# AN1227

# Using a Keyboard with the Microchip Graphics Library

*Author:* *Anton Alkhimenok*
*Microchip Technology Inc.*

## INTRODUCTION

Graphics displays are widely used in many applications and the number of products with displays grows every day. A primary reason is that a Graphical User Interface (GUI) can greatly simplify the use of a device.

The GUI can interface a display with a variety of input devices, such as keyboards, touch screens or mice. Keyboards range from devices with several side buttons to those enabling text entry. Low cost and the ability to quickly enter data are resulting in the frequent use of keyboards.

The Microchip Graphics Library simplifies the design of a keyboard-based GUI, as this application note will demonstrate. For more information on the library, see *AN1136, "How to Use Widgets in Microchip Graphics Library"*.

## MESSAGE INTERFACE

The Microchip Graphics Library supports several kinds of input devices, achieving that flexibility through a message interface. The interface has a structure containing information about input device events. The input device firmware is not a part of the library and must be implemented in the application.

The application must provide the event information in a a prescribed format and pass the data to the library's message manager. The rest of the work is done by the library with the on-screen widgets displaying the new state automatically.

The code structure of the message manager function is shown in Example 1.

**EXAMPLE 1: MESSAGE MANAGER FUNCTION PROTOTYPE**

```
void  GOLMsg(GOL_MSG *pMsg)
```

The `pMsg` parameter is a pointer to the message structure filled by the input device.

The format of the graphics library message structure is shown in Example 2.

**EXAMPLE 2: MESSAGE STRUCTURE FORMAT**

```
typedef  struct {
BYTE             type,
BYTE             uiEvent,
SHORT            param1,
SHORT            param2
} GOL_MSG;
```

The keyboard related definitions for the preceding example's fields are given in Table 1.

**TABLE 1: MESSAGE STRUCTURE FIELDS' DEFINITIONS**

| Field | Description | Bytes |
|---|---|---|
| type | The type of input device.<br>For a keyboard, this value must be `TYPE_KEYBOARD`. | 1 |
| uiEvent | The input event.<br>A keyboard has two kinds of events:<br>• `EVENT_KEYSCAN` – When the `param2` field contains a scan code<br>• `EVENT_KEYCODE` – When the `param2` field contains a character code. | 1 |
| param1 | The ID of the widget receiving the message.<br>This unique ID is assigned by the application when the object is created. | 2 |
| param2 | The scan code or character code, depending on the value in the `uiEvent` field. | 2 |

Keyboard messages use the standard AT keyboard scan codes. The Most Significant bit of the AT scan code defines the key state of pressed or released.

Frequently used scan codes for the graphics library widgets are listed in Table 2. The constants definitions for the codes is in the `ScanCodes.h` file included in the graphics library.

**TABLE 2: AT KEYBOARD SCAN CODES**

| Description | Press | | Release | |
|---|---|---|---|---|
| | Name Defined in ScanCodes.h | Code | Name Defined in ScanCodes.h | Code |
| Carriage Return | SCAN_CR_PRESSED | 0x1C | SCAN_CR_RELEASED | 0x9C |
| Delete | SCAN_DEL_PRESSED | 0x53 | SCAN_DEL_RELEASED | 0xD3 |
| Back Space | SCAN_BS_PRESSED | 0x0E | SCAN_BS_RELEASED | 0x8E |
| Tabulation | SCAN_TAB_PRESSED | 0x0F | SCAN_TAB_RELEASED | 0x8F |
| Home | SCAN_HOME_PRESSED | 0x47 | SCAN_HOME_RELEASED | 0xC7 |
| End | SCAN_END_PRESSED | 0x4F | SCAN_END_RELEASED | 0xCF |
| Page Up | SCAN_PGUP_PRESSED | 0x49 | SCAN_PGUP_RELEASED | 0xC9 |
| Page Down | SCAN_PGDOWN_PRESSED | 0x51 | SCAN_PGDOWN_RELEASED | 0xD1 |
| Arrow Up | SCAN_UP_PRESSED | 0x48 | SCAN_UP_RELEASED | 0xC8 |
| Arrow Down | SCAN_DOWN_PRESSED | 0x50 | SCAN_DOWN_RELEASED | 0xD0 |
| Arrow Left | SCAN_LEFT_PRESSED | 0x4B | SCAN_LEFT_RELEASED | 0xCB |
| Arrow Right | SCAN_RIGHT_PRESSED | 0x4D | SCAN_RIGHT_RELEASED | 0xCD |
| Space | SCAN_SPACE_PRESSED | 0x39 | SCAN_SPACE_RELEASED | 0xB9 |

