IDTECH®

Value through Innovation

USER MANUAL

Sign&Pay™

API Reference Guide

V1.05

80098501-001-A

Sept 27, 2011

Target Device:

Sign&Pay

Description:

Support Sign&Pay device USBHID and RS232 interface.

Platform:

Microsoft Windows XP, Windows 2000, Vista

DLL Usage (Microsoft Visual C++ 6.0)

Add Sign_PayKit.lib to Project->Settings->Link->Object/library modules and include the head file Sign_PayKit.h , then call the DLL function directly

Command Summary

All commands supported are listed below.

- Pay_OpenHid
- Pay_OpenPort
- Pay_Close
- Pay_GetSerialNumber
- Pay_SetSerialNumber
- Pay_GetVersion
- Pay_ControlLED
- Pay_GenerateTone
- Pay_ControlAudio
- Pay_GetSdkVersion
- Pay_ReSet
- Pay_GetModelNumber
- uSign_SetPenColor
- uSign_DrawLine
- uSign_DrawRectangle
- uSign_DrawArc
- uSign_BrushColor
- uSign_FillRectangle
- uSign_FillArc
- uSign_SetFont
- uSign_SetTextColor
- uSign_SetBackgroundColor
- uSign_SelectBackgroundMode
- uSign_DrawString
- uSign_DrawStringInRectangle
- uSign_GetPicture
- uSign_ShowPicture
- uSign_StorePicture
- uSign_ShowStorePicture
- uSign_RetrieveStorePicture

uSign_CalibrateDevice
uSign_SetClipArea
uSign_CreateRegion
uSign_StartCapture
uSign_AddPointHandle
uSign_ContinueCapture
uSign_PauseCapture
uSign_GetPointCount
uSign_ClearSignature
uSign_ExitCapture
uSign_GetSignFormat
uSign_SetSignFormat
MSR_EnableSecureHead
MSR_DisableSecureHead
MSR_AddDataHandle
MSR_GetOutputSetting
MSR_GetDecodingSetting
MSR_SetDecodingSetting
MSR_GetAllSettings
MSR_GetSecureHeadFirmware
MSR_ReviewSecureHeadPrePANID
MSR_ReviewSecureHeadPostPANID
MSR_ReviewSecureHeadMaskPAN
MSR_ReviewKSNAndCountID
MSR_SetSecureHeadPrePANID
MSR_SetSecureHeadPostPANID
MSR_SetSecureHeadMaskPAN
MSR_SetExpirationData
MSR_SetEncryption
MSR_SerialNumber
MSR_GetKeyModel
MSR_SetKeyMode
MSR_GetEncryptedData
MSR_SetEncryptedData
MSR_LoadDeviceKey
MSR_GetSecurityLevel
MSR_GetMSROutputStatus
MSR_GetDataOutputFormat
MSR_SetDataOutputFormat
MSR_SetOutputEncryptedFormat (supported in firmware v1.00.027 and above)
MSR_SetEncryptOption (supported in firmware v1.00.027 and above)
MSR_SetHashOption (supported in firmware v1.00.027 and above)

MSR_SetMaskOption (supported in firmware v1.00.027 and above)
MSR_SetDataPINorDataKey (supported in firmware v1.00.027 and above)
MSR_getDataPINorDataKey (supported in firmware v1.00.027 and above)
MSR_ManualInputCardData (supported in firmware v1.00.027 and above)
PIN_GetPINBlock
PIN_GetEncryptedData
PIN_CancelPIN
PIN_GetNumericOrAmount
PIN_GetKey
PIN_ClearKey
PIN_GetCardAccount
PIN_InvalidateKey

Function description

General function API

Function:	Pay_OpenHid	
Description:	Open USBHID device	
Format:	BYTE Pay_OpenHid(UINT auiVid, UINT auiPid)	
Parameter:	auiVid	The Vendor ID
	auiPid	The Product ID
Return:	Appendix A	
Example:	Pay_OpenHid(0x0ACD,0x2310);	
Function:	Pay_OpenPort	
Description:	Open RS232 device	
Format:	BYTE Pay_OpenPort(int Comport,long Baud, char Parity, int Stop, int Data)	
Parameter:	Comport	Port number.
	Baud	Baud rate,current baud rate is 38400.
	Parity	Parity check,curren parity is None.
	stop	Stop bit,current stop is 1.
	data	Data bit,current data is 8.
Return:	Appendix A	
Example:	Pay_OpenPort(1,38400, 'N', 1, 8)	
Function:	Pay_Close	
Description:	Close device	
Format:	bool Pay_Close()	
Parameter:	None	
Return:	Appendix A	
Example:	Pay_Close();	

Function:	Pay_GetSerialNumber	
Description:	Get the Serial Number of device	
Format:	BYTE Pay_GetSerialNumber(char *sNumber, int *length)	
Parameter:	sNumber	Serial Number string
	length	The length of Serial Number string
Return:	Appendix A	
Example:	Pay_GetSerialNumber(Serial, &length);	
Function:	Pay_SetSerialNumber	
Description:	Set Serial Number to device	
Format:	BYTE Pay_SetSerialNumber(char *sNumber, int length)	
Parameter:	sNumber	Serial Number string
	length	The length of Serial Number string,it must be eight.
Return:	Appendix A	
Example:	Pay_SetSerialNumber("IDTECH-2010", 11);	
Function:	Pay_GetVersion	
Description:	Get the version of device	
Format:	BYTE Pay_GetVersion(char *sVersion, int *length)	
Parameter:	sVersion	The version string
	length	The length of version string
Return:	Appendix A	
Example:	Pay_GetVersion(Version, &length);	
Function:	Pay_ControlLED	
Description:	Control LED	
Format:	BYTE Pay_ControlLED(BYTE L_Led,BYTE R_Led)	
Parameter:	L_Led	0x00:Left led is OFF; 0x01:Left Red led is ON; 0x02:Left Green led is ON; 0x03:Left Red led is flash; 0x04:Left Green led is flash;
	R_Led	0x00:Right led is OFF; 0x01: Right Red led is ON; 0x02: Right Green led is ON; 0x03: Right Red led is flash; 0x04: Right Green led is flash;
Return:	Appendix A	
Example:	Pay_ControlLED(0x02,0x00);	

Function:	Pay_GenerateTone	
Description:	Generate tone with specified frequency and duration	
Format:	BYTE Pay_GenerateTone(int Frequency, int Duration)	
Parameter:	Frequency	5 < Frequency < 40,000 Hz
	Duration	0 =< Duration < 65536 mS.
Return:	Appendix A	
Example:	Pay_GenerateTone(2000,300);	
Function:	Pay_ControlAudio	
Description:	Sound control enable/disable set	
Format:	BYTE Pay_ControlAudio (bool f_Audio)	
Parameter:	f_Audio	0 – disable
		1 – enable
Return:	Appendix A	
Example:	Pay_ControlAudio(true);	
Function:	Pay_GetSdkVersion	
Description:	Get the version of SDK	
Format:	BYTE Pay_GetSdkVersion(char *DllVersion, int *Length)	
Parameter:	DllVersion	The SDK version string
	Length	The length of SDK version string
Return:	Appendix A	
Example:	Pay_GetSdkVersion(Serial, &length);	
Function:	Pay_ReSet 返回值	
Description:	ReSet the device	
Format:	BYTE Pay_ReSet ()	
Parameter:	None	
Return:	Appendix A	
Example:	Pay_ReSet ()	
Function:	Pay_GetModelNumber	
Description:	Get the model number of device.	
Format:	BYTE Pay_GetModelNumber(char *sNumber, int *length)	
Parameter:	sNumber	The mode number string
	length	The length of mode number string
Return:	Appendix A	
Example:	Pay_GetModelNumber(sNumber, &length)	

uSign function API

Function:	uSign_SetPenColor（宽度变了无法画出）	
Description:	Set the pen color used to draw line on the LCD	
Format:	BYTE uSign_SetPenColor(BYTE P_Width1,BYTE P_Width2,BYTE P_Width3,BYTE P_Width4,BYTE C_Red,BYTE C_Green,BYTE C_Blue)	
Parameter	P_Width1~ P_Width4	the pen's width, 4 bytes long.
	C_Red, C_Green, C_Blue	the pen's color,0~255
Return:	Appendix A	
Example:	uSign_SetPenColor(0x01,0x00,0x00,0x00,0x00,0x00,0xff);	
Function:	uSign_DrawLine	
Description:	Draw line from point <left><top> to <right><bottom> using the pen.	
Format:	BYTE uSign_DrawLine(int left,int top,int right,int bottom)	
Parameter:	left	X-coordinate of start point,319 >= left >= 0;
	top	Y-coordinate of start point, 239 >= top >=0;
	right	X-coordinate of end point, 319 >= right >= 0;
	bottom	Y-coordinate of end point, 239 >= bottom >=0;
Return:	Appendix A	
Example:	uSign_DrawLine(0,0,319,239);	
Function:	uSign_DrawRectangle	
Description:	Draw rectangle defined by top left point <left><top> and bottom right point <right><bottom> using the pen.	
Format:	BYTE uSign_DrawRectangle(int left,int top,int right,int bottom)	
Parameter:	left	X-coordinate of start point,319 => left >= 0; left < right
	top	Y-coordinate of start point,239 => top >= 0; top < bottom
	right	X-coordinate of end point,319 => right >= 0;
	bottom	Y-coordinate of end point,239 => bottom >= 0;
Return:	Appendix A	
Example:	uSign_DrawRectangle(10,10,300,200);	
Function:	uSign_DrawArc	
Description:	Draw arc defined by center point, radius, start angle and sweep angle use pen.	
Format:	BYTE uSign_DrawArc(int left,int top,int I_Radius,int I_StartAngle,int I_SweepAngle)	
Parameter:	left	specifies the x-coordinate of the center of the related circle
	top	specifies the y-coordinate of the center of the related circle
	I_Radius	specifies the radius of the related circle
	I_StartAngle	specifies the starting angle in degrees relative to the x-axis. Unit is 0.1.

