HMIWorks
API Reference

Version 1.15, July 2012

Notices

Warranty

All products manufactured by ICP DAS are under warranty regarding defective materials for a period of one year, beginning from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for any damage resulting from the use of this product.ICP DAS reserves the right to change this manual at any time without notice.The information furnished by ICP DAS is believed to be accurate and reliable.However, no responsibility is assumed by ICP DAS for its use, not for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only may be registered trademarks of their respective companies.

TABLE OF CONTENTS

1. GEOMETRY API

This chapter describes how to draw different shapes, such as rectangles, circles, etc.

1.1. HMI_DRAWLINE

Draw a line on TouchPAD.

Syntax

```
void hmi_DrawLine(
    tContext *pContext,
    int x1,
    int y1,
    int x2,
    int y2
);
```

Parameter

pContext

[out] Specify the context.

x1

[in] The x-coordinate of the first vertex of the line segment to draw

y1

[in] The y-coordinate of the first vertex of the line segment to draw

x2

[in] The x-coordinate of the second vertex of the line segment to draw

y2

[in] The y-coordinate of the second vertex of the line segment to draw

Return Values

None.

Examples

[C]

```
hmi_DrawLine (pContext, x1, y1, x2, y2);
```

Remark

None

1.2. HMI_DRAWRECT

Draw a rectangle on TouchPAD.

Syntax

```
void hmi_DrawRect(
    tContext *pContext,
    int x1,
    int y1,
    int x2,
    int y2
);
```

Parameter

pContext

[out] Specify the context.

x1

[in] The x-coordinate of the first diagonal vertex of the rectangle to draw

y1

[in] The y-coordinate of the first diagonal vertex of the rectangle to draw

x2

[in] The x-coordinate of the second diagonal vertex of the rectangle to draw

y2

[in] The y-coordinate of the second diagonal vertex of the rectangle to draw

Return Values

None.

Examples

[C]

```
hmi_DrawRect (pContext, x1, y1, x2, y2);
```

Remark

None

1.3. HMI_FILLRECT

Draw a rectangle and fill it with a specified color on TouchPAD.

Syntax

```
void hmi_FillRect(
    tContext *pContext,
    int x1,
    int y1,
    int x2,
    int y2
);
```

Parameter

pContext

[out] Specify the context.

x1

[in] The x-coordinate of the first diagonal vertex of the rectangle to fill

y1

[in] The y-coordinate of the first diagonal vertex of the rectangle to fill

x2

[in] The x-coordinate of the second diagonal vertex of the rectangle to fill

y2

[in] The y-coordinate of the second diagonal vertex of the rectangle to fill

Return Values

None

Examples

[C]

```
void HotSpot4OnClick(tWidget *pWidget)
{
    static int count = 0;
    unsigned long color;

    count++;
    color = ClrRed >> ((count %3) * 8);
    // where ClrRed is defined in grlib.h

    // Here, thisContext means the scope of the whole screen
    hmi_SetForeground(thisContext, color);
    hmi_FillRect(thisContext, 0, 0, 320, 240);
}
```

Remark

None

1.4. HMI_DRAWCIRCLE

Draw a circle on TouchPAD.

Syntax

```
void hmi_DrawCircle(
        tContext *pContext,
        int x,
        int y,
        int w
);
```

Parameter

pContext

[out] Specify the context.

x

[in] The x-coordinate of the center of the circle to draw

y

[in] The y-coordinate of the center of the circle to draw

w

[in] The radius of the circle to draw

Return Values

None

Examples

[C]

```
hmi_DrawCircle (pContext, x, y, w);
```

Remark

None

1.5. HMI_FILLCIRCLE

Draw a circle and fill it with a specified color on TouchPAD.

Syntax

```
void hmi_FillCircle(
        tContext *pContext,
        int x,
        int y,
        int w
);
```

Parameter

pContext

[out] Specify the context.

x

[in] The x-coordinate of the center of the circle to fill

y

[in] The y-coordinate of the center of the circle to fill

w

[in] The radius of the circle to fill

Return Values

None

Examples

[C]

```
hmi_FillCircle (pContext, x, y, w);
```

Remark

None

1.6. HMI_DRAWELLIPSE

Draw an ellipse on TouchPAD.

Syntax

```
void hmi_DrawEllipse(
    tContext *pContext,
    int x1,
    int y1,
    int x2,
    int y2
);
```

Parameter

pContext

[out] Specify the context.

x1

[in] The x-coordinate of the first diagonal vertex of the rectangular that inscribes the ellipse to draw

y1

[in] The y-coordinate of the first diagonal vertex of the rectangular that inscribes the ellipse to draw

x2

[in] The x-coordinate of the second diagonal vertex of the rectangular that inscribes the ellipse to draw

y2

[in] The y-coordinate of the second diagonal vertex of the rectangular that inscribes the ellipse to draw

Return Values

None

Examples

[C]

```
hmi_DrawEllipse(pContext, x1, y1, x2, y2);
```

Remark

None

1.7. HMI_FillEllipse

Draw an ellipse and fill it with a specified color on TouchPAD.

Syntax

```
void hmi_FillEllipse(
    tContext *pContext,
    int x1,
    int y1,
    int x2,
    int y2
);
```

Parameter

pContext

[out] Specify the context.

x1

[in] The x-coordinate of the first diagonal vertex of the rectangular that inscribes the ellipse to fill

y1

[in] The y-coordinate of the first diagonal vertex of the rectangular that inscribes the ellipse to fill

x2

[in] The x-coordinate of the second diagonal vertex of the rectangular that inscribes the ellipse to fill

y2

[in] The y-coordinate of the second diagonal vertex of the rectangular that inscribes the ellipse to fill

Return Values

None.

Examples

[C]

```
hmi_FillEllipse (pContext, x1, y1, x2, y2);
```

Remark

None

1.8. HMI_DRAWPOLYLINE

Draw a polyline on TouchPAD.

Syntax

```
void hmi_DrawPolyLine(
        tContext *pContext,
        int x,
        int y,
        const int n,
        const short coordinates[]
);
```

Parameter

pContext

[out] Specify the context

x

[in] The x-coordinate of the reference point to which all the vertices of the polyline refer

y

[in] The y-coordinate of the reference point to which all the vertices of the polyline refer

n

[in] The number of vertices of the polygon to draw

coordinates

[in] The array of coordinates of the polygon to draw

Return Values

None.

Examples

[C]

```
hmi_DrawPolyLine (pContext, n, coordinates);
```

Remark

None

1.9. HMI_DRAWPOLYGON

Draw a polygon on TouchPAD.

Syntax

```
void hmi_DrawPolygon(
        tContext *pContext,
        int x,
        int y,
        const int n,
        const short coordinates[]
);
```

Parameter

pContext

[out] Specify the context

x

[in] The x-coordinate of the reference point to which all the vertices of the polygon refer

y

[in] The y-coordinate of the reference point to which all the vertices of the polygon refer

n

[in] The number of vertices of the polygon to draw

coordinates

[in] The array of coordinates of the polygon to draw

Return Values

None.

Examples

[C]

```
hmi_DrawPolygon (pContext, n, coordinates);
```

Remark

None

1.10. HMI_FILLPOLYGON

Draw a polygon and fill it with a specified color on TouchPAD.

Syntax

```
void hmi_FillPolygon(
        tContext *pContext,
        int x,
        int y,
        const int n,
        const short coordinates[]
);
```

Parameter

pContext

[out] Specify the context.

x

[in] The x-coordinate of the reference point to which all the vertices of the polygon refer

y

[in] The y-coordinate of the reference point to which all the vertices of the polygon refer

n

[in] The number of vertices of the polygon to fill

coordinates

[in] The array of coordinates of the polygon to fill

Return Values

None.

Examples

[C]

```
hmi_FillPolygon (pContext, n, coordinates);
```

Remark

None

1.11. HMI_SETFOREGROUND

Set the foreground color.

Syntax

```
void hmi_SetForeground(
        tContext *pContext,
        unsigned long color
);
```

Parameter

pContext

[out] Specify the context.

color

[in] Specify the color. A color is represented by a three-byte integer. The order of the bytes from the most significant byte to the least is Red, Green, Blue. For example, 0x0000FF represents Blue.

Return Values

None

Examples

[C]

```
int flag = 0;

void BitButton5OnClick(tWidget *pWidget)
{
   flag ++;
   WidgetPaint((tWidget*) &PaintBox4);
}
```

```
void PaintBox4OnPaint(tWidget *pWidget, tContext *pContext)
{
  if(flag % 2)
  {
    hmi_SetForeground(pContext, 0x0000FF);    //blue; R-G-B
    hmi_FillRect(pContext,
                 WidgetLeft(pWidget) -10,
                 WidgetTop(pWidget) -10,
                 WidgetRight(pWidget) -10,
                 WidgetBottom(pWidget) -10);
  }
  else
  {
    //piant a white rectangle to clear the PaintBox
    hmi_SetForeground(pContext, 0xFFFFFF);
    hmi_FillRect(pContext,
                 WidgetLeft(pWidget),
                 WidgetTop(pWidget),
                 WidgetRight(pWidget),
                 WidgetBottom(pWidget));
  }
}
```

Remark

Four macros used in the above example are described below:

1. WidgetLeft gets the x coordinate of the left-top vertex of the widget.
2. WidgetTop gets the y coordinate of the left-top vertex of the widget.
3. WidgetRight gets the x coordinate of the right-bottom vertex of the widget.
4. WidgetBottom gets the y coordinate of the right-bottom vertex of the widget.

2. FRAME API

TouchPAD supports multi-frame feature.

The chapter provides APIs to handle multi-frame functions.

2.1. HMI_GOTOFRAMEBYNAME

Goto the frame by specified name.

Syntax

```
void hmi_GotoFrameByName(
        const char *frame_name
);
```

Parameter

frame_name

[out] Specify the name of frame to goto.

Return Values

None.

Examples

[C]

```
hmi_GotoFrameByName (frame_name);
```

Remark

None

2.2. HMI_INDEXOFFRAMENAME

Get the index of a frame by specified frame name.

Syntax

```
int hmi_IndexOfFrameName(
        const char *frame_name
);
```

Parameter

frame_name

[out] Specify the frame name.

Return Values

Index of the frame with its name specified.

Examples

[C]

```
int index = hmi_IndexOfFrameName(frame_name);
```

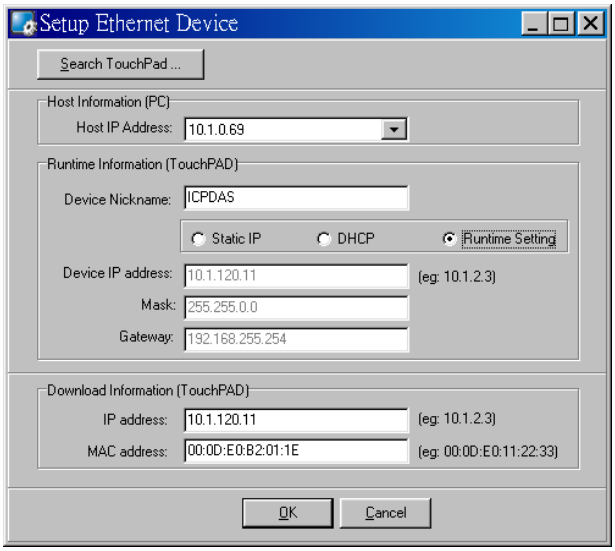
Remark

None

3. NETWORK CONFIGURATION API

This chapter provides network configuration APIs.

Be sure to set up the TouchPAD to have the Ethernet setting as “Runtime Setting” as shown below. Note that configured “Static IP” or “DHCP” disables the network configuration API functions.



3.1. HMI_NETWORKPARAMSGET

Get the network configuration of the TouchPAD from the parameter area in the internal flash.

Syntax

```
int hmi_NetworkParamsGet(
        unsigned long* ulDHCP,
        unsigned long* ulIP,
        unsigned long* ulMask,
        unsigned long* ulGateway
);
```

Parameter

ulDHCP

[out] Specify the pointer to the variable of the DHCP setting.

ulDHCP	0	1
IP Type	Static IP	DHCP Enabled

ulIP

[out] Specify the pointer to the variable of the IP address.

ulMask

[out] Specify the pointer to the variable of the subnet mask.

ulGateway

[out] Specify the pointer to the variable of gateway address.

Return Values

Always 1 (TRUE)

Examples

[C]

```
#define MAX_FUNS            6
unsigned long gulValue[MAX_FUNS] = {0};
// Funs: Read IP, DHCP, IP, Mask, Gateway, NetID
static char gszIP[BUF_SIZE]; // size: at least 16

void FConfig2OnShow()
{
    // Get DHCP, IP, Mask, Gateway
    hmi_NetworkParamsGet(gulValue +1, gulValue +2, gulValue +3,
gulValue +4);
    gulValue[5] = hmi_NetIDParamsGet();

    // Display the Gateway Address for example
    hmi_IPToStr(gulValue[4], gszIP);
    LabelTextSet(&lbValue24, gszIP);
}
```

Remark

None

3.2. HMI_NETWORKPARAMSSET

Set the network configuration of the TouchPAD to the parameter area in the internal flash.

Syntax

```
int hmi_NetworkParamsSet(
        unsigned long ulDHCP,
        unsigned long ulIP,
        unsigned long ulMask,
        unsigned long ulGateway
);
```

Parameter

ulDHCP

[in] Specify the variable of the DHCP setting.

ulDHCP	0	1
IP Type	Static IP	DHCP Enabled

ulIP

[in] Specify the variable of the IP address.

ulMask

[in] Specify the variable of the subnet mask.

ulGateway

[in] Specify the variable of gateway address.

Return Values

Always 1 (TRUE)

Examples

[C]

```
#define MAX_FUNS            6
unsigned long gulValue[MAX_FUNS] = {0};
// Funs: Read IP, DHCP, IP, Mask, Gateway, NetID

void btnSaveAll25OnClick(tWidget *pWidget)
{
    gulValue[1] = 0;    // DHCP=0, that is, static IP
    gulValue[2] = hmi_StrToIP("10.1.0.59");    // IP
    gulValue[3] = hmi_StrToIP("255.255.0.0");    // Mask
    gulValue[4] = hmi_StrToIP("10.1.0.254");    // Gateway
    gulValue[4] = 1;    // Net ID

    // Save DHCP, IP, Mask, Gateway
    hmi_NetworkParamsSet(gulValue[1], gulValue[2], gulValue[3],
gulValue[4]);

    // Save Net ID
    hmi_NetIDParamsSet(gulValue[5]);
}
```

Remark

None

3.3. HMI_LOCALIPADDRGET

Get the current used IP address.

Syntax

```
unsigned long hmi_LocalIPAddrGet();
```

Parameter

None

Return Values

The current used IP address

Examples

[C]

```
#define BUF_SIZE    22
static char gszIP[BUF_SIZE]; // size: at least 16
unsigned long ipaddr = 0;

void FConfig2OnShow()
{
    // Read the current IP address and display it
    ipaddr = hmi_LocalIPAddrGet();
    hmi_IPToStr(ipaddr, gszIP);
    LabelTextSet(&lbValue24, gszIP);
}
```

Remark

None

3.4. HMI_IPToStr

Convert the IP address of integer to the IP address string.

Syntax

```
int hmi_IPToStr(
        unsigned long ulIP,
        char szIP[]
);
```

Parameter

ulIP

[in] Specify the variable of the IP address integer.

szIP

[out] Specify the pointer to the string of the IP address. (16 bytes including null terminator char)

Return Values

Reserved for future use

Examples

[C]

```
#define MAX_FUNS            6
unsigned long gulValue[MAX_FUNS] = {0};
// Funs: Read IP, DHCP, IP, Mask, Gateway, NetID
static char gszIP[BUF_SIZE]; // size: at least 16

void FConfig2OnShow()
{
```

```
    // Get DHCP, IP, Mask, Gateway
    hmi_NetworkParamsGet(gulValue +1, gulValue +2, gulValue +3,
gulValue +4);
    gulValue[5] = hmi_NetIDParamsGet();

    // Display the Gateway Address for example
    hmi_IPToStr(gulValue[4], gszIP);
    LabelTextSet(&lbValue24, gszIP);
}
```

Remark

None

3.5. HMI_StrToIP

Convert the IP address string to the IP address of integer.

