HMIWorks
API Reference

Version 1.11, September 2011

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]

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

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

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. TCP API

This chapter provides TCP APIs.

3.1. HMI_TCPINIT

Initialize the session data for TCP communications.

Syntax

```
void hmi_TCPInit ();
```

Parameter

None

Return Values

None

Examples

[C]

```
hmi_TCPInit ();
```

Remark

None

3.2. 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]

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

Remark

None

3.3. 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;

hmi_TCPInit ();
tHandle h = hmi_TCPNew();
hmi_TCPListen(h, port_listen);
```

Remark

None

3.4. 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 the local port of the client.

Return Values

None

Examples

[C]

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

```
unsigned short local_port = 10001;

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

Remark

None

3.5. 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]

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

Remark

None

3.6. 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 the port which is used to connect to the server.

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]

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

Remark

None

3.7. 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]

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

Remark

None

3.8. 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]

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

Remark

None

3.9. 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];
```

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

Remark

None

3.10. 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];
```

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

Remark

None

3.11. HMI_TCPREAD

This function reads data through a TCP session.

Syntax

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

Parameter

h

[in] Specify the handle to the TCP session

data

[in] Specify the array to read

len

[in] Specify the length of the array to read

Return Values

The length of the receiving data

Examples

[C]

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

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

Remark

None

3.12. HMI_TCPSENDCMD

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

Syntax

```
int hmi_TCPSendCmd(
        tHandle h,
        unsigned char *send_data,
        int send_data_len,
        unsigned char *receive_data,
        int recv_data_len
);
```

Parameter

h

[in] Specify the handle to the TCP session

send_data

[in] Specify the sending array

send_data_len

[in] Specify the sending data len

receive_data

[in] Specify the receiving array

recv_data_len

[in] Specify the length of the recv data len

Return Values

The length of the receiving data

Examples

[C]

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

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

Remark

None

4. MODBUS TCP MASTER API

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

4.1. MBM_REGISTER

Register a modbus communication to the TouchPAD.

Syntax

```
tHandle mbm_Register (
    int net_id,
    DWORD modbus_ip,
    int modbus_port
);
```

Parameter

net_id

[in] Specify the net id of the device (Range: 1 ~ 255)

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.

If not, -1 is returned.

Examples

[C]

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

Remark

None

4.2. MBM_UNREGISTER

Unregister a modbus communication from the TouchPAD.

Syntax

```
BOOL mbm_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;
…
mbm_Unregister(h);
```

Remark

None

4.3. MBM_WRITEDO

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

Syntax

```
BOOL mbm_WriteDO(
    tHandle h,
    int addr,
    int ch_count,
    DWORD DO_value
);
```

Parameter

h

[in] Specify the handle to 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.

DO_value

[in] Specify the value of DO in which the least significant bit represents the channel 0, the second lowest bit represents the channel 1, etc. For example, if we turn on only channel 0 and channel 1, its DO value is 3 (whose binary equivalent is 00…011).

Return Values

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

Examples

[C]

```
int addr = 1;
int ch_count = 16;
DWORD DO_value = 3; //that is, turn on the ch 0 and ch1.
tHandle h = mbm_Register(addr, TCP_IPADDR(10,1,102,64), 502);
mbm_WriteDO(h, addr, ch_count, DO_value);
mbm_Unregister(h);
```

Remark

None

4.4. MBM_READDO

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

Syntax

```
DWORD mbm_ReadDO(
    tHandle h,
    int addr,
    int ch_count,
);
```

Parameter

h

[in] Specify the handle to 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.

Return Values

The DO value in which the least significant bit represents the channel 0, the second lowest bit represents the channel 1, etc.
For example, if we turn on only channel 0 and channel 1, its DO value is 3 (whose binary equivalent is 00…011).

