

---

## PIC32 Bootloader

---

<i>Author: Ganapathi Ramachandra Microchip Technology Inc.</i>
--

### INTRODUCTION

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

The bootloader consists of the following applications:

- The bootloader firmware, which is to be programmed into the target PIC32 device
- A demo application, which can be downloaded into the target PIC32 device using the bootloader
- A PC host application 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 running the bootloader application are as follows:

- A PC with MPLAB® IDE v8.60 or higher version installed
- Either a PIC32 Starter Kit or an Explorer 16 Development Board with a PIC32 Plug-in Module (PIM)
- A USB-to-serial port converter (if the COM port is not available on the PC)
- A traditional programming tool for initially writing the bootloader firmware into the PIC32 device (such as the MPLAB® REAL ICE™ In-Circuit Emulator or the ICD 3 In-Circuit Debugger). Starter kits do not require any programming tools.

The user should be familiar with the following concepts:

- Device Configuration registers
- Compiling and programming the PIC32 device
- PIC32 linker scripts

### CONCEPT

The bootloader application is placed in a dedicated section of the PIC32 Flash memory based on the following conditions:

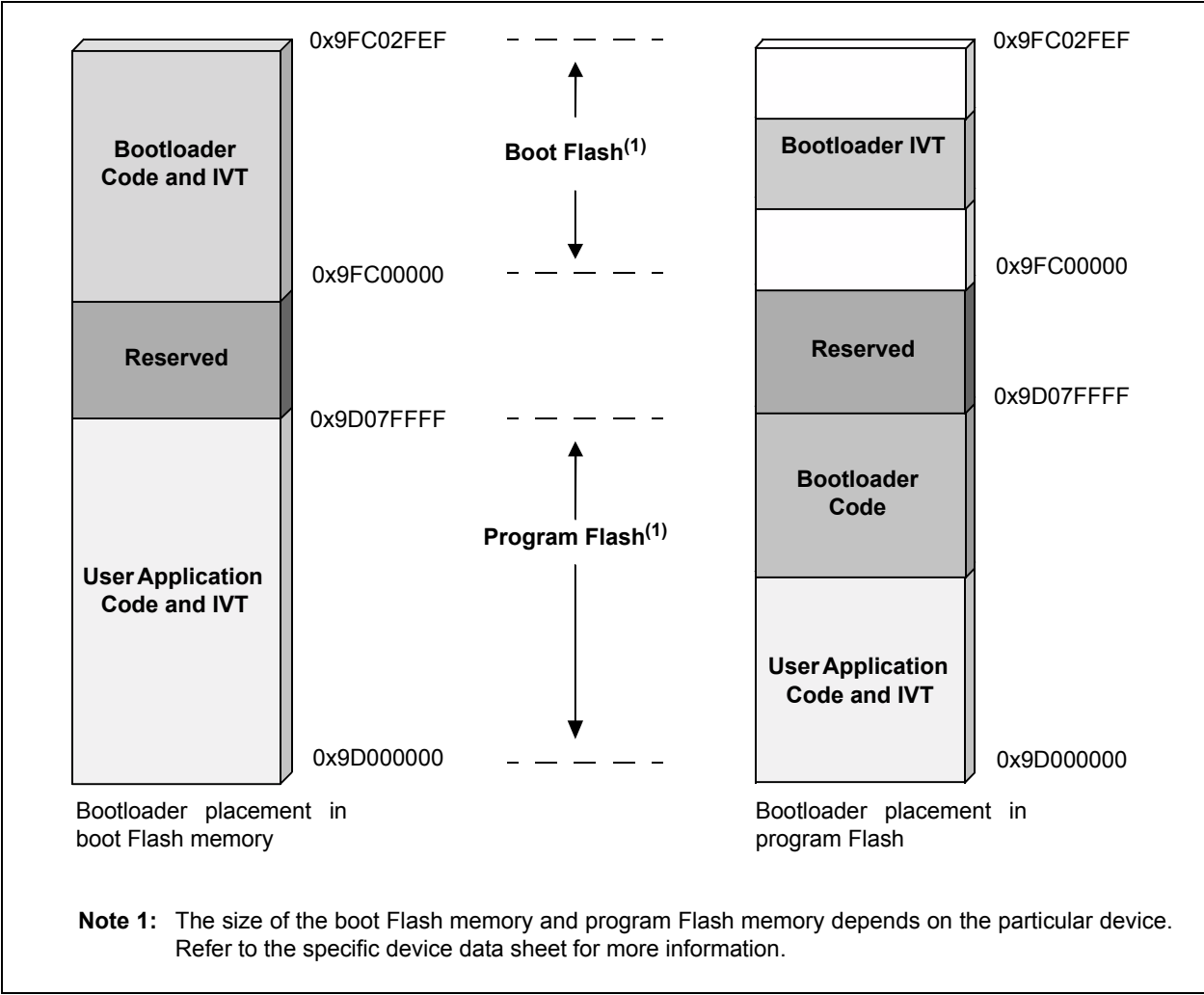
- If the size of the bootloader application is small, it can be placed within the PIC32 boot Flash memory. Fitting the bootloader application within the boot Flash memory provides the complete program Flash memory for the user application.
- If the size of the bootloader application is large, it must be placed in the program Flash memory. The user application can occupy the remaining area of the program Flash memory. In this case, the PIC32 boot Flash memory can be utilized to place the Interrupt Vector Table (IVT) of the bootloader application. [Figure 1](#) illustrates the placement of the bootloader and user application in the PIC32 Flash memory.