	I_SweepAngle	specifies the sweep angle in degrees relative to the starting angle. Unit is 0.1.
Return:	Appendix A	
Example:	uSign_DrawArc(1,1,200,0,6000);	
Function:	uSign_BrushColor	
Description:	Set the brush's color used to fill region on the LCD	
Format:	BYTE uSign_BrushColor(BYTE C_Red,BYTE C_Green,BYTE C_Blue)	
Parameter:	C_Red	the pen's color; 0~255
	C_Green	the pen's color; 0~255
	C_Blue	the pen's color; 0~255
Return:	Appendix A	
Example:	uSign_BrushColor(0xFF,0xff,0xff);	
Function:	uSign_FillRectangle	
Description:	Fill rectangle define by top left point <left><top> and bottom right point <right><bottom> using the brush	
Format:	BYTE uSign_FillRectangle(int left, int top, int right, int bottom)	
Parameter:	left	X-coordinate of top left point, left < right.
	top	Y-coordinate of top left point
	right	X-coordinate of bottom right point, top < bottom
	bottom	Y-coordinate of bottom right point
Return:	Appendix A	
Example:	uSign_FillRectangle(0,0,319,239);	
Function:	uSign_FillArc	
Description:	Draw arc defined by center point, radius, start angle and sweep angle use brush.	
Format:	BYTE uSign_FillArc(int left,int top,int I_Radius,int I_StartAngle,int I_SweepAngle)	
Parameter:	left	specifies the x-coordinate of the center of the related circle
	top	specifies the y-coordinate of the center of the related circle
	I_Radius	specifies the radius of the related circle
	I_StartAngle	specifies the starting angle in degrees relative to the x-axis, Unit is 0.1
	I_SweepAngle	specifies the sweep angle in degrees relative to the starting angle, Unit is 0.1
Return:	Appendix A	
Example:	uSign_FillArc(1,1,200,0,6000);	
Function:	uSign_SetFont	

Description:	Set the font for text display on the LCD	
Format:	BYTE uSign_SetFont(BYTE Height,BYTE Width,BYTE Weight,BYTE Italic,BYTE Underline,BYTE CharSet)	
Parameter:	Height	specifies the height of a char
	Width	specifies the width of a char.
	Weight	specifies the weight of the char
	Italic	specifies the italic of the char
	Underline	specifies the underline of the char
	CharSet	specifies the char set. 1 byte. The valid size is 1 –6
Return:	Appendix A	
Example:	uSign_SetFont(0x10,0x0C,0x10,0x00,0x00,0x03);	
Function:	uSign_SetTextColor	
Description:	Set the text's color	
Format:	BYTE uSign_SetTextColor(BYTE C_Red,BYTE C_Green,BYTE C_Blue)	
Parameter:	C_Red	the text's color, 0~255;
	C_Green	the text's color, 0~255;
	C_Blue	the text's color, 0~255;
Return:	Appendix A	
Example:	uSign_SetTextColor(0,0,0xff);	
Function:	uSign_SelectBackgroundMode	
Description:	Set the background mode for text display on the LCD	
Format:	BYTE uSign_SelectBackgroundMode(BYTE F_Mode)	
Parameter:	F_Mode	specifies background mode. 1 byte. 0x00 means OPAQUE and others means TRANSPARENT.
Return:	Appendix A	
Example:	uSign_SelectBackgroundMode(0x01);	
Function:	uSign_SetBackgroundColor	
Description:	Set the background color when display text on the LCD	
Format:	BYTE uSign_SetBackgroundColor(BYTE C_Red,BYTE C_Green,BYTE C_Blue)	
Parameter:	C_Red	the backgound's color, 0~255;
	C_Green	the backgound's color, 0~255;
	C_Blue	the backgound's color, 0~255;
Return:	Appendix A	
Example:	uSign_SetBackgroundColor(255,255,255);	
Function:	uSign_DrawString	

Description:	Draw string using the selected font and colors on the LCD	
Format:	BYTE uSign_DrawString(int left, int top, int sLength, char *strData)	
Parameter:	left	specifies the X-coordinate of the start point
	top	specifies the Y-coordinate of the start point
	sLength	specifies the length of the string in chars
	strData	specifies the string to be displayed
Return:	Appendix A	
Example:	uSign_DrawString(35,140,len,str);	
Function:	uSign_DrawStringInRectangle	
Description:	Draw string using the selected font and colors on the LCD The string will be displayed in the specified rectangle, from the top left of the rectangle to the right bottom of the rectangle	
Format:	BYTE uSign_DrawStringInRectangle(int left, int top, int right, int bottom, int sLength, char *strData)	
Parameter:	left	specifies the X-coordinate of the start point
	top	specifies the Y-coordinate of the start point
	right	specifies the X-coordinate of the end point
	bottom	specifies the Y-coordinate of the end point
	sLength	specifies the length of the string in chars
	strData	specifies the string to be displayed
Return:	Appendix A	
Example:	uSign_DrawStringInRectangle (0,0,40,140,len,str);	
Function:	uSign_GetPicture	
Description:	Get picture on the LCD defined by top left point <left><top> and bottom right point <right><bottom>	
Format:	BYTE uSign_GetPicture(int left,int top,int right,int bottom, BYTE *p_Data,unsigned int sLength,unsigned int *rLength)	
Parameter:	left	X-coordinate of top left point
	top	Y-coordinate of top left point
	right	X-coordinate of bottom right point
	bottom	Y-coordinate of bottom right point
	p_Data	Get picture data
	sLength	The length of p_Data buffer.
	rLength	The length of getting picture data
Return:	Appendix A	
Example:	uSign_GetPicture(0,0,20,20,rec,240000,&rlen);	
Function:	uSign_ShowPicture	

Description:	Show picture on the LCD defined by top left point <left><top> and bottom right point <right><bottom>.	
Format:	BYTE uSign_ShowPicture(int left,int top,int right,int bottom, BYTE *p_Data,unsigned int sLength)	
Parameter:	left	X-coordinate of top left point
	top	Y-coordinate of top left point
	right	X-coordinate of bottom right point
	bottom	Y-coordinate of bottom right point
	P_Data	The picture data
	sLength	The length of picture data
Return:	Appendix A	
Example:	uSign_ShowPicture (0,0,20,20,p_Data, rlen);	
Function:	uSign_StorePicture	
Description:	Store picture in the device.	
Format:	BYTE uSign_StorePicture(BYTE ID, BYTE type,BYTE *p_Data,unsigned int sLength)	
Parameter:	ID	the identifier for the picture
	type	the picture's type. . 0x00 means RAW format, 0x01 means 24-bit true color BMP format, 0x02 means JPEG format
	P_Data	The picture data
	sLength	The length of picture data
Return:	Appendix A	
Example:	uSign_StorePicture(0x01,0x02,p_Data,len);	
Function:	uSign_ShowStorePicture	
Description:	Show stored picture on the LCD defined by top left point <left><top> and bottom right point <right><bottom>	
Format:	BYTE uSign_ShowStorePicture(BYTE ID, int left,int top,int right,int bottom)	
Parameter:	ID	the identifier for the picture
	left	X-coordinate of top left point
	top	Y-coordinate of top left point
	right	X-coordinate of bottom right point
	bottom	Y-coordinate of bottom right point
Return:	Appendix A	
Example:	uSign_ShowStorePicture(0x01,10,10,200,200);	
Function:	uSign_RetrieveStorePicture	
Description:	Retrieve stored picture in the device.	
Format:	BYTE uSign_RetrieveStorePicture(BYTE *p_ID,int *rLength)	

Parameter:	P_ID	The ID buffer(two bytes for one ID)
	rLength	The length of p_ID buffer
Return:	Appendix A	
Example:	uSign_RetrieveStorePicture(p_ID,&len);	
Function:	uSign_CalibrateDevice	
Description:	Calibrate the device	
Format:	BYTE uSign_CalibrateDevice()	
Parameter:	None	
Return:	Appendix A	
Example:	uSign_CalibrateDevice()	
Function:	uSign_SetClipArea	
Description:	Set new clip area. The max area is (0,0) – (319,239)	
Format:	BYTE uSign_SetClipArea(int left,int top,int right,int bottom,BYTE ShowMode,BYTE C_Red,BYTE C_Green,BYTE C_Blue)	
Parameter:	left	X-coordinate of top left point
	top	Y-coordinate of top left point
	right	X-coordinate of bottom right point
	bottom	Y-coordinate of bottom right point
	ShowMode	a bitmap for 4 lines. Bit 1 for left line, Bit 2 for right line, Bit 3 for top line and Bit 4 for bottom line. Value 1b means show this line, 0b means don't show this line
	C_Red	The color of rectangle lines,0~255
	C_Greem	The color of rectangle lines,0~255
	C_Blue	The color of rectangle lines,0~255
Return:	Appendix A	
Example:	uSign_SetClipArea(20,70,300,160,8,255,255,255);	
Function:	uSign_CreateRegion	
Description:	Create object like picture, button and text showed on LCD when during signature. The object can be notified when touched.	
Format:	BYTE uSign_CreateRegion (BYTE ID,BYTE type,BYTE state,int left,int top,int right,int bottom,int sLength,char *strData)	
Parameter:	ID	specifies the region's ID
	type	specifies the region's type. 1 byte. 0x01 means BUTTON, 0x02 means PICTURE and 0x03 means TEXT, 0x04 means owner draw button
	state	specifies the region's state, Bit 0 Exists; Bit 1 Visable;