The key code in a keyboard message can have different encoding. An application must ensure that a message's encoding matches the one used for the font of the widget receiving the message.

For example, if the Edit Box widget has an ASCII encoded font, the key code in the keyboard message also must be in ASCII.

## WIDGETS KEYBOARD MESSAGES

Each widget has a set of valid keyboard messages. Invalid messages result in no action.

Some widgets – such as Window, Static Text, Picture, Progress Bar and Group Box – cannot accept keyboard input. If a widget is in a disabled state, it ignores all messages.

Table 3 summarizes the keyboard messages for the different widgets.

**TABLE 3: MICROCHIP GRAPHICS LIBRARY KEYBOARD MESSAGES FOR WIDGETS**

| Widget | Translated Message in the Message Callback Function | Message from the Input Device | | | |
|---|---|---|---|---|---|
| | | **Type of the Input Device (the `Type` field in the message structure)** | **Event Description (the `uiEvent` field in the message structure)** | **Parameter 1 Description (`param1` field in the message structure)** | **Parameter 2 Description (`param2` field in the message structure)** |
| Button | The button is pressed. (BTN_MSG_PRESSED constant) | Keyboard (TYPE_KEYBOARD constant) | Key scan code event. (EVENT_KEYSCAN constant) | ID of the button assigned by application when the button was created. | Carriage return pressed scan code (SCAN_CR_PRESSED constant) or Space pressed scan code (SCAN_SPACE_PRESSED constant) |
| | The button is released. (BTN_MSG_RELEASED constant) | | | | Carriage return released scan code (SCAN_CR_RELEASED constant) or Space released scan code (SCAN_SPACE_RELEASED constant) |
| Check Box | • The check box is checked. (BTN_MSG_CHECKED constant)<br>• The check box is unchecked. (BTN_MSG_UNCHECKED constant) | Keyboard (TYPE_KEYBOARD constant) | Key scan code event. (EVENT_KEYSCAN constant) | ID of the check box assigned by application when the check box was created. | Carriage return pressed scan code (SCAN_CR_PRESSED constant) or Space pressed scan code (SCAN_SPACE_PRESSED constant) |
| Radio Button | The same radio button in the group is checked. (RB_MSG_CHECKED constant) | Keyboard (TYPE_KEYBOARD constant) | Key scan code event. (EVENT_KEYSCAN constant) | ID of any radio button in the group assigned by application when the radio button was created. | Carriage return pressed scan code (SCAN_CR_PRESSED constant) or Space pressed scan code (SCAN_SPACE_PRESSED constant) |
| Edit Box | A new character is added to the edit box. (EB_MSG_CHAR constant) | Keyboard (TYPE_KEYBOARD constant) | Character code event. (EVENT_KEYCODE constant) | ID of the edit box assigned by application when the edit box was created. | Character code |
| | The last character is removed from edit box. (EB_MSG_DEL constant) | | Key scan code event. (EVENT_KEYSCAN constant) | | Back space pressed scan code (SCAN_BS_PRESSED constant) |