The bootloader code starts executing on a device Reset. If there is no valid application, or if there is an external trigger, the bootloader application enters Firmware Upgrade mode.

If there is no external trigger, but there is a valid user application, the bootloader code branches the control to the user application and starts executing the user application. The external trigger is a push button switch that the user must press during device boot-up. The bootloader application tests the validity of the user application by reading its reset address content. The bootloader application stays in Firmware Upgrade mode if this content is found to have been erased.

In Firmware Upgrade mode, the bootloader application continues checking for any commands from the PC host application. When a command is received from the PC host application, the bootloader performs the appropriate action, such as erasing or programming the program Flash memory, and so on. The flowchart in [Figure 3](#) illustrates the operation of the bootloader application.

FIGURE 1: BOOTLOADER PLACEMENT

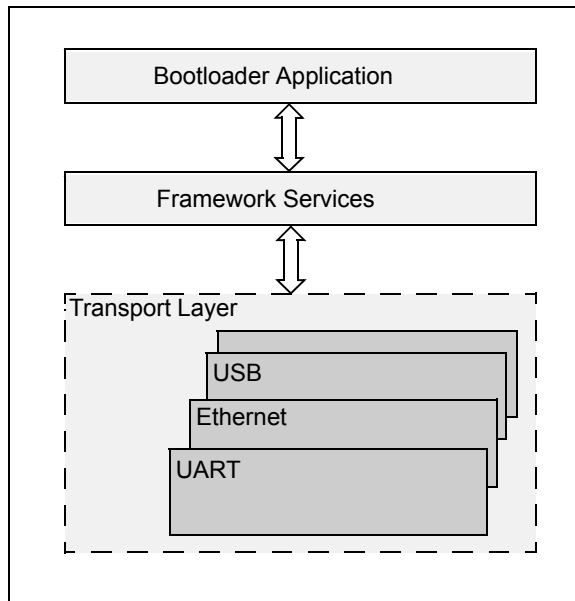


## IMPLEMENTATION OVERVIEW

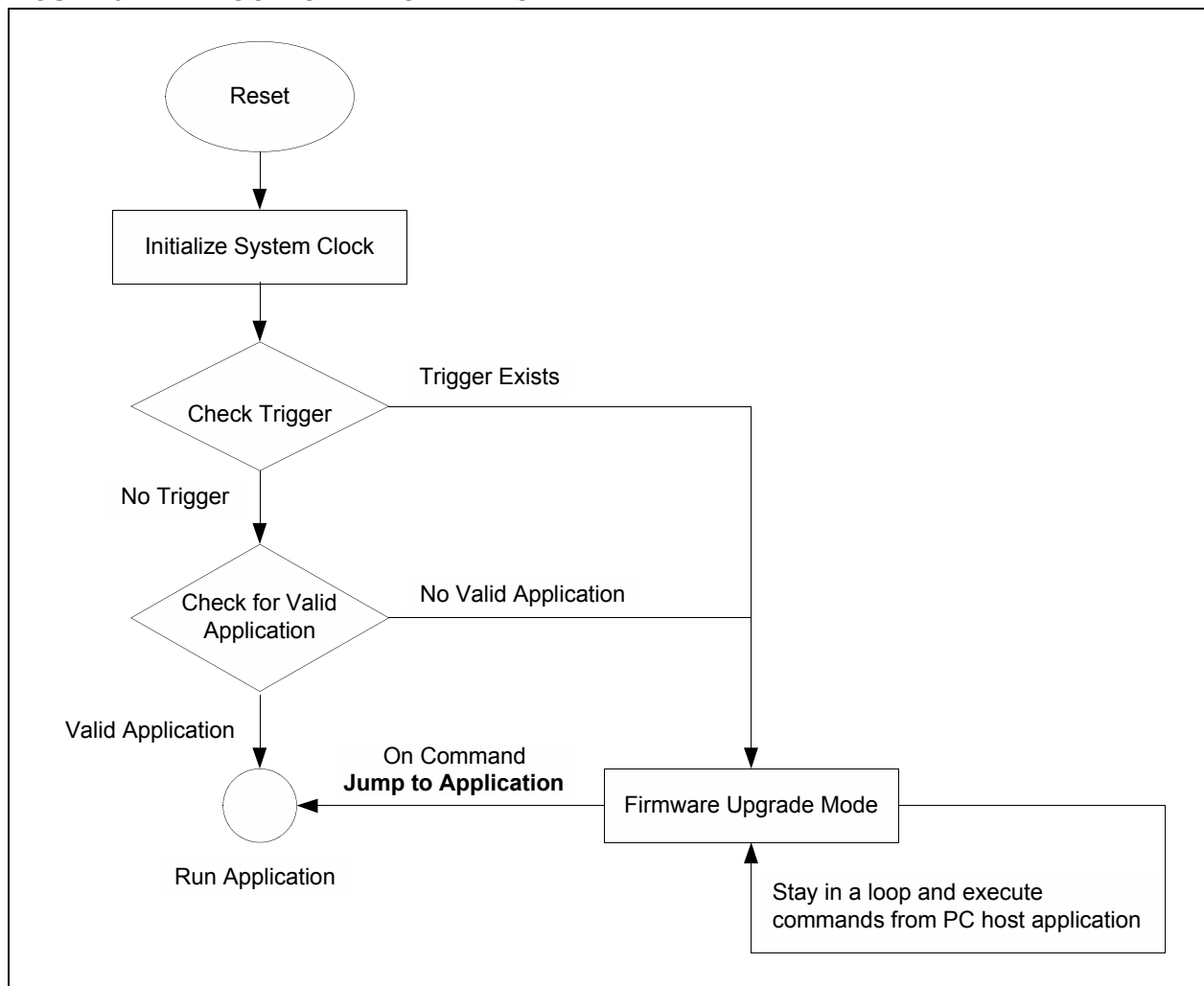
The bootloader application is implemented using a framework. The bootloader firmware communicates with the PC host application by using a predefined communication protocol. The bootloader framework provides Application Programming Interface (API) functions to handle the protocol related frames from the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

## FRAMEWORK

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