		Bit 2 Enabled; Bit 3 Notify.
	left	X-coordinate of top left point
	top	Y-coordinate of top left point
	right	X-coordinate of bottom right point
	bottom	Y-coordinate of bottom right point
	sLength	specifies the strData length
	strData	specifies the object's data For Button, <Data> is the text showed on the button. For Picture, <Data> is the picture data. Picture is arranged as top left point first and bottom right end. Each point occupies three bytes defined as: RED GREEN BLUE. For Text, <Data> is arranged as: Font(Height 1 byte, Width 1 byte, Weight 1 byte, Italic 1 byte, Underline 1 byte, CharSet 1 byte) TextColour(RED GREEN BLUE) TextBkMode(1 byte) TextBkColour(RED GREEN BLUE) String. For owner draw button, <Data> is: Font(Height 1 byte, Width 1 byte, Weight 1 byte, Italic 1 byte, Underline 1 byte, CharSet 1 byte) TextColour(RED GREEN BLUE) TextBkMode(1 byte) TextBkColour(RED GREEN BLUE) String Offset(X, Y 4 bytes) String.
Return:	Appendix A	
Example:	uSign_CreateRegion (0x04,0x01,0x0f,45,190,125,230,5,"Clear");	
Function:	uSign_StartCapture	
Description:	Start capture using specified parameters	
Format:	BYTE uSign_StartCapture(BYTE f_Mode,BYTE f_Interval,BYTE s_Red,BYTE s_Green,BYTE s_Blue,BYTE b_Red,BYTE b_Green,BYTE b_Blue)	
Parameter:	f_Mode	specifies the capture mode, 0x01~0x05 0x01: Out signature data using FBP format, pen up is 0x8C and pen down is 0x9C. 0x02: Out signature data (the difference of the current point and previous point) using FBP format, pen up is 0x80 and pen down is 0x90. 0x03: Out signature data using CMP format. 0x04: Out signature data using FBP format, pen up is 0x80 and pen down is 0x90. 0x05: Data is buffered and not send out.
	F_interval	specifies the maximum points' interval during signature. If

		exceeds, the signature will be cleared.
	s_Red	specifies signature's color
	s_Green	specifies signature's color
	s_Blue	specifies signature's color
	b_Red	specifies background color
	b_Green	specifies background color
	b_Blue	specifies background color
Return:	Appendix A	
Example:	uSign_StartCapture(0x04,0x7A,0,0,0,255,255,255);	
Function:	uSign_AddPointHandle	
Description:	Register a call-back function for StartCapture function, the function will be called when receiving sign data	
Format:	BYTE uSign_AddPointHandle(PSIGN_FUNC func,LPVOID pParam)	
Parameter:	func	The name of call-back function
	pParam	The currently pointer
Return:	Appendix A	
Example:	uSign_AddPointHandle(point_handle,this);	
Function:	uSign_ContinueCapture	
Description:	Continue capture using specified parameters	
Format:	BYTE uSign_ContinueCapture()	
Parameter:	None	
Return:	Appendix A	
Example:	uSign_ContinueCapture()	
Function:	uSign_PauseCapture	
Description:	Pause capture	
Format:	BYTE uSign_PauseCapture(unsigned int *p_Count)	
Parameter:	P_Count	The current point count
Return:	Appendix A	
Example:	uSign_PauseCapture(&p_Count);	
Function:	uSign_GetPointCount	
Description:	Get script point count	
Format:	BYTE uSign_GetPointCount(unsigned int *p_Count)	
Parameter:	P_Count	The point count
Return:	Appendix A	
Example:	uSign_GetPointCount(&p_Count);	

Function:	uSign_ClearSignature	
Description:	Clear signature	
Format:	BYTE uSign_ClearSignature()	
Parameter:	None	
Return:	Appendix A	
Example:	uSign_ClearSignature()	
Function:	uSign_ExitCapture	
Description:	Exit signature	
Format:	BYTE uSign_ExitCapture()	
Parameter:	None	
Return:	Appendix A	
Example:	uSign_ExitCapture()	
Function:	uSign_GetSignFormat	
Description:	Get buffered signature data.	
Format:	BYTE uSign_GetSignFormat(int s_Type, BYTE *Sign_Data, unsigned int sLength, unsigned int *rLength)	
Parameter:	s_Type	signature format; 1-SIG format 2-CMP format 3-RAW format 4- BMP format
	Sign_Data	signature data buffer
	sLength	The length of signature data buffer
	rLength	The length of signature data
Return:	Appendix A	
Example:	uSign_GetSignFormat(1,Sign_Data,24000,&len);	
Function:	uSign_SetSignFormat	
Description:	Send signature data to uSign	
Format:	BYTE uSign_SetSignFormat(int s_Type, BYTE *Sign_Data, unsigned int sLength)	
Parameter:	s_Type	signature format; 1-SIG format 2-CMP format 3-RAW format 4- BMP format
	Sign_Data	signature data
	sLength	The length of signature data
Return:	Appendix A	

Example:	uSign_SetSignFormat(1,Sign_Data,1000);

MSR API

Function:	MSR_EnableSecureHead	
Description:	Enable SecureHead	
Format:	BYTE MSR_EnableSecureHead()	
Parameter:	None	
Return:	Appendix A	
Example:	MSR_EnableSecureHead ();	
Function:	MSR_DisableSecureHead	
Description:	Disable SecureHead	
Format:	BYTE MSR_DisableSecureHead()	
Parameter:	None	
Return:	Appendix A	
Example:	MSR_DisableSecureHead()	
Function:	MSR_AddDataHandle	
Description:	Register a call-back function for MSR_EnableSecureHead function,the function will be called when receiving SecureHead data.please see Appendix C for more information about data format.	
Format:	BYTE MSR_AddDataHandle(PMSR_FUNC func,LPVOID pParam)	
Parameter:	func	The name of call-back function
	pParam	The currently pointer
Return:	Appendix A	
Example:	MSR_AddDataHandle(Data_Handle,this)	
Function:	MSR_GetOutputSetting	
Description:	Get outputting setting of SecureHead	
Format:	BYTE MSR_GetOutputSetting(unsigned char *t_Out)	
Parameter:	t_Out	0x30: SecureHead Outputting Disabled **0x31:SecureHead Outputting Enabled**
Return:	Appendix A	
Example:	MSR_GetOutputSetting(&t_Out)	
Function:	MSR_GetDecodingSetting	
Description:	Get decoding method of SecureHead	
Format:	BYTE MSR_GetDecodingSetting(unsigned char *t_Dec)	
Parameter:	t_Dec	0x30: Raw Data Decoding in Both Directions, send out in ID

		TECH mode 0x31: Decoding in Both Directions**.** If the encryption feature is enabled, the key management method used is DUKPT. 0x32: Moving stripe along head in direction of encoding. If the encryption feature is enabled, the key management method used is DUKPT. 0x33: Moving stripe along head against direction of encoding. If the encryption feature is enabled, the key management method used is DUKPT. **0x34: Raw Data Decoding in Both Directions, send out in other mode, Its format is** <0x01> <0x01> <0x1A><0x02> <0x00> <8 bytes Device Serial Number> < 0x30> <0x31> <264 bytes of Sampling data>**.** If the encryption feature is enabled, the key management method used is fixed key.
Return:	Appendix A	
Example:	MSR_GetDecodingSetting(&t_Dec)	
Function:	MSR_SetDecodingSetting	
Description:	Set decoding method for SecureHead	
Format:	BYTE MSR_SetDecodingSetting(unsigned char t_Dec)	
Parameter:	t_Dec	0x30: Raw Data Decoding in Both Directions, send out in ID TECH mode 0x31: Decoding in Both Directions**.** If the encryption feature is enabled, the key management method used is DUKPT. 0x32: Moving stripe along head in direction of encoding. If the encryption feature is enabled, the key management method used is DUKPT. 0x33: Moving stripe along head against direction of encoding. If the encryption feature is enabled, the key management method used is DUKPT. **0x34: Raw Data Decoding in Both Directions, send out in other mode, Its format is** <0x01> <0x01> <0x1A><0x02> <0x00> <8 bytes Device Serial Number> < 0x30> <0x31> <264 bytes of Sampling data>**.** If the encryption feature is enabled, the key management method used is fixed key.
Return:	Appendix A	
Example:	MSR_SetDecodingSetting(t_Dec)	
Function:	MSR_GetAllSettings	
Description:	Get all settings of SecureHead	

