Bootloader Generator

User's Guide

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

ISBN: 978-1-63277-095-0

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

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.

BOOTLOADER GENERATOR USER'S GUIDE

Table of Contents

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the Bootloader Generator. Items discussed in this chapter include:

- Document Layout
- Conventions Used in this Guide
- Recommended Reading
- Recommended Reading
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support
- Revision History

DOCUMENT LAYOUT

This document describes the installation and use of the Bootloader Generator. The document is organized as follows:

- **Chapter 1. "Overview"**
- **Chapter 2. "Options, Parts and Pieces"**
- **Chapter 3. "Hex File"**
- **Chapter 4. "Generator Tool"**
- **Chapter 5. "Bootloader Host Application"**
- **Chapter 6. "Manual Configuration Requirements"**
- **Chapter 7. "Bootloader Operation"**
- **Appendix A. "Protocol"**
- **Appendix B. "How to Calculate and Embed a Checksum Using XC8"**

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

RECOMMENDED READING

This user's guide describes how to use the Bootloader Generator. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources.

- *"AN851 – A Flash Bootloader for PIC16 and PIC18 Devices"* (DS00851). This document demonstrates a very powerful bootloader implementation for the PIC16F87XA and PIC18F families of microcontrollers with a maximum performance and functionality, while requiring a minimum of code space.
- *"High-Speed Serial Bootloader for PIC16 and PIC18 Devices"* (DS01310). Please consult this document for information regarding the serial bootloader features, implementation basics and firmware, hardware, software and application considerations.

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB® C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators.This includes the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICkit™ 3 debug express.
- **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are nonproduction development programmers such as PICSTART® Plus and PICkit 2 and 3.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at:

http://www.microchip.com/support.

REVISION HISTORY

Revision A (February, 2015)

Initial release of the document.

NOTES:

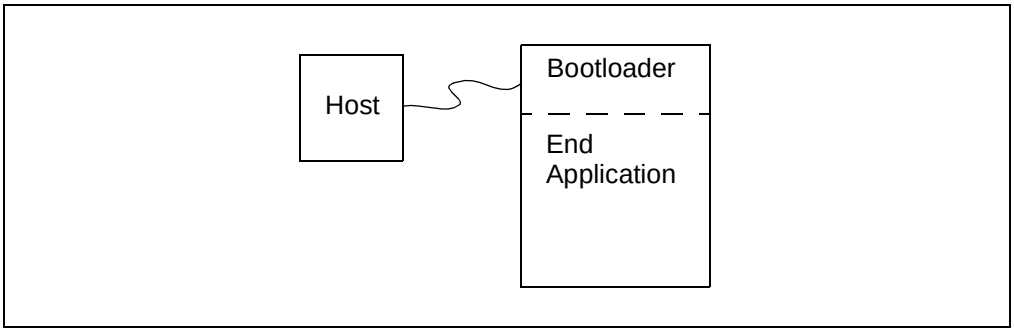
Chapter 1. Overview

1.1 BOOTLOADER OVERVIEW

The bootloader is made up of three parts (see Figure 1-1):

- The host application
- The bootloader that actually loads in the new code
- The end application

FIGURE 1-1: BOOTLOADER UNIVERSE



The host application reads the new hex file and sends it to the bootloader. The end application has to be aware of the bootloader and understand how and when to transfer control to the bootloader. The bootloader needs to run on start-up and decide whether there is a valid application loaded and transfer control there, or whether there is no valid application loaded and, in this case, stay in the bootloader. This is further explained in **Chapter 5. "Bootloader Host Application"**.

The host application may be a stand-alone application or part of the normal host application. Either way, the functionality remains the same (i.e., it has to get a new version of the end application and send it to the bootloader). This user's guide implements one version of a host application which will be able to communicate with any bootloader generated, using the generator.

The end application may be aware of the boot-loading possibility. Under some triggering condition, it should transfer control to the bootloader.