Examples

[C]

```
int addr = 1;
```

```
int ch_count = 16;
DWORD DO_value;
tHandle h = mbm_Register(addr, TCP_IPADDR(10,1,102,64), 502);
DO_value = mbm_ReadDO(h, addr, ch_count);
mbm_Unregister(h);
```

Remark

None

4.5. MBM_READDI

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

Syntax

```
DWORD mbm_ReadDI(
    tHandle h,
    int addr,
    int ch_count,
);
```

Parameter

h

[in] Specify the handle to 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.

Return Values

Specify the value of DI in which the least significant bit represents the channel 0, the second lowest bit represents the channel 1, etc. For example, if only channel 0 and channel 1 are input, its DI value is 3 (whose binary equivalent is 00…011).

Examples

[C]

```
int addr = 1;
int ch_count = 16;
```

```
DWORD DI_value;
tHandle h = mbm_Register(addr, TCP_IPADDR(10,1,102,64), 502);
DI_value = mbm_ReadDI(h, addr, ch_count);
mbm_Unregister(h);
```

Remark

None

4.6. MBM_WRITEAO

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

Syntax

```
BOOL mbm_WriteAO(
    tHandle h,
    int addr,
    int ch_count,
    WORD * AO_value
);
```

Parameter

h

[in] Specify the handle to 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]

```
int addr = 1;
int ch_count = 16;
WORD AO_value[2]; //for example, we have a two-channel AO module

AO_value[0] = 65535; //set channel 0 to its maximum analog output
tHandle h = mbm_Register(addr, TCP_IPADDR(10,1,102,64), 502);
mbm_WriteAO(h, addr, ch_count, AO_value);
mbm_Unregister(h);
```

Remark

None

4.7. MBM_READAO

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

Syntax

```
BOOL mbm_ReadAO(
    tHandle h,
    int addr,
    int ch_count,
    WORD * AO_value
);
```

Parameter

h

[in] Specify the handle to 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 to store the values of 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]

```
int addr = 1;
int ch_count = 16;
WORD AO_value[2]; //for example, we have a two-channel AO module

tHandle h = mbm_Register(addr, TCP_IPADDR(10,1,102,64), 502);
mbm_ReadAO(h, addr, ch_count, AO_value);
mbm_Unregister(h);
```

Remark

None

4.8. MBM_READAI

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

Syntax

```
BOOL mbm_ReadAI(
    tHandle h,
    int addr,
    int ch_count,
    WORD * AI_value
);
```

Parameter

h

[in] Specify the handle to the modbus communication.

addr

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

ch_count

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

AO_value

[out] Specify the pointer to an array to store the values of 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]

```
int addr = 1;
int ch_count = 16;
WORD AI_value[2]; //for example, we have a two-channel AI module

tHandle h = mbm_Register(addr, TCP_IPADDR(10,1,102,64), 502);
mbm_ReadAI(h, addr, ch_count, AI_value);
mbm_Unregister(h);
```

Remark

None

5. MODBUS RTU MASTER API

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

5.1. 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

5.2. 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

5.3. 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

5.4. 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

5.5. 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

5.6. 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

6. 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.

6.1. 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:

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 open COM port.

Examples

[C]

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

Remark

None

6.2. UART_CLOSE

This function closes the COM port which has been opened.

Syntax

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

Parameter

hPort

[in] Handle to the open COM port to close.

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.

6.3. UART_SEND

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

Syntax

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

Parameter

hPort

[in] Handle to the open COM port.

buf

[in] A pointer to a buffer that send the data.

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

6.4. 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 open COM port.

buf

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

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

None

6.5. UART_SENDCMD

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

Syntax

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

Parameter

hPort

[in] Handle to the open COM.

cmd

[in] A pointer to a command.

szResult

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

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

6.6. UART_SETTIMEOUT

This function sets the time out timer.

Syntax

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

Parameter

hPort

[in] Handle to the open 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

6.7. 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 open 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

None

6.8. UART_SETTERMINATOR

This function sets the terminate characters.