**FIGURE 2: BOOTLOADER ARCHITECTURE**

The `Bootloader.c` file contains the bootloader application code. This file includes the bootloader functionality, and is illustrated in the form of a flowchart 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.

The API functions provided by the framework are listed in [Table 1](#).

**TABLE 1: FRAMEWORK API DESCRIPTIONS**

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><b>Input Parameters:</b> None</p> <p><b>Return:</b> 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><b>Input Parameters:</b> <code>*RxData</code>: Pointer to the received data buffer <code>RxLen</code>: Length of the received data bytes</p>
<code>UINT GetTransmitFrame(UINT8* TxData)</code>	<p>Returns a non-zero value if there is 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><b>Input Parameters:</b> <code>*TxData</code>: Pointer to a data buffer which will carry response frame</p> <p><b>Return:</b> Length of the response frame. Zero indicates no response frame</p>
<code>BOOL ExitFirmwareUpgradeMode(void)</code>	<p>Framework 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><b>Input Parameters:</b> None</p> <p><b>Return:</b> If value is true, exit Firmware Upgrade mode</p>

## BASIC SETTINGS

The bootloader application provides the option to change the baud rate, if the UART module is selected as the communication port. By default, the bootloader baud rate is set to 115200. To change the baud rate, change the value of the `DEFAULT_BAUDRATE` macro found inside the `UART.h` file to the desired value.