1.2 BOOTLOADER REQUIREMENTS

The main requirements for the bootloader are:

- Determine if there is a valid application loaded
- Communicate with the host
- Erase and rewrite the part
- Transfer control to end application

There are a few more features that might also be required:

- Ensure the erase and write addresses will not affect the bootloader
- Allow the host to read the Program Memory – some developers see this as a security hole and do not want the functionality
- Detect a failed boot load and recover gracefully

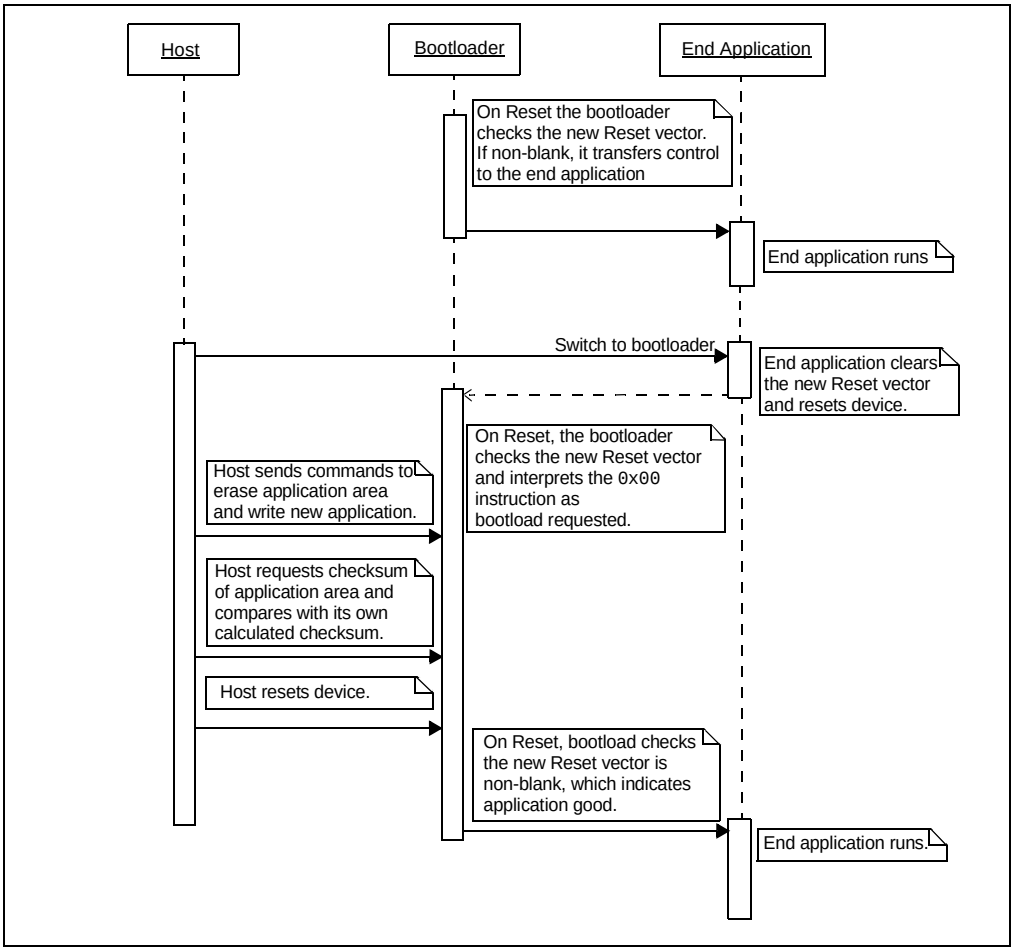
Chapter 2. Options, Parts and Pieces

There are many choices to make when creating a bootloader. Some of the questions to consider are the following:

- How will the bootloader decide if there is a valid application loaded?
- How will it communicate with the host?
- Does the bootloader need to support writing to or reading from EE Data memory?
- Is there a valid application loaded?
- How will the bootloader know when it's time to boot load or instead run the end application?