Syntax

```
unsigned long hmi_StrToIP(
        char szIP[]
);
```

Parameter

szIP

[out] Specify the pointer to the string of the IP address. (16 bytes including null terminator char)

Return Values

The IP address of integer

Examples

[C]

```
#define MAX_FUNS            6
unsigned long gulValue[MAX_FUNS] = {0};
// Funs: Read IP, DHCP, IP, Mask, Gateway, NetID

void btnSaveAll25OnClick(tWidget *pWidget)
{
    gulValue[1] = 0;    // DHCP=0, that is, static IP
    gulValue[2] = hmi_StrToIP("10.1.0.59");    // IP
    gulValue[3] = hmi_StrToIP("255.255.0.0");    // Mask
    gulValue[4] = hmi_StrToIP("10.1.0.254");    // Gateway
```

```
    gulValue[4] = 1;    // Net ID

    // Save DHCP, IP, Mask, Gateway
    hmi_NetworkParamsSet(gulValue[1], gulValue[2], gulValue[3],
gulValue[4]);

    // Save Net ID
    hmi_NetIDParamsSet(gulValue[5]);
}
```

Remark

None

4. TCP API

This chapter provides TCP APIs.

4.1. HMI_TCPNEW

Allocate a TCP session if possible.

Syntax

```
tHandle hmi_TCPNew();
```

Parameter

None

Return Values

typedef int tHandle;
If successful, a handle for the session is returned.
If not successful, -1 is returned.

Examples

[C]

```
tHandle h = hmi_TCPNew();
```

Remark

None

4.2. HMI_TCPLISTEN

This function establishes a TCP session for listening as a server.

Syntax

```
void hmi_TCPListen(
        tHandle h,
        unsigned short usPort
);
```

Parameter

h

[in] Specify the handle to the TCP session

usPort

[in] Specify the port of the TCP communication to listen.

Return Values

None

Examples

[C]

```
unsigned short port_listen = 10000;
tHandle h = hmi_TCPNew();
hmi_TCPListen(h, port_listen);
```

Remark

None

4.3. HMI_TCPOPEN

This function is used in a client to establish a TCP session for connecting to a server.

Syntax

```
void hmi_TCPOpen (
    tHandle h,
    unsigned long ulIPAddr,
    unsigned short usRemotePort,
    unsigned short usLocalPort
);
```

Parameter

h

[in] Specify the handle to the TCP session

ulIPAddr

[in] Specify the IP address of the server to connect.

usRemotePort

[in] Specify the remote listening port of the server.

usLocalPort

[in] Specify any value which is greater than zero. This is reserved for future use.

Return Values

None

Examples

[C]

```
unsigned short remote_port = 10000;
```

```
unsigned short local_port = 10001;

tHandle h = hmi_TCPNew();
hmi_TCPOpen(h, TCP_IPADDR(10,1,0,45), remote_port, local_port);
```

Remark

None

4.4. HMI_TCPCLOSE

This function closes and deallocates a TCP session.

Syntax

```
void hmi_TCPClose (
      tHandle h
);
```

Parameter

h

[in] Specify the handle to the TCP session

Return Values

None

Examples

[C]

```
tHandle h = hmi_TCPNew();
…
hmi_TCPClose (h);
```

Remark

None

4.5. HMI_TCPGETLOCALPORT

This function gets local port of the TCP session.
If operating as a server, the local port is the listening port. If operating as a client, the local port is meaningless, and reserved for future use.

Syntax

```
unsigned short hmi_TCPGetLocalPort (
        tHandle h
);
```

Parameter

h

[in] Specify the handle to the TCP session

Return Values

The local port of the TCP session

Examples

[C]

```
tHandle h = hmi_TCPNew();
…
unsigned short local_port = hmi_TCPGetLocalPort (h);
```

Remark

None

4.6. HMI_TCPGETREMOTEPORT

This function gets remote port of the TCP session.
If operating as a server, the remote port is 0. If operating as a client, the remote port is the port that the server uses for connection.

Syntax

```
unsigned short hmi_TCPGetRemotePort (
    tHandle h
);
```

Parameter

h

[in] Specify the handle to the TCP session

Return Values

The remote port of the TCP session

Examples

[C]

```
tHandle h = hmi_TCPNew();
…
unsigned short remote_port = hmi_TCPGetRemotePort (h);
```

Remark

None

4.7. HMI_TCPSTATE

This function gets the state of the TCP session.

Syntax

```
int hmi_TCPState (
        tHandle h
);
```

Parameter

h

[in] Specify the handle to the TCP session

Return Values

The state of the TCP session

STATE_TCP_IDLE	0
STATE_TCP_LISTEN	1
STATE_TCP_CONNECTING	2
STATE_TCP_CONNECTED	3

Examples

[C]

```
tHandle h = hmi_TCPNew();
…
int state = hmi_TCPState (h);
```

Remark

None

4.8. HMI_TCPWRITE

This function writes data through a TCP session. Actually hmi_TCPWrite puts data to the queue for next flush to output to the destination.

Syntax

```
void hmi_TCPWrite(
        tHandle h,
        unsigned char data[],
        int len
);
```

Parameter

h

[in] Specify the handle to the TCP session

data

[in] Specify the array to write

len

[in] Specify the length of the array to write

Return Values

None

Examples

[C]

```
unsigned short remote_port = 10000;
unsigned short local_port = 10001;
unsigned char data[1024];
```

```
tHandle h = hmi_TCPNew();
hmi_TCPOpen(h, TCP_IPADDR(10,1,0,45), remote_port, local_port);
hmi_TCPWrite(h, data, 1024);
```

Remark

None

4.9. HMI_TCPOUTPUT

This function writes data through a TCP session to the destination immediately (no waiting in the queue).

Syntax

```
void hmi_TCPOutput (
        tHandle h,
        unsigned char data[],
        int len
);
```

Parameter

h

[in] Specify the handle to the TCP session

data

[in] Specify the array to write

len

[in] Specify the length of the array to write

Return Values

None

Examples

[C]

```
unsigned short remote_port = 10000;
unsigned short local_port = 10001;
unsigned char data[1024];
```

```
tHandle h = hmi_TCPNew();
hmi_TCPOpen(h, TCP_IPADDR(10,1,0,45), remote_port, local_port);
hmi_TCPOutput (h, data, 1024);
```

Remark

None

4.10. HMI_TCPREADEX

This function reads data through a TCP session.

Syntax

```
int hmi_TCPReadEx (
        tHandle h,
        unsigned char data[],
        int len,
        DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the TCP session

data

[out] Specify the pointer to the array for the reading data

len

[in] Specify the length of the array to read

timeout

[in] Specify the timeout value of the TCP communications

Return Values

The length of the receiving data

Examples

[C]

```
unsigned short remote_port = 10000;
```

```
unsigned short local_port = 10001;
unsigned char data[1024];

tHandle h = hmi_TCPNew();
hmi_TCPOpen(h, TCP_IPADDR(10,1,0,45), remote_port, local_port);
int length_received = hmi_TCPReadEx (h, data, 1024);
```

Remark

Backward Compatibility:
The old version for TCPRead is as below:
int hmi_TCPRead(tHandle h, unsigned char *buf, int buf_len);

It reads the TCP data without waiting, that is, returns immediately.
It is the same as the hmi_TCPReadEx with timeout = 0.

4.11. HMI_TCPSENDCMDEX

This function sends data and then receives data through a TCP session.

Syntax

```
int hmi_TCPSendCmdEx(
        tHandle h,
        unsigned char *send_data,
        int send_data_len,
        unsigned char *receive_data,
        int recv_data_len,
        DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the TCP session

send_data

[out] Specify the pointer to the sending data array

send_data_len

[in] Specify the sending data len

receive_data

[out] Specify the pointer to the receiving data array

recv_data_len

[in] Specify the length of the recv data len

timeout

[in] Specify the timeout value for the TCP communications

Return Values

The length of the receiving data

Examples

[C]

```
unsigned char send_data[1024];
unsigned char receive_data[1024];

tHandle h = hmi_TCPNew();
hmi_TCPOpen(h, TCP_IPADDR(10,1,0,45), remote_port, local_port);
int length_received = hmi_TCPSendCmdEx (h, send_data, 1024,
receive_data, 1024);
```

Remark

Backward Compatibility:
The old version for TCPSendCmd is as below:
int hmi_TCPSendCmd(tHandle h, unsigned char *send_data, int send_len, unsigned char *recv_data, int recv_len);

It is the same as the hmi_TCPSendCmdEx with timeout = 200.

4.12. HMI_TCPTIMEOUTBEEP

This function beeps when timeout occurs.

Syntax

```
void hmi_TCPTimeoutBeep (
      int iConfig
);
```

Parameter

int

[in] Specify an integer to configure the timeout beep.

Possible range: 0 = Disable, others = Enable

Return Values

None

Examples

[C]

```
int config = 1;    // Enable the timeout beep

hmi_TCPTimeoutBeep(config);
```

Remark

None

5. MODBUS TCP MASTER API

This chapter provides the Modbus TCP Master API functions.
Modbus is a commonly-used communication protocol in the industry field.

The old mbm_ series Modbus TCP master API is still supported for backward compatility.

Mapping between the function code and the Modbus TCP Master API

Function Code	HMIWorks Modbus TCP Master API
1	mtm_ReadDO
2	mtm_ReadDI
3	mtm_ReadAO
4	mtm_ReadAI
5	mtm_WriteDO with ch_count = 1
6	mtm_WriteAO with ch_count = 1
15	mtm_WriteDO with ch_count > 1
16	mtm_WriteAO with ch_count > 1

5.1. MTM_REGISTER

Register a modbus communication on the TouchPAD.

Syntax

```
tHandle mtm_Register (
    int NetID,
    DWORD modbus_ip,
    int modbus_port
);
```

Parameter

NetID

[in] Specify the Net ID of the device (Range: 1 ~ 247)

modbus_ip

[in] Specify the IP of the device to communicate.

modbus_port

[in] Specify the port of the modbus communication.

Return Values

typedef int tHandle

If successful, a handle to a modbus communication is returned. The range of the possible value of tHandle is 0 ~ 7.
If not, -1 is returned.

Examples

[C]

```
tHandle h = mtm_Register(1, TCP_IPADDR(10,1,102,64), 502);
```

Remark

Backward Compatibility:
The mbm_Register function is exactly the same as the mtm_Register function.

5.2. MTM_UNREGISTER

Unregister a modbus communication from the TouchPAD.

Syntax

```
BOOL mtm_Unregister (
    tHandle h
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

Return Values

True, if unregistering the modbus communication is successful.
False, if not.

Examples

[C]

```
tHandle h;
…
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The mbm_Unregister function is exactly the same as the mtm_Unregister function.

5.3. MTM_WRITEDO

Write DO Values to the digital output module through modbus communications.

Syntax

```
BOOL mtm_WriteDO(
    tHandle h,
    int NetID,
    int addr,
    int ch_count,
    char *pcData,
    DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Modbus TCP Network ID (usually 1 ~ 247).

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the DO module.

pcData

[out] Specify the pointer to an array of char in which each bit of every byte represents the status of a single channel of an I/O module.

timeout

[in] Specify the value of the timeout value for the TCP communications. (unit: ms) When accessing more I/O points, larger timeout value may be required depending on I/O device's performance.

Return Values

True, if writing DO successfully.
False, if not.

Examples

[C]

```
int addr = 1;
int NetID = 1;
int ch_count = 16;
char DOValue[2];
DWORD timeout = 200;

// Turn on the ch 0 and ch1.
DOValue[0] = 1;
DOValue[1] = 1;

tHandle h = mtm_Register(NetID, TCP_IPADDR(10,1,102,64), 502);
mtm_WriteDO(h, NetID, addr, ch_count, DOValue, timeout);
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The old WriteDO function has the declaration as below:
BOOL mbm_WriteDO(tHandle h, int addr, int ch_count, DWORD DOValue);

The mbm_WriteDO function does the the same job as the mtm_WriteDO function, except with

1. NetID: uses what mtm_Register specifies
2. ch_count: is not greater than 32.
3. DOValue: is replaced with the char array in the mtm_WriteDO function.
4. timeout: uses the default value, 200 ms.

5.4. MTM_READDO

Read DO Values from the digital output module through modbus communications.

Syntax

```
BOOL mtm_ReadDO(
    tHandle h,
    int NetID,
    int addr,
    int ch_count,
    char *pcData,
    DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Modbus TCP Network ID (usually 1 ~ 247).

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the DO module.

pcData

[out] Specify the pointer to an array of char in which each bit of every byte represents the status of a single channel of an I/O module.

timeout

[in] Specify the value of the timeout value for the TCP communications. (unit: ms) When accessing more I/O points, larger timeout value may be required depending on I/O device's performance.

Return Values

True, if reading DO successfully.
False, if not.

Examples

[C]

```
int addr = 1;
int NetID = 1;
int ch_count = 16;
char DOValue[2];
DWORD timeout = 200;

tHandle h = mtm_Register(NetID, TCP_IPADDR(10,1,102,64), 502);
mtm_ReadDO(h, NetID, addr, ch_count, DOValue, timeout);
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The old ReadDO function has the declaration as below:
BOOL mbm_ReadDO(tHandle h, int addr, int ch_count, DWORD * DOValue);

The mbm_ReadDO function does the the same job as the mtm_ReadDO function, except with

1. NetID: uses what mtm_Register specifies
2. ch_count: is not greater than 32.
3. DOValue: is replaced with the char array in the mtm_ReadDO function.
4. timeout: uses the default value, 200 ms.

5.5. MTM_READDI

Read DI Values from the digital input module through modbus communications.

Syntax

```
DWORD mtm_ReadDI(
    tHandle h,
    int NetID,
    int addr,
    int ch_count,
    char *pcData,
    DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Modbus TCP Network ID (usually 1 ~ 247).

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the DI module.

pcData

[out] Specify the pointer to an array of char in which each bit of every byte represents the status of a single channel of an I/O module.

timeout

[in] Specify the value of the timeout value for the TCP communications. (unit: ms) When accessing more I/O points, larger timeout value may be required depending on I/O device's performance.

Return Values

True, if reading DI successfully.
False, if not.

Examples

[C]

```
int addr = 1;
int NetID = 1;
int ch_count = 16;
char DIValue[2];
DWORD timeout = 200;

tHandle h = mtm_Register(NetID, TCP_IPADDR(10,1,102,64), 502);
mtm_ReadDI(h, NetID, addr, ch_count, DIValue, timeout);
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The old ReadDI function has the declaration as below:
BOOL mbm_ReadDI(tHandle h, int addr, int ch_count, DWORD *DIValue);

The mbm_ReadDI function does the the same job as the mtm_ReadDI function, except with

1. NetID: uses what mtm_Register specifies
2. ch_count: is not greater than 32.
3. DIValue: is replaced with the char array in the mtm_ReadDI function.
4. timeout: uses the default value, 200 ms.

5.6. MTM_WRITEAO

Write AO Values to the analog output module through modbus communications.

Syntax

```
BOOL mtm_WriteAO(
    tHandle h,
    int NetID,
    int addr,
    int ch_count,
    WORD *pwData,
    DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Modbus TCP Network ID (usually 1 ~ 247).

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the AO module.

pwData

[out] Specify the pointer to an array of WORD (2 byte) in which each element of the array represents the state of a single channel of an I/O module.

timeout

[in] Specify the value of the timeout value for the TCP communications. (unit: ms) When accessing more I/O points, larger timeout value may be required depending on I/O device's performance.

Return Values

True, if writing AO successfully.
False, if not.

Examples

[C]

```
int addr = 1;
int NetID = 1;
int ch_count = 2;
WORD AOValue[2]; //for example, we have a two-channel AO module
DWORD timeout = 200;

// Set channel 0 to its maximum analog output
// The meaning of AOValue is defined differently among users.
AOValue[0] = 65535;

tHandle h = mtm_Register(NetID, TCP_IPADDR(10,1,102,64), 502);
mtm_WriteAO(h, NetID, addr, ch_count, AOValue, timeout);
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The old WriteAO function has the declaration as below:
BOOL mbm_WriteAO(tHandle h, int addr, int ch_count, WORD* AOValue);

The mbm_WriteAO function does the the same job as the mtm_WriteAO function, except with

1. NetID: uses what mtm_Register specifies
2. timeout: uses the default value, 200 ms.

5.7. MTM_READAO

Read AO Values from the analog output module through modbus communications.