Format:	BYTE MSR_GetAllSettings(unsigned char *all_Set,int *length)	
Parameter:	all_Set	The buffer of all settings
	length	The length of all settings
Return:	Appendix A	
Example:	MSR_GetAllSettings(all_Set,&length)	
Function:	MSR_GetSecureHeadFirmware	
Description:	Read firmware version of SecureHead	
Format:	BYTE MSR_GetSecureHeadFirmware(char *sVersion, int *length)	
Parameter:	sVersion	The version string
	lenght	The length of version string
Return:	Appendix A	
Example:	MSR_GetSecureHeadFirmware(sVersion &len);	
Function:	MSR_ReviewSecureHeadPrePANID	
Description:	Review First N Digits in PAN which can be clear data	
Format:	BYTE MSR_ReviewSecureHeadPrePANID(unsigned char *tNumber)	
Parameter:	tNumber	the first count in PAN
Return:	Appendix A	
Example:	MSR_ReviewSecureHeadPrePANID(&tNumber)	
Function:	MSR_ReviewSecureHeadPostPANID	
Description:	Review Last M Digits in PAN which can be clear data	
Format:	BYTE MSR_ReviewSecureHeadPostPANID(unsigned char *tNumber)	
Parameter:	tNumber	the last count in PAN
Return:	Appendix A	
Example:	MSR_ReviewSecureHeadPostPANID(&tNumber)	
Function:	MSR_ReviewSecureHeadMaskPAN	
Description:	Read character that used to mask PAN	
Format:	BYTE MSR_ReviewSecureHeadMaskPAN(unsigned char *tChar);	
Parameter:	tChar	the character that used to mask PAN
Return:	Appendix A	
Example:	MSR_ReviewSecureHeadMaskPAN(&tChar);	
Function:	MSR_ReviewKSNAndCountID	
Description:	Review the Key Serial Number and Encryption Counter	
Format:	BYTE MSR_ReviewKSNAndCountID(unsigned char *tData, int *length)	
Parameter:	tData	The buffer of Data, includes the Initial Key Serial Number in the leftmost 59 bits and a value for the Encryption Counter in

		the right most 21 bits.
	length	The length of tData
Return:	Appendix A	
Example:	MSR_ReviewKSNAndCountID(tData, &length)	
Function:	MSR_SetSecureHeadPrePANID	
Description:	Set first N Digits in PAN which can be clear data	
Format:	BYTE MSR_SetSecureHeadPrePANID(unsigned char tNumber)	
Parameter:	tNumber	The first count in PAN
Return:	Appendix A	
Example:	MSR_SetSecureHeadPrePANID(tNumber)	
Function:	MSR_SetSecureHeadPostPANID	
Description:	Set last M Digits in PAN which can be clear data	
Format:	BYTE MSR_SetSecureHeadPostPANID(unsigned char tNumber)	
Parameter:	tNumber	the last count in PAN
Return:	Appendix A	
Example:	MSR_SetSecureHeadPostPANID(tNumber)	
Function:	MSR_SetSecureHeadMaskPAN	
Description:	Set character that used to mask PAN	
Format:	BYTE MSR_SetSecureHeadMaskPAN(unsigned char tChar)	
Parameter:	tChar	The character to mask PAN
Return:	Appendix A	
Example:	MSR_SetSecureHeadMaskPAN(tChar)	
Function:	MSR_SetExpirationData	
Description:	Display expiration data as mask data or clear data	
Format:	BYTE MSR_SetExpirationData(unsigned char t_Mode)	
Parameter:	t_Mode	0x30 Display expiration data as mask data
		0x31 Display expiration data as clear data
Return:	Appendix A	
Example:	MSR_SetExpirationData(t_Mode)	
Function:	MSR_SetEncryption	
Description:	Set securityalgorithm for SecureHead	
Format:	BYTE MSR_SetEncryption(unsigned char t_Encry)	
Parameter:	t_Encry	0x30: Encryption Disabled 0x31: Enable TDES Encryption 0x32: Enable AES Encryption (Not for Raw Data Decoding in

		Both Directions, send out in other mode.)
Return:	Appendix A	
Example:	MSR_SetEncryption(t_Encry)	
Function:	MSR_SerialNumber	
Description:	Read Serial Number of SecureHead	
Format:	BYTE MSR_SerialNumber(char *sNumber, int *length)	
Parameter:	sNumber	The buffer of Serial Number string
	length	The length of Serial Number string
Return:	Appendix A	
Example:	MSR_SerialNumber(sNumber, &length)	
Function:	MSR_GetKeyModel	
Description:	Get model of key management.	
Format:	BYTE MSR_GetKeyModel(unsigned char *t_mKey)	
Parameter:	t_mKey	0x30: Fixed Key 0x31: DUKPT Key
Return:	Appendix A	
Example:	MSR_GetKeyModel(&t_mKey)	
Function:	MSR_SetKeyModel	
Description:	Set model of key management	
Format:	BYTE MSR_SetKeyModel(unsigned char t_mKey)	
Parameter:	t_mKey	0x30: Fixed Key 0x31: DUKPT Key
Return:	Appendix A	
Example:	MSR_SetKeyModel(t_mKey)	
Function:	MSR_GetEncryptedData	
Description:	Get 8 bytes of TDES-encrypted random data. Then use FIX key of SecureHead to encrypte these bytes,and send the result to SecureHead. If these steps are ok, then SecureHead allow you to change FIX key.	
Format:	BYTE MSR_GetEncryptedData(unsigned char *tData, int *length)	
Parameter:	tData	The buffer of 8 bytes of TDES-encrypted random data
	length	The length of 8 bytes of TDES-encrypted random data,it is always 8.
Return:	Appendix A	
Example:	MSR_GetEncryptedData(tData, &length)	
Function:	MSR_SetEncryptedData	

Description:	Send external authenticate data to SecureHead	
Format:	BYTE MSR_SetEncryptedData(unsigned char *tData, int length)	
Parameter:	tData	The buffer of external authenticate data. After executing the API function MSR_GetEncryptedData then get 8 bytes random data and encrypt these data with Fix key of SecureHead
	length	The length of external authenticate data,it must be 8
Return:	Appendix A	
Example:	MSR_SetEncryptedData(tData, length)	
Function:	MSR_LoadDeviceKey	
Description:	Change Fix Key. If you want to change Fix key, you must execute the API function MSR_GetEncryptedData,Get 8-bytes Random Data, and encrypt Random data with Fix key of SecureHead, then send encrypt-data to SecureHead. If all is ok, this command can be executed success.	
Format:	BYTE MSR_LoadDeviceKey(unsigned char *tKey, int length)	
Parameter:	tKey	The buffer of device key
	length	The length of device key,it must be 16.
Return:	Appendix A	
Example:	MSR_LoadDeviceKey(tKey, length)	
Function:	MSR_GetSecurityLevel	
Description:	Read out SecureHead Security level	
Format:	BYTE MSR_GetSecurityLevel(unsigned char *t_Level)	
Parameter:	t_Level	0x30: Security Level 0; 0x31: Security Level 1; 0x32: Security Level 2; 0x33: Security Level 3.
Return:	Appendix A	
Example:	MSR_GetSecurityLevel(t_Level)	
Function:	MSR_GetMSROutputStatus	
Description:	Get SecureHead card data output model	
Format:	BYTE MSR_GetMSROutputStatus(unsigned char *t_Status);	
Parameter:	t_Status	0x30: SecureHead output card data with clear data. 0x31: SecureHead output card data with masked data.
Return:	Appendix A	
Example:	MSR_GetMSROutputStatus(t_Status)	

Function:	MSR_GetDataOutputFormat	
Description:	Get SecureHead card data output format.	
Format:	BYTE MSR_GetDataOutputFormat(unsigned char *t_Format)	
Parameter:	t_Format	0x30: SecureHead output clear card data with no LRC , and there isn`t data '0x0d' in the end track if the track has no data exist. 0x31: SecureHead output clear card data with LRC, and there is a data '0x0d' in the end every track.
Return:	Appendix A	
Example:	MSR_GetDataOutputFormat(&t_Format);	
Function:	MSR_SetDataOutputFormat	
Description:	Set SecureHead card data output format.	
Format:	BYTE MSR_SetDataOutputFormat(unsigned char t_Format)	
Parameter:		0x30: SecureHead output clear card data with no LRC , and there isn`t data '0x0d' in the end track if the track has no data exist. 0x31: SecureHead output clear card data with LRC, and there is a data '0x0d' in the end every track.
Return:	Appendix A	
Example:	MSR_SetDataOutputFormat(0x30);	
Function:	MSR_SetOutputEncryptedFormat (supported in firmware v1.00.027 and above)	
Description:	Set securehead encrypted structure of outputting, SecureHead output structure has two format, one is default structure which Track 1 and Track 2 is encrypted together with AES or Tri-DES, and Track 3 is clear data; the other is new format which every Track is individual encrypted with Aes or Tri-DES.	
Format:	BYTE MSR_SetOutputEncryptedFormat(unsigned char e_Format)	
Parameter:	e_Format	0x30: Default: original encrypt output structure 0x31: enhanced encrypt output structure will send bytes 8 and 9 and CardType will be 1xxxxxxx (high bit =1)
Return:	Appendix A	
Example:	MSR_SetOutputEncryptedFormat(0x30);	
Function:	MSR_SetEncryptOption (supported in firmware v1.00.027 and above)	
Description:	Set encrypted output format of card data in new structure.	
Format:	BYTE MSR_SetEncryptOption(bool f_Track1,bool f_Track2,bool f_Track3,bool f_Other)	