There are several ways to decide if there is a valid application loaded. "AN851 – *A Flash Bootloader for PIC16 and PIC18 Devices*" (DS00851) looks at the value of the new Reset vector. If the contents of the Reset vector is the erased value, it assumes the rest of memory is erased and it must run the bootloader application. However, it also requires the Reset vector to be the last location written once it decides that the bootload was successful (see Figure 2-1).

FIGURE 2-1: CHECK RESET VECTOR ADDRESS



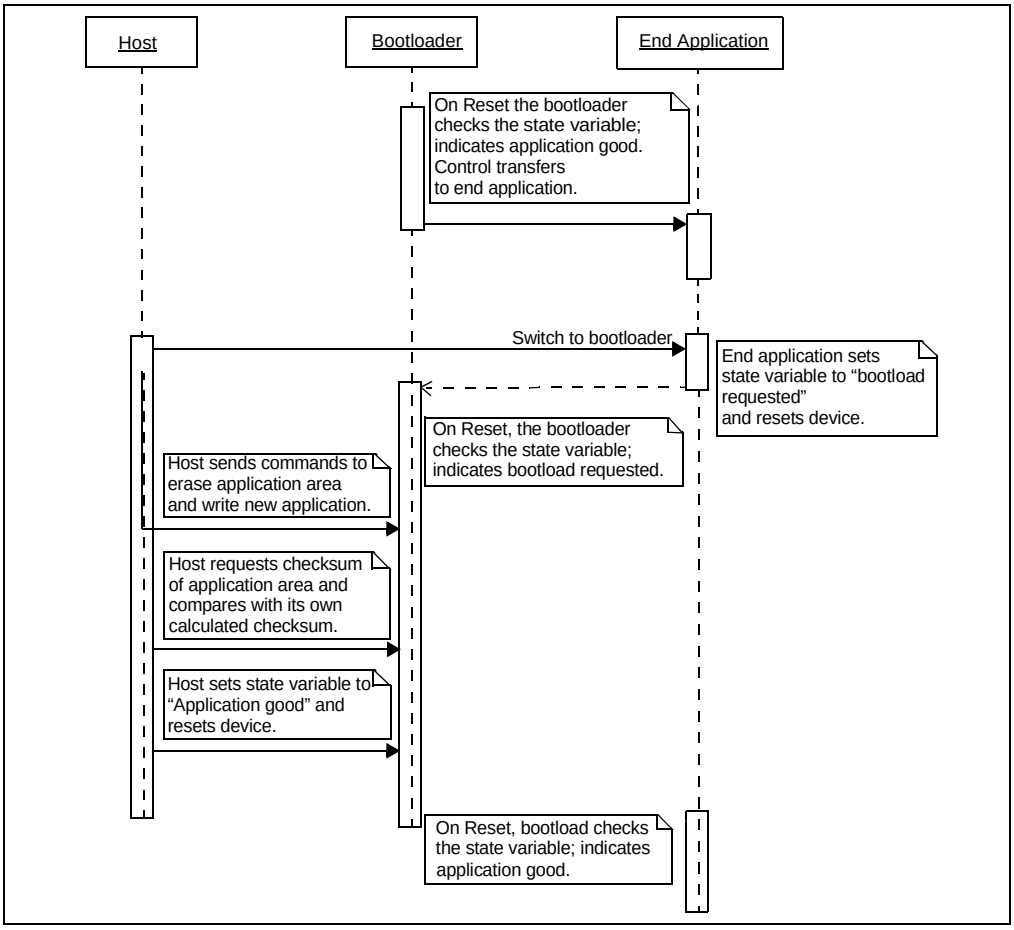
Another scheme involves using a state variable in nonvolatile memory. The bootloader on start-up will check the value for the “application good” code. The state variable needs three states:

- Erased
- Valid
- Boot load requested

The scenario is as follows (see Figure 2-2):