Syntax

```
BOOL mtm_ReadAO(
    tHandle h,
    int NetID,
    int addr,
    int ch_count,
    WORD *pwData,
    DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Modbus TCP Network ID (usually 1 ~ 247).

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the AO module.

pwData

[out] Specify the pointer to an array of WORD (2 byte) in which each element of the array represents the state of a single channel of an I/O module.

timeout

[in] Specify the value of the timeout value for the TCP communications. (unit: ms) When accessing more I/O points, larger timeout value may be required depending on I/O device's performance.

Return Values

True, if reading AO successfully.
False, if not.

Examples

[C]

```
int addr = 1;
int NetID = 1;
int ch_count = 2;
WORD AOValue[2]; //for example, we have a two-channel AO module
DWORD timeout = 200;

tHandle h = mtm_Register(NetID, TCP_IPADDR(10,1,102,64), 502);
mtm_ReadAO(h, NetID, addr, ch_count, AOValue, timeout);
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The old ReadAO function has the declaration as below:
BOOL mbm_ReadAO(tHandle h, int addr, int ch_count, WORD* AOValue);

The mbm_ReadAO function does the the same job as the mtm_ReadAO function, except with

1. NetID: uses what mtm_Register specifies
2. timeout: uses the default value, 200 ms.

5.8. MTM_READAI

Read AI Values from the analog input module through modbus communications.

Syntax

```
BOOL mtm_ReadAI(
    tHandle h,
    int NetID,
    int addr,
    int ch_count,
    WORD *pwData,
    DWORD timeout
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Modbus TCP Network ID (usually 1 ~ 247).

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the AI module.

pwData

[out] Specify the pointer to an array of WORD (2 byte) in which each element of the array represents the state of a single channel of an I/O module.

timeout

[in] Specify the value of the timeout value for the TCP communications. (unit: ms) When accessing more I/O points, larger timeout value may be required depending on I/O device's performance.

Return Values

True, if reading AI successfully.
False, if not.

Examples

[C]

```
int addr = 1;
int NetID = 1;
int ch_count = 2;
WORD AIValue[2]; //for example, we have a two-channel AI module
DWORD timeout = 200;

tHandle h = mtm_Register(NetID, TCP_IPADDR(10,1,102,64), 502);
mtm_ReadAI(h, NetID, addr, ch_count, AIValue, timeout);
mtm_Unregister(h);
```

Remark

Backward Compatibility:
The old ReadAI function has the declaration as below:
BOOL mbm_ReadAI(tHandle h, int addr, int ch_count, WORD* AIValue);

The mbm_ReadAI function does the the same job as the mtm_ReadAI function, except with

1. NetID: uses what mtm_Register specifies
2. timeout: uses the default value, 200 ms.

6. MODBUS RTU MASTER API

This chapter provides the Modbus RTU Master APIs.
Modbus is a commonly-used serial communications in the industry field.

Mapping between the function code and the Modbus RTU Master API

Function Code	HMIWorks Modbus RTU Master API
1	mrm_ReadDO
2	mrm_ReadDI
3	mrm_ReadAO
4	mrm_ReadAI
5	mrm_WriteDO with ch_count = 1
6	mrm_WriteAO with ch_count = 1
15	mrm_WriteDO with ch_count > 1
16	mrm_WriteAO with ch_count > 1

6.1. MRM_WRITEDO

Write DO Value to the digital output module through modbus communications.

Syntax

```
BOOL mrm_WriteDO (
    Handle h,
    int NetID,
    int addr,
    int ch_count,
    char * data
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Net ID of the modbus communication.

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the DO module.

data

[in] Specify the pointer to an array in which the least significant bit of the first element represents the channel 0, the second lowest bit represents the channel 1, etc. Each 8-channel DI/DO uses a byte to store the data. Channel 0 ~ 7 are stored in data[0], channel 8 ~ 15 are stored in data[1], and so on. For example, if we turn on only channel 0 and channel 1, data[0] has the value of 3 (whose binary equivalent is 0000,0011).

Return Values

True, if writing DO successfully. False, if not.

Examples

[C]

```
HANDLE h;
int NetID = 1;
int addr = 1;
int ch_count = 8;
char DO_value[1];
DO_value[0] = 3; //that is, turn on the ch 0 and ch1.

h = uart_Open("COM1,9600,N,8,1");
mrm_WriteDO (h, NetID, addr, ch_count, DO_value);
uart_Close(h);
```

Remark

None

6.2. MRM_READDO

Read DO Value from the digital output module through modbus communications.

Syntax

```
BOOL mrm_ReadDO (
    Handle h,
    int NetID,
    int addr,
    int ch_count,
    char * data
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Net ID of the modbus communication.

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the DO module.

data

[out] Specify the pointer to an array in which the least significant bit of the first element represents the channel 0, the second lowest bit represents the channel 1, etc. Each 8-channel DI/DO uses a byte to store the data. Channel 0 ~ 7 are stored in data[0], channel 8 ~ 15 are stored in data[1], and so on. For example, if we turn on only channel 0 and channel 1, data[0] has the value of 3 (whose binary equivalent is 0000,0011).

Return Values

True, if reading DO successfully. False, if not.

Examples

[C]

```
HANDLE h;
int NetID = 1;
int addr = 1;
int ch_count = 8;
char DO_value[1];

h = uart_Open("COM1,9600,N,8,1");
mrm_ReadDO (h, NetID, addr, ch_count, DO_value);
uart_Close(h);
```

Remark

None

6.3. MRM_READDI

Read DI Value from the digital input module through modbus communications.

Syntax

```
BOOL mrm_ReadDI (
    Handle h,
    int NetID,
    int addr,
    int ch_count,
    char * data
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Net ID of the modbus communication.

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the DI module.

data

[out] Specify the pointer to an array in which the least significant bit of the first element represents the channel 0, the second lowest bit represents the channel 1, etc. Each 8-channel DI/DO uses a byte to store the data. Channel 0 ~ 7 are stored in data[0], channel 8 ~ 15 are stored in data[1], and so on. For example, if we only channel 0 and channel 1 are input, data[0] has the value of 3 (whose binary equivalent is 0000,0011).

Return Values

True, if reading DI successfully. False, if not.

Examples

[C]

```
HANDLE h;
int NetID = 1;
int addr = 1;
int ch_count = 8;
char DI_value[1];

h = uart_Open("COM1,9600,N,8,1");
mrm_ReadDI (h, NetID, addr, ch_count, DI_value);
uart_Close(h);
```

Remark

None

6.4. MRM_WRITEAO

Write AO Value to the analog output module through modbus communications.

Syntax

```
BOOL mrm_WriteAO (
    Handle h,
    int NetID,
    int addr,
    int ch_count,
    WORD * AO_value
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Net ID of the modbus communication.

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the AO module.

AO_value

[out] Specify the pointer to an array whose values are the Analog Outputs. Each AI/AO channel uses a WORD type to store a data and the data format strongly depends on the devices.

Return Values

True, if Writing AO successfully. False, if not.

Examples

[C]

```
HANDLE h;
int NetID = 1;
int addr = 1;
int ch_count = 2;
WORD AO_value[2]; //for example, we have a two-channel AO module

AO_value[0] = 65535; //arbitrarily set channel 0, simply for example
AO_value[1] = 65535; //arbitrarily set channel 1, simply for example

h = uart_Open("COM1,9600,N,8,1");
mrm_WriteAO (h, NetID, addr, ch_count, AO_value);
uart_Close(h);
```

Remark

None

6.5. MRM_READAO

Read AO Value from the analog output module through modbus communications.

Syntax

```
BOOL mrm_ReadAO (
    Handle h,
    int NetID,
    int addr,
    int ch_count,
    WORD * AO_value
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Net ID of the modbus communication.

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the AO module.

AO_value

[out] Specify the pointer to an array whose values are the Analog Outputs. Each AI/AO channel uses a WORD type to store a data and the data format strongly depends on the devices.

Return Values

True, if reading AO successfully. False, if not.

Examples

[C]

```
HANDLE h;
int NetID = 1;
int addr = 1;
int ch_count = 2;
WORD AO_value[2]; //for example, we have a two-channel AO module

h = uart_Open("COM1,9600,N,8,1");
mrm_ReadAO (h, NetID, addr, ch_count, AO_value);
uart_Close(h);
```

Remark

None

6.6. MRM_READAI

Read AI Value from the analog input module through modbus communications.

Syntax

```
BOOL mrm_ReadAI (
    Handle h,
    int NetID,
    int addr,
    int ch_count,
    WORD * AI_value
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

NetID

[in] Specify the Net ID of the modbus communication.

addr

[in] Specify the starting address of the modbus communication.

ch_count

[in] Specify the number of the channels of the AO module.

AI_value

[out] Specify the pointer to an array whose values are the Analog Inputs. Each AI/AO channel uses a WORD type to store a data and the data format strongly depends on the devices.

Return Values

True, if reading AI successfully. False, if not.

Examples

[C]

```
HANDLE h;
int NetID = 1;
int addr = 1;
int ch_count = 2;
WORD AI_value[2]; //for example, we have a two-channel AI module

h = uart_Open("COM1,9600,N,8,1");
mrm_ReadAI (h, NetID, addr, ch_count, AI_value);
uart_Close(h);
```

Remark

None

7. MODBUS RTU SLAVE API

This chapter provides the Modbus RTU Slave APIs.
Modbus is a commonly-used serial communications in the industry field.

7.1. MRS_REGISTERSLAVE

Register a Modbus slave operations on the TouchPAD.

Syntax

```
BOOL mrs_RegisterSlave (
    unsigned char NetID,
    WORD DIO_StartAddr,
    WORD DIO_count,
    char *pcDioBuf,
    WORD AIO_StartAddr,
    WORD AIO_count,
    WORD *pwAioBuf
);
```

Parameter

NetID

[in] Specify the Net ID of the modbus communication.

DIO_StartAddr

[in] Specify the starting address for the modbus communications of DI/DO.

DIO_count

[in] Specify the number of the channels for DI/DO.

pcDioBuf

[out] Specify the pointer to an array of char in which each bit of every byte represents the status of a single DI or DO channel.

AIO_StartAddr

[in] Specify the starting address for the modbus communications of AI/AO.

AIO_count

[in] Specify the number of the channels for AI/AO.

pcAioBuf

[out] Specify the pointer to an array of WORD in which each word of the array

represents the state of a single AI or AO channel.

Return Values

TRUE = OK, FALSE = Parameter Error.

Examples

[C]

```
HANDLE hPort = 0;
char DioBuf[2]; // 8-bit x2 (=16-bit) DIO (can be more)
// We arbitrarily take the first byte as the reserved byte.
// Users NEED NOT to do this if they have their own considerations.
// The first byte is used to store some information which is used by the
// host. In this example, we take
// bit 0: Initial flag, 1=initialized by master, 0 = not yet.
// bit 1~ 7: Reserved.
// bit 8~11: DIs; bit 12~15: DOs

WORD     AioBuf[6]; // 16-bit x6 AIO buffers (can be more)

void Frame12OnCreate()
{
   if ( ! hPort )    // port is not opened?
   {
         hPort = uart_Open("COM1,115200,N,8,1");    // Open com port
         memset(DioBuf, 0, sizeof(DioBuf));    // Clear buffer
         memset(AioBuf, 0, sizeof(AioBuf));
         mrs_RegisterSlave(1, 0, 16, DioBuf, 0, 6, AioBuf);
   }
}

void Timer4OnExecute(tWidget *pWidget)
{
   unsigned long mrs_status;
   static int iCnt = 0;
```

```
    if ( hPort )    // port is opened?
    {
        // retrieve Modbus RTU command and process it
        mrs_status = mrs_ProcessCmd(hPort);
        if ( (mrs_status & MRS_DIO_DIRTY )    // DIO is updated
        ||     ( iCnt == 0 )    )    // First time, or periodically updating
        {
            // Bit 12 ~ 15 as DO
            CheckBoxSelectedSet(&CheckBox5, DioBuf[1] & (1 << 4));
            CheckBoxSelectedSet(&CheckBox7, DioBuf[1] & (1 << 5));
            CheckBoxSelectedSet(&CheckBox8, DioBuf[1] & (1 << 6));
            CheckBoxSelectedSet(&CheckBox9, DioBuf[1] & (1 << 7));
        }
    }
    iCnt ++;
    if (iCnt > 10) iCnt = 0;
}
```

Remark

1. Set 0 to both parameters, DIO_count and AIO_count, to **unregister** those slave functions.
2. Currently, we support only one slave device on TouchPAD.

7.2. MRS_PROCESSCMD

Process the Modbus RTU Slave command (suggested for every 10 ms).

Syntax

```
unsigned long mrs_ProcessCmd (
    Handle h,
);
```

Parameter

h

[in] Specify the handle opened by the uart_Open function.

Return Values

The returning unsigned long value has 32 bits, in which

Bit that is set	Representing macros	Descriptions
0	MRS_DIO_DIRTY	DIO is updated
1	MRS_AIO_DIRTY	AIO is updated
8	MRS_DIO_READ	Read DI
9	MRS_AIO_READ	Read AI
31	MRS_CMD_ERROR	Error (0x8000 0000)

Examples

[C]

```
HANDLE hPort = 0;
char DioBuf[2]; // 8-bit x2 (=16-bit) DIO (can be more)
// We arbitrarily take the first byte as the reserved byte.
// Users NEED NOT to do this if they have their own considerations.
```

```
// The first byte is used to store some information which is used by the
// host. In this example, we take
// bit 0: Initial flag, 1=initialized by master, 0 = not yet.
// bit 1~ 7: Reserved.
// bit 8~11: DIs; bit 12~15: DOs

WORD     AioBuf[6]; // 16-bit x6 AIO buffers (can be more)

void Frame12OnCreate()
{
   if ( ! hPort )    // port is not opened?
   {
         hPort = uart_Open("COM1,115200,N,8,1");    // Open com port
         memset(DioBuf, 0, sizeof(DioBuf));    // Clear buffer
         memset(AioBuf, 0, sizeof(AioBuf));
         mrs_RegisterSlave(1, 0, 16, DioBuf, 0, 6, AioBuf);
   }
}

void Timer4OnExecute(tWidget *pWidget)
{
   unsigned long mrs_status;
   static int iCnt = 0;

   if ( hPort )    // port is opened?
   {
       // retrieve Modbus RTU command and process it
       mrs_status = mrs_ProcessCmd(hPort);
       if ( (mrs_status & MRS_DIO_DIRTY )    // DIO is updated
       ||     ( iCnt == 0 )    )    // First time, or periodically updating
       {
           // Bit 12 ~ 15 as DO
           CheckBoxSelectedSet(&CheckBox5, DioBuf[1] & (1 << 4));
           CheckBoxSelectedSet(&CheckBox7, DioBuf[1] & (1 << 5));
           CheckBoxSelectedSet(&CheckBox8, DioBuf[1] & (1 << 6));
           CheckBoxSelectedSet(&CheckBox9, DioBuf[1] & (1 << 7));
       }
```

```
    }
    iCnt ++;
    if (iCnt > 10) iCnt = 0;
}
```

Remark

None

7.3. MRS_GETIOSTATUS

Process the Modbus RTU Slave command (suggested for every 10 ms).

Syntax

```
unsigned long mrs_GetIOStatus (
        Handle h,
);
```

Parameter

h

[in] Specify the handle opened by the uart_Open function.

Return Values

Bit that is set	Representing macros	Descriptions
0	MRS_DIO_DIRTY	DIO is updated
1	MRS_AIO_DIRTY	AIO is updated
8	MRS_DIO_READ	Read DI
9	MRS_AIO_READ	Read AI
31	MRS_CMD_ERROR	Error (0x8000 0000)

Examples

[C]

