

PIC32 Bootloader

*Author: Ganapathi Ramachandra
Microchip Technology Inc.*

INTRODUCTION

The bootloader for PIC32 devices is used to upgrade firmware on a target device without the need for an external programmer or debugger.

The bootloader consists of the following applications:

- Five bootloader firmware implementations:
 - Universal Asynchronous Receiver Transmitter (UART)
 - Universal Serial Bus (USB) device based on the Human Interface Device (HID) class
 - USB host based on the Mass Storage Device (MSD) class
 - Ethernet
 - Secure Digital (SD) card
- A demonstration application, which can be downloaded into the target PIC32 device using the bootloader
- A PC host application (required for UART, USB HID and Ethernet bootloaders only) to communicate with the bootloader firmware running inside the PIC32 device. This application is used to perform erase and programming operations.

PREREQUISITES

The prerequisites for operating the bootloader application are as follows:

- A PC with MPLAB® IDE version 8.60 or later, or MPLAB X Beta version 7.12 or later installed, and the C32 compiler version 2.01 or later installed
- Development hardware based on the selected project, as listed in [TABLE 9: "Available Bootloader Workspaces"](#)
- A USB-to-serial port converter (if the COM port is not available on the PC) for the UART bootloader
- A USB Flash drive for use with the USB mass storage bootloader
- A SD card for use with the SD card bootloader
- An Ethernet (RJ-45) crossover cable for use with the Ethernet bootloader
- A traditional programming tool for initially writing the bootloader firmware into the PIC32 device (such as MPLAB® REAL ICE™ In-Circuit Emulator or the MPLAB ICD 3 In-Circuit Debugger). PIC32 starter kits do not require any programming tools.

Before using the PIC32 Bootloader, the user should be familiar with the following concepts:

- PIC32 device Configuration registers
- Compiling and programming a PIC32 device
- PIC32 linker scripts

BASIC FLOW OF THE BOOTLOADER

The flowchart in [Figure 3](#) illustrates the operation of the bootloader application. The bootloader code starts executing on a device Reset. If there are no conditions to enter the firmware upgrade mode, the bootloader starts executing the user application. The bootloader performs Flash erase/program operations while in the firmware upgrade mode.

Entering the Firmware Upgrade Mode

On a device Reset, the bootloader forces itself into the firmware upgrade mode if the content of the user application's reset vector address is erased. To manually force the bootloader into the firmware upgrade mode, press and hold the switch, S3, on the Explorer 16 Development Board during power-up. On PIC32 starter kits, press and hold the switch, SW3, during power-up. While in firmware upgrade mode, the LED labeled D5 on the Explorer 16 Development Board and the LED labeled LED3 on the PIC32 starter kit will blink.

Exiting the Firmware Upgrade Mode

For USB HID, Ethernet, or the UART bootloader, the firmware upgrade mode can be exited either by applying a hard Reset to the device, or by sending a "Jump to Application" command from the PC. For the USB Flash drive or SD card bootloader, the firmware upgrade mode is exited either by a hard Reset or upon completion of firmware programming.

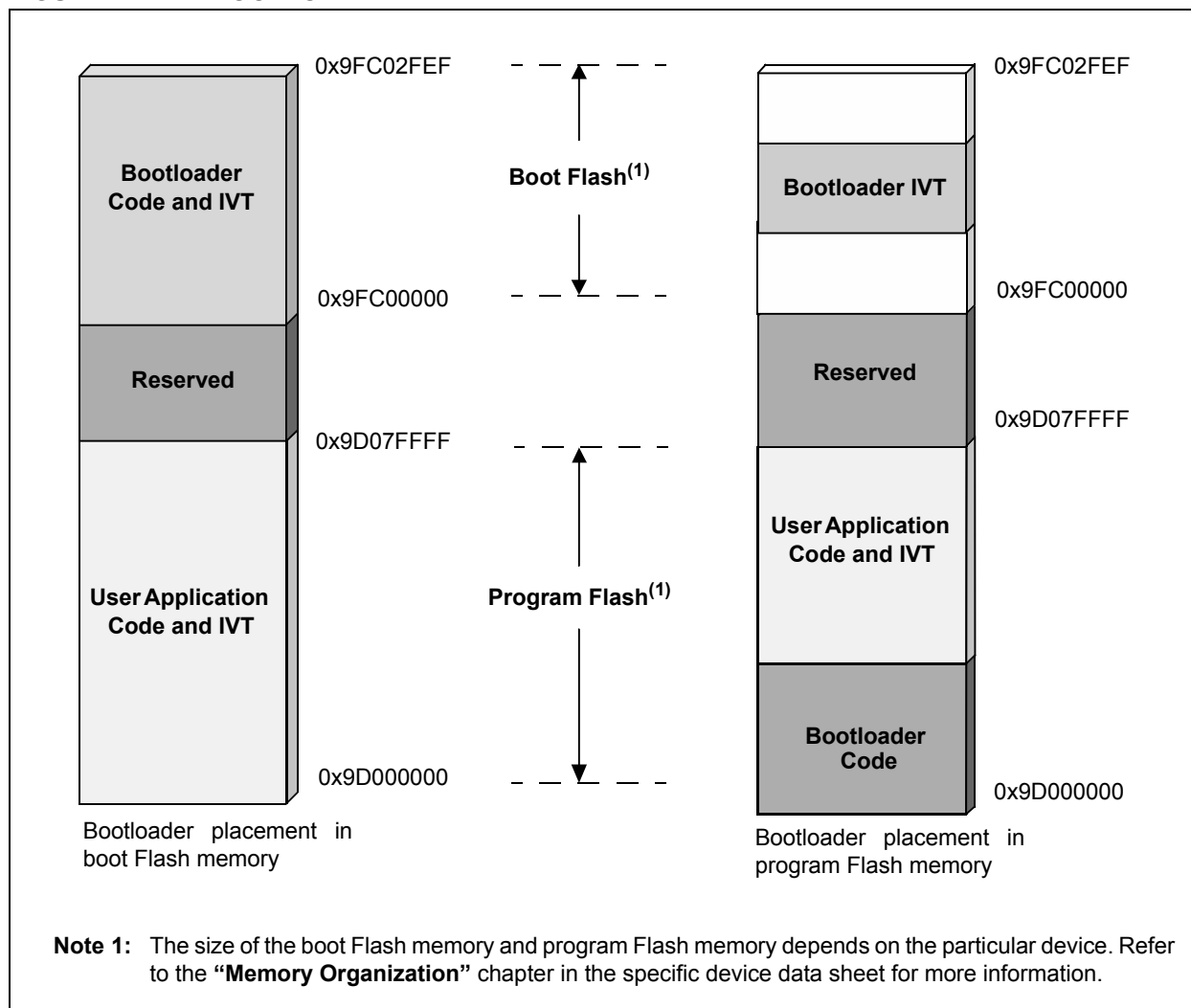
Note: The bootloader should disable and clear any enabled interrupts before running the user application. The stray interrupts from the bootloader may interfere with the user application and cause the application to fail. If applicable, interrupts and peripherals should be reinitialized in the user application.