**TABLE 3: MICROCHIP GRAPHICS LIBRARY KEYBOARD MESSAGES FOR WIDGETS (CONTINUED)**

| Widget | Translated Message in the Message Callback Function | Message from the Input Device | | | |
|---|---|---|---|---|---|
| | | **Type of the Input Device (the `Type` field in the message structure)** | **Event Description (the `uiEvent` field in the message structure)** | **Parameter 1 Description (`param1` field in the message structure)** | **Parameter 2 Description (`param2` field in the message structure)** |
| Slider | Slider position is incremented. (SLD_MSG_INC constant) | Keyboard (TYPE_KEYBOARD constant) | Key scan code event. (EVENT_KEYSCAN constant) | ID of the slider assigned by application when the slider was created. | Arrow down pressed scan code (SCAN_UP_PRESSED constant) or Arrow right pressed scan code (SCAN_LEFT_PRESSED constant) |
| | Slider position is decremented. (SLD_MSG_DEC constant) | | | | Arrow up pressed scan code (SCAN_DOWN_PRESSED constant) or Arrow left pressed scan code (SCAN_RIGHT_PRESSED constant) |
| List Box | Current item mark is moved to the next item. (LB_MSG_MOVE constant) | Keyboard (TYPE_KEYBOARD constant) | Key scan code event. (EVENT_KEYSCAN constant) | ID of the list box assigned by application when the list box was created. | Arrow up pressed scan code (SCAN_UP_PRESSED constant) or Arrow down pressed scan code (SCAN_DOWN_PRESSED constant) |
| | Current item is selected. (LB_MSG_SEL constant) | | | | Carriage return pressed scan code (SCAN_CR_PRESSED constant) or Space pressed scan code (SCAN_SPACE_PRESSED constant) |
| Dial | Keyboard messages are not supported for these objects. | | | | |
| Group Box | | | | | |
| Meter | | | | | |
| Picture | | | | | |
| Progress Bar | | | | | |
| Static Text | | | | | |
| Window | | | | | |

Example 3 shows how to pass the button's "press" or "release" events to the library. As a keyboard key, the switch connected to the RD6 port is used. If the switch is pressed, '0' is presented on this port. If the key is in a released state, '1' is read from this port.

**EXAMPLE 3: PASSING BUTTON EVENTS TO GRAPHICS LIBRARY**

```
#define BUTTON1_ID         1111                        // button unique ID

int main(void)
{
GOL_MSG msg;                                           // message interface structure, should
                                                       // be filled by the keyboard driver and
                                                       // passed to the message manager

BYTE     previousKey1State;                            // previous state of the key

         // initialize the keyboard
         TRISDbits.TRISD6 = 1;                         // set port RD6 to be an input
         previousKey1State = PORTDbits.RD6;            // previous state equals the current state

         GOLInit();                                    // initialize the graphics library

         BtnCreate(                                    // create a button widget
                  BUTTON1_ID,                          // button unique ID
                  0,40,                                // left, top corner coordinates
                  100,90,                              // right, bottom corner
                  0,                                   // corner radius is zero, it's a square
                                                       // button
                  BTN_DRAW,                            // will be dislayed after creation
                  NULL,                                // no bitmap
                  "Released",                          // text for released state
                  NULL                                 // default color scheme is used
                  );

         while(1)
         {
                  if(GOLDraw())                        // drawing manager to display widgets
                  {
                           // Keyboard driver
                           if(PORTDbits.RD6 != previousKey1State)
                           // check if the button has changed its state
                           {
                                    if(previousKey1State)
                                    {        // if RD6 equals zero it means the key is pressed
                                             msg.type = TYPE_KEYBOARD;
                                             msg.uiEvent =  EVENT_KEYSCAN;
                                             msg.param1 = BUTTON1_ID;
                                             msg.param2 = SCAN_CR_PRESSED;
                                    }else{
                                             // if RD6 equals one it means the key is released
                                             msg.type = TYPE_KEYBOARD;
                                             msg.uiEvent =  EVENT_KEYSCAN;
                                             msg.param1 = BUTTON1_ID;
                                             msg.param2 = SCAN_CR_RELEASED;
                                    } // end of else

                                    // state of the key was changed
                                    previousKey1State = ! previousKey1State;

                                    // pass the message to the graphics library
                                    GOLMsg(&msg);
                           } // end of if
                  } // end of if
         } // end of while
         return 0;

} // end of main

WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg)
{
         // Application should process messages here
         return 1; // process the message by default
} // end of GOLMessageCallback
```

## MESSAGE CALLBACK FUNCTION

After the library's message manager, GOLMsg(…), has received a message from the input device, the graphics library finds the widgets affected in the active link list and uses a special callback function so the program reacts on the event. This function must be implemented in the application. Example 4 shows this function's prototype.