```
HANDLE h;
…
if(mrs_GetIOStatus(h) & MRS_CMD_ERROR)
   LabelTextSet(&Label5, “Error!”);
…
```

Remark

None

8. UART API

This chapter introduces UART APIs.
This set of UART APIs is designed for the serial port in TouchPAD.

Uart Reference

Uart operations include basic management operations, such as opening, sending, receiving, and closing. The following topics describe how you can operate uart programmatically using the uart functions.

Classification

Functions	Descriptions	Memo
uart_Send, uart_Recv	Used for sending/receiving the DCON commands. They add/check the checksum if enabled. They also add a terminator when sending and read until the terminator received. The DCON commands always terminated at 0xD (CR, Carriage Return).	
uart_Write, uart_Read	Used for sending/receiving ASCII string. Both functions are without checksum and the uart_Read returns if 0xD (CR, Carriage Return) is received.	We suggest users to use uart_Send and uart_Recv instead of uart_Write and uart_Read.
uart_BinSend, uart_BinRecv	Used for sending/receiving binary data or ASCII string. Both functions handle a specified length of data.	
uart_BinWrite, uart_BinRead	uart_BinWrite is exactly the same as uart_BinSend and uart_BinRead is exactly the same as uart_BinRecv.	We suggest users to use uart_BinSend and uart_BinRecv instead of uart_BinWrite and uart_BinRead.

8.1. UART_OPEN

This function opens the COM port and specifies the baud rate, parity bits, data bits, and stop bits.

Syntax

```
HANDLE uart_Open(
    LPCSTR ConnectionString
);
```

Parameter

connectionString

[in] Specifies the COM port, baud rate, parity bits, data bits, and stop bits.

The default setting is COM1,115200,N,8,1.

The format of ConnectionString is as follows:

"com_port,baud_rate,parity_bits,data_bits,stop_bits"

Com_port:

COM1, COM2……

baud_rate:

1200/2400/4800/9600/19200/38400/57600/115200

parity_bits:

'N' = NOPARITY

'O' = ODDPARITY

'E' = EVENPARITY

'M' = MARKPARITY

'S' = SPACEPARITY

Data_bits:

5/6/7/8

Stop_bits:

"1" = ONESTOPBIT

"2" = TWOSTOPBITS

"1.5" = ONE5STOPBITS

Return Values

A handle to the opening COM port
It is the number of the COM port.
For example, if COM 1 is opened, the value of the handle is 1.

Examples

[C]

```
HANDLE hOpen;
hOpen = uart_Open("COM1,9600,N,8,1");
```

Remark

None

8.2. UART_CLOSE

This function closes the COM port which has been opened.

Syntax

```
BOOL uart_Close(
    HANDLE hPort
);
```

Parameter

hPort

[in] Handle to the opened COM port.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_Close(hOpen);
```

Remark

The function for a specified COM port should not be used after it has been closed.

8.3. UART_SEND

This function sends data through the COM port which have been opened. The sending string is automatically appended with the checksum (2 byte) if it is enabled by uart_EnableCheckSum and appended with a terminating character (default: CR) if it is set by uart_SetTerminator.

Syntax

```
BOOL uart_Send(
    HANDLE hPort,
    LPSTR buf
);
```

Parameter

hPort

[in] Handle to the opened COM port

buf

[in] A pointer to a buffer that send the data (null-terminated)

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_Send(hOpen, buf);
uart_Close(hPort);
```

Remark

None

8.4. UART_RECV

This function retrieves data through the COM port which have been opened.

Syntax

```
BOOL uart_Recv(
    HANDLE hPort,
    LPSTR buf
);
```

Parameter

hPort

[in] Handle to the opened COM port

buf

[out] A pointer to a buffer that receives the data (null-terminated)

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_Recv(hOpen, buf);
uart_Close(hPort);
```

Remark

Both uart_EnableCheckSum and uart_SetTimeOut can influence the behavior of the uart_Recv function.
Though FALSE returned if timeout occurs or checksum has errors, uart_Recv still fill the data into the buffer (the second parameter). Of course, the data of the buffer may not be useful.

8.5. UART_SENDCMD

This function sends commands through the COM port which have been opened and then receive data from the COM port.
This function mainly consists of 3 functions, uart_Purge, uart_Send, and uart_Recv.

Syntax

```
BOOL uart_SendCmd(
    HANDLE hPort,
    LPSTR cmd,
    LPSTR szResult
);
```

Parameter

hPort

[in] Handle to the opened COM

cmd

[in] A pointer to a command (null-terminated)

szResult

[out] A pointer to a buffer that receives the data (null-terminated)

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
```

```
ret = uart_SendCmd(hOpen,"$00M", buf);      // $00M: ask the device name
uart_Close(hOpen);
```

Remark

None

8.6. UART_SETTIMEOUT

This function sets the time out timer.

Syntax

```
void uart_SetTimeOut(
    HANDLE hPort,
    DWORD msec,
);
```

Parameter

hPort

[in] Handle to the opened COM port.

msec

[in] Millisecond to the timer

Return Values

None

Examples

[C]

```
HANDLE hOpen;
DWORD mes;
hOpen = uart_Open("COM1,9600,N,8,1");
uart_SetTimeOut(hOpen, mes);
```

Remark

None

8.7. UART_ENABLECHECKSUM

This function turns on the check sum or not.

Syntax

```
void uart_EnableCheckSum(
    HANDLE hPort,
    BOOL bEnable
);
```

Parameter

hPort

[in] Handle to the opened COM port.

bEnable

[in] Decide the check sum turning on or not.

Default is disabling.

Return Values

None

Examples

[C]

```
HANDLE hUart;
char result[32];
hUart = uart_Open("");
uart_EnableCheckSum(hUart , true);
uart_SendCmd(hUart, "$00M", result);
uart_Close(hPort);
```

Remark

1. uart_EnableCheckSum does not apply to the binary UART API functions, that is, uart_BinSend, uart_BinRecv, and uart_BinSendCmd.

8.8. UART_SETTERMINATOR

This function sets the terminate characters.

Syntax

```
void uart_SetTerminator(
    HANDLE hPort,
    LPCSTR szTerm
);
```

Parameter

hPort

[in] Handle to the opened COM port.

szTerm

[in] Pointer the terminate characters.
Default is CR.

Return Values

None

Examples

[C]

```
HANDLE hUart;
char result[32];
hUart = uart_Open("");
uart_SetTerminator(hUart, "\015");
uart_SendCmd(hUart, "$00M", result);
uart_Close(hPort);
```

Remark

None

8.9. UART_BINSEND

This function sends binary data through the COM port which have been opened.

Syntax

```
BOOL uart_BinSend(
    HANDLE hPort,
    LPSTR buf,
    int buf_len
);
```

Parameter

hPort

[in] Handle to the opened COM port.

buf

[in] A pointer to a buffer that send the data (null-terminated)

buf_len

[in] The length of the buffer

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_BinSend(hOpen, buf, Length);
uart_Close(hPort);
```

Remark

uart_BinSend does not support uart_EnableCheckSum.

8.10. UART_BINRECV

This function retrieves binary data through the COM port which have been opened.

Syntax

```
BOOL uart_BinRecv(
    HANDLE hPort,
    LPSTR buf,
    int buf_len
);
```

Parameter

hPort

[in] Handle to the open COM port.

buf

[out] A pointer to a buffer that receives the data.

buf_len

[in] The length of the buffer

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
char buf[Length];
hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_BinRecv(hOpen, buf, Length);
uart_Close(hPort);
```

Remark

uart_BinRecv does not support uart_EnableCheckSum.

8.11. UART_BINSENDCMD

This function sends binary commands through the COM port which have been opened and then receive data from the COM port.

Syntax

```
BOOL uart_BinSendCmd(
    HANDLE hPort,
    LPSTR cmd,
    int cmd_len,
    LPSTR buf,
    int buf_len
);
```

Parameter

hPort

[in] Handle to the opened COM.

cmd

[in] A pointer to a command.

cmd_len

[in] The length of the command

buf

[out] A pointer to a buffer that receives the data.

buf_len

[in] The length of the buffer

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
BOOL ret;
HANDLE hOpen;
char cmd[Length1];
char buf[Length2];

hOpen = uart_Open("COM1,9600,N,8,1");
ret = uart_BinSendCmd(hOpen,"$00M", Length1, buf, Length2);
// $00M: ask the device name
uart_Close(hPort);
```

Remark

uart_BinSendCmd does not support uart_EnableCheckSum.

8.12. UART_GetRxDataCount

This function returns the count of bytes which are presently in the receiver buffer.

Syntax

```
unsigned long uart_GetRxDataCount (
    HANDLE hPort,
);
```

Parameter

hPort

[in] Handle to the opened COM.

Return Values

The count of bytes which are presently in the receiver buffer

Examples

[C]

```
// The example of an echo server

HANDLE hOpen=-1;
#define LENGTH 20 //256
int iRxData=0;

void Timer5OnExecute(tWidget *pWidget)
{
    //--- The buffer to storage the data, it's size is User-defined value ---
    static char recv_str[LENGTH];
    int res=0,ret=0;
```

```
    //--- If handle is invalid, return ---
    if(hOpen < 0)    return;
    //--- If no data received, return ---
    if(uart_GetRxDataCount(hOpen)==0) return;
    //--- Check whether the data has been transferred completely     ---
    if (iRxData != uart_GetRxDataCount(hOpen))
    {
      iRxData = uart_GetRxDataCount(hOpen);
      return;
    }
    //--- Make sure the message don't overflow the buffer ---
    iRxData = (iRxData<LENGTH)?iRxData:LENGTH;

    //--- Receive the data from COM port ---
    res = uart_BinRecv(hOpen, recv_str,iRxData);
    recv_str[iRxData]=0;
    //--- Purge Rx Buffer ---
    uart_Purge(hOpen, 0, 1);

    //--- Process all recevied data ---
    if (res)
    {
        hmi_Beep();
        //--- echo the received message ---
        LabelTextSet(&Label4, recv_str);
        ret = uart_BinSend(hOpen, recv_str, iRxData);
        iRxData = 0;

        if (ret)    LabelTextSet(&Label9, "Echo OK");
        else        LabelTextSet(&Label9, "Echo Error");
    }
}

void BitButton10OnClick(tWidget *pWidget)    // Start
{
    //--- Close the existing handle ---
   if(hOpen>=0)
```

```
    {
        uart_Close(hOpen);
        hOpen = -1;
    }

    //--- Establish a new handle ---
    if(hOpen<0)
    {
        hOpen = uart_Open("COM1,115200,N,8,1");
    }

    //---   If success, display current COM Port settings on screen ---
    if(hOpen>=0)
    {
       LabelTextSet(&Label8, "COM1,115200,N,8,1");
       uart_SetTimeOut(hOpen, 300);
    }
}

void BitButton11OnClick(tWidget *pWidget)    // Stop
{
    if(hOpen>=0)
    {
        uart_Close(hOpen);
        hOpen = -1;
        LabelTextSet(&Label8, "Press 'Start' to Begin");
        LabelTextSet(&Label4, "");
        LabelTextSet(&Label9, "");
    }
}
```

Remark

None

8.13. UART_PURGE

This function sends binary commands through the COM port which have been opened and then receive data from the COM port.

Syntax

```
int uart_Purge (
    HANDLE hPort,
    int ClearTx,
    int ClearRx
);
```

Parameter

hPort

[in] Handle to the opened COM.

ClearTx

[in] A integer to tell the uart_Purge to clear the transmitter buffer.

Possible value:

0: DO NOT clear

Otherwise: DO clear

ClearRx

[in] A integer to tell the uart_Purge to clear the receiver buffer.

Possible value:

0: DO NOT clear

Otherwise: DO clear

Return Values

It always returns zero.

The returning value is reserved for future use.

Examples

[C]

```
// The example of an echo server

HANDLE hOpen=-1;
#define LENGTH 20 //256
int iRxData=0;

void Timer5OnExecute(tWidget *pWidget)
{
    //--- The buffer to storage the data, it's size is User-defined value ---
    static char recv_str[LENGTH];
    int res=0,ret=0;

    //--- If handle is invalid, return ---
    if(hOpen < 0)    return;
    //--- If no data received, return ---
    if(uart_GetRxDataCount(hOpen)==0) return;
    //--- Check whether the data has been transferred completely      ---
    if (iRxData != uart_GetRxDataCount(hOpen))
    {
      iRxData = uart_GetRxDataCount(hOpen);
      return;
    }
    //--- Make sure the message don't overflow the buffer ---
    iRxData = (iRxData<LENGTH)?iRxData:LENGTH;

    //--- Receive the data from COM port ---
    res = uart_BinRecv(hOpen, recv_str,iRxData);
    recv_str[iRxData]=0;
    //--- Purge Rx Buffer ---
    uart_Purge(hOpen, 0, 1);

    //--- Process all recevied data ---
    if (res)
    {
        hmi_Beep();
```

```
        //--- echo the received message ---
        LabelTextSet(&Label4, recv_str);
        ret = uart_BinSend(hOpen, recv_str, iRxData);
        iRxData = 0;

        if (ret)    LabelTextSet(&Label9, "Echo OK");
        else        LabelTextSet(&Label9, "Echo Error");
    }
}

void BitButton10OnClick(tWidget *pWidget)    // Start
{
    //--- Close the existing handle ---
  if(hOpen>=0)
  {
      uart_Close(hOpen);
      hOpen = -1;
  }

  //--- Establish a new handle ---
  if(hOpen<0)
  {
      hOpen = uart_Open("COM1,115200,N,8,1");
  }

  //---   If success, display current COM Port settings on screen ---
  if(hOpen>=0)
  {
     LabelTextSet(&Label8, "COM1,115200,N,8,1");
     uart_SetTimeOut(hOpen, 300);
  }
}

void BitButton11OnClick(tWidget *pWidget)    // Stop
{
  if(hOpen>=0)
  {
```

```
        uart_Close(hOpen);
        hOpen = -1;
        LabelTextSet(&Label8, "Press 'Start' to Begin");
        LabelTextSet(&Label4, "");
        LabelTextSet(&Label9, "");
    }
}
```

Remark

None

9. DCON_IO API

DCON_IO Reference

DCON_IO API supports to operate I-7000 series I/O modules of ICP DAS.
For more details of I-7000 series:
http://www.icpdas.com/products/Remote_IO/i-7000/i-7000_introduction.htm

9.1. dcon_WriteDO

This function writes the DO values to DO modules.

Syntax

```
BOOL dcon_WriteDO(
    HANDLE hPort,
    int iAddress,
    int iDO_TotalCh,
    DWORD lDO_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iDO_TotalCh

[in] The total number of DO channels of the DO modules.

iDO_Value

[in] The value which is the binary representation of DOs.

1 is to turn on the DO channel; 0 is off.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
int addr = 1;
```

```
int total_channel = 8;
DWORD do_value = 4; // turn on the channel two

hPort = uart_Open("COM3,9600");
BOOL ret = dcon_WriteDO(hPort, addr, total_channel, do_value);
uart_Close(hPort);
```

Remark

None

9.2. dcon_WriteDOBit

This function writes a single bit of value to the DO module, that is, only the channel corresponding to the bit is changed.

Syntax

```
BOOL dcon_WriteDOBit(
        HANDLE hPort,
        int iAddress,
        int iChannel,
        int iDO_TotalCh,
        int iBitValue
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in]The DO channel to change

iDO_TotalCh

[in] The total number of DO channels of the DO modules.

iBitValue

[in] 1 is to turn on the DO channel; 0 is off.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
int iAddress = 1;
int iChannel = 2;
int iDO_TotalCh = 8;
int iBitValue = 1;

hPort = uart_Open("COM1,115200");
BOOL ret = dcon_WriteDOBit(hPort, iAddress, iChannel, iDO_TotalCh,
iBitValue);
uart_Close(hPort);
```

Remark

None

9.3. DCON_READDO

This function reads the DO value of the DO module.