BOOTLOADER PLACEMENT IN MEMORY

Figure 1 illustrates two schemes for the bootloader placement based on the size of the bootloader. Bootloaders that are smaller in size are placed within the PIC32 boot Flash memory. Fitting the bootloader application within the boot Flash memory provides the complete program Flash memory for the user application.

In the case of bootloaders that exceed the size of PIC32 boot Flash, the bootloader is split into two parts. The Interrupt Vector Table (IVT) and the C start-up code are placed in boot Flash, and the remaining portion of the bootloader is placed inside the program Flash.

FIGURE 1: BOOTLOADER PLACEMENT



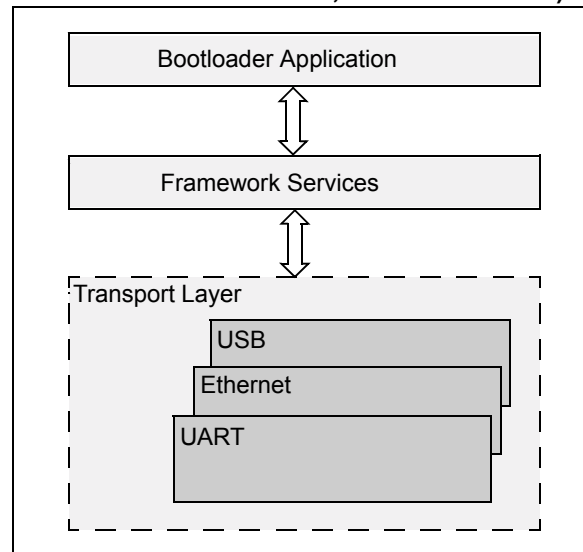
IMPLEMENTATION OVERVIEW (UART, USB HID, AND ETHERNET)

The bootloader application is implemented using a framework. The bootloader firmware communicates with the PC host application by using a predefined communication protocol. The bootloader framework provides Application Programming Interface (API) functions to handle the protocol related frames from the PC application. For more information on the communication protocol, see [Appendix B: “Bootloader Communication Protocol \(UART, USB HID, and Ethernet\)”](#).

FRAMEWORK (UART, USB HID, AND ETHERNET)