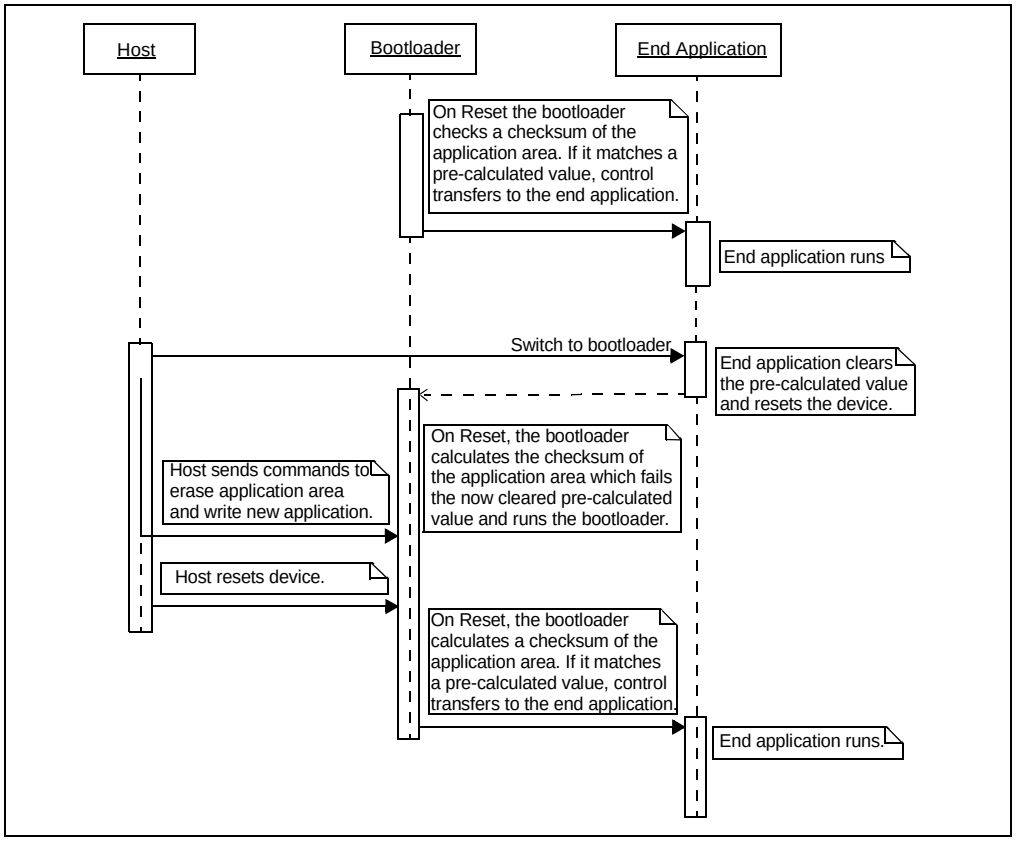
1. Host notifies end application: “Switch to bootloader”
2. End application sets the variable to “boot load requested” and resets the device
3. Device resets and on start-up the bootloader runs, checks the value and sees the “Boot load requested” state and runs the bootloader
4. Host erases and downloads a new application
5. Host calculates a checksum of the new application and asks the bootloader to do the same
6. If the two match, the host sets the state variable to “application good” and resets the device