Syntax

```
BOOL dcon_ReadDO(
    HANDLE hPort,
    int iAddress,
    int iDO_TotalCh,
    DWORD *lDO_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iDO_TotalCh

[in] The total number of DO channels of the DO modules.

lDO_Value

[out] The pointer to the DO value to read from the DO module. The DO value is the binary representation of DOs.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress = 1;
```

```
int total_channel = 8;
DWORD do_value;

hPort = uart_Open("COM1,115200");
BOOL ret = dcon_ReadDO(hPort, iAddress, total_channel, &do_value );
uart_Close(hPort);
```

Remark

None

9.4. DCON_READDI

This function reads the DI value of the DI module.

Syntax

```
BOOL dcon_ReadDI(
HANDLE hPort,
int iAddress,
int iDI_TotalCh,
DWORD *lDI_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iDI_TotalCh

[in] The total channels of the DI module.

lDI_Value

[out] The pointer to read-back value which is the binary representation of DIs.
1 means high in the DI channel; 0 is low.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress = 2;
```

```
int iDI_TotalCh = 8;
DWORD lDI_Value;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadDI(hPort, iAddress, iDI_TotalCh, &lDI_Value);
uart_Close(hPort);
```

Remark

None

9.5. DCON_READDIO

This function reads the DI and the DO values of the DIO module.

Syntax

```
BOOL dcon_ReadDIO(
    HANDLE hPort,
    int iAddress,
    int iDI_TotalCh,
    int iDO_TotalCh,
    DWORD* lDI_Value,
    DWORD* lDO_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iDI_TotalCh

[in] The total number of DI channels of the DIO module.

iDO_TotalCh

[in] The total number of DO channels of the DIO module.

lDI_Value

[out] The pointer to the read-back value which is the binary representation of DIs. 1 means high in the DI channel; 0 is low.

lDO_Value

[out] The pointer to the read-back value which is the binary representation of DOs. 1 means high in the DO channel; 0 is low.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
DWORD lDI_Value;
DWORD lDO_Value;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadDIO(hPort, iAddress, iDI_TotalCh,
iDO_TotalCh, &lDI_Value, &lDO_Value);
uart_Close(hPort);
```

Remark

None

9.6. DCON_READDILATCH

This function reads the DI latch value of the DI module.

Syntax

```
BOOL dcon_ReadDILatch(
HANDLE hPort,
int iAddress,
int iDI_TotalCh,
int iLatchType,
DWORD *lDI_Latch_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iDI_TotalCh

[in] The total number of the DI channels of the DI module.

iLatchType

[in] The latch type specified to read latch value back.

1 = high status latched

0 = low status latched

lDI_Latch_Value

[out] The pointer to the latch value read back from the DI module.

The latch value of a particular channel is

1 if there's at least one time that the DI channel is high for latch type = 1;

0 if there's at least one time that the DI channel is low for latch type = 0.

Take latch value of each channel as a bit of a binary value, then the binary value is the DI latch value.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iDI_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadDILatch(hPort, iAddress, iDI_TotalCh,
iLatchType, &lDI_Latch_Value);
uart_Close(hPort);
```

Remark

None

9.7. DCON_CLEARDILATCH

This function clears the latch value of the DI module.

Syntax

```
BOOL dcon_ClearDILatch(
HANDLE hPort,
int iAddress
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ClearDILatch(hPort, iAddress);
uart_Close(hPort);
```

Remark

None

9.8. DCON_READDIOLATCH

This function reads the latch values of the DI and DO channels of the DIO module.

Syntax

```
BOOL dcon_ReadDIOLatch(
HANDLE hPort,
int iAddress,
int iDI_TotalCh,
int iDO_TotalCh,
int iLatchType,
DWORD *lDI_Latch_Value,
DWORD *lDO_Latch_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iDI_TotalCh

[in] The total number of the DI channels of the DIO module.

iDO_TotalCh

[in] The total number of the DO channels of the DIO module.

iLatchType

[in] The type of the latch value read back.

1 = high status latched

0 = low status latched

lDI_Latch_Value

[out] The pointer to the latch value read back from the DI channels of DIO module.

The latch value of a particular channel is

1 if there's at least one time that the DI channel is high for latch type = 1;

0 if there's at least one time that the DI channel is low for latch type = 0.
Take latch value of each channel as a bit of a binary value, then the binary value is the DI latch value.

lDO_Latch_Value

[out] The pointer to the latch value read back from the DO channels of DIO module.
The latch value of a particular channel is
1 if there's at least one time that the DO channel is high for latch type = 1;
0 if there's at least one time that the DO channel is low for latch type = 0.
Take latch value of each channel as a bit of a binary value, then the binary value is the DO latch value.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iDI_TotalCh=8;
int iDO_TotalCh=8;
int iLatchType=0;
DWORD lDI_Latch_Value;
DWORD lDO_Latch_Value;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadDIOLatch(hPort, iAddress, iDI_TotalCh,
iDO_TotalCh, iLatchType, &lDI_Latch_Value,&lDO_Latch_Value);
uart_Close(hPort);
```

Remark

None

9.9. DCON_CLEARDIOLATCH

This function clears the latch values of DI and DO channels of the DIO module.

Syntax

```
BOOL dcon_ClearDIOLatch(
    HANDLE hPort,
    int iAddress
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ClearDIOLatch(hPort, iAddress);
uart_Close(hPort);
```

Remark

None

9.10. DCON_READDICNT

This function reads the counts of the DI channels of the DI module.

Syntax

```
BOOL dcon_ReadDICNT(
HANDLE hPort,
int iAddress,
int iChannel,
int iDI_TotalCh,
DWORD *lCounter_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel to which the counter value belongs

iDI_TotalCh

[in] Total number of the DI channels of the DI module.

lCounter_Value

[out] The pointer to the counter value

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel =2;
int iDI_TotalCh=8;
DWORD lCounter_Value;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadDICNT(hPort, iAddress,iChannel,iDI_TotalCh,
&lCounter_Value);
uart_Close(hPort);
```

Remark

None

9.11. DCON_CLEARDICNT

This function clears the counter value of the DI channel of the DI module.

Syntax

```
BOOL dcon_ClearDICNT(
HANDLE hPort,
int iAddress,
int iChannel,
int iDI_TotalCh
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel to which the counter value belongs

iDI_TotalCh

[in] Total number of the DI channels of the DI module.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel=2;
```

```
int iDI_TotalCh=8;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ClearDICNT(hPort, iAddress,iChannel,iDI_TotalCh);
uart_Close(hPort);
```

Remark

None

9.12. DCON_WRITEAO

This function writes the AO value to the AO modules.

Syntax

```
BOOL dcon_WriteAO(
    HANDLE hPort,
    int iAddress,
    int iChannel,
    int iAO_TotalCh,
    float fValue
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel to which the AO value is written

iAO_TotalCh

[in] The total number of the AO channels of the AO module.

float fValue

[in] The AO value to write to the AO module

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue=5;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_WriteAO(hPort, iAddress, iChannel, iAO_TotalCh,
fValue);
uart_Close(hPort);
```

Remark

None

9.13. DCON_READAO

This function reads the AO value of the AO module.

Syntax

```
BOOL dcon_ReadAO(
HANDLE hPort,
int iAddress,
int iChannel,
int iAO_TotalCh,
float *fValue
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel from which the AO value is read back

iAO_TotalCh

[in] The total number of the AO channels of the AO module.

float fValue

[in] The pointer to the AO value that is read back from the AO module

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel=2;
int iAO_TotalCh=8;
float fValue;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadAO(hPort, iAddress,iChannel,iAO_TotalCh,
&fValue);
uart_Close(hPort);
```

Remark

None

9.14. DCON_READAI

This function reads the AI value of engineering-mode (floating-point) from the AI module.

Syntax

```
BOOL dcon_ReadAI(
HANDLE hPort,
int iAddress,
int iChannel,
int iAI_TotalCh,
float *fValue
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel from which the AI value is read back

iAI_TotalCh

[in] The total number of the AI channels of the AI module.

fValue

[in] The pointer to the AI value that is read back from the AI module.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel=2;
int iAI_TotalCh=8;
float fValue;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadAI(hPort, iAddress, iChannel, iAI_TotalCh,
&fValue);
uart_Close(hPort);
```

Remark

None

9.15. DCON_READAIHEX

This function reads the AI value of 2's complement-mode (hexadecimal) from the AI module.

Syntax

```
BOOL dcon_ReadAIHex(
        HANDLE hPort,
        int iAddress,
        int iChannel,
        int iAI_TotalCh,
        int *iValue
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel from which the AI value is read back

iAI_TotalCh

[in] The total number of the AI channels of the AI module.

iValue

[in] The pointer to the AI value that is read back from the AI module

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel=2;
int iAI_TotalCh=8;
int iValue;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadAIHex(hPort, iAddress, iChannel, iAI_TotalCh,
&iValue);
uart_Close(hPort);
```

Remark

None

9.16. DCON_READAIALL

This function reads all the AI values of all channels in engineering-mode (floating-point) from the AI module.

Syntax

```
BOOL dcon_ReadAIAll(
    HANDLE hPort,
    int iAddress,
    float fValue[]
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

fValue[]

[out] The array which contains the AI values that read back from the AI module.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
float fValue[8];

hPort = uart_Open("COM1,115200");
```

```
BOOL iRet = dcon_ReadAIAll(hPort, iAddress, fValue);
uart_Close(hPort);
```

Remark

None

9.17. DCON_READAIALLHEX

This function reads all the AI values of all channels in 2's complement-mode (hexadecimal) from the AI module.

Syntax

```
BOOL dcon_ReadAIAllHex(
    HANDLE hPort,
    int iAddress,
    int iValue[]
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iValue[]

[out] The array which contains the AI values that read back from the AI module.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iValue[8];

hPort = uart_Open("COM1,115200");
```

```
BOOL iRet = dcon_ReadAIAllHex(hPort, iAddress, iValue);
uart_Close(hPort);
```

Remark

None

9.18. DCON_READCNT

This function reads the counter values of the counter/frequency modules.

Syntax

```
BOOL dcon_ReadCNT(
        HANDLE hPort,
        int iAddress,
        int iChannel,
        DWORD *lCounter_Value
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel from which the count value is read back from the counter/frequency module

lCounter_Value

[out] The pointer to the counter value that reads back from the counter/frequency module

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
```

```
BYTE iAddress=1;
int iChannel=0;
DWORD lCounter_Value;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadCNT(hPort, iAddress, iChannel,
&lCounter_Value);
uart_Close(hPort);
```

Remark

None

9.19. DCON_CLEARCNT

This function clears the counter values of the counter/frequency modules.

Syntax

```
BOOL dcon_ClearCNT(
    HANDLE hPort,
    int iAddress,
    int iChannel
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel where the count value is cleared

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
BYTE iAddress=1;
int iChannel=0;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ClearCNT(hPort, iAddress, iChannel);
```

```
uart_Close(hPort);
```

Remark

None

9.20. DCON_READCNTOVERFLOW

This function reads the overflow value of the channel from the counter/frequency modules.

Syntax

```
BOOL dcon_ReadCNTOverflow(
    HANDLE hPort,
    int iAddress,
    int iChannel,
    int *iOverflow
);
```

Parameter

hPort

[in] The serial port HANDLE opened by uart_Open()

iAddress

[in] The address of the command-receiving I/O module

iChannel

[in] The channel from which the overflow value is read back from the counter/frequency module

iOverflow

[out] The pointer to the overflow value, 1: overflow; 0: not.

Return Values

TRUE indicates success. FALSE indicates failure.

Examples

[C]

```
HANDLE hPort;
```

```
BYTE iAddress=1;
int iChannel=0;
int iOverflow;

hPort = uart_Open("COM1,115200");
BOOL iRet = dcon_ReadCNT_Overflow(hPort, iAddress, iChannel,
&iOverflow);
uart_Close(hPort);
```

Remark

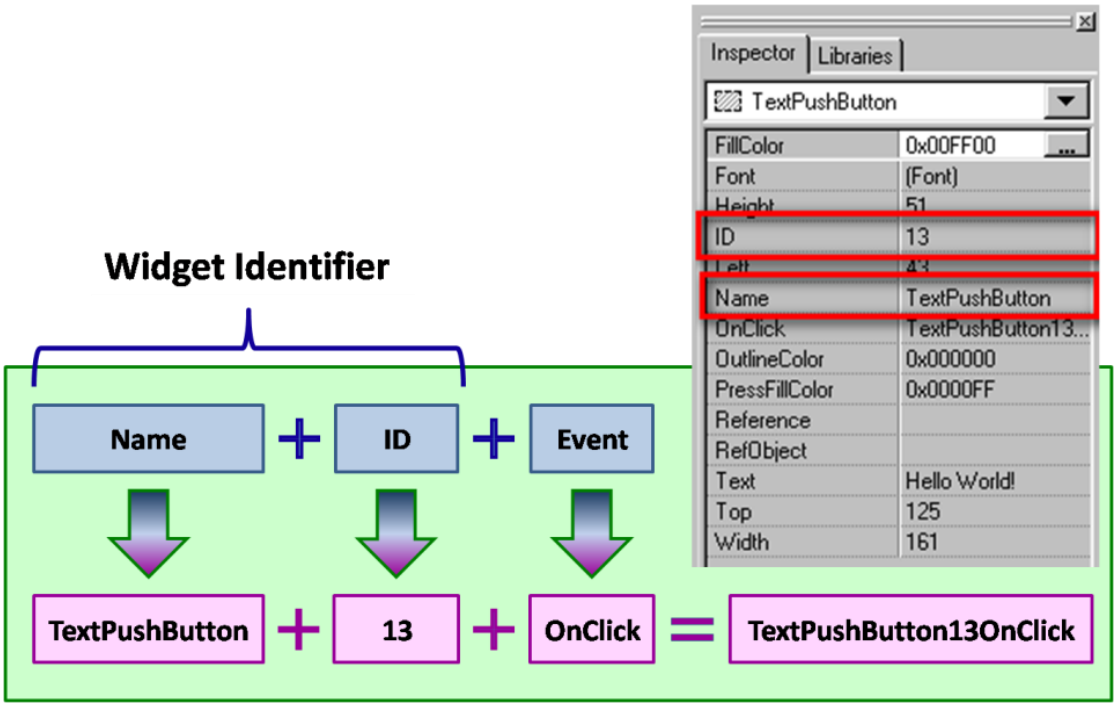
None

10. WIDGET API

This chapter provides APIs that are not specified in the Stellaris Graphics Library. (The API functions that we made some modifications)
“The Stellaris Graphics Library is a royalty-free set of graphics primitives and a widget set for creating graphical user interfaces …”

For more details:
http://www.luminarymicro.com/products/stellaris_graphics_library.html

Note that the naming convention of the event handler of the widget (here the widget is TextPushButton) is as followings:



10.1. TEXTBUTTONTEXTGET

Get the “Text” property of the TextPushButton.
Users can set the “Text” property in the inspector in the design time.

This function is used to get a static string from the “Text” property. The widget has no buffer for the “Text”, so the string must be static string (static char[]).

Syntax

```
const char * TextButtonTextGet (
        tTextButton * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the TextPushButton, to get the “Text” property.

Return Values

A constant pointer to the static string to store the “Text” of the TextPushButton

Examples

[C]

```
int tag = 0;
int count = 0;
static char str[16];
static char str2[16];

void TextPushButton4OnClick(tWidget *pWidget)
{
```

```
    count ++;

    //Set the value of count to the "Tag" property
    TextButtonTagSet((tTextButton*)pWidget, count);

    //Get the value of the "Tag" property
    tag = TextButtonTagGet((tTextButton*)pWidget);

    //Set the "Text" property of the TextPushButton 4
    usprintf(str, "%d", tag);
    TextButtonTextSet((tTextButton*)pWidget, str);

    //Get the "Text" property of the TextPushButton
    //And then show it on the Label 5 (in this example)
    strcpy(str2, TextButtonTextGet((tTextButton*)pWidget));
    LabelTextSet(&LabelWidget5, str2);
}
```

Remark

None

10.2. TEXTBUTTONTEXTSET

Set the “Text” property of the TextPushButton.
Users can set the “Text” property in the inspector in the design time.

This function is used to set a static string to the “Text” property. The widget has no buffer for the “Text”, so the string must be static string (static char[]).

Syntax

```
void TextButtonTextSet (
        tTextButton * pWidget,
        char * text
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the CheckBox, to set the “Selected” property.

text

[in] Specify the static string of the “Text” property.