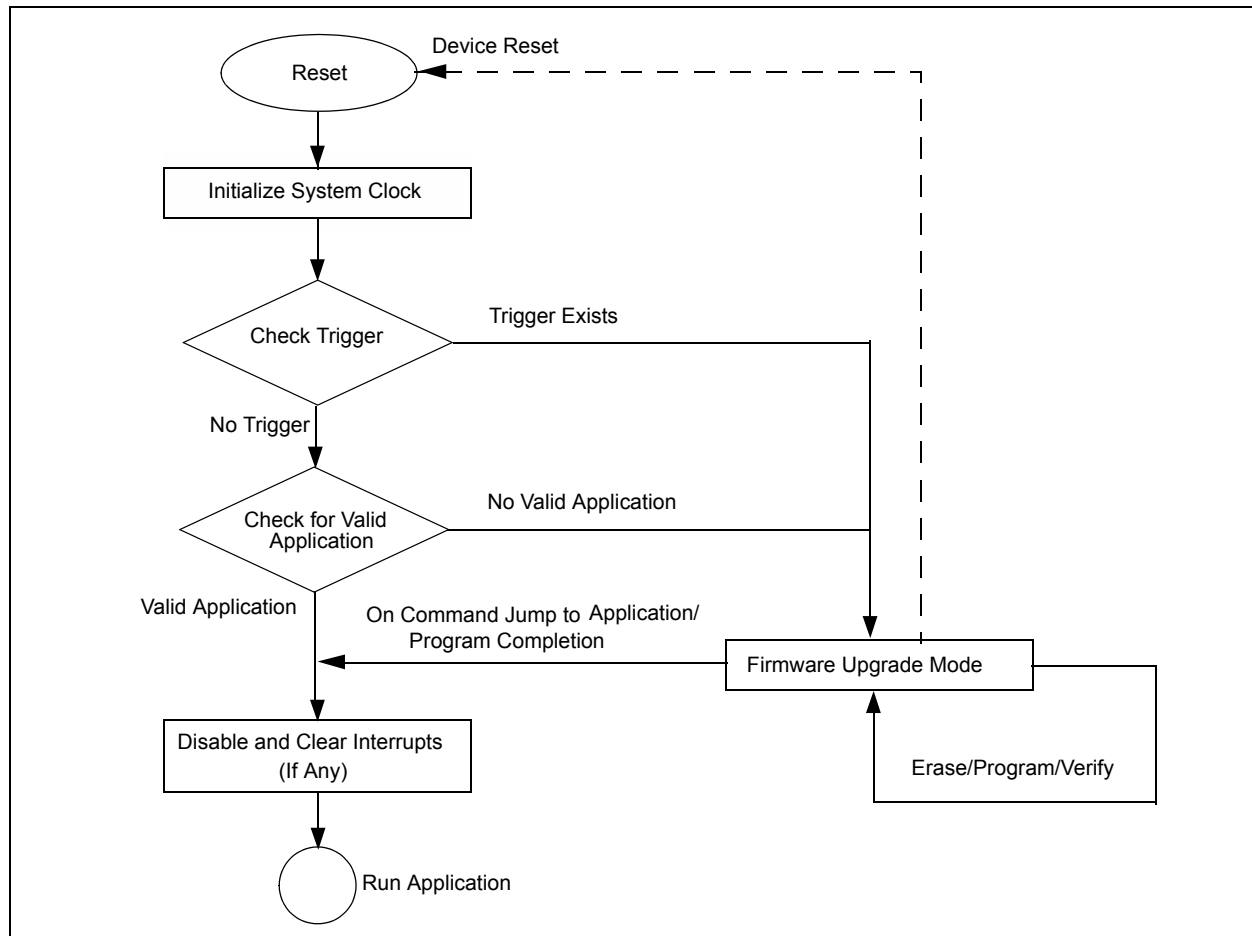
[Figure 2](#) illustrates the bootloader architecture. The bootloader framework provides several API functions, which can be called by the bootloader application and the transport layer. The bootloader framework assists the user to easily modify the bootloader application to adapt to different requirements.

FIGURE 2: BOOTLOADER ARCHITECTURE (UART, USB HID, AND ETHERNET)



The `Bootloader.c` file contains the bootloader application code. This file includes the bootloader functionality, and is illustrated in [Figure 3](#).

FIGURE 3: BOOTLOADER OPERATION



The `Framework.c` file contains the framework functions. The framework handles the communication protocol frames, and executes commands received from the PC host application.

The transport layer files, `Uart.c` and `Usb_HID_tasks.c`, include the functionality for transmitting and receiving the raw bytes to and from the PC host application.

[Table 1](#) lists the API functions provided by the framework.

TABLE 1: FRAMEWORK API DESCRIPTIONS (UART, USB HID, AND ETHERNET)

API	Description
<code>void FrameWorkTask(void)</code>	<p>This function executes the command if there is a valid frame from the PC host application. It must be called periodically from the bootloader application task.</p> <p>Input Parameters: None.</p> <p>Return: None.</p>
<code>void BuildRxFrame(UINT8 *RxData, INT16 RxLen)</code>	<p>The transport layer calls this function when it receives data from the PC host application.</p> <p>Input Parameters: <code>*RxData</code> – Pointer to the received data buffer <code>RxLen</code> – Length of the received data bytes</p> <p>Return: None.</p>
<code>UINT GetTransmitFrame(UINT8* TxData)</code>	<p>Returns a non-zero value if there is a valid response frame from the framework. The transport layer calls this function to check if there is a frame to be transmitted to the PC host application.</p> <p>Input Parameters: <code>*TxData</code> – Pointer to a data buffer that will carry the response frame</p> <p>Return: Length of the response frame. Zero indicates no response frame.</p>
<code>BOOL ExitFirmwareUpgradeMode(void)</code>	<p>This function directs the bootloader application to exit the Firmware Upgrade mode, and executes the user application. This can happen when the PC host application issues the <code>run application</code> command.</p> <p>Input Parameters: None.</p> <p>Return: If value is true, exit Firmware Upgrade mode.</p>

HANDLING DEVICE CONFIGURATION BITS

The bootloader does not erase or write the device Configuration words while programming the new application firmware. This is because the device Configuration words settings are shared by both the bootloader and the user application. Any modification to the device Configuration words may make the bootloader non-functional. Therefore, it is highly recommended to have common device Configuration words settings for both the user application and the bootloader.

DEMONSTRATION APPLICATION

The `Demo_Application` folder contains a sample demonstration application that controls two LEDs causing them to blink. [Table 2](#) lists the two workspaces that support the Explorer 16 Development Board and PIC32 starter kits. The demonstration application uses the custom linker script file to map the entire application into the program Flash without overlapping the bootloader.

Use the following procedure to configure and build the project:

1. In MPLAB or MPLAB X, open the desired workspace.
2. Select the desired device part number in the IDE.
3. Build the project in Release mode. The project compiles and generates a `.hex` file.