Parameter:	f_Track1	Track1 force encrypt
	f_Track2	Track2 force encrypt
	f_Track3	Track3 force encrypt
	f_Other	Track3 force encrypt when Card type is 0
Return:	Appendix A	
Example:	MSR_SetEncryptOption(true, true, true,false);	
Function:	MSR_SetHashOption (supported in firmware v1.00.027 and above)	
Description:	Set hash output format of card data in new structure	
Format:	BYTE MSR_SetHashOption(bool f_Track1,bool f_Track2,bool f_Track3)	
Parameter:	f_Track1	Track1 hash will be sent if data is encrypted
	f_Track2	Track2 hash will be sent if data is encrypted
	f_Track3	Track3 hash will be sent if data is encrypted
Return:	Appendix A	
Example:	MSR_SetHashOption(true, true, true);	
Function:	MSR_SetMaskOption (supported in firmware v1.00.027 and above)	
Description:	Set mask output format of card data in new structure	
Format:	BYTE MSR_SetMaskOption(bool f_Track1,bool f_Track2,bool f_Track3)	
Parameter:	f_Track1	Track1 mask data allow to send when encrypted
	f_Track2	Track2 mask data allow to send when encrypted
	f_Track3	Track3 mask data allow to send when encrypted
Return:	Appendix A	
Example:	MSR_SetMaskOption(true,true,true);	
Function:	MSR_SetDataPINorDataKey (supported in firmware v1.00.027 and above)	
Description:	Set the unit card encrypted data output format with PIN key or Data key	
Format:	BYTE MSR_SetDataPINorDataKey(unsigned char d_Format)	
Parameter:	d_Format	0x30: The unit always output clear card data if user don't load MSR DUKPT key, or output encrypted card data with Data key if user had loaded MSR DUKPT key. 0x31: The unit always output clear card data if user don't load MSR DUKPT key, or output encrypted card data with PIN key if user had loaded MSR DUKPT ke
Return:	Appendix A	
Example:	MSR_SetDataPINorDataKey(0x30);	
Function:	MSR_getDataPINorDataKey (supported in firmware v1.00.027 and above)	
Description:	Get the unit card endrypted data output format with PIN key or Data key.	
Format:	BYTE MSR_getDataPINorDataKey(unsigned char *d_Format)	

Parameter:	d_Format	0x30: the unit always output clear card data if user don't load MSR DUKPT key, or output encrypted card data with Data key if user had loaded MSR DUKPT key. 0x31: the unit always output clear card data if user don't load MSR DUKPT key, or output encrypted card data with PIN key if user had loaded MSR DUKPT key.
Return:	Appendix A	
Example:	MSR_getDataPINorDataKey(&d_Format);	
Function:	MSR_ManualInputCardData (supported in firmware v1.00.027 and above)	
Description:	Get manual input card data, only support format of new structure with PIN key, and no LRC to set for every track data.	
Format:	BYTE MSR_ManualInputCardData(unsigned char *CardData, int *Length);	
Parameter:	CardData	The buffer for output ISO/ABA card data format
	Length	The length of output data.
Return:	Appendix A	
Example:	MSR_ManualInputCardData(CardData, &Length);	

PIN API

Function:	PIN_GetPINBlock	
Description:	Get encrypted PIN	
Format:	BYTE PIN_GetPINBlock(char *EncryptedPIN, int *Length,PIN_PARAM p_Param)	
Parameter:	EncryptedPIN	The encrypted PIN string
	Length	The length of encrypted PIN string
	p_Param	Input parameters,See Appendix B for more information
Return:	Appendix A	
Example:	PIN_GetPINBlock(EncryptedPIN, &length,p_Param);	
Function:	PIN_GetEncryptedData	
Description:	Get encrypted data from keypad.	
Format:	BYTE PIN_GetEncryptedData(bool s_Type,unsigned char *EncryptedData, int *Length,PIN_PARAM p_Param);	
Parameter:	s_Type	End_flag = 0 : Not to send back encrypted key entry. Must followed by another Get encrypted data command. End_flag = 1: Final key entry command. Data will be encrypted and sent back
	EncryptedData	All keys entered encrypted by DUKPT 3DES key.When the encrypted data is decrypted, it contains cleat text of all the

		keys entered separated by '/'.Format: <First entered key string> '/' <Second entered key string> '/' ....<Last entered key string>
	Length	The length of encrypted Data string
	p_Param	Input parameters,See Appendix B for more information
Return:	Appendix A	
Example:	PIN_GetEncryptedData(0x01,EncryptedData,&Length，p_Param); 例子	
Function:	PIN_CancelPIN	
Description:	Cancel PIN enter state	
Format:	BYTE PIN_CancelPIN()	
Parameter:	None	
Return:	Appendix A	
Example:	PIN_CancelPIN()	
Function:	PIN_GetNumericOrAmount	
Description:	Get pinpad input as numeric or amount.	
Format:	BYTE PIN_GetNumericOrAmount(BYTE s_Type,char *EncryptedPIN, int *Length,PIN_PARAM p_Param,R_RSA_PRIVATE_KEY PrivateKey)	
Parameter:	s_Type	Output PIN type,it must be 1 or 2 1- numeric pinpad 2- amount pinpad
	EncryptedPIN	The PIN string
	Length	The length of PIN string
	p_Param	Input parameters,See Appendix B for more information
	PrivateKey	Key parameters, See Appendix B for more information
Return:	Appendix A	
Example:	PIN_GetNumericOrAmount(1,EncryptedPIN, &length,p_Param,p_Key);	
Function:	PIN_GetKey	
Description:	Get one Key Pad buffered pressed non-numeric key	
Format:	BYTE PIN_GetKey(char *Key, int *Length)	
Parameter:	Key	The non-numeric key string
	Length	The length of non-numeric key string
Return:	Appendix A	
Example:	PIN_GetKey(Key, &Length)	
Function:	PIN_ClearKey	
Description:	Clear Key Pad buffer	
Format:	BYTE PIN_ClearKey()	

Parameter:	None	
Return:	Appendix A	
Example:	PIN_ClearKey();	
Function:	PIN_GetCardAccount	
Description:	Get card account	
Format:	BYTE PIN_GetCardAccount(unsigned char *EncryptedData, int *Length,PIN_PARAM p_Param)	
Parameter:	EncryptedData	The buffer of Encrypted data, SessionID and KSN SessionID is only used at security level 4 of SecureHead , it is part of the encrypted data KSN is a 10 bytes string, in the case of fix key management, use serial number plus two bytes null characters instead of KSN.
	Length	The length of data
	p_Param	Input parameters,See Appendix B for more information
Return:	Appendix A	
Example:	PIN_GetCardAccount(EncryptedData,&Length,p_Param)	
Function:	PIN_InvalidateKey	
Description:	Make the numeric key and account key invalid.	
Format:	BYTE PIN_InvalidateKey()	
Parameter:	None	
Return:	Appendix A	
Example:	PIN_InvalidateKey()	

Example for DLL call:

```
//include head file
#include "uSign_PayKit.h"
//add Lib(uSign_PayKit.lib):
Add uSign_PayKit.lib to Project->Settings->Link->Object/library
//Call DLL functions using single-thread method:
// Pay_OpenHid
    BYTE res = Pay_OpenHid(0x0ACD,0x2310);
// Pay_Close
    BYTE res = Pay_Close();
// Pay_GetSerialNumber
    char Serial[128];
    m_GetSerial = "";
    int length = 0;
```

```
	BYTE res = Pay_GetSerialNumber(Serial, &length);
	if(res == 1)
	{
		for(int i = 0; i < length; i++)
		{
			m_GetSerial += Serial[i];
		}
	}
	UpdateData(FALSE);
// Pay_SetSerialNumber
	BYTE res = Pay_SetSerialNumber("IDTECH-2010", 11);
// Pay_ControlLED
	BYTE res = Pay_ControlLED(0x02,0x00);
// Pay_GetSdkVersion
	char Serial[128];
	m_GetSerial = "";
	int length = 0;
	BYTE res = Pay_GetSdkVersion(Serial, &length);
// Pay_GenerateTone
	res = Pay_GenerateTone(2000,300);
// Pay_ControlAudio
	BYTE res = Pay_ControlAudio(false);
// Pay_GetVersion
	char Version[128];
	m_Version = "";
	int length = 0;
	BYTE res = Pay_GetVersion(Version, &length);
// uSign_SetPenColor
	BYTE res = uSign_SetPenColor(0x01,0x00,0x00,0x00,0x00,0x00,0xff);
// uSign_DrawLine
	BYTE res = uSign_DrawLine(0,0,319,239);
// uSign_DrawRectangle
	res = uSign_DrawRectangle(10,10,300,200);
// uSign_DrawArc
	res = uSign_DrawArc(1,1,200,0,6000);
// uSign_BrushColor
	res = uSign_BrushColor(0xFF,0xff,0xff);
// uSign_FillRectangle
	res = uSign_FillRectangle(0,0,319,239);
// uSign_FillArc
	res = uSign_FillArc(1,1,200,0,6000);
// uSign_SetFont
```

```
        res = uSign_SetFont(0x10,0x0C,0x10,0x00,0x00,0x03);
//uSign_SetText
        res = uSign_SetTextColor(0,0,0xff);
// uSign_SelectBackgroundMode
        res = uSign_SelectBackgroundMode(0x01);
// uSign_SetBackgroundColor
        res = uSign_SetBackgroundColor(255,255,255);
// uSign_GetPicture
        BYTE res = 0;
        BYTE rec[240000];
        unsigned int rlen = 0;
        res = uSign_GetPicture(0,0,20,20,rec,240000,&rlen);
// uSign_ClearSignature
        p_Count = 0;
        p_Draw = 0;
        m_pArray.RemoveAll();
        Invalidate(TRUE);
        BYTE res = uSign_ClearSignature();
// uSign_ExitCapture
        BYTE res = uSign_ExitCapture();
// uSign_ShowPicture
        BYTE pData[2000];
        BYTE res = uSign_ShowPicture(10,10,20,20, pData,2000);
// uSign_StorePicture
        BYTE pData[3000];
        BYTE res = uSign_StorePicture(1,1,pData,3000);
// uSign_ShowStorePicture
        BYTE res = uSign_ShowStorePicture(1,0,0,319,239);
// PIN_CancelPIN
        BYTE res = PIN_CancelPIN();
// PIN_GetKey
        m_Key = "";
        char key[128];
        int len = 0;
        BYTE res = PIN_GetKey(key,&len);
        for(int i = 0; i < len; i++)
              m_Key += key[i];
        UpdateData(false);
// PIN_ClearKey
        BYTE res = PIN_ClearKey();
// MSR_ArmToRead
        BYTE res = MSR_ArmToRead();
```

```
    if(res == 1)
        GetDlgItem(IDC_GETDATABUF)->EnableWindow(true);
    m_Msr = " ";
    UpdateData(false);
```