**EXAMPLE 4: MESSAGE CALLBACK FUNCTION PROTOTYPE**

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg);
```

The first parameter – objMsg – is a translated message. The graphics library parses the message from the input device and translates it into a form for the particular widget.

For example, if the keyboard sends the button the carriage return pressed code, the library returns the translated message, BUTTON IS PRESSED (BTN_MSG_PRESSED constant), to the message callback function in the objMsg parameter.

The second parameter, pObj, is a pointer to the widget affected by the message. The third parameter, pMsg, is a pointer to the original message from the input device.

This information is enough for the application to perform any action on an event.

The graphics library has a default action of all events for each widget. (For the button, the library can display a pressed or released state). If the callback function returns non-zero, the message for the object will be processed by default. If '0' is returned, the library will not perform any default action.

Example 5 adds application code to the previous example for processing messages for the button with the ID BUTTON1_ID. This example shows the code for changing the text on the face of the button for "pressed" and "released" events. GOLMsgCallback() returns '1' to enable the default action on the button which is the change in state from released to pressed and pressed to released.

**EXAMPLE 5: CHANGING BUTTON TEXT FOR PRESSED AND RELEASED EVENTS**

```
WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg)
{
        // Application should process messages here

        if(GetobjID(pObj) == BUTTON1_ID)                // if the button with BUTTON_ID is
                                                        // receiving the message
        {
                if(objMsg == BTN_MSG_PRESSED)
                {
                        BtnSetText(pObj,"Pressed");     // set text for pressed state
                }
                if(objMsg == BTN_MSG_RELEASED)
                {
                        BtnSetText(pObj,"Released");    // set text for released state
                }
        }

        return 1; // process the message by default
} // end of GOLMessageCallback
```

## KEYBOARD FOCUS

The keyboard focus determines which widget receives the information typed on the keyboard. The Microchip Graphics Library shows which widget has focus by putting a dashed rectangle around it.

Focus is especially useful when the number of widgets is greater than the number of keyboard keys. In such cases, some keys can be assigned to navigate between controls on the screen.

Widgets not supporting the keyboard cannot accept focus. To allow focus, the USE_FOCUS compile-time option must be defined in the GraphicsConfig.h file. If another type of input device, such as a touch screen, is used simultaneously with a keyboard, the second device will move the keyboard focus automatically to the active widget.

The following functions are available to control focus.

**WORD GOLCanBeFocused (OBJ_HEADER* object)**

This function returns non-zero if the object can be focused. Only the button, check box, radio button, slider, edit box and list box can accept focus. If the object is disabled, it cannot be set to the focused state.

**OBJ_HEADER *GOLGetFocusNext()**

This function returns the pointer of the next object in the active list that is capable of receiving keyboard input. If there is no such object, NULL is returned.

**void GOLSetFocus (OBJ_HEADER* object)**

This function sets the keyboard input focus to the object. If the object cannot accept keyboard messages, focus will not be changed.

This function resets the focused state for the object that previously was in focus, sets the focused state for the required object and marks the objects to be redrawn.

**OBJ_HEADER *GOLGetFocus(void)**

This macro returns the pointer to the object receiving keyboard input. If there is no object in focus, NULL is returned.

Example 6 illustrates the use of focus by adding second and third button widgets. The initial state of the third button widget is disabled such that it will not accept the keyboard focus. The second keyboard key, connected to port RD13, also is added to move the input focus between widgets on the current screen.

**EXAMPLE 6: USING FOCUS WITH THREE BUTTON WIDGETS**