Syntax

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

Parameter

hPort

[in] Handle to the open 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

6.9. 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 open COM port.

buf

[in] A pointer to a buffer that send 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_BinSend(hOpen, buf, Length);
uart_Close(hPort);
```

Remark

None

6.10. 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

None

6.11. 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 open 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

None

7. 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

7.1. 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

7.2. 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

7.3. 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

7.4. 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

7.5. 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

7.6. 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

7.7. 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

7.8. 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

7.9. 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

7.10. 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

7.11. 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

7.12. 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

7.13. 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

7.14. 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

7.15. 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

7.16. 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

7.17. 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

7.18. 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

7.19. 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

7.20. 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

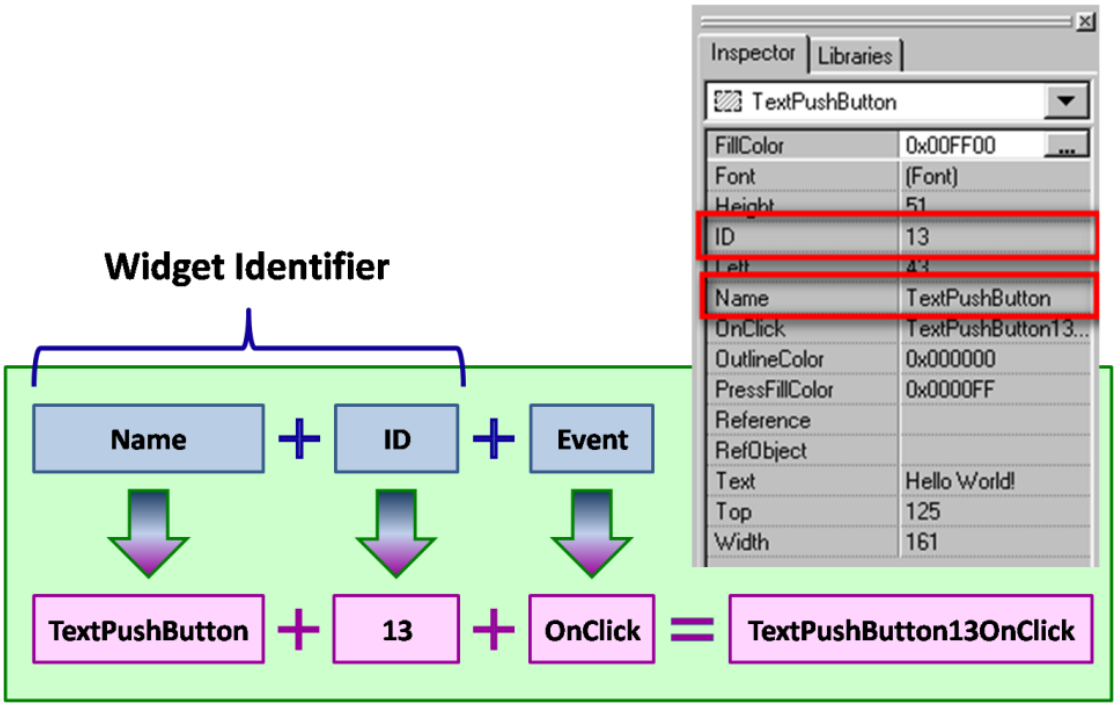
None

8. 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:



8.1. TEXTBUTTONTAGGET

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

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
int TextButtonTagGet (
        tTextButton * pWidget,
);
```

Parameter

pWidget

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

Return Values

The pointer to an integer 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

8.2. TEXTBUTTONTAGSET

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

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
void TextButtonTagSet (
        tTextButton * pWidget,
        int tag
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the TextPushButton, to set the “Tag” property.

tag

[in] Specify the value of the “Tag” 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

8.3. OBJBUTTONTAGGET

Get the “Tag” property of the TextPushButton **while its “RefObject” property is set to an ObjectList widget**.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
int ObjButtonTagGet (
        tObjButton * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the TextPushButton, to get the “Tag” property **while its “RefObject” property is set to an ObjectList widget**.