The resulting application Hex file can be downloaded into the selected development board through the bootloader. This demonstration application controls two LEDs labeled D9 and D10 on the Explorer 16 Development Board, or the LEDs labeled LED1 and LED2 on a PIC32 starter kit, causing them to blink.

TABLE 2: EXPLORER 16 DEVELOPMENT BOARD AND PIC32 STARTER KIT WORKSPACES

Workspace	Hardware Resource	Compatible Devices	Action
For MPLAB®: <code>Demo_App_Explorer16.mcp</code> For MPLAB X: <code>Demo_App_Explorer16.X</code>	Explorer 16 Development Board	PIC32MX1XX ⁽¹⁾ PIC32MX2XX ⁽¹⁾ PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	Blinks LEDs D9 and D10.
For MPLAB: <code>Demo_App_PIC32_Starter_Kits.mcp</code> For MPLAB X: <code>Demo_App_PIC32_Starter_Kits.X</code>	PIC32 USB Starter Kit II or PIC32 Ethernet Starter Kit	PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	Blinks LEDs LED1 and LED2.

Note 1: The demonstration application requires firmware modification to map the LEDs to the correct I/O ports.

USING THE BOOTLOADER APPLICATION (UART, USB HID, AND ETHERNET BOOTLOADERS)

Use the following procedure to run the UART/USB HID/Ethernet bootloader:

1. Using [Table 9](#), select the specific hardware setup for the selected bootloader.
2. With a debugger or programmer, program the bootloader into the device using Release Build.
3. Run `PIC32UBL.exe`, which is located in the `..\PC_Application\` folder.
4. Depending on the bootloader workspace used, enable either the USB, Serial port, or Ethernet using the **Communication Settings** menu in the PC application. For the Serial Port bootloader, the default baud rate is 115200. For the USB bootloader, the default values of Vendor ID number (VID) and Product ID number (PID) are 0x4D8 and 0x03C, respectively. For the Ethernet bootloader, the default IP address is 192.168.1.11.
5. If you are using the Ethernet bootloader, set the IP address and subnet mask of the PC to 192.168.1.12 and 255.255.255.0, respectively.
6. To enter the firmware upgrade mode, use the procedure as described in [“Entering the Firmware Upgrade Mode”](#).
7. Depending on the type of bootloader programmed, connect the serial cable, micro-USB cable, or the Ethernet (RJ-45) crossover cable to the Explorer 16 Development Board.
8. Click **Connect**. The device connects and the bootloader version information is read.
9. Click **Load Hex File**, and browse to the `Demo_Application` folder.
10. Select the specific demonstration application file `Demo_App_Explorer16.hex`/`Demo_App_PIC32_Starter_Kits.hex`, located in the `Firmware\Demo_Application` folder.
11. Click **Erase** to erase the device.
12. Click **Program** to program the previously loaded `.hex` file into the device Flash.
13. Click **Verify** to verify the Flash contents. If the Flash is written correctly, the “Verification successful” message is displayed in the console.
14. Click **Run Application**. The application must run and perform an action, as listed in the **Remarks** column in [Table 2](#). Optionally, steps 11, 12 and 13 can be performed as a single action by clicking **Erase-Program-Verify**.
15. Once the device starts running the programmed application firmware, the PC application will not be able to communicate with the device. To reconnect to the bootloader, return to step 6.

RUNNING THE USB HOST FLASH DRIVE BOOTLOADER

Use the following procedure to run the USB host Flash drive bootloader:

1. Using [Table 9](#), select the hardware setup for the USB host Flash drive bootloader.
2. Using a debugger or programmer, program the USB host MSD bootloader into the device.
3. After programming the device, perform the procedure as described in [“Entering the Firmware Upgrade Mode”](#) to place the bootloader into the firmware upgrade mode.
4. Copy the application `.hex` file to a USB Flash drive and rename the hex file to `image.hex`.
5. Insert the USB Flash drive into the selected development hardware. The bootloader begins programming the image into the program Flash memory of the device. If applicable, the LED on the Flash drive blinks during the programming operation. After programming has completed, the bootloader exits and starts running the application.

Note: If the device does not recognize the Flash drive (or the file on the Flash drive), format the Flash drive in the FAT32 format.

RUNNING THE SD CARD BOOTLOADER

Use the following procedure to run the SD card bootloader:

1. Using [Table 9](#), select the specific hardware setup for the SD card bootloader.
2. Copy the application `.hex` file to a SD card, and rename the hex file to `image.hex`. Insert the SD card into the selected hardware setup.
3. Program the SD card bootloader into the device using Release Build.
4. After programming the device, perform the procedure as described in [“Entering the Firmware Upgrade Mode”](#) to place the bootloader into the firmware upgrade mode.
5. Once inside the firmware upgrade mode, the bootloader begins programming the hex image into program Flash. During the programming operation, the indicator LED on the selected development hardware will blink at a faster rate. After programming has completed, the bootloader exits and starts running the user application.

Note: If the device does not recognize the file on SD card, format the SD card in the FAT32 format.
--

CONCLUSION

This application note provides the concepts of the PIC32 bootloader, bootloader memory mapping, bootloader framework API calls, and usage of the bootloader PC application.

[Appendix B: “Bootloader Communication Protocol \(UART, USB HID, and Ethernet\)”](#) explains the communication protocol and the commands used by the bootloader.

[Appendix C: “Considerations While Moving the Application Image”](#) explains the step-by-step approach to map the application to a different region inside the program Flash.

[Appendix D: “Bootloader Configurations”](#) explains the compile time settings available in the bootloader source code.