By default, the bootloader software version is set to 1.0. The `MAJOR_VERSION` and `MINOR_VERSION` macros defined in the `Bootloader.h` file can be used to change the version of the bootloader application. The user must make sure that the complete project is rebuilt after performing these changes. The PC application can read the bootloader version and can be used by the user for version control of the bootloader firmware.

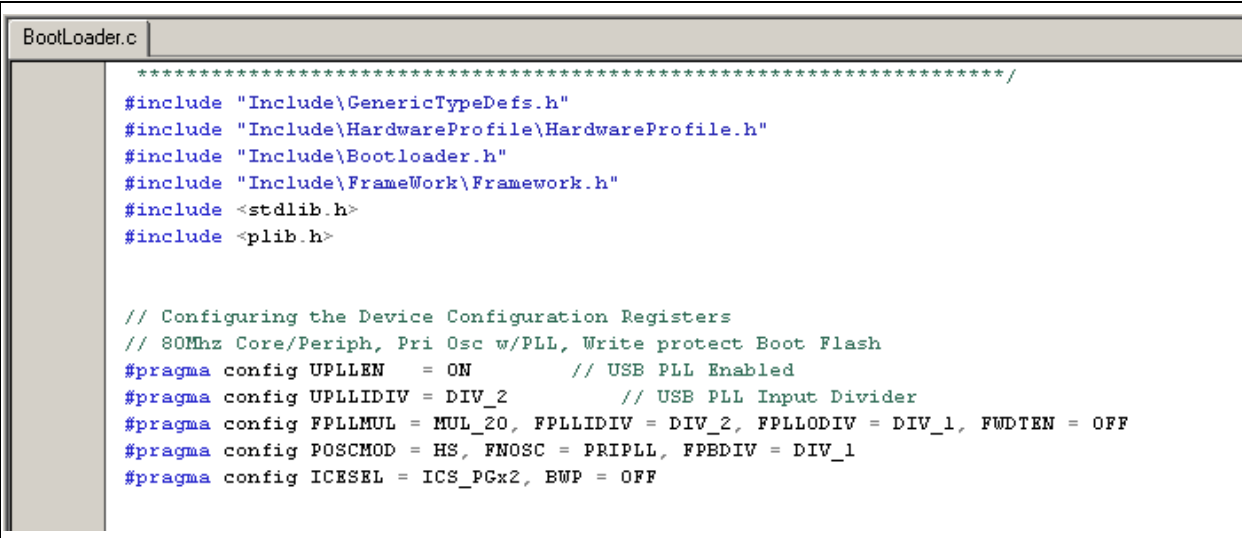
## DEVICE CONFIGURATION BITS

The PIC32 device family has programmable nonvolatile device Configuration registers that define the behavior of the device. For more information on the device Configuration registers, refer to the specific device data sheet.

**Note:** The device Configuration registers are located in the boot Flash memory section; therefore, these registers must be programmed along with the bootloader code. The bootloader does not modify the device Configuration registers while programming the application. Therefore, it is recommended to have the same configuration settings in both the application and the bootloader code.

The device Configuration registers can be configured by using the `#pragma config` compiler directive. Figure 4 shows a code snippet for configuring the device Configuration registers in the `bootloader.c` file.

**FIGURE 4: CONFIGURATION SETTINGS IN CODE**



```

BootLoader.c
*****/
#include "Include\GenericTypeDefs.h"
#include "Include\HardwareProfile\HardwareProfile.h"
#include "Include\Bootloader.h"
#include "Include\Framework\Framework.h"
#include <stdlib.h>
#include <plib.h>

// Configuring the Device Configuration Registers
// 80Mhz Core/Periph, Pri Osc w/PLL, Write protect Boot Flash
#pragma config UPLLEN = ON           // USB PLL Enabled
#pragma config UPLLIDIV = DIV_2      // USB PLL Input Divider
#pragma config FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FPLLODIV = DIV_1, FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL, FPBIDIV = DIV_1
#pragma config ICSEL = ICS_PGx2, BWP = OFF

```

Table 2 lists the device Configuration bits used in the bootloader application, and their settings. The remaining device Configuration bits, which are not included in this table, retain their default values.