Return Values

None

Examples

[C]

```
int tag = 0;
int count = 0;
static char str[16];
static char str2[16];
```

```
void TextPushButton4OnClick(tWidget *pWidget)
{
   count ++;

   //Set the value of count to the "Tag" property
   TextButtonTagSet((tTextButton*)pWidget, count);

   //Get the value of the "Tag" property
   tag = TextButtonTagGet((tTextButton*)pWidget);

   //Set the "Text" property of the TextPushButton 4
   usprintf(str, "%d", tag);
   TextButtonTextSet((tTextButton*)pWidget, str);

   //Get the "Text" property of the TextPushButton
   //And then show it on the Label 5 (in this example)
   strcpy(str2, TextButtonTextGet((tTextButton*)pWidget));
   LabelTextSet(&LabelWidget5, str2);
}
```

Remark

None

10.3. SLIDERRANGEGET

Get the range of the Slider.
That is, get the Min and the Max properties of the Slider.

Syntax

```
void SliderRangeGet (
        tWidget* pWidget,
        long lMinimum,
        long lMaximum
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the Slider, to get its Max and Min property.

lMinimum

[in] Specify the integer to store the minimum of the Slider value, that is, the Min property.

lMaximum

[in] Specify the integer to store the maximum of the Slider value, that is, the Max property.

Return Values

None

Examples

[C]

```
void BitButton5OnClick(tWidget *pWidget)
```

```
{
    static char msg[32];
    long min;
    long max;

    SliderRangeGet(&Slider4, min, max);

    usprintf(msg, "%d, %d", min, max);
    LabelTextSet(&Label6, msg);
}
```

Remark

The parameters lMinimum and lMaximum are not pointers (actually SliderRangeGet is a macro), while the pWidget is a pointer.

10.4. HotSpotLastXGet

Get the last clicked point's coordinate X.
(The left-top vertex of the screen is the origin.)

Syntax

```
int HotSpotLastXGet (
        tHotSpot * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the HotSpot, to get the last clicked point's coordinate X.

Return Values

The last clicked point's coordinate X

Examples

[C]

```
int tag = 0;
int count = 0;
static char str[16];

void HotSpotWidget4OnClick(tWidget *pWidget)
{
   count ++;

   //Get the last clicked coordinate X, Y
   int x = HotSpotLastXGet((tHotSpot*)pWidget);
```

```
    int y = HotSpotLastYGet((tHotSpot*)pWidget);

    //Set the value of count to the "Tag" property
    HotSpotTagSet((tHotSpot*)pWidget, count);

    //Get the value of the "Tag" property
    tag = HotSpotTagGet((tHotSpot*)pWidget);

    //Show the "Tag" and (x, y) on the Label 6 (in this example)
    usprintf(str, "tag=%d, x=%d, y=%d", tag, x, y);
    LabelTextSet(&LabelWidget6, str);
}
```

Remark

None

10.5. HotSpotLastYGet

Get the last clicked point's coordinate Y.
(The left-top vertex of the screen is the origin.)

Syntax

```
int HotSpotLastYGet (
        tHotSpot * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the HotSpot, to get the last clicked point's coordinate Y.

Return Values

The last clicked point's coordinate Y

Examples

[C]

```
int tag = 0;
int count = 0;
static char str[16];

void HotSpotWidget4OnClick(tWidget *pWidget)
{
   count ++;

   //Get the last clicked coordinate X, Y
   int x = HotSpotLastXGet((tHotSpot*)pWidget);
```

```
    int y = HotSpotLastYGet((tHotSpot*)pWidget);

    //Set the value of count to the "Tag" property
    HotSpotTagSet((tHotSpot*)pWidget, count);

    //Get the value of the "Tag" property
    tag = HotSpotTagGet((tHotSpot*)pWidget);

    //Show the "Tag" and (x, y) on the Label 6 (in this example)
    usprintf(str, "tag=%d, x=%d, y=%d", tag, x, y);
    LabelTextSet(&LabelWidget6, str);
}
```

Remark

None

10.6. CHECKBOXSELECTEDGET

Get the “Selected” property of the CheckBox.

Syntax

```
int CheckBoxSelectedGet (
        tCheckBox * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the CheckBox, to get the “Selected” property.

Return Values

0 is for the unchecked state (un-Selected).
1 is for the checked state (Selected).

Examples

[C]

```
int count = 0;
static char str2[16];

void BitButton5OnClick(tWidget *pWidget)
{
   count ++;

   if(count % 2)
   {
      //Set the "Selected" state to un-checked
```

```
        CheckBoxSelectedSet(&CheckBoxWidget4, 0);
        WidgetPaint((tWidget*)&CheckBoxWidget4);
    }
    else
    {
        //Set the "Selected" state to checked
        CheckBoxSelectedSet(&CheckBoxWidget4, 1);
        WidgetPaint((tWidget*)&CheckBoxWidget4);
    }

    //Show the "Selected" state on the Label 6 (in this example)
    usprintf(str2, "Sel: %d", CheckBoxSelectedGet(&CheckBoxWidget4));
    LabelTextSet(&LabelWidget6, str2);
}
```

Remark

None

10.7. CHECKBOXSELECTEDSET

Set the “Selected” property of the CheckBox.

Syntax

```
void CheckBoxSelectedSet (
        tCheckBox * pWidget,
        int bFlag
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the CheckBox, to set the “Selected” property.

bFlag

[in] Specify the state of the “Selected” property. 0 is for the unchecked state while 1 is for the checked state.

Return Values

None

Examples

[C]

```
int count = 0;
static char str2[16];

void BitButton5OnClick(tWidget *pWidget)
{
   count ++;

   if(count % 2)
```

```
    {
        //Set the "Selected" state to un-checked
        CheckBoxSelectedSet(&CheckBoxWidget4, 0);
        WidgetPaint((tWidget*)&CheckBoxWidget4);
    }
    else
    {
        //Set the "Selected" state to checked
        CheckBoxSelectedSet(&CheckBoxWidget4, 1);
        WidgetPaint((tWidget*)&CheckBoxWidget4);
    }

    //Show the "Selected" state on the Label 6 (in this example)
    usprintf(str2, "Sel: %d", CheckBoxSelectedGet(&CheckBoxWidget4));
    LabelTextSet(&LabelWidget6, str2);
}
```

Remark

None

10.8. LABELTEXTGET

Get the “Text” property of the Label.
Users can set the “Text” property in the inspector in the design time.

This function is used to get a static string from the “Text” property.
Because the widget has no buffer for the “Text”, the string must be static strings (static char[]). Also different Labels must use different static char arrays or these labels (with the same static char array pointers) may display the same content.

Syntax

```
const char * LabelTextGet (
      tLabel * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the Label, to get the “Text” property.

Return Values

A constant pointer to the static string to store the “Text” of the Label

Examples

[C]

```
int count = 0;
static char str[16];

void BitButton6OnClick(tWidget *pWidget)
{
   const char * p;
```

```
    count ++;

    //Set the value of count to the Text of the Label
    usprintf(str, "%d", count);
    LabelTextSet(&LabelWidget4, str);

    //Get the value of count to compare with 5
    //if count reaches 5, reset it to zero.
    p = LabelTextGet(&LabelWidget4);
    if(strcmp("5", p) == 0)
    {
        count = 0;
        usprintf(str, "5, reset to 0 !");
        LabelTextSet(&LabelWidget4, str);
    }
}
```

Remark

None

10.9. LABELTEXTSET

Set the “Text” property of the Label.
Users can set the “Text” property in the inspector in the design time.

This function is used to set a static string to the “Text” property.
Because the widget has no buffer for the “Text”, the string must be static strings (static char[]). Also different Labels must use different static char arrays or these labels (with the same static char array pointers) may display the same content.

Syntax

```
void LabelTextSet (
        tLabel * pWidget,
        char * text
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the Label, to set the “Text” property.

text

[in] Specify the static string of the “Text” property

Return Values

None

Examples

[C]

```
int count = 0;
static char str[16];
```

```
void BitButton6OnClick(tWidget *pWidget)
{
   const char * p;

   count ++;

   //Set the value of count to the Text of the Label
   usprintf(str, "%d", count);
   LabelTextSet(&LabelWidget4, str);

   //Get the value of count to compare with 5
   //if count reaches 5, reset it to zero.
   p = LabelTextGet(&LabelWidget4);
   if(strcmp("5", p) == 0)
   {
      count = 0;
      usprintf(str, "5, reset to 0 !");
      LabelTextSet(&LabelWidget4, str);
   }
}
```

Remark

None

10.10. TIMERENABLEDGET

Get the “Enabled” property of the Timer.

Syntax

```
int TimerEnabledGet (
        tTimer * pTimer,
);
```

Parameter

pTimer

[out] Specify the pointer to the Timer to set the “Enabled” property.

Return Values

0 is for the “Disabled” state while 1 is for the “Enabled” state.

Examples

[C]

```
static char str[16];

void Timer4OnExecute(tWidget *pWidget)
{
   hmi_Beep();
}

void BitButton5OnClick(tWidget *pWidget)
{
   //Get the status of the timer
   int bEnabled = TimerEnabledGet(&Timer4);
```

```
    if(bEnabled)
    {
        TimerEnabledSet(&Timer4, 0);    //disable the timer
        usprintf(str, "Timer disabled");
        LabelTextSet(&LabelWidget6, str);
    }
    else
    {
        TimerEnabledSet(&Timer4, 1);    //enable the timer
        usprintf(str, "Timer enabled");
        LabelTextSet(&LabelWidget6, str);
    }
}
```

Remark

None

10.11. TIMERENABLEDSET

Set the “Enabled” property of the Timer.

Syntax

```
void TimerEnabledSet (
        tTimer * pTimer,
        int bFlag
);
```

Parameter

pTimer

[out] Specify the pointer to the Timer to set the “Enabled” property.

bFlag

[in] Specify the state of the “Enabled” property. 0 is for the “Disabled” state while 1 is for the “Enabled” state.

Return Values

None

Examples

[C]

```
static char str[16];

void Timer4OnExecute(tWidget *pWidget)
{
   hmi_Beep();
}
```

```
void BitButton5OnClick(tWidget *pWidget)
{
   //Get the status of the timer
   int bEnabled = TimerEnabledGet(&Timer4);

   if(bEnabled)
   {
      TimerEnabledSet(&Timer4, 0);    //disable the timer
      usprintf(str, "Timer disabled");
      LabelTextSet(&LabelWidget6, str);
   }
   else
   {
      TimerEnabledSet(&Timer4, 1);    //enable the timer
      usprintf(str, "Timer enabled");
      LabelTextSet(&LabelWidget6, str);
   }
}
```

Remark

None

10.12. TIMERINTERVALGET

Get the “Interval” property of the Timer. (The interval of a timer is its period.)

Syntax

```
unsigned long TimerIntervalGet (
        tTimer * pTimer,
);
```

Parameter

pTimer

[out] Specify the pointer to the Timer to set the “Interval” property.

Return Values

The Interval (period) of the Timer widget

Examples

[C]

```
// Circularly assign 1000 ~ 5000 (ms) to the interval of a Timer
// The Timer beeps from every 1 second to 5 second
#define ONE_SECOND      1000

int count = 0;
void HotSpot68OnClick(tWidget *pWidget)
{
    static char str[16];
    unsigned long intervalGet;

    count++;
```

```
    unsigned long interval = (count % 5 + 1) * ONE_SECOND;
    TimerIntervalSet(&Timer69, interval);

    intervalGet = TimerIntervalGet(&Timer69);
    usprintf(str, "beep every %d sec", intervalGet);
    LabelTextSet(&Label64, str);
}

void Timer69OnExecute(tWidget *pWidget)
{
    hmi_Beep();
}
```

Remark

None

10.13. TimerIntervalSet

Set the “Interval” property of the Timer. (The interval of a timer is its period.)

Syntax

```
void TimerIntervalSet (
        tTimer * pTimer,
        unsigned long ulInterval
);
```

Parameter

pTimer

[out] Specify the pointer to the Timer to set the “Interval” property.

bFlag

[in] Specify the “Interval” property of the Timer widget.

Return Values

None

Examples

[C]

```
// Circularly assign 1000 ~ 5000 (ms) to the interval of a Timer
// The Timer beeps from every 1 second to 5 second
#define ONE_SECOND     1000

int count = 0;
void HotSpot68OnClick(tWidget *pWidget)
{
   static char str[16];
   unsigned long intervalGet;
```

```
    count++;
    unsigned long interval = (count % 5 + 1) * ONE_SECOND;
    TimerIntervalSet(&Timer69, interval);

    intervalGet = TimerIntervalGet(&Timer69);
    usprintf(str, "beep every %d sec", intervalGet);
    LabelTextSet(&Label64, str);
}

void Timer69OnExecute(tWidget *pWidget)
{
    hmi_Beep();
}
```

Remark

None

10.14. FUNCTIONS FOR TAG

This section introduces functions (actually macros) of the Tag property for widgets as shown below. For each widget, there are two functions for the Tag property, xTagSet and xTagGet. (x is the widget name)

We can set the “Tag” property in the design time, and then read it in the run-time. The "Tag" property would be useful when using several widgets in a single event handler function. This property can be used to indicate which widget is clicked at this time. It looks like a widget array index.

Widget	Function
TextPushButton without an associated ObjectList	int TextButtonTagGet (tTextButton * pWidget);
	void TextButtonTagSet (tTextButton * pWidget, int tag);
TextPushButton with an associated ObjectList	void ObjButtonTagSet (tObjButton * pWidget, int tag);
	int ObjButtonTagGet (tObjButton * pWidget);
Slider	void SliderTagSet (tSlider * pWidget, int tag);
	int SliderTagGet (tSlider * pWidget);
BitButton	void BitButtonTagSet (tBitButton * pWidget, int tag);
	int BitButtonTagGet (tBitButton * pWidget);
HotSpot	void HotSoptTagSet (tHotSpot * pWidget, int tag);
	int HotSoptTagGet (tHotSpot * pWidget);
CheckBox	void CheckBoxTagSet (tCheckBox * pWidget, int tag);
	int CheckBoxTagGet (tCheckBox * pWidget);

Note: We say a widget is associated with an ObjectList if its “RefObject” property is set to an ObjectList.

We take TextPushButton for example to introduce how to use these functions.

Syntax

```
void TextButtonTagSet (
```

```
        tTextButton * pWidget,
        int tag
    );

    int TextButtonTagGet (
        tTextButton * pWidget
    );
```

Parameter

pWidget

[out] Specify the pointer to the widget (tTextButton is used for TextPushButton without an associated ObjectList) to set/get the Tag property.

tag

[in] Specify the value of the “Tag” property.

Return Values

The returning value of TextButtonTagGet is the value of the “Tag” property.

Examples

[C]

```
int tag = 0;
int count = 0;
static char str[16];
static char str2[16];

void TextPushButton4OnClick(tWidget *pWidget)
{
   count ++;

   //Set the value of count to the "Tag" property
   TextButtonTagSet((tTextButton*)pWidget, count);
```

```
    //Get the value of the "Tag" property
    tag = TextButtonTagGet((tTextButton*)pWidget);

    //Set the "Text" property of the TextPushButton 4
    usprintf(str, "%d", tag);
    TextButtonTextSet((tTextButton*)pWidget, str);