[Appendix E: “Bootloader Workspace”](#) lists all the available workspaces for the bootloader, and explains the procedure to program the bootloader firmware into the device.

REFERENCES

The following resources are available from Microchip Technology Inc.

These documents provide information on the terminologies used in the linker script file:

- *“MPLAB® Assembler, Linker and Utilities for PIC32 MCUs User’s Guide”* (DS51833)
- *“MPLAB® C Compiler for PIC32 MCUs User’s Guide”* (DS51686)

These documents provide information on the PIC32 device and its peripherals:

- *“PIC32MX1XX/2XX Data Sheet”* (DS61168)
- *“PIC32MX3XX/4XX Data Sheet”* (DS61143)
- *“PIC32MX5XX/6XX/7XX Data Sheet”* (DS61156)

APPENDIX A: SOURCE CODE

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

All of the software covered in this application note is available as a single WinZip archive file. This archive can be downloaded from the Microchip corporate Web site at:

www.microchip.com

APPENDIX B: BOOTLOADER COMMUNICATION PROTOCOL (UART, USB HID, AND ETHERNET)

The PC host application uses a communication protocol to interact with the bootloader firmware. The PC host application acts as a master and issues commands to the bootloader firmware to perform specific operations.

Frame Format

The communication protocol follows the frame format, as shown in [Example 1](#). The frame format remains the same in both directions, that is, from the host application to the bootloader, and from the bootloader to the host application.

EXAMPLE 1: FRAME FORMAT

```
[<SOH>...] <SOH> [<DATA>...] <CRCL><CRCH><EOT>
```

Where:

<...> Represents a byte

[...] Represents an optional or variable number of bytes

The frame starts with a control character, Start of Header (SOH), and ends with another control character, End of Transmission (EOT). The integrity of the frame is protected by two bytes of Cyclic Redundancy Check (CRC)-16, represented by CRCL (low-byte) and CRCH (high-byte).

Control Characters

Some bytes in the Data field may imitate the control characters, SOH and EOT. The Data Link Escape (DLE) character is used to escape such bytes that could be interpreted as control characters. The bootloader always accepts the byte following a <DLE> as data, and always sends a <DLE> before any of the control characters.

TABLE 3: CONTROL CHARACTER DESCRIPTIONS

Control	Hex Value	Description
<SOH>	0x01	Marks the beginning of a frame
<EOT>	0x04	Marks the end of a frame
<DLE>	0x10	Data link escape

Commands

The PC host application can issue the commands listed in [Table 4](#) to the bootloader. The first byte in the data field carries the command.

TABLE 4: COMMAND DESCRIPTION

Command Value in Hexadecimal	Description
0x01	Read the bootloader version information
0x02	Erase Flash
0x03	Program Flash
0x04	Read CRC
0x05	Jump to application

READ BOOTLOADER VERSION INFORMATION

The PC host application request for version information to the bootloader is shown in [Example 2](#).

EXAMPLE 2: REQUEST

```
[<SOH>...] <SOH> [<0x01>] <CRCL><CRCH><EOT>
```

The bootloader responds to the PC request for version information in two bytes, as shown in [Example 3](#).

EXAMPLE 3: RESPONSE

```
[<SOH>...] <SOH><0x01><MAJOR_VER><MINOR_VER>  
<CRCL><CRCH><EOT>
```

Where:

MAJOR_VER – Major version of the bootloader

MINOR_VER – Minor version of the bootloader

ERASE FLASH

On receiving the erase Flash command from the PC host application, the bootloader erases that part of the program Flash, which is allocated for the user application. The request frame from the PC host application to the bootloader is shown in [Example 4](#).

EXAMPLE 4: REQUEST

```
[<SOH>...] <SOH><0x02><CRCL><CRCH><EOT>
```

The response frame from the bootloader to the PC host application is shown in [Example 5](#).

EXAMPLE 5: RESPONSE

```
[<SOH>...] <SOH><0x02><CRCL><CRCH><EOT>
```

PROGRAM FLASH

The PC host application sends one or multiple hex records in Intel Hex format along with the program Flash command. The MPLAB C32 compiler generates the image in the Intel Hex format. Each line in the Intel hexadecimal file represents a hexadecimal record. Each hexadecimal record starts with a colon (:) and is in ASCII format. The PC host application discards the colon and converts the remaining data from ASCII to hexadecimal, and then sends the data to the bootloader. The bootloader extracts the destination address and data from the hex record, and writes the data into program Flash.

The request frame from the PC host application to the bootloader is shown in [Example 6](#).

EXAMPLE 6: REQUEST

```
[<SOH>...]<SOH><0x03>[<HEX_RECORD>...]<CRCL>  
<CRCH><EOT>
```

Where:

HEX_RECORD is the Intel Hex record in hexadecimal format

The response from the bootloader to the PC host application is shown in [Example 7](#).

EXAMPLE 7: RESPONSE

```
[<SOH>...]<SOH><0x03><CRCL><CRCH><EOT>
```

READ CRC

The read CRC command is used to verify the content of the program Flash after programming. The request frame from the PC host application to the bootloader is shown in [Example 8](#).

EXAMPLE 8: REQUEST

```
[<SOH>...]<SOH><0x04><ADRS_LB><ADRS_HB>  
<ADRS_UB><ADRS_MB><NUMBYTES_LB><NUMBYTES_HB>  
<NUMBYTES_UB><NUMBYTES_MB><CRCL><CRCH><EOT>
```

ADRS_LB, ADRS_HB, ADRS_UB and ADRS_MB, as shown in [Example 8](#), represent the 32-bit Flash addresses from where the CRC calculation begins.