**TABLE 2: BOOTLOADER CONFIGURATION SETTINGS**

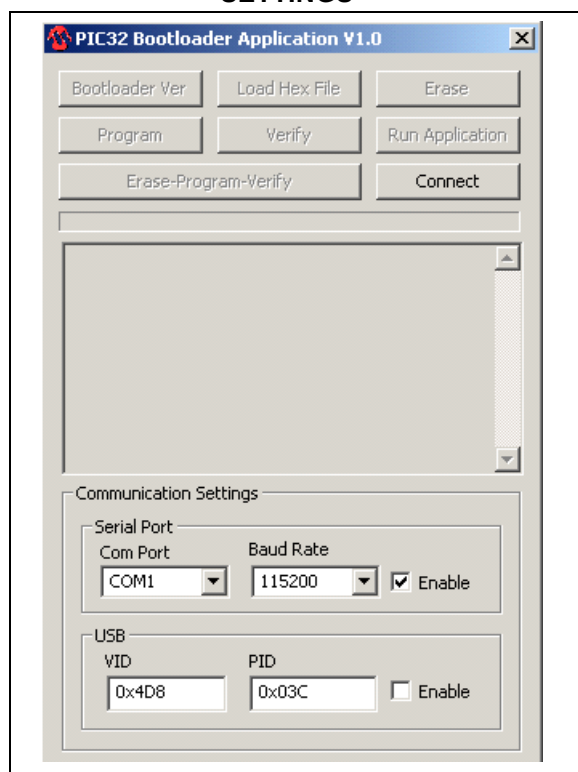
Configuration Bits	Setting
USB PLL Enable bit (UPLLEN)	Enabled.
PLL Input Divider bit (UPLLIDIV)	Set to 2x Divider.
PLL Multiplier bit (FPLLMUL)	Set to 20x Multiplier.
PLL Input Divider bit (FPLLIDIV)	Set to 2x Divider.
Default Postscaler for PLL bit (FPLLODIV)	PLL output divided by 1.
Watchdog Timer Enable bit (FWDTEN)	Disabled.
Primary Oscillator Configuration bit (POSCMOD)	High-speed Oscillator mode.
Oscillator Selection bit (FNOSC)	Primary oscillator with PLL mode.
Peripheral Bus Clock Divisor Default Value bit (FPBDIV)	System clock divided by 1.
In-Circuit Emulator/Debugger Communication Channel Select bit (ICESEL)	Set to 1 (In-Circuit Emulator uses EMUC2/EMUD2 pins; In-Circuit Debugger uses PGC2/PGD2 pins).

## USING THE BOOTLOADER APPLICATION

Follow these steps to use the bootloader application:

1. Refer to the associated `Readme.html` file to select an appropriate workspace and select and open the workspace in MPLAB IDE. Workspaces are located in the `Bootloader\Firmware\Bootloader` folder.
2. From the IDE menu, select an appropriate device part number, and then compile the bootloader project. Program the bootloader into the device using a programming tool such as MPLAB ICD 3, REAL ICE, or the PIC32 Starter Kit.
3. Disconnect the programmer from the PIC32 device.
4. Run the PC application file, `PIC32UBL.exe`, which is located in the `BootLoader\PC Application` folder.
5. Based on the workspace selected, choose the COM port or USB in the Communication Settings, as shown in [Figure 5](#).

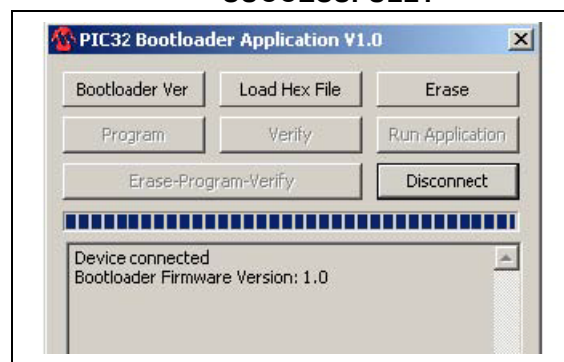
**FIGURE 5: COMMUNICATION SETTINGS**



6. Reset the PIC32 device and set the bootloader in Firmware Upgrade mode. Refer to the associated `Readme.html` file for the procedure to set the bootloader in Firmware Upgrade mode.