```
#define BUTTON1_ID          1111                    // button unique ID
#define BUTTON2_ID          2222                    // button unique ID
#define BUTTON3_ID          3333                    // button unique ID

int main(void)
{
GOL_MSG msg;                                        // message interface structure, should be
                                                    // filled by the keyboard driver and
                                                    // passed to the message manager
OBJ_HEADER* pFocusedObj;                            // temporary variable for the widget
                                                    // receiving the keyboard focus
BYTE     previousKey1State;                         // previous state of the button 1
BYTE     previousKey2State;                         // previous state of the button 2

         // initialize the keyboard's keys
         TRISDbits.TRISD6 = 1;                      // set port RD6 to be an input
         previousKey1State = PORTDbits.RD6;         // previous state equals the current state
         TRISDbits.TRISD13 = 1;                     // set port RD13 to be an input
         previousKey2State = PORTDbits.RD13;        // previous state equals the current state

         GOLInit();                                 // initialize the graphics library
```

**EXAMPLE 6: USING FOCUS WITH THREE BUTTON WIDGETS (CONTINUED)**

```
        // create button widgets
        BtnCreate(
                BUTTON1_ID,                     // button unique ID
                10,40,                          // left, top corner coordinates
                110,90,                         // right, bottom corner
                0,                              // corner radius is zero, it's a square
                                                // button
                BTN_DRAW,                       // will be dislayed after creation
                NULL,                           // no bitmap
                "Released",                     // text for released state
                NULL                            // default color scheme is used
                );

        BtnCreate(
                BUTTON2_ID,                     // button unique ID
                10,100,                         // left, top corner coordinates
                110,150,                        // right, bottom corner
                0,                              // corner radius is zero, it's a square
                                                // button
                BTN_DRAW,                       // will be dislayed after creation
                NULL,                           // no bitmap
                "Button 2",                     // text
                NULL                            // default color scheme is used
                );

        BtnCreate(
                BUTTON3_ID,                     // button unique ID
                10,160,                         // left, top corner coordinates
                110,210,                        // right, bottom corner
                0,                              // corner radius is zero, it's a square
                                                // button
                BTN_DRAW|BTN_DISABLED,          // will be dislayed and disabled after
                                                // creation
                NULL,                           // no bitmap
                "Disabled",                     // text
                NULL                            // default color scheme is used
                );

        pFocusedObj = NULL;                     // there are no widgets in focus

        while(1)
        {
                if(GOLDraw())                   // drawing manager to display widgets
                {
                        // Keyboard driver
                        if(GOLGetFocus() != NULL)  // if there's a widget in focus send a
                                                   // message
                        {
                                // check if the button has changed its state
                                if(PORTDbits.RD6 != previousKey1State)
                                {
                                        if(previousKey1State)
                                        {
                                                // if RD6 equals zero it means the button
                                                // is pressed
                                                msg.type = TYPE_KEYBOARD;
                                                msg.uiEvent =  EVENT_KEYSCAN;
                                                // the focused button will receive the
                                                // message
                                                msg.param1 = GetObjID(GOLGetFocus());
                                                msg.param2 = SCAN_CR_PRESSED;
                                        }else{
                                                // if RD6 equals one it means the button is
                                                // released
                                                msg.type = TYPE_KEYBOARD;
                                                msg.uiEvent =  EVENT_KEYSCAN;
                                                // the focused button will receive the
                                                // message
                                                msg.param1 = GetObjID(GOLGetFocus());
                                                msg.param2 = SCAN_CR_RELEASED;
                                        } // end of else
```

**EXAMPLE 6: USING FOCUS WITH THREE BUTTON WIDGETS (CONTINUED)**

```
                                        // state of the button was changed
                                        previousKey1State = ! previousKey1State;

                                        // pass the message to the graphics ibrary
                                        GOLMsg(&msg);
                                        continue;
                                } // end of if
                        } // end of if