NUMBYTES_LB, NUMBYTES_HB, NUMBYTES_UB and NUMBYTES_MB, as shown in [Example 8](#), represent the total number of bytes in 32-bit format for which the CRC is to be calculated.

The response from the bootloader to the PC host application is shown in [Example 9](#).

EXAMPLE 9: RESPONSE

```
[<SOH>...]<SOH><0x04><FLASH_CRCL><FLASH_CRCH>  
<CRCL><CRCH><EOT>
```

JUMP TO APPLICATION

The Jump to Application command from the PC host application commands the bootloader to execute the application. The request frame from the PC host application to the bootloader is shown in [Example 10](#).

EXAMPLE 10: REQUEST

```
[<SOH>...]<SOH><0x05><CRCL><CRCH><EOT>
```

There is no response to this command from the bootloader because the bootloader immediately exits the firmware upgrade mode and begins executing the application.

APPENDIX C: CONSIDERATIONS WHILE MOVING THE APPLICATION IMAGE

This section describes the procedure to place a user application into a desired program Flash memory region. It must be ensured that the user application's memory region does not overlap with the memory region reserved for the bootloader.

1. Create a new text file and save it with a `.ld` file extension.
2. Add the new `*.ld` file to your project. The new `*.ld` file appears in the project tree.
3. Starting with the default linker script is easier than starting from the scratch. Use a text editor to copy the contents of the `\pic32mx\lib\ldscripts\elf32pic32mx.x` default linker script into the newly created `*.ld` file. The `INCLUDE procdefs.ld` directive should be replaced with the contents of device specific `\pic32mx\lib\proc\device\procdefs.ld` portion of the linker script. The path `\pic32mx\lib` is located inside the folder where C32 compiler tools are installed.
4. Edit the newly created `*.ld` file to remap the linker script memory regions `exception_mem`, `kseg0_boot_mem`, `kseg1_boot_mem` and `kseg0_program_mem` into the **program Flash reserved for the user application**. The `exception_mem` must align on a 4K address boundary of the program Flash as shown in [Example 11](#).

For more information on linker script memory regions, refer to the “*MPLAB® Assembler, Linker and Utilities for PIC32 MCUs User's Guide*” (DS51833) and the “*MPLAB® C Compiler for PIC32 MCUs User's Guide*” (DS51686).

5. Set the value of `_ebase_address` to the `ORIGIN` value of `exception_mem`.
6. Change the values of memory addresses `_RESET_ADDR`, `_BEV_EXCPT_ADDR` and `_DBG_EXCPT_ADDR`, so that all these addresses fall inside `kseg1_boot_mem`.
7. Clean and build the application project to get a remapped application image.

Note: The modified application linker script file is not suitable for building the application in debug or stand-alone (to use without bootloader) mode.

The next step following the application remap is informing the bootloader about the new location and reset address of the user application in the program Flash.

The bootloader code provides compile time options for this purpose. The macros, `APP_FLASH_BASE_ADDRESS` and `APP_FLASH_END_ADDRESS`, define the start and the end addresses of the program Flash reserved for the user application. The bootloader performs an erase or program operation only if the target address of the Flash is within these addresses. Therefore, the user must set new start and end address values to these macros following the application remap. Set the addresses so that `exception_mem`, `kseg0_boot_mem`, `kseg1_boot_mem`, and `kseg0_program_mem` defined in the `procdefs.ld` file of the user application are within these addresses, as shown in [Example 12](#).

The macro, `USER_APP_RESET_ADDRESS`, specifies the reset address of the user application. The bootloader branches to this address when it must run the user application. The value of this macro must be changed to `_RESET_ADDR` defined in the `procdefs.ld` file of the user application project. The bootloader project must be recompiled and programmed into the PIC32 device after modifying these macros.