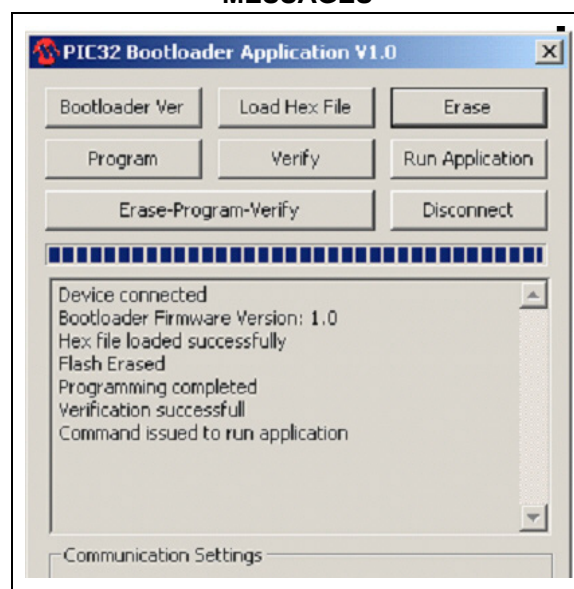
7. If the device is connected successfully, a confirmation message appears, as shown in [Figure 6](#).

**FIGURE 6: BOOTLOADER CONNECTED SUCCESSFULLY**



8. Click **Load Hex File**, and then browse to the location where the application image is located. The application image must be in the Intel Hex format. Select and load the appropriate application image file. The **Program** and **Erase-Program-Verify** buttons are enabled only if the Hex file is loaded successfully. The PC host application console displays the message "Hex file loaded successfully" indicating the successful loading of a Hex file, as shown in [Figure 7](#).

**FIGURE 7: PC APPLICATION MESSAGES**



9. Click **Erase** to erase the program Flash memory. The message "Flash Erased" appears in the console window indicating a successful Flash erase, as shown in [Figure 7](#).

10. Click **Program** to program the Flash. The PC host application begins writing the application image into the device. This step may take considerable time depending on the selected baud rate and the size of the selected application image. Upon completion, the "Programming completed" message appears in the console window, as shown in [Figure 7](#).
11. The integrity of the data programmed into program Flash memory is verified using a 16-bit Cyclic Redundancy Check (CRC). The CRC calculated by the PC host application and the CRC calculated by the device must match for the verification to pass. Upon CRC match, the "Verification successful" message appears in the console window, as shown in [Figure 7](#).
12. Click **Run Application** to run the user application firmware. After this step, the bootloader branches the control to the user application and the bootloader stops running. To invoke the bootloader again, execute step 6.

After successful completion of all of these steps, the PC application console window should look like [Figure 7](#). Erasing, programming and verifying can be performed in a single step by clicking **Erase-Program-Verify**.

## CONCLUSION

This application note presents 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"** 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.

## 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:

- "PIC32MX3XX/4XX Data Sheet" (DS61143)
- "PIC32MX5XX/6XX/7XX Family 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](http://www.microchip.com)

## APPENDIX B: BOOTLOADER COMMUNICATION PROTOCOL

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 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 CRC-16, represented by CRCL (low-byte) and CRCH (high-byte).

### 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

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.

### 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's 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 – Intel hex record in hex 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>
```

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>
```

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, shown in [Example 8](#), represent the total number of bytes in 32-bit format for which the CRC needs to be calculated.

## 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 Firmware Upgrade mode and begins executing the application.

## APPENDIX C: CONSIDERATIONS WHILE MOVING THE APPLICATION IMAGE

The bootloader code has three macros which specify the location and the reset address of the user application. These macros can be found in the `bootloader.h` file.

The `APP_FLASH_BASE_ADDRESS` and `APP_FLASH_END_ADDRESS` macros specify the start and end addresses of the program Flash that is reserved for the user application. The bootloader performs an erase or programming operation only if the target address falls within these addresses. The `USER_APP_RESET_ADDRESS` macro specifies the reset address of the user application from where it starts executing. The bootloader branches to this address when it must run the user application. The linker script file of the application project has to be modified to move the application image to a different section inside the program Flash.

The application image contains three sections: C start-up code, IVT and the program (text and data). The linker script file maps these sections of the application to `kseg0_boot_mem`, `exception_mem` and `kseg0_program_mem` memory sections, respectively. User has to modify only these three memory sections in the linker script file of the application project, to change the application mapping. For more information on these memory sections, 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).