Return Values

The pointer to an integer of the “Tag” property

Examples

[C]

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

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

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

   //Get the value of the "Tag" property
   tag = ObjButtonTagGet((tObjButton*)pWidget);

   //Set the "Tag" property to the Label
   usprintf(str, "%d", tag);
   LabelTextSet(&LabelWidget5, str);
}
```

Remark

None

8.4. OBJBUTTONTAGSET

Set the “Tag” property of the TextPushButton **while its “RefObject” property is set to an ObjectList widget**.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
void TextButtonTagSet (
        tObjButton * pWidget,
        int tag
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the TextPushButton, to set the “Tag” property **while its “RefObject” property is set to an ObjectList widget**.

tag

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

Return Values

None

Examples

[C]

```
int tag = 0;
```

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

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

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

   //Get the value of the "Tag" property
   tag = ObjButtonTagGet((tObjButton*)pWidget);

   //Set the "Tag" property to the Label
   usprintf(str, "%d", tag);
   LabelTextSet(&LabelWidget5, str);
}
```

Remark

None

8.5. 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

8.6. 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

8.7. SLIDERTAGGET

Get the “Tag” property of the Slider.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
int SliderTagGet (
        tSlider * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the Slider, to get the “Tag” property.

Return Values

The pointer to an integer of the “Tag” property

Examples

[C]

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

void SliderWidget4OnSliderChange(tWidget *pWidget, long lValue)
```

```
{
   count ++;

   //Get the "Position" property and show it on the Slider
   //To have visible text of the Slider, be sure to
   //Set the "BackgroundTextColor" to the color except white (0xFFFFFF)
   //Set the Font color except black (0x000000)
   usprintf(str, "%d", lValue);
   SliderTextSet(&SliderWidget4, str);

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

   //Get the value of the "Tag" property
   tag = SliderTagGet((tSlider*)pWidget);

   //Show the "Tag" on the Label 5 (in this example)
   usprintf(str2, "%d", tag);
   LabelTextSet(&LabelWidget5, str2);
}
```

Remark

None

8.8. SLIDERTAGSET

Set the “Tag” property of the Slider.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
void SliderTagSet (
        tSlider * pWidget,
        int tag
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the Slider, to get the “Tag” property.

tag

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

Return Values

None

Examples

[C]

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

```
static char str2[16];

void SliderWidget4OnSliderChange(tWidget *pWidget, long lValue)
{
   count ++;

   //Get the "Position" property and show it on the Slider
   //To have visible text of the Slider, be sure to
   //Set the "BackgroundTextColor" to the color except white (0xFFFFFF)
   //Set the Font color except black (0x000000)
   usprintf(str, "%d", lValue);
   SliderTextSet(&SliderWidget4, str);

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

   //Get the value of the "Tag" property
   tag = SliderTagGet((tSlider*)pWidget);

   //Show the "Tag" on the Label 5 (in this example)
   usprintf(str2, "%d", tag);
   LabelTextSet(&LabelWidget5, str2);
}
```

Remark

None

8.9. BitButtonTagGet

Get the “Tag” property of the BitButton.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
int BitButtonTagGet (
        tBitButton * pWidget,
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the BitButton, to get the “Tag” property.

Return Values

The pointer to an interger of the “Tag” property

Examples

[C]

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

void BitButton4OnClick(tWidget *pWidget)
{
```

```
    count ++;

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

    //Get the value of the "Tag" property
    tag = BitButtonTagGet((tBitButton*)pWidget);

    //Show the "Tag" on the Label 5 (in this example)
    usprintf(str, "%d", tag);
    LabelTextSet(&LabelWidget5, str);
}
```

Remark

None

8.10. BitButtonTagSet

Set the “Tag” property of the BitButton.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
void BitButtonTagSet (
    tBitButton * pWidget,
    int tag
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the BitButton, to set the “Tag” property.

tag

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

Return Values

None

Examples

[C]

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

```
void BitButton4OnClick(tWidget *pWidget)
{
   count ++;

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

   //Get the value of the "Tag" property
   tag = BitButtonTagGet((tBitButton*)pWidget);

   //Show the "Tag" on the Label 5 (in this example)
   usprintf(str, "%d", tag);
   LabelTextSet(&LabelWidget5, str);
}
```

Remark

None

8.11. 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

8.12. 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

8.13. HOTSOPTTAGGET

Get the “Tag” property of the HotSpot.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

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

Parameter

pWidget

[out] Specify the pointer to the widget, the HotSpot, to get the “Tag” property.