    //Get the "Text" property of the TextPushButton
    //And then show it on the Label 5 (in this example)
    strcpy(str2, TextButtonTextGet((tTextButton*)pWidget));
    LabelTextSet(&LabelWidget5, str2);
}
```

Remark

None

10.15. FUNCTIONS FOR ENABLED

This section introduces functions (actually macros) of the Enabled property for widgets as shown below. For each widget, there are two functions for the Enabled property, xEnabledSet and xEnabledGet. (x is the widget name)

Widget	Function
TextPushButton without an associated ObjectList	void TextButtonEnabledSet(tTextButton *pWidget, BOOL bEnabled);
	BOOL TextButtonEnabledGet(tTextButton *pWidget);
TextPushButton with an associated ObjectList	void ObjButtonEnabledSet(tObjButton *pWidget, BOOL bEnabled);
	BOOL ObjButtonEnabledGet(tObjButton *pWidget);
Slider	void SliderEnabledSet(tSlider *pWidget, BOOL bEnabled);
	BOOL SliderEnabledGet(tSlider *pWidget);
BitButton	void BitButtonEnabledSet(tBitButton *pWidget, BOOL bEnabled);
	BOOL BitButtonEnabledGet(tBitButton *pWidget);
HotSpot	void HotSpotEnabledSet(tHotSpot *pWidget, BOOL bEnabled);
	BOOL HotSpotEnabledGet(tHotSpot *pWidget);
CheckBox	void CheckBoxEnabledSet(tCheckBox *pWidget, BOOL bEnabled);
	BOOL CheckBoxEnabledGet(tCheckBox *pWidget);
Label	void LabelEnabledSet(tLabel *pWidget, BOOL bEnabled);
	BOOL LabelEnabledGet(tLabel *pWidget);

Note: We say a widget is associated with an ObjectList if its “RefObject” property is set to an ObjectList.

We take TextPushButton for example to introduce how to use these functions.

Syntax

```
void TextButtonEnabledSet (
    tTextButton *pWidget,
```

```
        BOOL bEnabled
    );

    BOOL TextButtonEnabledGet(
        tTextButton *pWidget
    );
```

Parameter

pWidget

[out] Specify the pointer to the widget (tTextButton is used for TextPushButton without an associated ObjectList) to set/get the Enabled property.

bEnabled

[in] Specify the “Enabled” property of the widget.

Possbile value: TRUE or FALSE

Return Values

The returning value of TextButtonEnabledGet is the status of the Enabled property.

TRUE: the widget is enabled

FALSE: the widget is not enabled

Examples

[C]

```
int status = 0;

void TextPushButton5OnClick(tWidget *pWidget)
{
   hmi_Beep();
}

void BitButton4OnClick(tWidget *pWidget)
{
      BOOL en = FALSE;
      BOOL vi = FALSE;
      tTextButton * pt = &TextPushButton5;
```

```
    static char msg[32];

    status++;

    switch (status % 4)
    {
        default:
        case 0:
            TextButtonEnabledSet(pt, TRUE);
            TextButtonVisibleSet(pt, TRUE);
            break;
        case 1:
            TextButtonEnabledSet(pt, TRUE);
            TextButtonVisibleSet(pt, FALSE);
            break;
        case 2:
            TextButtonEnabledSet(pt, FALSE);
            TextButtonVisibleSet(pt, TRUE);
            break;
        case 3:
            TextButtonEnabledSet(pt, FALSE);
            TextButtonVisibleSet(pt, FALSE);
            break;
    }

    // WidgetPaint must be executed to redraw full screen after
    // changing the visibility of any widget.
    WidgetPaint(WIDGET_ROOT);

    en = TextButtonEnabledGet(pt);
    vi = TextButtonVisibleGet(pt);
    strcpy(msg, en ? "Enabled, " : "Disenabled, ");
    strcat(msg, vi ? "Visible" : "Unvisible");
    LabelTextSet(&Label8, msg);
}
```

Remark

None

10.16. FUNCTIONS FOR VISIBLE

This section introduces functions (actually macros) of the Visible property for widgets as shown below. For each widget, there are two functions for the Visible property, xVisibleSet and xVisibleGet. (x is the widget name)

Widget	Function
TextPushButton without an associated ObjectList	void TextButtonVisibleSet(tTextButton *pWidget, BOOL bVisible);
	BOOL TextButtonVisibleGet(tTextButton *pWidget);
TextPushButton with an associated ObjectList	void ObjButtonVisibleSet(tObjButton *pWidget, BOOL bVisible);
	BOOL ObjButtonVisibleGet(tObjButton *pWidget);
Slider	void SliderVisibleSet(tSlider *pWidget, BOOL bVisible);
	BOOL SliderVisibleGet(tSlider *pWidget);
BitButton	void BitButtonVisibleSet(tBitButton *pWidget, BOOL bVisible);
	BOOL BitButtonVisibleGet(tBitButton *pWidget);
CheckBox	void CheckBoxVisibleSet(tCheckBox *pWidget, BOOL bVisible);
	BOOL CheckBoxVisibleGet(tCheckBox *pWidget);
Label	void LabelVisibleSet(tLabel *pWidget, BOOL bVisible);
	BOOL LabelVisibleGet(tLabel *pWidget);

Note: We say a widget is associated with an ObjectList if its “RefObject” property is set to an ObjectList.

We take TextPushButton for example to introduce how to use these functions.

Syntax

```
void TextButtonVisibleSet (
        tTextButton *pWidget,
        BOOL bVisible
);

BOOL TextButtonVisibleGet(
```

```
        tTextButton *pWidget
    );
```

Parameter

pWidget

[out] Specify the pointer to the widget (tTextButton is used for TextPushButton without an associated ObjectList) to set/get the Visible property.

bVisible

[in] Specify the “Visible” property of the widget.

Possbile value: TRUE or FALSE

Return Values

The returning value of TextButtonVisibleGet is the status of the Visible property.

TRUE: the widget is visible

FALSE: the widget is not visible

Examples

[C]

```
int status = 0;

void TextPushButton5OnClick(tWidget *pWidget)
{
  hmi_Beep();
}

void BitButton4OnClick(tWidget *pWidget)
{
    BOOL en = FALSE;
    BOOL vi = FALSE;
    tTextButton * pt = &TextPushButton5;
    static char msg[32];
```

```
    status++;

    switch (status % 4)
    {
        default:
        case 0:
            TextButtonEnabledSet(pt, TRUE);
            TextButtonVisibleSet(pt, TRUE);
            break;
        case 1:
            TextButtonEnabledSet(pt, TRUE);
            TextButtonVisibleSet(pt, FALSE);
            break;
        case 2:
            TextButtonEnabledSet(pt, FALSE);
            TextButtonVisibleSet(pt, TRUE);
            break;
        case 3:
            TextButtonEnabledSet(pt, FALSE);
            TextButtonVisibleSet(pt, FALSE);
            break;
    }

    // WidgetPaint must be executed to redraw full screen after
    // changing the visibility of any widget.
    WidgetPaint(WIDGET_ROOT);

    en = TextButtonEnabledGet(pt);
    vi = TextButtonVisibleGet(pt);
    strcpy(msg, en ? "Enabled, " : "Disenabled, ");
    strcat(msg, vi ? "Visible" : "Unvisible");
    LabelTextSet(&Label8, msg);
}
```

Remark

WidgetPaint(WIDGET_ROOT) must be called to redraw all widgets after changing

visibility of the widget to make the changing take effect. WidgetPaint costs system resources so we don't suggest users frequently changing the visibility of widgets.

11. FLASH API

This chapter introduces API functions for flash reading and writing.

For users' convenience, there are two sets of API functions for data storage in the flash on the TouchPAD devices. One is for the MCU (micro-controller unit) internal flash and the other is the external serial flash (total 8 MB).

To user these features, install the HMIWorks software with version 2.03 or above.
ftp://ftp.icpdas.com/pub/cd/touchpad/setup/

No.	1	2
Target Flash	MCU internal flash	External serial flash
Possible Target Device	All devices in the TouchPAD series	All devices in the TouchPAD series, except TPD-280 and TPD-283 (for those having external flash)
API Functions Provided	hmi_UserParamsGet, hmi_UserParamsSet	hmi_UserFlashReadEx, hmi_UserFlashWriteEx, hmi_UserFlashConfig, hmi_UserFlashErase
Size of Storage	256 byte	4 KB ~ 7 MB
Suggested Users	Any TouchPAD users	**For advanced users only.** Any undetermined use will damage the application image.

11.1. HMI_USERPARAMSGET

Get data from the 256-byte parameter area in the MCU (MicroController Unit) internal flash.

Syntax

```
int hmi_UserParamsGet(
    int iOffset,
    int iSize,
    char *pcBuffer
);
```

Parameter

iOffset

[in] Specify the offset to the base of the 256-byte parameter area to read data from it.

Possible range: 0 ~ 255. (iOffset + iSize cannot be larger than 256)

iSize

[in] Specify the size of the data to read from the 256-byte parameter area.

Possible range: 0 ~ 255. (iOffset + iSize cannot be larger than 256)

pcBuffer

[out] Specify the pointer to the char array to store the data got from the 256-byte parameter area.

Return Values

1 (true) = OK;
0 (false) = Failure

Examples

[C]

```
void BitButton4OnClick(tWidget *pWidget)
{
    static int iLog = 0;
    static char szMsg[30];

    // User Parameter Area is 256 bytes only.
    // Get the last record
    if ( hmi_UserParamsGet(0, 4, (char *)&iLog) )
    {
        iLog++;
        usprintf(szMsg, "%d", iLog);
        LabelTextSet(&Label5, szMsg);

        // Update data
        if ( hmi_UserParamsSet(0, 4, (char *)&iLog) )
        {
            hmi_Beep();
        }
    }
}
```

Remark

1. There is a write/erase limit for the flashes.
   Frequent uses may damage the flash.
2. The old g_sParameters.UserParamsData is no longer available.

11.2. HMI_USERPARAMSSET

Set data to the 256-byte parameter area in the MCU (MicroController Unit) internal flash.

Syntax

```
int hmi_UserParamsSet (
    int iOffset,
    int iSize,
    char *pcBuffer
);
```

Parameter

iOffset

[in] Specify the offset to the base of the 256-byte parameter area to write data to it.

Possible range: 0 ~ 255. (iOffset + iSize cannot be larger than 256)

iSize

[in] Specify the size of the data to write to the 256-byte parameter area.

Possible range: 0 ~ 255. (iOffset + iSize cannot be larger than 256)

pcBuffer

[out] Specify the pointer to the char array which is used to write to the 256-byte parameter area.

Return Values

1 (true) = OK;
0 (false) = Failure

Examples

[C]

```
void BitButton4OnClick(tWidget *pWidget)
{
    static int iLog = 0;
    static char szMsg[30];

    // User Parameter Area is 256 bytes only.
    // Get the last record
    if ( hmi_UserParamsGet(0, 4, (char *)&iLog) )
    {
        iLog++;
        usprintf(szMsg, "%d", iLog);
        LabelTextSet(&Label5, szMsg);

        // Update data
        if ( hmi_UserParamsSet(0, 4, (char *)&iLog) )
        {
            hmi_Beep();
        }
    }
}
```

Remark

1. There is a write/erase limit for the flashes.
   Frequent uses may damage the flash.
2. The old g_sParameters.UserParamsData is no longer available.

11.3. HMI_USERFLASHCONFIG

Configure how many blocks of the external serial flash can be used for reading and writing. Each block has size of 4KB.

Syntax

```
int hmi_UserFlashConfig (
      unsigned long iNumberOfBlocks
);
```

Parameter

iNumberOfBlocks

[in] Specify the number of blocks to be used for data storage.
Possible range: 1 ~ 1792 blocks (= 4 KB ~ 7 MB, 0 block = disable)

Number of Blocks	Size	Number of Blocks	Size
1	4 KB	768	3 MB
2	8 KB	1024	4 MB
4	16 KB	1280	5 MB
8	32 KB	1536	6 MB
256	1 MB	1792	7 MB
512	2 MB		

Note:

1. The application image is put on the external flash started from the lowest address (that is, zero). Be careful to configure an area in order not to overwrite the application image.
2. The area claimed by hmi_UserFlashConfig occupies the highest addresses of the external flash and started from its lowest address inside the claimed area.

Return Values

The number of blocks that are claimed

Examples

[C]

```
void btnConfig4OnClick(tWidget *pWidget)
{
    // Enable 1792 blocks (=7 MB) for reading/writing by user
    if ( hmi_UserFlashConfig(1792) == 1792 )
        LabelTextSet(&Label7, "Configure OK");
}

void btnWrite5OnClick(tWidget *pWidget)
{
    int i;
    for (i=0; i<1792; i++)    // loop through the upper 7 MB
    {
        // MUST: Erase the external serial flash block
        hmi_UserFlashErase(i);
        // Write 4 bytes to each block for example
        hmi_UserFlashWriteEx(i, 0, 4, (char *)&i);
    }
}

void btnRead6OnClick(tWidget *pWidget)
{
    static char szMsg[30];
    static int iBlock = 0;
    int iVal;

    // Read 4 bytes from a block
    if ( hmi_UserFlashReadEx(iBlock, 0, 4, (char *)&iVal) )
    {
        usprintf(szMsg, "%d", iVal);
        LabelTextSet(&Label7, szMsg);
```

```
    }
    else
        LabelTextSet(&Label7, "Read Error");

    iBlock++;
}
```

Remark

1. There is a write/erase limit for the flashes.
   Frequent uses may damage the flash.
2. The old functions, such as hmi_UserFlashRead and hmi_UserFlashWrite, are depreciated.
3. **For advanced users only**. Any wrongly-configured area will overwrite the application area and damage the application image.

11.4. HMI_USERFLASHREADEX

Read data from the configured user flash area by hmi_UserFlashConfig.

Syntax

```
unsigned long hmi_UserFlashReadEx(
    unsigned long iBlock,
    unsigned long iOffset,
    unsigned long iLength,
    char *pBuffer
);
```

Parameter

iBlock

[in] Specify the block index to read data from the configured user flash area. Possible range: 0 to iNumberOfBlocks - 1. (where iNumberOfBlocks is the number of blocks claimed by the hmi_UserFlashConfig function)

iOffset

[in] Specify the offset to the base of the block to read data from that block which has index equal to iBlock.

Possible range: 0 ~ 4095. (iOffset + iLength cannot be larger than 4096)

iLength

[in] Specify the size of the data to read from the block of the flash whose index is iBlock.

Possible range: 1 ~ 4096. (iOffset + iLength cannot be larger than 4096)

pcBuffer

[out] Specify the pointer to the char array to store the data read from the configured user flash area.

Return Values

Data length read;
0 (false) = failure

Examples

[C]

```
void btnConfig4OnClick(tWidget *pWidget)
{
    // Enable 1792 blocks (=7 MB) for reading/writing by user
    if ( hmi_UserFlashConfig(1792) == 1792 )
        LabelTextSet(&Label7, "Configure OK");
}

void btnWrite5OnClick(tWidget *pWidget)
{
    int i;
    for (i=0; i<1792; i++)    // loop through the upper 7 MB
    {
        // MUST: Erase the external serial flash block
        hmi_UserFlashErase(i);
        // Write 4 bytes to each block for example
        hmi_UserFlashWriteEx(i, 0, 4, (char *)&i);
    }
}

void btnRead6OnClick(tWidget *pWidget)
{
    static char szMsg[30];
    static int iBlock = 0;
    int iVal;

    // Read 4 bytes from a block
    if ( hmi_UserFlashReadEx(iBlock, 0, 4, (char *)&iVal) )
    {
        usprintf(szMsg, "%d", iVal);
        LabelTextSet(&Label7, szMsg);
```

```
    }
    else
        LabelTextSet(&Label7, "Read Error");

    iBlock++;
}
```

Remark

1. There is a write/erase limit for the flashes.
   Frequent uses may damage the flash.
2. The old functions, such as hmi_UserFlashRead and hmi_UserFlashWrite, are depreciated.
3. **For advanced users only**. Any wrongly-configured area will overwrite the application area and damage the application image.

11.5. HMI_USERFLASHERASE

Erase a block of data in the configured user flash area by hmi_UserFlashConfig. This function **MUST** be used before hmi_UserFlashWriteEx is used.

Syntax

```
int hmi_UserFlashErase (
      unsigned long iBlock
);
```

Parameter

iBlock

[in] Specify the block index to erase in the configured user flash area. Possible range: 0 to iNumberOfBlocks - 1. (where iNumberOfBlocks is the number of blocks claimed by the hmi_UserFlashConfig function)

Return Values

TRUE = Success;
0 (FALSE) = Failure

Examples

[C]