EXAMPLE 11: LINES TO MODIFY IN APPLICATION LINKER SCRIPT

```

/*****
 * Processor-specific object file. Contains SFR definitions.
 *****/
INPUT("processor.o")

/*****
 * For interrupt vector handling
 *****/
PROVIDE(_vector_spacing = 0x00000001);
/* _ebase_address value must be same as the ORIGIN value of exception_mem (see below) */
_ebase_address = 0x9D000000;

/*****
 * Memory Address Equates
 *****/
/* Equate _RESET_ADDR to the ORIGIN value of kseg1_boot_mem (see below) */
_RESET_ADDR = (0x9D000000 + 0x1000 + 0x970);

/*Map _BEV_EXCPT_ADDR and _DBG_EXCPT_ADDR in to kseg1_boot_mem (see below) */
/* Place _BEV_EXCPT_ADDR at an offset of 0x380 to _RESET_ADDR */
/* Place _DBG_EXCPT_ADDR at an offset of 0x480 to _RESET_ADDR */

_BEV_EXCPT_ADDR = (0x9D000000 + 0x1000 + 0x970 + 0x380);
_DBG_EXCPT_ADDR = (0x9D000000 + 0x1000 + 0x970 + 0x480);

_DBG_CODE_ADDR = 0xBFC02000;
_DBG_CODE_SIZE = 0xFF0;
_GEN_EXCPT_ADDR = _ebase_address + 0x180;

/*****
 * Memory Regions
 *
 * Memory regions without attributes cannot be used for orphaned sections.
 * Only sections specifically assigned to these regions can be allocated
 * into these regions.
 *****/
MEMORY
{
    /* IVT is mapped into the exception_mem. ORIGIN value of exception_mem must align with 4K
       address boundary. Keep the default value for the length */

    exception_mem : ORIGIN = 0x9D000000, LENGTH = 0x1000

    /* Place kseg0_boot_mem adjacent to exception_mem. Keep the default value for the length */
    kseg0_boot_mem : ORIGIN = (0x9D000000 + 0x1000), LENGTH = 0x970

    /* C Start-up code is mapped into kseg1_boot_mem. Place kseg1_boot_mem adjacent to
       kseg0_boot_mem. Keep the default value for the length */
    kseg1_boot_mem : ORIGIN = (0x9D000000 + 0x1000 + 0x970), LENGTH = 0x490

    /*All C files (Text and Data) are mapped into kseg0_program_mem. Place kseg0_program_mem
       adjacent to kseg1_boot_mem. Change the length of kseg0_program_mem as required. In this
       example, 512 KB Flash size is shrunk as follows: */

    kseg0_program_mem (rx):ORIGIN = (0x9D000000 + 0x1000 + 0x970 + 0x490),
        LENGTH = (0x80000 - (0x1000 + 0x970 + 0x490))

    debug_exec_mem : ORIGIN = 0xBFC02000, LENGTH = 0xFF0
    config3 : ORIGIN = 0xBFC02FF0, LENGTH = 0x4
    config2 : ORIGIN = 0xBFC02FF4, LENGTH = 0x4
    config1 : ORIGIN = 0xBFC02FF8, LENGTH = 0x4
    config0 : ORIGIN = 0xBFC02FFC, LENGTH = 0x4
    kseg1_data_mem (w!x) : ORIGIN = 0xA0000000, LENGTH = 0x20000
    sfrs : ORIGIN = 0xBF800000, LENGTH = 0x100000
    configsfrs : ORIGIN = 0xBFC02FF0, LENGTH = 0x10
}

```

EXAMPLE 12: LINES TO MODIFY IN BOOTLOADER CODE

```
/* APP_FLASH_BASE_ADDRESS and APP_FLASH_END_ADDRESS reserves program Flash for the application */
/* Rule:
    1) The memory regions kseg0_program_mem, kseg0_boot_mem, exception_mem and
       kseg1_boot_mem of the application linker script must fall within APP_FLASH_BASE_ADDRESS
       and APP_FLASH_END_ADDRESS

    2) The base address and end address must align on 4K address boundary
*/

#define APP_FLASH_BASE_ADDRESS    0x9D000000
#define APP_FLASH_END_ADDRESS    0x9D07FFFF

/* Address of the Flash from where the application starts executing */
/* Rule: Set APP_FLASH_BASE_ADDRESS to _RESET_ADDR value of application linker script */

#define USER_APP_RESET_ADDRESS (0x9D000000 + 0x1000 + 0x970)
```

APPENDIX D: BOOTLOADER CONFIGURATIONS

The bootloader code provides a few macros for configuring the settings while compiling. [Table 5](#) through [Table 8](#) list these macros and their usage. Depending on user requirements, the values in these macros may need to be changed.

TABLE 5: GENERAL MACROS

Macro	Usage
APP_FLASH_BASE_ADDRESS	Base address of the program Flash reserved for the user application. The address value must point to the beginning of a 4K Flash page.
APP_FLASH_END_ADDRESS	End address of the program Flash reserved for the user application. The address value must point to the end of a 4K Flash page.
USER_APP_RESET_ADDRESS	Address of user Reset vector. The bootloader branches to this address when it must run the user application.
MAJOR_VERSION	Major version of the bootloader firmware.
MINOR_VERSION	Minor version of the bootloader firmware.

TABLE 6: UART BOOTLOADER MACRO

Macro	Usage
DEFAULT_BAUDRATE	Sets the UART baud rate.

TABLE 7: USB HID BOOTLOADER MACROS

Macro	Usage
USB_VENDOR_ID	Sets the vendor ID.
USB_PRODUCT_ID	Sets the product ID.

TABLE 8: ETHERNET BOOTLOADER MACROS

Macro	Usage
MY_DEFAULT_MAC_BYTE1 MY_DEFAULT_MAC_BYTE2 MY_DEFAULT_MAC_BYTE3 MY_DEFAULT_MAC_BYTE4	Sets the MAC address.
MY_DEFAULT_IP_ADDR_BYTE1 MY_DEFAULT_IP_ADDR_BYTE2 MY_DEFAULT_IP_ADDR_BYTE3 MY_DEFAULT_IP_ADDR_BYTE4	Sets the IP address.

APPENDIX E: BOOTLOADER WORKSPACE

The `Bootloader` folder contains the bootloader firmware source code. The available workspaces are listed in [Table 9](#).

Programming the Bootloader Firmware

Use the following procedure to select a suitable workspace depending on the hardware configuration and the selected bootloader type.

1. Using MPLAB or MPLAB X, open the specific bootloader workspace.
2. Select the specific device part number in IDE, and rebuild the project.
3. After building the project, program the bootloader into the device.
4. Use REAL ICE or ICD 3 to program the device on an Explorer 16 Development Board. The PIC32 starter kits do not require a programmer.

