libIDT\_Device.h, [243](#)
  - libIDT\_SpectrumPro.h, [513](#)
- parsePINData
  - libIDT\_Device.h, [244](#)
  - libIDT\_SpectrumPro.h, [513](#)
- pin\_cancelPINEntry
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [244](#)
  - libIDT\_NEO2.h, [447](#)
  - libIDT\_SpectrumPro.h, [514](#)
- pin\_capturePin
  - libIDT\_Device.h, [244](#)
  - libIDT\_NEO2.h, [447](#)
- pin\_capturePinExt
  - libIDT\_Device.h, [244](#)
  - libIDT\_NEO2.h, [447](#)
- pin\_getEncryptedOnlinePIN
  - libIDT\_Device.h, [245](#)
  - libIDT\_VP8800.h, [792](#)
- pin\_getEncryptedPIN
  - libIDT\_Device.h, [246](#)
  - libIDT\_L100.h, [305](#)
- pin\_getFunctionKey
  - libIDT\_Device.h, [246](#)
  - libIDT\_L100.h, [305](#)
- pin\_getPAN
  - libIDT\_Device.h, [246](#)
  - libIDT\_VP8800.h, [792](#)
- pin\_getPIN
  - libIDT\_Device.h, [248](#)
  - libIDT\_SpectrumPro.h, [514](#)
- pin\_getPanEntry
  - libIDT\_Device.h, [248](#)
  - libIDT\_NEO2.h, [448](#)
- pin\_inputFromPrompt
  - libIDT\_Device.h, [249](#)
  - libIDT\_NEO2.h, [449](#)
- pin\_promptCreditDebit
  - libIDT\_Device.h, [249](#)
  - libIDT\_VP8800.h, [792](#)
- pin\_promptForAmount
  - libIDT\_Device.h, [250](#)
- pin\_promptForAmountInput
  - libIDT\_Device.h, [250](#)
  - libIDT\_L100.h, [306](#)
- pin\_promptForKeyInput
  - libIDT\_Device.h, [253](#)
  - libIDT\_L100.h, [309](#)
- pin\_promptForNumericKey
  - libIDT\_Device.h, [257](#)
  - libIDT\_NEO2.h, [450](#)
- pin\_promptForNumericKeyWithSwipe
  - libIDT\_Device.h, [258](#)
  - libIDT\_NEO2.h, [450](#)
- pin\_registerCallBk
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [258](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_L100.h, [312](#)
  - libIDT\_MiniSmartII.h, [359](#)
  - libIDT\_NEO2.h, [451](#)
  - libIDT\_PipReader.h, [474](#)
  - libIDT\_SpectrumPro.h, [514](#)
  - libIDT\_SREDKey2.h, [533](#)
  - libIDT\_UniPayI\_V.h, [559](#)
  - libIDT\_Vendi.h, [580](#)
  - libIDT\_VP3300\_AJ.h, [619](#)
  - libIDT\_VP3300\_BT.h, [655](#)
  - libIDT\_VP3300\_COM.h, [692](#)
  - libIDT\_VP3300\_USB.h, [728](#)
  - libIDT\_VP8800.h, [793](#)
- pin\_sendBeep
  - libIDT\_Device.h, [258](#)
  - libIDT\_L100.h, [312](#)
- pin\_setKeyValues
  - libIDT\_Device.h, [259](#)
  - libIDT\_L100.h, [313](#)
  - libIDT\_NEO2.h, [451](#)
- registerHotplugCallBk
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [259](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_L100.h, [313](#)
  - libIDT\_MiniSmartII.h, [359](#)
  - libIDT\_NEO2.h, [451](#)
  - libIDT\_PipReader.h, [474](#)
  - libIDT\_SpectrumPro.h, [514](#)
  - libIDT\_SREDKey2.h, [533](#)
  - libIDT\_UniPayI\_V.h, [559](#)
  - libIDT\_Vendi.h, [580](#)
  - libIDT\_VP3300\_AJ.h, [619](#)
  - libIDT\_VP3300\_BT.h, [655](#)
  - libIDT\_VP3300\_COM.h, [692](#)
  - libIDT\_VP3300\_USB.h, [728](#)
  - libIDT\_VP8800.h, [793](#)
- registerLogCallBk
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [259](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_L100.h, [313](#)

- libIDT\_MiniSmartII.h, [359](#)
- libIDT\_NEO2.h, [451](#)
- libIDT\_PipReader.h, [474](#)
- libIDT\_SpectrumPro.h, [515](#)
- libIDT\_SREDKey2.h, [533](#)
- libIDT\_UniPayI\_V.h, [559](#)
- libIDT\_Vendi.h, [580](#)