Return Values

The pointer to an integer of the “Tag” property

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

8.14. HOTSOPTTAGSET

Set the “Tag” property of the HotSpot.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
void HotSoptTagSet (
        tHotSpot * pWidget,
        int tag
);
```

Parameter

pWidget

[out] Specify the pointer to the widget, the HotSpot, to set the “Tag” property.

tag

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

Return Values

None

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

8.15. 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

8.16. CHECKBOXTAGGET

Get the “Tag” property of the CheckBox.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

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

Parameter

pWidget

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

Return Values

The pointer to an integer of the “Tag” property

Examples

[C]

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

void CheckBoxWidget4OnChange(tWidget *pWidget, unsigned long
bSelected)
```

```
{
   count ++;

   if(bSelected)
   {
      //Set the value of count to the "Tag" property
      CheckBoxTagSet((tCheckBox*)pWidget, count);

      //Get the value of the "Tag" property
      tag = CheckBoxTagGet((tCheckBox*)pWidget);
   }

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

Remark

None

8.17. CHECKBOXTAGSET

Set the “Tag” property of the CheckBox.
Users can set the “Tag” property in the inspector in the design time.

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 indicates which widget is clicked at this time. This looks like a widget array index.

Syntax

```
void CheckBoxTagSet (
        tCheckBox * pWidget,
        int tag
);
```

Parameter

pWidget

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

tag

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

Return Values

None

Examples

[C]

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

```
void CheckBoxWidget4OnChange(tWidget *pWidget, unsigned long
bSelected)
{
   count ++;

   if(bSelected)
   {
      //Set the value of count to the "Tag" property
      CheckBoxTagSet((tCheckBox*)pWidget, count);

      //Get the value of the "Tag" property
      tag = CheckBoxTagGet((tCheckBox*)pWidget);
   }

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

Remark

None

8.18. 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. The widget has no buffer for the “Text”, so the string must be static string (static char[]).

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

8.19. 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. The widget has no buffer for the “Text”, so the string must be static string (static char[]).

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

8.20. 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

8.21. 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

9. MISCELLANEOUS API

This chapter provides APIs that are not classified.

9.1. 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.

Return Values

None

Examples

[C]

```
hmi_SetForeground (pContext, color);
```

Remark

None

9.2. HMI_BEEP

Sound the beep.

Syntax

```
void hmi_Beep();
```

Parameter

None

Return Values

None

Examples

[C]

```
hmi_Beep ();
```

Remark

None

9.3. 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

9.4. 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

9.5. 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

9.6. HMI_BACKLIGHTSET

Set the brightness of the TPD-430/TPD-430-EU.

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.

Return Values

None

Examples

[C]

```
unsigned char b = 128;

hmi_BacklightSet(b);    //set the brightness to 128
```

Remark

None

9.7. 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

9.8. 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.

9.9. HMI_USERFLASHERASE

Erase the data on the user's flash area.

There's a **4kB** size flash allocated for users to store their data. It is the last sector of the external flash disk (8 MB) of the TPD-280U/ TPD-430/ TPD-430-EU/ VPD-130 /VPD-130N. Note that the flash has 100,000 erase/program cycles limitation.

Syntax

```
int hmi_UserFlashErase (
        int iSector
);
```

Parameter

iSector

[in] Reserved (Any arbitrarily specified integer is OK.)