```
void btnConfig4OnClick(tWidget *pWidget)
{
    // Enable 1792 blocks (=7 MB) for reading/writing by user
    if ( hmi_UserFlashConfig(1792) == 1792 )
        LabelTextSet(&Label7, "Configure OK");
}
```

```
void btnWrite5OnClick(tWidget *pWidget)
{
    int i;
    for (i=0; i<1792; i++)    // loop through the upper 7 MB
    {
        // MUST: Erase the external serial flash block
        hmi_UserFlashErase(i);
        // Write 4 bytes to each block for example
        hmi_UserFlashWriteEx(i, 0, 4, (char *)&i);
    }
}

void btnRead6OnClick(tWidget *pWidget)
{
    static char szMsg[30];
    static int iBlock = 0;
    int iVal;

    // Read 4 bytes from a block
    if ( hmi_UserFlashReadEx(iBlock, 0, 4, (char *)&iVal) )
    {
        usprintf(szMsg, "%d", iVal);
        LabelTextSet(&Label7, szMsg);
    }
    else
        LabelTextSet(&Label7, "Read Error");

    iBlock++;
}
```

Remark

1. There is a write/erase limit for the flashes.
   Frequent uses may damage the flash.
2. The old functions, such as hmi_UserFlashRead and hmi_UserFlashWrite, are depreciated.

3. **For advanced users only**. Any wrongly-configured area will overwrite the application area and damage the application image.

11.6. HMI_USERFLASHWRITEEX

Write data to the configured user flash area by hmi_UserFlashConfig.

Syntax

```
unsigned long hmi_UserFlashWriteEx (
    unsigned long iBlock,
    unsigned long iOffset,
    unsigned long iLength,
    char *pBuffer
);
```

Parameter

iBlock

[in] Specify the block index to write data to the configured user flash area.
Possible range: 0 to iNumberOfBlocks - 1. (where iNumberOfBlocks is the number of blocks claimed by the hmi_UserFlashConfig function)

iOffset

[in] Specify the offset to the base of the block to write data to that block which has index equal to iBlock.
Possible range: 0 ~ 4095. (iOffset + iLength cannot be larger than 4096)

iLength

[in] Specify the size of the data to write to the block of the flash whose index is iBlock.
Possible range: 1 ~ 4096. (iOffset + iLength cannot be larger than 4096)

pcBuffer

[out] Specify the pointer to the char array which is used to write to the configured user flash area.

Return Values

Data length written

Examples

[C]

```
void btnConfig4OnClick(tWidget *pWidget)
{
    // Enable 1792 blocks (=7 MB) for reading/writing by user
    if ( hmi_UserFlashConfig(1792) == 1792 )
        LabelTextSet(&Label7, "Configure OK");
}

void btnWrite5OnClick(tWidget *pWidget)
{
    int i;
    for (i=0; i<1792; i++)    // loop through the upper 7 MB
    {
        // MUST: Erase the external serial flash block
        hmi_UserFlashErase(i);
        // Write 4 bytes to each block for example
        hmi_UserFlashWriteEx(i, 0, 4, (char *)&i);
    }
}

void btnRead6OnClick(tWidget *pWidget)
{
    static char szMsg[30];
    static int iBlock = 0;
    int iVal;

    // Read 4 bytes from a block
    if ( hmi_UserFlashReadEx(iBlock, 0, 4, (char *)&iVal) )
    {
        usprintf(szMsg, "%d", iVal);
        LabelTextSet(&Label7, szMsg);
    }
```

```
    else
        LabelTextSet(&Label7, "Read Error");

    iBlock++;
}
```

Remark

1. There is a write/erase limit for the flashes.
   Frequent uses may damage the flash.
2. The old functions, such as hmi_UserFlashRead and hmi_UserFlashWrite, are depreciated.
3. **For advanced users only**. Any wrongly-configured area will overwrite the application area and damage the application image.

12. MISCELLANEOUS API

This chapter provides APIs that are not classified.

12.1. HMI_BEEP

Sound the beep.

Syntax

```
void hmi_Beep();
```

Parameter

None

Return Values

None

Examples

[C]

```
hmi_Beep ();
```

Remark

None

12.2. HMI_CONFIGBEEP

Configure the beep of the TPD-430.

Syntax

```
void hmi_ConfigBeep(
        unsigned short usFreq,
        unsigned short usTicksMS
);
```

Parameter

usFreq

[in] Specify the pitch (the frequency value) of the beep. Range: 30 ~ 4,000 Hz.

usTicksMS

[in] Specify the elapsing interval of the beep. Range: 1 ~ 30,000 ms.

Return Values

None

Examples

[C]

```
//beep at the the specified pitch 100 Hz, and 5ms long a beep
hmi_ConfigBeep (100, 5);
```

Remark

None

12.3. HMI_GETROTARYID

Get the ID of the rotary switch.

Syntax

```
int hmi_GetRotaryID();
```

Parameter

None

Return Values

The ID of the rotary switch in the back of the TouchPAD
The possible values are 0 ~ 9.

Examples

[C]

```
int id = hmi_GetRotaryID();
```

Remark

None

12.4. HMI_SETLED

Set the LED indicator of TPD-430/TPD-430-EU.

Syntax

```
void hmi_SetLED (
      int status
);
```

Parameter

status

[in] Specify the status of the LED indicator. There are two states of the LED indicator, HMI_LED_ON and HMI_LED_OFF.

Return Values

None

Examples

[C]

```
int count = 0;

if(count % 2)
  hmi_SetLED (HMI_LED_ON);    //turn on the LED indicator
else
  hmi_SetLED (HMI_LED_OFF);    //turn off the LED indicator
```

Remark

None

12.5. HMI_BACKLIGHTSET

Set the brightness of the TPD-430/TPD-430-EU.
Or turn on or off the back light of other devices in the TouchPAD series.

Syntax

```
void hmi_BacklightSet (
      unsigned char ucBrightness
);
```

Parameter

ucBrightness

[in] Specify the brightness of the TPD-430/TPD-430-EU.
Range: 0 ~ 255. 0=the darkest, ..., 255=the brightest.
Or specify the status of the backlight of other devices in the TouchPAD series.
0: Turn off the backlight; 1: Turn on the backlight

Return Values

None

Examples

[C]

```
unsigned char b = 128; // For TPD-430

hmi_BacklightSet(b);    //set the brightness to 128
```

Remark

None

12.6. HMI_READPANELKEY

Get the states of the panel key on the front panel of the VPD-130 devices.

Syntax

```
int hmi_ReadPanelKey ();
```

Parameter

None

Return Values

When the state of the key is pressed, the corresponding value in the table below is returned.

Key	Value	Define #1	Define #2
	1	PANEL_KEY_F1	PANEL_KEY_UP

	2	PANEL_KEY_F2	PANEL_KEY_DOWN
	4	PANEL_KEY_F3	PANEL_KEY_LEFT
	8	PANEL_KEY_F4	PANEL_KEY_RIGHT
	16	PANEL_KEY_F5	PANEL_KEY_ENTER

Examples

[C]

```
void Timer6OnExecute(tWidget *pWidget)
{
   static char str[20];
   int reading = -1;

   reading = hmi_ReadPanelKey();

   usprintf(str, "%d", reading);
   LabelTextSet(&Label5, str);
}
```

Remark

Only VPD-130 has panel keys in the TouchPAD series.

12.7. HMI_GETTICKCOUNT

Get the tick count of the TouchPAD.

Syntax

```
int hmi_GetTickCount ();
```

Parameter

None

Return Values

The system tick count in the **unit** of **milisecond**. The **resolution** is about 10 ms. That is, this hmi_GetTickCount function is based on a fixed time interval of 100 ticks/second.

Examples

[C]

```
void BitButton4OnClick(tWidget *pWidget)
{
   static char str[16];
   int tick = hmi_GetTickCount();

   usprintf(str, "tick= %d", tick);
   LabelTextSet(&LabelWidget5, str);
}
```

Remark

None

12.8. HMI_DELAYUS

Delay a specified interval in micro-second.

Syntax

```
void hmi_DelayUS(
      unsigned long ulDelayTime
);
```

Parameter

ulDelayTime

[in] Specify the delay time in micro-second. Suggested range: 1 ~ 50 (us) **in order not to block the system**.

Return Values

None

Examples

[C]

```
hmi_DelayUS(10); //delay 10 us
```

Remark

TouchPAD is not a multitasking system.
Delay too much severely blocks the system.

12.9. HMI_GETDATETIME

Get the date and time from the RTC chip on the TouchPAD devices.
Not all the devices in the TouchPAD series equips with a RTC chip.

Syntax

```
void hmi_GetDateTime(
    int *year,
    int *month,
    int *day,
    int *hour,
    int *minute,
    int *second
);
```

Parameter

year
[out] Specify the pointer to the integer used to represent the year.
month
[out] Specify the pointer to the integer used to represent the month.
day
[out] Specify the pointer to the integer used to represent the day.
hour
[out] Specify the pointer to the integer used to represent the hour.
minute
[out] Specify the pointer to the integer used to represent the minute.
second
[out] Specify the pointer to the integer used to represent the second.

Return Values

None

Examples

[C]

```
void TextPushButton4OnClick(tWidget *pWidget)
{
   static char temp[32];
   int yy, mm, dd, hh, nn, ss;

   // set date and time to RTC (Real Time Clock)
   // Set to 13:31:22, Jan 2nd, 2012
   yy = 2012;    mm = 1;    dd = 2;
   hh = 13;       nn =31;    ss = 22;
   hmi_SetDateTime(yy, mm, dd, hh, nn, ss);

   // get date and time from RTC
   hmi_GetDateTime(&yy, &mm, &dd, &hh, &nn, &ss);

   usprintf(temp, "%04d/%02d/%02d %02d:%02d:%02d\n", yy, mm, dd,
hh, nn, ss);
   LabelTextSet(&Label5, temp);
}
```

Remark

A RTC chip is required for hmi_GetDateTime and hmi_SetDateTime.

12.10. HMI_SETDATETIME

Set the date and time to the RTC chip on the TouchPAD devices.
Not all the devices in the TouchPAD series equips with a RTC chip.

Syntax

```
void hmi_SetDateTime(
    int year,
    int month,
    int day,
    int hour,
    int minute,
    int second
);
```

Parameter

year
[in] Specify the value of the year to set to the RTC chip.
month
[in] Specify the value of the month to set to the RTC chip.
day
[in] Specify the value of the day to set to the RTC chip.
hour
[in] Specify the value of the hour to set to the RTC chip.
minute
[in] Specify the value of the minute to set to the RTC chip.
second
[in] Specify the value of the second to set to the RTC chip.

Return Values

None

Examples

[C]

```
void TextPushButton4OnClick(tWidget *pWidget)
{
   static char temp[32];
   int yy, mm, dd, hh, nn, ss;

   // set date and time to RTC (Real Time Clock)
   // Set to 13:31:22, Jan 2nd, 2012
   yy = 2012;    mm = 1;    dd = 2;
   hh = 13;       nn =31;    ss = 22;
   hmi_SetDateTime(yy, mm, dd, hh, nn, ss);

   // get date and time from RTC
   hmi_GetDateTime(&yy, &mm, &dd, &hh, &nn, &ss);

   usprintf(temp, "%04d/%02d/%02d %02d:%02d:%02d\n", yy, mm, dd,
hh, nn, ss);
   LabelTextSet(&Label5, temp);
}
```

Remark

A RTC chip is required for hmi_GetDateTime and hmi_SetDateTime.

12.11. CRC16

Get a 16-bit cyclic redundancy check value of an array of bytes.

Syntax

```
unsigned short CRC16(
    const unsigned char *nData,
    unsigned short wLength
);
```

Parameter

nData

[in] Specify the pointer to the data to calculate its CRC value

wLength

[in] the length of data to calculate

Return Values

The 16-bit CRC value

Examples

[C]

```
//
// Use the CRC16 in the Modbus coil command
//
// DO command (Modbus Coil)
//str[0] = slave address
//str[1] = function
//str[2,3] = start address
//str[4,5] = length
//str[6] = Byte ((length+7)/8)
```

```
//str[7] = value
//str[9, 10] = checksum

char recv[1+1+2+2+2];

// a cmd of DO (Modbus coil) for example
unsigned char cmd[] = {0x01, 0x0F, 0x00, 0x00, 0x00, 0x10, 0x03, 0x11,
0, 0};

// calculate the 16-bit CRC value of the cmd
unsigned short ret_crc = CRC16(cmd, sizeof(cmd)-2);

// put the CRC value in the last 2 bytes of the cmd
cmd[sizeof(cmd)-2]  = ret_crc & 0xff;
cmd[sizeof(cmd)-1]  = ret_crc >>8;
recv[0] = 0;

HANDLE h = uart_Open("COM1,115200,N,8,1");
uart_BinSendCmd(h, cmd, sizeof(cmd), recv, sizeof(recv));
uart_Close(h);
```

Remark

None

12.12. FLOATTOSTR

Convert a floating-point number of the type float to string.

Syntax

```
int FloatToStr (
        char *buf,
        float fVal,
        int precision
);
```

Parameter

buf

[out] Specify the pointer to a char array to represent the floating-point number.

fVal

[in] the floating-point number to be converted

precision

[in] the number of digit after the decimal point (round off)

Possible range: 0 ~ 5.

Return Values

0 (false), 1 (OK)

Examples

[C]

```
void BitButton4OnClick(tWidget *pWidget)
{
    static char buf[16];
    float fVal = -2.3617;
```

```
    int precision = 3;    // 0 ~ 5 assigned by user

    FloatToStr(buf, fVal, precision);
    LabelTextSet(&Label5, buf);    // the result is -2.362
}
```

Remark

None

12.13. HMI_LCDIdleSetCallback

This function is used to configure the callback function as TouchPAD's idle or wake up event handlers.

Syntax

```
void hmi_LCDIdleSetCallback(
        unsigned long ulTimeoutMS,
        PFN_VOIDCALLBACK pfnIdle,
        PFN_VOIDCALLBACK pfnWakeup
);
```

Parameter

ulTimeoutMS

[in] Specify the timeout value to enter the idle mode. (unit: ms)

pfnIdle

[out] Specify the function pointer to the callback function that is fired when entering the idle state.

pfnWakeup

[out] Specify the function pointer to the callback function that is fired when the touch screen wakes up.

Note: The prototype of the idle or wakeup callback functions is defined as below:
typedef void (*PFN_VOIDCALLBACK)(void);

Return Values

None

Examples

[C]

```
// Callback function when TouchPAD entering the idle state.
void TLCDIdle()
{
    // go to the frame of the screen saver when idle
    hmi_GotoFrameByName("FScreenSaver");
}

// Callback function when TouchPAD wakes up.
void TLCDWakeup()
{
    hmi_BacklightSet(255); // Enable LCD
    hmi_GotoFrameByName("FHome");
}

void FHome2OnCreate()
{
    // Configuration for the callback functions when idle atfter 5 seconds.
    hmi_LCDIdleSetCallback(5 * 1000, TLCDIdle, TLCDWakeup);
}
```

Remark

How to disable the screen saver?
Simply set the first argument of hmi_LCDIdleSetCallback to zero and run it.
That is,

---

hmi_LCDIdleSetCallback(**0**, TLCDIdle, TLCDWakeup);

12.14. HMI_LCDIDLESTATUSRESET

This function is used to reset the idle state of TouchPAD and enable the next callback when idle state is entered again.

When to use? For example, assume that we have an application that make TouchPAD change from the idle page to the alarm page when needed. And after the alarm turned off, TouchPAD then goes back to the home page automatically.

In this case at this time, hmi_LCDIdleStatusReset is used to tell TouchPAD to start to listen to the idle callback again for the next possible idleness because if you don't do this, TouchPAD is still in its idle state and cannot turn off the screen.

Remember that the idle and wakeup state is based on touch or no-touch on the LCD. When in the idle state, the program is still running, not sleeping.

Syntax

```
void hmi_LCDIdleStatusReset();
```

Parameter

None

Return Values

None

Examples

[C]

```
hmi_LCDIdleStatusReset();
```

Remark

None