The steps to modify the application mapping with an example are as follows:

1. Select a memory region in the program Flash to place the user application. The user application must not overlap with the memory region reserved for the bootloader. The base address of the user application must align with the beginning of a 4K Flash segment. The end address must align with the boundary of a 4K Flash segment. In this example, the base address and the end address of the user application are chosen as `0x9D005000` and `0x9D07FFFF`, respectively.
2. Inform the bootloader about the address range of the program Flash reserved for the application. For this example, the same can be done by setting the values of `APP_FLASH_BASE_ADDRESS` and `APP_FLASH_END_ADDRESS` to `0x9D005000` and `0x9D07FFFF`, respectively.

3. **Finding the Linker Script File:** Select an appropriate `procdefs.ld` linker header file from the path where MPLAB C32 compiler is installed. For example, for the PIC32MX795F512L device, copy the linker header file from: `C:\Program Files\Microchip\MPLAB32\pic32mx\lib\proc\32MX795F512L` (or from the path where MPLAB C32 compiler is installed). Copy the linker file into the root directory of the user application project folder.
4. **Mapping the IVT:** The base address of the IVT must align with the beginning of a 4K Flash segment. The interrupts on PIC32 may not work, if the IVT is not aligned with the beginning of a 4K Flash segment. In this example, the IVT will be placed in the last 4K segment of the program Flash reserved for the application. That is, the IVT will be mapped from `0x9D07F000` to `0x9D07FFFF`. Change the origin value of `exception_mem` to `0x9D07F000` in `procdefs.ld` file to modify the IVT mapping. [Example 11](#) shows the IVT mapping.

### EXAMPLE 11: SETTING THE EXCEPTION MEMORY

```
exception_mem : ORIGIN = 0x9D07F000, LENGTH
= 0x1000
```

The next line to change in the `procdefs.ld` file is the value of `_ebase_address`. The value of `_ebase_address` must be same as the origin value of `exception_mem`. Therefore, change the line, as shown in [Example 12](#).

### EXAMPLE 12: SETTING THE ebase REGISTER

```
_ebase_address = 0x9D07F000
```

5. **Mapping the C Start-up Code:** Place the `kseg0` boot memory just before the exception memory region. The `kseg0` boot memory contains the C start-up code; therefore, it is advised to keep the original value for the length of the boot memory. Calculate the origin value of the `kseg0` boot memory by subtracting its length (`0x970`) from the origin value of the exception memory (`0x9D07F000`). The line that defines `kseg0_boot_mem` is then changed, as shown in [Example 13](#).

### EXAMPLE 13: SETTING THE kseg0 BOOT MEMORY

```
kseg0_boot_mem : ORIGIN = 0x9D07E690,
LENGTH=0x970
```

6. **Mapping the Program Code:** Allocate the remaining memory region (from the user application base address to the beginning of the kseg0 boot memory) to kseg0 program memory. Set the kseg0 program memory origin to the user application base address. The length can be calculated by subtracting the user application base address value (0x9D005000) from the kseg0 boot memory base address value (0x9D07E690). The line that defines `kseg0_program_mem` is then changed, as shown in [Example 14](#).