// Call DLL functions using multi-threads methods:

```
// uSign_StartCapture(See Sign_PayKitTest demo software for more information)
    void __stdcall point_handle (int *buf, int rev, LPVOID pParam)
    {
        CSign_PayKitTestDlg* pthis = (CSign_PayKitTestDlg*)pParam;
        pthis->p_Draw = 0;
        memset(pthis->p_Array, 0, 50);
        pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
        CPoint p;
        for(int i = 0; i < rev;)
        {
            p.x = buf[i++];
            p.y = buf[i++];
            pthis->m_pArray.Add(p);
            pthis->p_Count++;
            pthis->p_Draw++;
        }
        pthis->ClearRect(false);
        pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
    }
    static UINT ThreadProc_Capture( LPVOID pParam )
  {

    CSign_PayKitTestDlg* pthis = (CSign_PayKitTestDlg*)pParam;
    int res = uSign_StartCapture(0x04,0x7A,0,0,0,255,255,255);
    TRACE("Start capturing result:%d\n", res);
    return 0;
  }

  void CSign_PayKitTestDlg::OnStartcapturing()
  {
    // TODO: Add your control notification handler code here
    uSign_AddPointHandle(point_handle,this);
    AfxBeginThread(ThreadProc_Capture, this);
  }
  //SecureHead data
void __stdcall SecureHead_handle (unsigned char *buf, int rev, LPVOID pParam)
```

```
{
	CSign_PayKitTestDlg* pthis = (CSign_PayKitTestDlg*)pParam;
	pthis->m_Track = "";
	CString str;
	pthis->Parse_TDES_Data(buf,rev);//TDES please see Sign_PayKitTest demo for more
information
}
	static UINT ThreadProc_MSR( LPVOID pParam )
	{

		CSign_PayKitTestDlg* pthis = (CSign_PayKitTestDlg*)pParam;
		int res = 0;
		res = MSR_GetEncryption(&pthis->m_Encrypt);
		res = MSR_EnableSecureHead();



		return 0;
	}
void CSign_PayKitTestDlg::OnGetdatabuf()
{
	// TODO: Add your control notification handler code here
	MSR_AddDataHandle(SecureHead_handle,this);
	AfxBeginThread(ThreadProc_MSR, this);
}
// PIN_GetPINBlock
	static UINT ThreadProc( LPVOID pParam )
	{
			CString temp;
		CSign_PayKitTestDlg* pthis = (CSign_PayKitTestDlg*)pParam;
		R_RSA_PRIVATE_KEY Private_key;

		char EncryptedPIN[256];
		int length = 0;
		BYTE res = 0;
		pthis->m_PIN = "";
		PIN_INPUT p_Input;
		PIN_MSG p_Msg[2];
		PIN_PARAM p_Param;
		//input
		p_Input.f_Height = 0x16;
		p_Input.f_Width = 0x18;
		p_Input.f_Weight = 0x00;
```

```
p_Input.f_Italic = 0x00;
p_Input.f_Underline = 0x00;
p_Input.f_CharSet = 0x05;
p_Input.t_Blue = p_Input.t_Green = p_Input.t_Red = 0x00;
p_Input.b_Mode = 0x01;
p_Input.b_Blue = p_Input.b_Green = p_Input.b_Red = 0xff;
p_Input.x_Start = 5;
p_Input.y_Start = 96;
p_Input.x_End = 310;
p_Input.y_End = 144;
p_Input.Show_Mode = 0x0f;
//msg 1    2
p_Msg[0].f_Height =    0x10;
p_Msg[0].f_Width =    0x10;
p_Msg[1].f_Height = 0x0c;
p_Msg[1].f_Width = 0x0c;
p_Msg[0].f_Weight = p_Msg[1].f_Weight = 0x00;
p_Msg[0].f_Italic = p_Msg[1].f_Italic = 0x00;
p_Msg[0].f_Underline = p_Msg[1].f_Underline = 0x00;
p_Msg[0].f_CharSet = p_Msg[1].f_CharSet = 0x03;
p_Msg[0].t_Red = p_Msg[1].t_Red = 0x00;
p_Msg[0].t_Green = p_Msg[1].t_Green = 0x00;
p_Msg[0].t_Blue = p_Msg[1].t_Blue = 0xff;
p_Msg[0].b_Mode = p_Msg[1].b_Mode = 0x01;

p_Msg[0].b_Red = p_Msg[1].b_Red = 0xff;
p_Msg[0].b_Green = p_Msg[1].b_Green = 0xff;
p_Msg[0].b_Blue = p_Msg[1].b_Blue = 0xff;

p_Msg[0].x_Start = 64;
p_Msg[0].y_Start = 32;

p_Msg[1].x_Start = 6;
p_Msg[1].y_Start = 180;
if(pthis->m_rPIN == 0)
{
      p_Msg[0].s_Message = "Enter PIN:";
}
if(pthis->m_rPIN == 1)
{
      p_Msg[0].s_Message = "Numeric PIN:";
}
```

```
if(pthis->m_rPIN == 2)
{
    p_Msg[0].s_Message = "Amount PIN:";
}
p_Msg[1].s_Message = "Press Enter Key When Done";
p_Param.k_Type = 0x31;
p_Param.k_MaxLen = 0x0c;
p_Param.k_MinLen = 0x04;
p_Param.k_Account = "0123456789123456";
p_Param.LCD_Status = 0x03;
p_Param.b_Red = 0xff;
p_Param.b_Green = 0xff;
p_Param.b_Blue = 0xcc;
p_Param.m_Number = 2;
p_Param.p_Input = p_Input;
p_Param.p_Message[0] = p_Msg[0];
p_Param.p_Message[1] = p_Msg[1];


if(pthis->m_rPIN == 0)
{
    res = PIN_GetPINBlock(EncryptedPIN, &length,p_Param);
}
if(pthis->m_rPIN == 1)
{
    if(pthis->ReadNumericPrivateKeys(&Private_key) == 0)
    {
        AfxMessageBox("Read private keys error!");
        return 0;
    }

    res = PIN_GetNumericOrAmount(1,EncryptedPIN,
&length,p_Param,Private_key);
}
if(pthis->m_rPIN == 2)
{
    if(pthis->ReadNumericPrivateKeys(&Private_key) == 0)
    {
        AfxMessageBox("Read private keys error!");
        return 0;
    }
    res = PIN_GetNumericOrAmount(2,EncryptedPIN,
```

```
&length,p_Param,Private_key);
        }
        if(res == 1)
        {
            for(int j = 0; j < length; j++)
            {
                pthis->m_PIN += EncryptedPIN[j];
            }

            pthis->SendMessage(WM_SWITCH_UPDATE, 0, 0);
            return 0;
        }
        if(res == 108)
            pthis->MessageBox("KEYS_NOT_LOADED");
        return 0;
    }

    void CSign_PayKitTestDlg::OnPin()
    {
        // TODO: Add your control notification handler code here
        UpdateData(true);
        AfxBeginThread(ThreadProc, this);
    }
```

Appendix A Return Value

Return Value	Description
0	FAIL
1	SUCCESS
99	PARAMETER_ERR
100	COMMAND_UNSUPPORTED
101	INVALID_COMMAND
102	COMMAND_PROCESS
103	TIME_OUT
104	NO_DATA_AVAILABLE
105	ACTION_CANCELED
106	ACTION_ABORTED
107	WRONG_KEY_TYPE
108	KEYS_NOT_LOADED
109	DUKPT_OVER
110	KEY_EXIST
200	PORT_ OPENED
201	PORT_CLOSED
202	PIN_MODEL
203	CLEAR_TEXT_MODEL
204	MSR_MODEL
205	USIGN_MODEL
206	KEY_FUNCT_MODEL
207	ENCRYPTED_DATA_MODEL