- libIDT\_VP3300\_AJ.h, [619](#)
- libIDT\_VP3300\_BT.h, [655](#)
- libIDT\_VP3300\_COM.h, [692](#)
- libIDT\_VP3300\_USB.h, [728](#)
- libIDT\_VP8800.h, [793](#)
- rs232\_device\_init
  - libIDT\_Device.h, [259](#)
  - libIDT\_KioskIII.h, [282](#)
  - libIDT\_MiniSmartII.h, [359](#)
  - libIDT\_NEO2.h, [452](#)
  - libIDT\_PipReader.h, [474](#)
  - libIDT\_SpectrumPro.h, [515](#)
  - libIDT\_SREDKey2.h, [533](#)
- SDK\_Version
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [260](#)
  - libIDT\_KioskIII.h, [283](#)
  - libIDT\_L100.h, [313](#)
  - libIDT\_MiniSmartII.h, [359](#)
  - libIDT\_NEO2.h, [452](#)
  - libIDT\_PipReader.h, [474](#)
  - libIDT\_SpectrumPro.h, [515](#)
  - libIDT\_SREDKey2.h, [533](#)
  - libIDT\_UniPayI\_V.h, [559](#)
  - libIDT\_Vendi.h, [580](#)
  - libIDT\_VP3300\_AJ.h, [619](#)
  - libIDT\_VP3300\_BT.h, [655](#)
  - libIDT\_VP3300\_COM.h, [692](#)
  - libIDT\_VP3300\_USB.h, [728](#)
  - libIDT\_VP8800.h, [793](#)
- setAbsoluteLibraryPath
  - libIDT\_Augusta.h, [100](#)
  - libIDT\_Device.h, [260](#)
  - libIDT\_KioskIII.h, [283](#)
  - libIDT\_L100.h, [313](#)
  - libIDT\_MiniSmartII.h, [360](#)
  - libIDT\_NEO2.h, [452](#)
  - libIDT\_PipReader.h, [475](#)
  - libIDT\_SpectrumPro.h, [515](#)
  - libIDT\_SREDKey2.h, [533](#)
  - libIDT\_UniPayI\_V.h, [559](#)
  - libIDT\_Vendi.h, [581](#)
  - libIDT\_VP3300\_AJ.h, [619](#)
  - libIDT\_VP3300\_BT.h, [655](#)
  - libIDT\_VP3300\_COM.h, [692](#)
  - libIDT\_VP3300\_USB.h, [728](#)
  - libIDT\_VP8800.h, [793](#)
- Source\_C/libIDT\_Augusta.h, [48](#)
- Source\_C/libIDT\_Device.h, [101](#)
- Source\_C/libIDT\_KioskIII.h, [262](#)
- Source\_C/libIDT\_L100.h, [283](#)
- Source\_C/libIDT\_MiniSmartII.h, [314](#)
- Source\_C/libIDT\_NEO2.h, [360](#)
- Source\_C/libIDT\_PipReader.h, [453](#)
- Source\_C/libIDT\_SREDKey2.h, [516](#)
- Source\_C/libIDT\_SpectrumPro.h, [475](#)
- Source\_C/libIDT\_UniPayI\_V.h, [535](#)
- Source\_C/libIDT\_VP3300\_AJ.h, [581](#)
- Source\_C/libIDT\_VP3300\_BT.h, [620](#)
- Source\_C/libIDT\_VP3300\_COM.h, [656](#)
- Source\_C/libIDT\_VP3300\_USB.h, [692](#)
- Source\_C/libIDT\_VP8800.h, [729](#)
- Source\_C/libIDT\_Vendi.h, [560](#)
- v4Comm\_callBack
  - libIDT\_Augusta.h, [52](#)
  - libIDT\_Device.h, [110](#)
  - libIDT\_KioskIII.h, [265](#)
  - libIDT\_L100.h, [287](#)
  - libIDT\_MiniSmartII.h, [317](#)
  - libIDT\_NEO2.h, [367](#)
  - libIDT\_PipReader.h, [456](#)
  - libIDT\_SpectrumPro.h, [479](#)
  - libIDT\_SREDKey2.h, [519](#)
  - libIDT\_UniPayI\_V.h, [538](#)
  - libIDT\_Vendi.h, [563](#)
  - libIDT\_VP3300\_AJ.h, [585](#)
  - libIDT\_VP3300\_BT.h, [624](#)
  - libIDT\_VP3300\_COM.h, [660](#)
  - libIDT\_VP3300\_USB.h, [696](#)
  - libIDT\_VP8800.h, [734](#)
- ws\_deleteSSLCert
  - libIDT\_Device.h, [260](#)
  - libIDT\_VP8800.h, [793](#)
- ws\_getCertChainType
  - libIDT\_Device.h, [260](#)
  - libIDT\_VP8800.h, [794](#)
- ws\_loadSSLCert
  - libIDT\_Device.h, [260](#)
  - libIDT\_VP8800.h, [794](#)
- ws\_requestCSR
  - libIDT\_Device.h, [261](#)
  - libIDT\_VP8800.h, [794](#)
- ws\_revokeSSLCert
  - libIDT\_Device.h, [261](#)
  - libIDT\_VP8800.h, [794](#)
- ws\_updateRootCertificate
  - libIDT\_Device.h, [261](#)
  - libIDT\_VP8800.h, [795](#)