                        // check if the button has changed its state
                        if(PORTDbits.RD13 != previousKey2State)
                        {
                                if(previousKey2State)
                                {       // if RD13 equals zero it means the button
                                        // is pressed
                                        // get the object can be focused next
                                        pFocusedObj = GOLGetFocusNext();
                                        //  move focus
                                        GOLSetFocus(pFocusedObj);
                                }else{
                                        // if RD13 equals one it means the button is
                                        // released

                                } // end of else
                                                // state of the button was changed
                                previousKey2State = ! previousKey2State;
                                // pass the message to the graphics library
                                GOLMsg(&msg);
                        } // end of if

                } // end of if
        } // end of while
        return 0;

} // end of main

WORD GOLMsgCallback(WORD objMsg, OBJ_HEADER* pObj, GOL_MSG* pMsg)
{
        // Application should process messages here

        if(GetobjID(pObj) == BUTTON1_ID)                // if the button with BUTTON_ID is
                                                        // receiving the message
        {
                if(objMsg == BTN_MSG_PRESSED)
                {
                        BtnSetText(pObj,”Pressed”);    // set text for pressed state
                }
                if(objMsg == BTN_MSG_RELEASED)
                {
                        BtnSetText(pObj,”Released”);   // set text for released state
                }
        }

        return 1; // process the message by default
} // end of GOLMessageCallback
```

## CONCLUSION

Any type of keyboard can easily be integrated into an application using the Microchip Graphics Library. This is done with a message interface, widgets messages, message processing and keyboard focus control.

**Note the following details of the code protection feature on Microchip devices:**

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.



**Trademarks**



QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

*Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.*

# WORLDWIDE SALES AND SERVICE

## AMERICAS

**Corporate Office**
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

**Atlanta**
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

**Boston**
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

**Chicago**
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

**Dallas**
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

**Detroit**
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

**Kokomo**
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

**Los Angeles**
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

**Santa Clara**
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

**Toronto**
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

## ASIA/PACIFIC

**Asia Pacific Office**
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

**Australia - Sydney**
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

**China - Beijing**
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

**China - Chengdu**
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

**China - Hong Kong SAR**
Tel: 852-2401-1200
Fax: 852-2401-3431

**China - Nanjing**
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

**China - Qingdao**
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

**China - Shanghai**
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

**China - Shenyang**
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

**China - Shenzhen**
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

**China - Wuhan**
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

**China - Xiamen**
Tel: 86-592-2388138
Fax: 86-592-2388130

**China - Xian**
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

**China - Zhuhai**
Tel: 86-756-3210040
Fax: 86-756-3210049

## ASIA/PACIFIC

**India - Bangalore**
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

**India - New Delhi**
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

**India - Pune**
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

**Japan - Yokohama**
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

**Korea - Daegu**
Tel: 82-53-744-4301
Fax: 82-53-744-4302

**Korea - Seoul**
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

**Malaysia - Kuala Lumpur**
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

**Malaysia - Penang**
Tel: 60-4-227-8870
Fax: 60-4-227-4068

**Philippines - Manila**
Tel: 63-2-634-9065
Fax: 63-2-634-9069

**Singapore**
Tel: 65-6334-8870
Fax: 65-6334-8850

**Taiwan - Hsin Chu**
Tel: 886-3-572-9526
Fax: 886-3-572-6459

**Taiwan - Kaohsiung**
Tel: 886-7-536-4818
Fax: 886-7-536-4803

**Taiwan - Taipei**
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

**Thailand - Bangkok**
Tel: 66-2-694-1351
Fax: 66-2-694-1350

## EUROPE

**Austria - Wels**
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

**Denmark - Copenhagen**
Tel: 45-4450-2828
Fax: 45-4485-2829

**France - Paris**
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

**Germany - Munich**
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

**Italy - Milan**
Tel: 39-0331-742611
Fax: 39-0331-466781

**Netherlands - Drunen**
Tel: 31-416-690399
Fax: 31-416-690340

**Spain - Madrid**
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

**UK - Wokingham**
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08