Appendix B Input Parameter Structure

```
typedef struct {
   unsigned short int bits;
   unsigned char modulus[MAX_RSA_MODULUS_LEN];
   unsigned char exponent[MAX_RSA_MODULUS_LEN];
} R_RSA_PUBLIC_KEY;

//private key include p, q, d mod (p-1), d mod(q-1), q(-1) mod p
typedef struct {
   unsigned short int bits;
   unsigned char modulus[MAX_RSA_MODULUS_LEN];
   unsigned char publicExponent[MAX_RSA_MODULUS_LEN];
   unsigned char exponent[MAX_RSA_MODULUS_LEN];
   unsigned char prime[2][MAX_RSA_PRIME_LEN];
   unsigned char primeExponent[2][MAX_RSA_PRIME_LEN];
   unsigned char coefficient[MAX_RSA_PRIME_LEN];
} R_RSA_PRIVATE_KEY;

//PIN_MSG struct is defined: <Message Length includes self (2 bytes)><><Font(6
//bytes)><Text Color(R G B, total 3 bytes)><Background Mode(1 byte)><Backgroud
//color(R G B, total 3 bytes)> <X(2 bytes)><Y(2 bytes)><String Length(2 bytes)><String>

typedef struct{
      BYTE f_Height;
      BYTE f_Width;
      BYTE f_Weight;
      BYTE f_Italic;
      BYTE f_Underline;
      BYTE f_CharSet;
      BYTE t_Red;
      BYTE t_Green;
      BYTE t_Blue;
      BYTE b_Mode;
      BYTE b_Red;
      BYTE b_Green;
      BYTE b_Blue;
      int x_Start;
      int y_Start;
      char *s_Message;
}PIN_MSG;
```

```
//PIN_INPUT struct is defined: <Font(6 bytes)><Text Color(R G B, total 3
//bytes)><Background Mode(1 byte)><Backgroud color(R G B, total 3 bytes)> <X0 (2
//bytes)><Y0 (2 bytes)><X1 (2 bytes)><Y1 (2 bytes)><Show Mode (1 byte)>
typedef struct{
	BYTE f_Height;
	BYTE f_Width;
	BYTE f_Weight;
	BYTE f_Italic;
	BYTE f_Underline;
	BYTE f_CharSet;
	BYTE t_Red;
	BYTE t_Green;
	BYTE t_Blue;
	BYTE b_Mode;
	BYTE b_Red;
	BYTE b_Green;
	BYTE b_Blue;
	int x_Start;
	int y_Start;
	int x_End;
	int y_End;
	BYTE Show_Mode;
}PIN_INPUT;
typedef struct{
	BYTE k_Type; //'0' - Master key / Session key; '1'- DUKPT
	BYTE k_MaxLen; //the max length of input PIN
	BYTE k_MinLen; //the min length of input PIN
	char *k_Account; //16 ASCII code for digital (0x30 - 0x39) or NULL
	BYTE LCD_Status; //LCD status
	BYTE b_Red; //Background Color
	BYTE b_Green; //Background Color
	BYTE b_Blue; //Background Color
	PIN_INPUT p_Input;
	BYTE m_Number;
	PIN_MSG p_Message[10];
}PIN_PARAM;
```

Appendix C MagStripe Card Data Output Format

Unencrypted MSR Data Output Format

Track 1: <SS1><T_1 Data><ES><CR>*
Track 2: <SS2><T_2 Data><ES><CR>*
Track 3: <SS3><T_3 Data><ES><CR>*

where: SS1(start sentinel track 1) = %
SS2(start sentinel track 2) = ;
SS3(start sentinel track 3) = ; for ISO, ! for CDL, % for AAMVA
ES(end sentinel all tracks) = ?

Start or End Sentinel: Characters in encoding format which come before the first data character (start) and after the last data character (end), indicating the beginning and end, respectively, of data.

Track Separator: A designated character which separates data tracks.

Terminator: A designated character which comes at the end of the last track of data, to separate card reads.

LRC: Check character, following end sentinel.

CDL: Old California Drivers License format.

CR: Carriage Return.

**Note: The <CR> characters (shown above) between tracks 1 & 2 and 2 & 3 denote the default character for this position, the Track Separator position. The <CR> characters shown for track 3 denotes the default character for this position, the Terminator position.*

Unencrypted MSR setting:
0x30: clear text card data with no LRC, '0x0d' at the end of each data track only if it exists (default setting)

0x31: clear text card data with LRC, '0x0d' at the end of each track when the track data does not exist.

Encrypted MSR Data Output Format

Original Encryption Format

<STX><LenL><LenH><Card Data><CheckLRC><CheckSum><ETX>

Where <STX> = 02h, <ETX> = 03h
<LenL><LenH> is a two byte length of <Card Data>.
<CheckLRC> is a one byte Exclusive-OR sum calculated for all <Card Data>.
<CheckSum> is a one byte Sum value calculated for all <Card data>.
<Card Data> card data format is shown below.

ISO/ABA Data Output Format:

- card encoding type (0: ISO/ABA, 4: for Raw Mode)
- track status (bit 0,1,2:T1,2,3 decode, bit 3,4,5:T1,2,3 sampling)
- track 1 unencrypted length (1 byte, 0 for no track1 data)
- track 2 unencrypted length (1 byte, 0 for no track2 data)
- track 3 unencrypted length (1 byte, 0 for no track3 data)
- track 1 masked (Omitted if in Raw mode)
- track 2 masked (Omitted if in Raw mode)
- track 3 data (Omitted if in Raw mode)
- track 1 encrypted (AES/TDES encrypted data)
- track 2 encrypted (AES/TDES encrypted data)
- track 3 encrypted (Only used in Raw mode)
- track 1 hashed (20 bytes SHA1-Xor)
- track 2 hashed (20 bytes SHA1-Xor)
- DUKPT serial number (10 bytes)

Non ISO/ABA Data Output Format

- card encoding type (1: AAMVA, 3: Others)
- track status (bit 0,1,2:T1,2,3 decode, bit 3,4,5:T1,2,3 sampling)
- track 1 length (1 byte, 0 for no track1 data)
- track 2 length (1 byte, 0 for no track2 data)
- track 3 length (1 byte, 0 for no track3 data)
- track 1 data
- track 2 data
- track 3 data

Original Encryption Format Decryption Example

Decryption of a three track ABA card with the original encryption format.
Sign&Pay with default settings

Original encryption format can be recognized because the high bit of the fourth byte underlined (00) is 0.

027D0100 3F48236B252A343236362A2A2A2A2A2A2A2A393939395E42555348204A522F47
454F52474520572E4D525E2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A
A2A2A2A2A2A2A2A2A2A3F2A3B343236362A2A2A2A2A2A2A2A393939393D2A2A2A2A2A2
A2A2A2A2A2A2A2A2A2A2A2A3F2A3B33333333333333333333373637363736303730373037
736373637363333333333333333333337363736373630373037303737363736373633333333333
333333333736373637363037303730373736373637363333333333333333333337363736373637
03730373F32863E9E3DA28E455B28F7736B77E47A64EDDA3BF03A06E44F31D1818C0BC
D7A353FB1AD70EFD30FFC3DA08A4FBC9372E57E8B40848BAEAA3FE724B3550E2F4B2
23E6BF264BEAE9E39142B648CDB51FB8DAF8EA5B63913D29419B67582FCCCE9B37266
0F03668CC453216D9449C6B67EF33418AC88F65E1DB7ED4D10973F99DFC8463FF6DF113
B6226C4898A9D355057ECAF11A5598F02CA3162994901190000000001399F03

STX, Length (LSB, MSB), card type, track status, length track 1, length track 2, length track 3
02 7D01 00 3F 48 23 6B

The above broken down and interpreted
02—STX character
7D—low byte of total length
01—high byte of total length
00—card type byte (interpretation old format ABA card)
3F—3 tracks of data all good
48—length of track 1
23—length of track 2
6B—length of track 3

Track 1 data masked (length 0x48)
252A343236362A2A2A2A2A2A2A2A393939395E42555348204A522F47454F52474520572E
4D525E2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A
2A2A3F2A

Track 2 data in hex masked (length 0x23)
3B343236362A2A2A2A2A2A2A2A393939393D2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A3
F2A

Track 3 data unencrypted (length 0x6B)
3B3333333333333333333373637363736303730373037373637363736333333333333333333
337363736373630373037303737363736373633333333333333333333373637363736303730373
037373637363736333333333333333333333373637363736303730373F32

Track 1 & 2 encrypted length 0x48+0x23 rounded up to 8 bytes =0x6B -> 0x70 (112 decimal)
863E9E3DA28E455B28F7736B77E47A64EDDA3BF03A06E44F31D1818C0BCD7A35
3FB1AD70EFD30FFC3DA08A4FBC9372E57E8B40848BAEAA3FE724B3550E2F4B22
3E6BF264BEAE9E39142B648CDB51FB8DAF8EA5B63913D29419B67582FCCCE9B3
72660F03668CC453216D9449C6B67EF3

Track 1 hashed
3418AC88F65E1DB7ED4D10973F99DFC8463FF6DF

Track 2 hashed
113B6226C4898A9D355057ECAF11A5598F02CA31

KSN
62994901190000000001

LRC, checksum and ETX
39 9F 03

Masked Data:
Track 1 data masked in ASCII:
%*4266********9999^BUSH JR/GEORGE W.MR^********************************?*
Track 2 data masked in ASCII:
;4266********9999=****************?*
Track 3 data unencrypted in ASCII:
;333333333767676070707767676333333333376767607070776767633333333337676760707
7767676333333333376767607072

Key Value: F8 2A 7A 0D 7C 67 46 F1 96 18 9A FB 54 2C 65 A3
KSN: 62 99 49 01 19 00 00 00 00 01

Decrypted Data in ASCII:
%B4266841088889999^BUSH JR/GEORGE
W.MR^08091011000011000000000046000000?!;4266841088889999=0809101100000046?0
;333333333767676070707767676333333333376767607070776767633333333337676760707

776767633333333376767607 07?2

Decrypted Data in Hex:
2542343236363834313038383838393939395E42555348204A522F47454F52474520572E4D52
5E3038303931303131303030303131303030303030303030343630303030303F213B3432363
6383431303838383839393939 3D3038303931303131303030303034363F300000000000

Enhanced Encryption Format

This mode is used when all tracks must be encrypted, or encrypted OPOS support is required, or when the tracks must be encrypted separately or when cards other than type 0 (ABA bank cards) must be encrypted or when track 3 must be encrypted. This format is the standard encryption format, but not yet the default encryption format.