Return Values

1 = erase operation success; 0 = erase operation failure

Examples

[C]

```
//
//The flash has 100,000 erase/program cycles limitation.
//

static char str[16];
static char str2[16];
char * data = "Hello";
```

```
void BitButton4OnClick(tWidget *pWidget)
{
   int ret = hmi_UserFlashErase(0);
   if(ret == 0)
   {
      usprintf(str, "Erase Error");
      LabelTextSet(&LabelWidget5, str);
   }
   else
   {
      //supposed offset=0; and "Hello" is written to the flash
      //and then read back to check
      hmi_UserFlashWrite(0, strlen(data), data);
      hmi_UserFlashRead(0, strlen(data), str2);

      if(strncmp(data, str2, strlen(data)) == 0)
      {
         usprintf(str, "Read/Write flash correctly");
         LabelTextSet(&LabelWidget5, str);
      }
   }
}
```

Remark

The flash has 100,000 erase/program cycles limitation.

9.10. HMI_USERFLASHREAD

Read the data from the user's flash area.

There's a **4kB** size flash allocated for users to store their data. It is the last sector of the external flash disk (8 MB) of the TPD-280U/ TPD-430/ TPD-430-EU/ VPD-130 /VPD-130N. Note that the flash has 100,000 erase/program cycles limitation.

Syntax

```
unsigned long hmi_UserFlashRead (
        unsigned long offset,
        unsigned long length,
        char *pBuffer
);
```

Parameter

offset

[in] the starting point to read users' data in the reserved sector of the flash

length

[in] the amount of data to read

pTimer

[out] Specify the pointer to the data to read

Return Values

The amount of data that is read from the flash

Examples

[C]

```
//
//The flash has 100,000 erase/program cycles limitation.
```

```
//

static char str[16];
static char str2[16];
char * data = "Hello";

void BitButton4OnClick(tWidget *pWidget)
{
   int ret = hmi_UserFlashErase(0);
   if(ret == 0)
   {
      usprintf(str, "Erase Error");
      LabelTextSet(&LabelWidget5, str);
   }
   else
   {
      //supposed offset=0; and "Hello" is written to the flash
      //and then read back to check
      hmi_UserFlashWrite(0, strlen(data), data);
      hmi_UserFlashRead(0, strlen(data), str2);

      if(strncmp(data, str2, strlen(data)) == 0)
      {
         usprintf(str, "Read/Write flash correctly");
         LabelTextSet(&LabelWidget5, str);
      }
   }
}
```

Remark

The flash has 100,000 erase/program cycles limitation.

9.11. HMI_USERFLASHWRITE

Write the data to the user's flash area.

There's a **4kB** size flash allocated for users to store their data. It is the last sector of the external flash disk (8 MB) of the TPD-280U/ TPD-430/ TPD-430-EU/ VPD-130 /VPD-130N. Note that the flash has 100,000 erase/program cycles limitation.

Syntax

```
unsigned long hmi_UserFlashWrite (
    unsigned long offset,
    unsigned long length,
    char *pBuffer
);
```

Parameter

offset

[in] the starting point to write users' data to the reserved sector of the flash

length

[in] the amount of data to write

pTimer

[out] Specify the pointer to the data to write

Return Values

The amount of data that is written to the flash

Examples

[C]

```
//
```

```
//The flash has 100,000 erase/program cycles limitation.
//

static char str[16];
static char str2[16];
char * data = "Hello";

void BitButton4OnClick(tWidget *pWidget)
{
   int ret = hmi_UserFlashErase(0);
   if(ret == 0)
   {
      usprintf(str, "Erase Error");
      LabelTextSet(&LabelWidget5, str);
   }
   else
   {
      //supposed offset=0; and "Hello" is written to the flash
      //and then read back to check
      hmi_UserFlashWrite(0, strlen(data), data);
      hmi_UserFlashRead(0, strlen(data), str2);

      if(strncmp(data, str2, strlen(data)) == 0)
      {
         usprintf(str, "Read/Write flash correctly");
         LabelTextSet(&LabelWidget5, str);
      }
   }
}
```

Remark

The flash has 100,000 erase/program cycles limitation.

9.12. 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