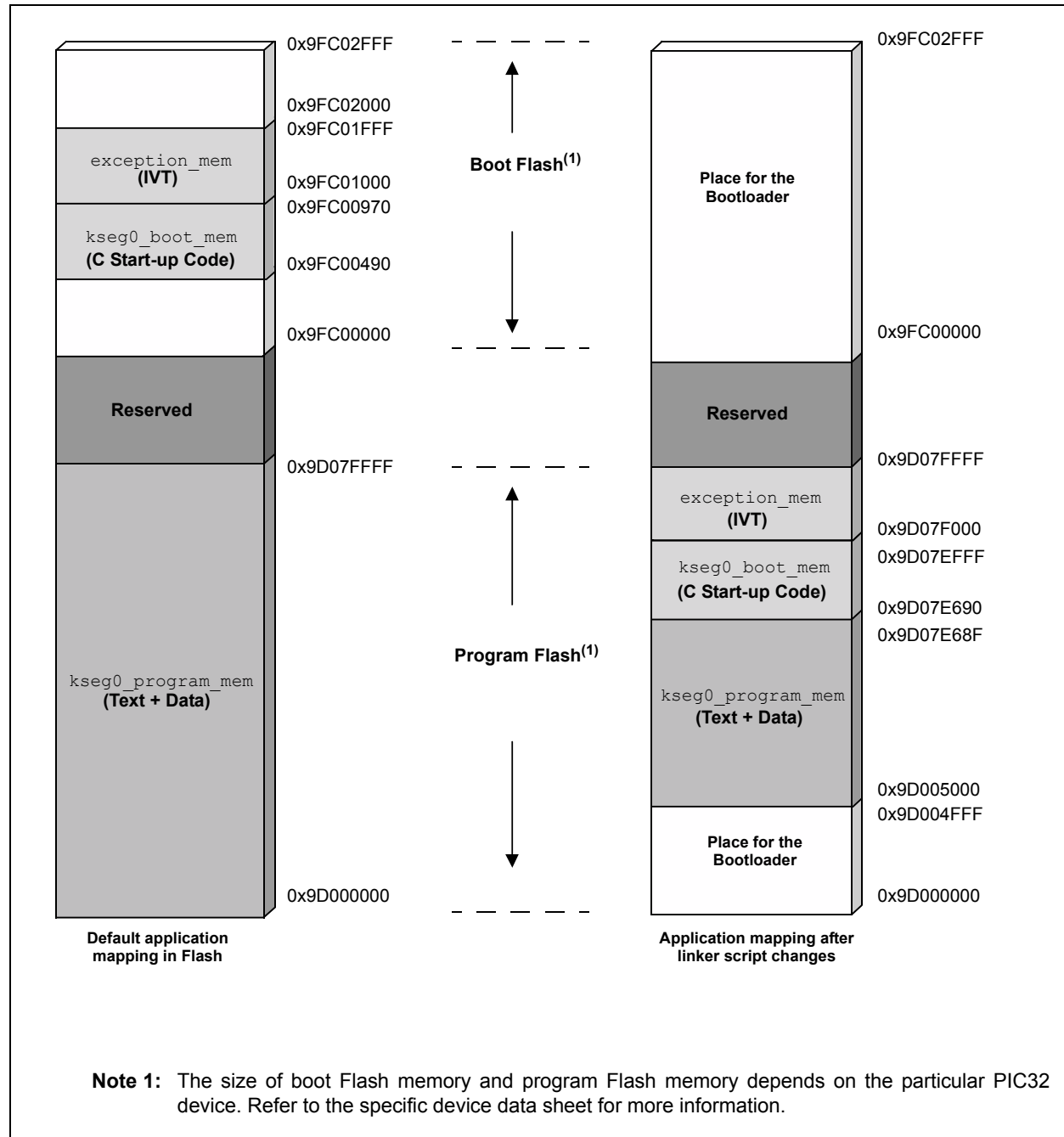
**EXAMPLE 14: SETTING THE kseg0 PROGRAM MEMORY**

```
kseg0_program_mem (rx) : ORIGIN=0x9D005000,  
LENGTH=0x79690
```

7. **Informing the Bootloader about the Reset Address of the Application:** The origin of `kseg0_boot_mem`, which holds the C start-up code, becomes the reset address of the application. Inform the bootloader about the application's reset address by setting the value of the `USER_APP_RESET_ADDRESS` macro to the origin value of the `kseg0_boot_mem`. In this example, the `USER_APP_RESET_ADDRESS` value has to be set to 0x9D07E690. The bootloader will branch the control to this address, to execute the user application.
8. Rebuild the bootloader project with all of the modified macro values.
9. Rebuild the application project with the modified linker header file included in the root directory of the application project. The MPLAB IDE automatically includes the `procdefs.ld` file from the root directory of the project during the linking process. Therefore, there is no need to add this linker script file explicitly into the workspace. The main body of the linker script is selected from the `elf32pic32mx.x` file, which is located in the same location as MPLAB C32 tools (C:\Program Files\Microchip\MPLAB C32\pic32mx\lib\ldscripts). The `elf32pic32mx.x` file maps the code sections to memory regions defined in `procdefs.ld` file. There is nothing for the user to modify inside this main linker script file.

[Figure 8](#) shows the default and modified mapping of the application image in the PIC32 Flash memory.

**FIGURE 8: APPLICATION MAPPING BEFORE AND AFTER LINKER SCRIPT CHANGES**



---

**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, PIC<sup>32</sup> 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, Omniscient 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, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-61341-321-0

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

*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://www.microchip.com/support>  
Web Address:  
[www.microchip.com](http://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: 86-571-2819-3180  
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 - 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: 886-3-6578-300  
Fax: 886-3-6578-370

**Taiwan - Kaohsiung**  
Tel: 886-7-213-7830  
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 - Druenen**  
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