Card data is sent out in the following format
<STX><LenL><LenH><Card Data><CheckLRC><CheckSum><ETX>

0	STX
1	Data Length low byte
2	Data Length high byte
3	Card Encode Type[1]
4	Track 1-3 Status[2]
5	Track 1 data length
6	Track 2 data length
7	Track 3 data length
8	Clear/masked data sent status [3]
9	Encrypted/Hash data sent status [4]
10	Track 1 clear/mask data
	Track 2 clear/mask data
	Track 3 clear/mask data
	Track 1 encrypted data
	Track 2 encrypted data
	Track 3 encrypted data
	Session ID (8 bytes) (Security level 4 only)
	Track 1 hashed (20 bytes each) (if encrypted and hash track 1 allowed)
	Track 2 hashed (20 bytes each) (if encrypted and hash track 2 allowed)
	Track 3 hashed (20 bytes each) (if encrypted and hash track 3 allowed)
	KSN (10 bytes)
	CheckLRC
	CheckSum
	ETX

Where <STX> = 02h, <ETX> = 03h

Note 1 : Card Encode Type

Card Type will be 8x for enhanced encryption format and 0x for original encryption format

Value	Encode Type Description
00h / 80h	ISO/ABA format
01h / 81h	AAMVA format
03h / 83h	Other
04h / 84h	Raw; un-decoded format

For Type 04 or 84 Raw data format, all tracks are encrypted and no mask data is sent. No track indicator '01', '02' or '03' in front of each track. Track indicator '01','02' and '03' will still exist for non-encrypted mode.

Note 2: Track 1-3 status byte

Field 4:
Bit 0: 1— track 1 decoded data present
Bit 1: 1— track 2 decoded data present
Bit 2: 1— track 3 decoded data present
Bit 3: 1— track 1 sampling data present
Bit 4: 1— track 2 sampling data present
Bit 5: 1— track 3 sampling data present
Bit 6, 7 — Reserved for future use

Note 3: Clear/mask data sent status

Field 8 (Clear/mask data sent status) and field 9 (Encrypted/Hash data sent status) will only be sent out in enhanced encryption format.

Field 8: Clear/masked data sent status byte:
Bit 0: 1 —track 1 clear/mask data present
Bit 1: 1— track 2 clear/mask data present
Bit 2: 1— track 3 clear/mask data present
Bit 3: 0— reserved for future use
Bit 4: 0— reserved for future use
Bit 5: 0— reserved for future use

Note 4: Encrypted/Hash data sent status

Field 9: Encrypted data sent status
Bit 0: 1— track 1 encrypted data present
Bit 1: 1— track 2 encrypted data present
Bit 2: 1— track 3 encrypted data present
Bit 3: 1— track 1 hash data present

Bit 4: 1— track 2 hash data present

Bit 5: 1— track 3 hash data present

Bit 6: 1—session ID present

Bit 7: 1—KSN present

Enhanced Encryption Format Decryption Example

Example of decryption of a three track ABA card with the enhanced encryption format.
Sign&Pay with default settings except enhanced encryption structure format.

Enhanced encryption Format (this can be recognized because the high bit of the fourth byte underlined (80) is 1.

029801<u>80</u>3F48236B03BF252A343236362A2A2A2A2A2A2A2A393939395E42555348204A52
2F47454F52474520572E4D525E2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2
A2A2A2A2A2A2A2A2A2A2A2A3F2A3B343236362A2A2A2A2A2A2A2A393939393D2A2A2
A2A2A2A2A2A2A2A2A2A2A2A2A3F2ADA7F2A52BD3F6DD8B96C50FC39C7E6AF22F06
ED1F033BE0FB23D6BD33DC5A1F808512F7AE18D47A60CC3F4559B1B093563BE7E0745
9072ABF8FAAB5338C6CC8815FF87797AE3A7BEAB3B10A3FBC230FBFB941FAC9E8264
9981AE79F2632156E775A06AEDAFAF6F0A184318C5209E55AD44A9CCF6A78AC240F79
1B63284E15B4019102BA6C505814B585816CA3C2D2F42A99B1B9773EF1B116E005B7CD
8681860D174E6AD316A0ECDBC687115FC89360AEE7E430140A7B791589CCAADB6D687
2B78433C3A25DA9DDAE83F12FEFAB530CE405B701131D2FBAAD970248A45600093341
8AC88F65E1DB7ED4D10973F99DFC8463FF6DF113B6226C4898A9D355057ECAF11A5598
F02CA31688861C157C1CE2E0F72CE0F3BB598A614EAABB16299490119000000000206E2
03

STX, Length(LSB, MSB), card type, track status, length track 1, length track 2, length track 3
02 9801 80 3F 48-23-6B 03BF

The above broken down and interpreted
02—STX character
98—low byte of total length
01—high byte of total length
80—card type byte (interpretation new format ABA card)
3F—3 tracks of data all good
48—length of track 1
23—length of track 2
6B—length of track 3
03—tracks 1 and 2 have masked/clear data
BF—bit 7=1—KSN included
Bit 6=0—no Session ID included so not level 4 encryption
Bit 5=1—track 3 hash data present
Bit 4=1—track 2 hash data present
Bit 3-1—track 1 hash data present

Bit 2=1—track 3 encrypted data present
Bit 1=1—track 2 encrypted data present
Bit 0=1—track 1 encrypted data present

Track 1 data masked (length 0x48)
252A343236362A2A2A2A2A2A2A2A393939395E42555348204A522F47454F52474520572E
4D525E2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A
2A2A3F2A

Track 1 masked data in ASCII
%*4266********9999^BUSH JR/GEORGE W.MR^********************************?*

Track 2 data in hex masked (length 0x23)
3B343236362A2A2A2A2A2A2A2A393939393D2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A2A3
F2A

Track2 masked data in ASCII
;4266********9999=****************?*

In this example there is no Track 3 data either clear or masked (encrypted and hashed data is below)

Track 1 encrypted length 0x48 rounded up to 8 bytes = 0x48 (72 decimal)
DA7F2A52BD3F6DD8B96C50FC39C7E6AF22F06ED1F033BE0FB23D6BD33DC5A1F8
08512F7AE18D47A60CC3F4559B1B093563BE7E07459072ABF8FAAB5338C6CC88
15FF87797AE3A7BE

Track 2 encrypted length 0x32 rounded up to 8 bytes =0x38 (56 decimal)
AB3B10A3FBC230FBFB941FAC9E82649981AE79F2632156E775A06AEDAFAF6F0A
184318C5209E55AD

Track 3 encrypted length 0x6B rounded up to 8 bytes =0x70 (64 decimal)
44A9CCF6A78AC240F791B63284E15B4019102BA6C505814B585816CA3C2D2F42
A99B1B9773EF1B116E005B7CD8681860D174E6AD316A0ECDBC687115FC89360A
EE7E430140A7B791589CCAADB6D6872B78433C3A25DA9DDAE83F12FEFAB530CE
405B701131D2FBAAD970248A45600093

Track 1 data hashed length 20 bytes
3418AC88F65E1DB7ED4D10973F99DFC8463FF6DF

Track 2 data hashed length 20 bytes
113B6226C4898A9D355057ECAF11A5598F02CA31

Track 3 data hashed length 20 bytes
688861C157C1CE2E0F72CE0F3BB598A614EAABB1

KSN length 10 bytes
62994901190000000002

LCR, check sum and ETX
06E203

Clear/Masked Data in ASCII:
Track 1: %*4266********9999^BUSH JR/GEORGE
W.MR^********************************?*
Track 2: ;4266********9999=****************?*

Key Value: 1A 99 4C 3E 09 D9 AC EF 3E A9 BD 43 81 EF A3 34
KSN: 62 99 49 01 19 00 00 00 00 02

Decrypted Data:
Track 1 decrypted
%B4266841088889999^BUSH JR/GEORGE W.MR^080910110000110000000046000000?!
Track 2 decrypted
;4266841088889999=08091011000046?0
Track 3 decrypted
;3333333337676760707077676763333333337676760707077676763333333337676760707077676763333333333376760707?2

Track 1 decrypted data in hex including padding zeros (but there are no pad bytes here)
2542343236363834313038383838393939395E42555348204A522F47454F52474520572E4D525E3038303931303131303030303131303030303030303030343630303030303F21

Track 2 decrypted data in hex including padding zeros
3B34323636383431303838383839393939D3038303931303131303030303034363F300000000000

Track 3 decrypted data in hex including padding zeros
3B33333333333333333333373637363736303730373037373637363736333333333333333333373637363736303730373037373637363736333333333333333333337363736373630373037373637363736333333333333333333337363736373630373037373F320000000000