TABLE 9: AVAILABLE BOOTLOADER WORKSPACES

Project Name	Development Hardware	Compatible Devices	Bootloader Type	Bootloader Mapping
For MPLAB®: UART_Btl_Explorer16.mcp For MPLAB X: UART_Btl_Explorer16.X	Explorer 16 Development Board (DM24001) and a PIC32 Plug-in Module (PIM)	PIC32MX1XX ^(1,2) PIC32MX2XX ^(1,2) PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	UART	See Note 3
For MPLAB: UART_HID_Btl_StarterKit.mcp For MPLAB X: UART_HID_Btl_StarterKit.X	PIC32 Ethernet Starter Kit (DM320004) or the PIC32 USB Starter Kit II (DM320003-2)	PIC32MX2XX ^(1,2) PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	USB HID	See Note 3
For MPLAB: UART_MSD_Btl_StarterKit.mcp For MPLAB X: UART_MSD_Btl_StarterKit.X	PIC32 Ethernet Starter Kit (DM320004) or the PIC32 USB Starter Kit II (DM320003-2)	PIC32MX2XX ^(1,2) PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	USB MSD	See Note 3
For MPLAB: ETH_Btl_ETH_StarterKit.mcp For MPLAB X: ETH_Btl_ETH_StarterKit.X	PIC32 Ethernet Starter Kit (DM320004)	PIC32MX6XX PIC32MX7XX	Ethernet (with internal MAC)	See Note 3

Note 1: The bootloader code will need modifications to I/O port mapping for these devices.

2: The example bootloader linker script for these devices is provided in the `Bootloader\linker_scripts\PIC32MX_1XX_2XX` folder. The user must compile the bootloader with the specific linker script files.

3: Refer to the “Mapping Note” in the corresponding linker script file.

TABLE 9: AVAILABLE BOOTLOADER WORKSPACES (CONTINUED)

Project Name	Development Hardware	Compatible Devices	Bootloader Type	Bootloader Mapping
For MPLAB: ETH_Btl_Explorer16_ENC28.mcp For MPLAB X: ETH_Btl_Explorer16_ENC28.X	Explorer 16 Development Board (DM24001), a PIC32 PIM, and the Ethernet PICtail™ Plus Daughter Board (AC164123)	PIC32MX1XX ^(1,2) PIC32MX2XX ^(1,2) PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	Ethernet (with external Ethernet Controller ENC28J60)	See Note 3
For MPLAB: ETH_Btl_Explorer16_ENC624.mcp For MPLAB X: ETH_Btl_Explorer16_ENC624.X	Explorer 16 Development Board (DM24001), a PIC32 PIM, and the Fast 100 Mbps Ethernet PICtail™ Plus Daughter Board (AC164132)	PIC32MX1XX ^(1,2) PIC32MX2XX ^(1,2) PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	Ethernet (with external Ethernet Controller ENC624J600)	See Note 3
For MPLAB: SD_Card_Btl_Explorer16.mcp For MPLAB X: SD_Card_Btl_Explorer16.X	Explorer 16 Development Board (DM24001), a PIC32 PIM and the PICtail™ Daughter Board for SD™ and MMC Cards (AC164122)	PIC32MX1XX ^(1,2) PIC32MX2XX ^(1,2) PIC32MX3XX PIC32MX4XX PIC32MX5XX PIC32MX6XX PIC32MX7XX	SD Card	See Note 3

Note 1: The bootloader code will need modifications to I/O port mapping for these devices.

2: The example bootloader linker script for these devices is provided in the `Bootloader\linker_scripts\PIC32MX_1XX_2XX` folder. The user must compile the bootloader with the specific linker script files.

3: Refer to the “Mapping Note” in the corresponding linker script file.

APPENDIX F: REVISION HISTORY

Revision A (June 2011)

This is the initial released version of this document.

Revision B (January 2012)

This revision includes the following updates:

- Updated the list of bootloader applications in **“Introduction”**
- Updated **“Prerequisites”**
- Updated all content in **“Basic Flow of the Bootloader”**
- Removed the section **“Basic Settings”**
- Updated **Figure 1**
- Updated the titles in **“Implementation Overview (UART, USB HID, and Ethernet)”** and **“Framework (UART, USB HID, and Ethernet)”**
- Updated the title in **Figure 2**
- Updated **Figure 3**
- Updated the title in **Table 1**
- Added **“Demonstration Application”**
- Updated all content in **“Handling Device Configuration Bits”**
- Updated the title and the procedure in **“Using the Bootloader Application (UART, USB HID, and Ethernet Bootloaders)”**:
- Removed **Figure 5: Communication Settings**
- Removed **Figure 6: Bootloader Connected Successfully**
- Removed **Figure 7: PC Application Messages**
- Added **“Running the USB Host Flash drive Bootloader”**
- Added **“Running the SD Card Bootloader”**
- Updated **“Conclusion”**
- Updated the title in **Appendix B: “Bootloader Communication Protocol (UART, USB HID, and Ethernet)”**
- Updated all content in **Appendix C: “Considerations While Moving the Application Image”**
- Added **Appendix D: “Bootloader Configurations”**
- Added **Appendix E: “Bootloader Workspace”**
- Minor changes to the text and formatting were incorporated throughout the document

NOTES:

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.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscent Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICKit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2011-2012, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-61341-942-7

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2009 ==

Microchip received ISO/TS-16949:2009 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://www.microchip.com/support>
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

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

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

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

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

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: 86-10-8569-7000
Fax: 86-10-8528-2104

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

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189

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

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

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

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

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 - Osaka
Tel: 81-66-152-7160
Fax: 81-66-152-9310

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: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-330-9305

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