FIGURE 2-2: NONVOLATILE STATE VARIABLE



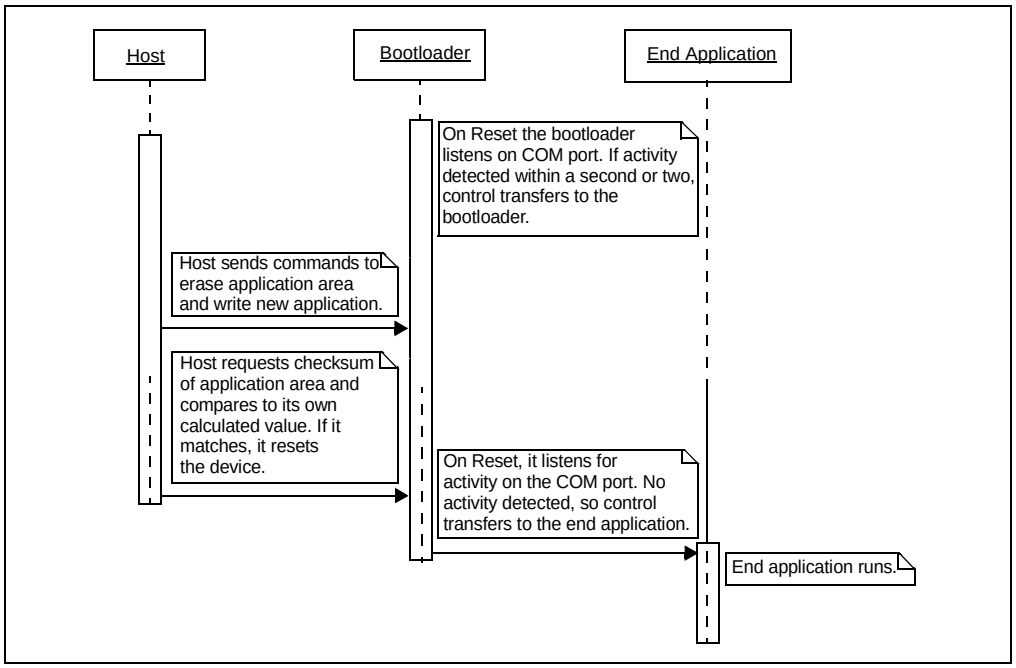
Some customers have a requirement to check application validity on each start-up. A way to achieve this would be to calculate a checksum of the application area and compare it to a pre-calculated value (see Figure 2-3). XC8 includes a tool that can do the pre-calculation and store it in the hex file. See **Appendix B. “How to Calculate and Embed a Checksum Using XC8”** for details.

FIGURE 2-3: CALCULATE CHECKSUM ON START-UP



If the device does not normally connect to the host, connecting to the host at start-up could be the trigger to run the bootloader. On start-up, the bootloader would monitor the comm port and switch to the bootloader if activity is detected. If after a second or two no activity is observed, it runs the end application (see Figure 2-4).

FIGURE 2-4: LISTEN FOR COM PORT ACTIVITY



2.1 COMMUNICATION METHOD

What protocol will the host use to communicate with the bootloader? Initially, only the UART is supported, but I^{2}C™, SPI and USB will be added in the future.

2.2 VERIFICATION OF BOOTLOAD INTEGRITY

It is prudent following the boot load to verify that the Flash accurately represents the application code.

If read Flash is supported, the host application can read back the Flash and compare it to the original file.

The bootloader itself includes a checksum command that calculates a 16-bit checksum of the application area. The host can then compare this to its own calculated checksum and confirm that the write was successful.

2.3 SELF PROTECTION

The bootloader should protect itself from accidental over-write. Therefore, attempts to write into the memory where the bootloader resides should be prevented. PIC® microcontrollers have two means to implement this: hardware and software. Write-Protect Configuration bits can selectively write-protect various regions of the Program Memory. The advantage to hardware protection is a smaller code footprint. The downside is that the block size is fixed and may leave memory wasted. The address protection can also be accomplished in software. The code can check the destination address of each Write and Erase command. If it conflicts with the bootloader region, the command is rejected. The software check takes a little more code space, but it has the advantage that a special bootloader application could be written to replace the original bootloader in case a bug emerges in the bootloader. This is not possible if the Write-Protect Configuration bit is set.

2.4 ENCRYPTION

It is possible to encrypt the data as it is sent to the device. In order for this encryption to be effective, a robust key management system needs to be in place. No encryption methods are supported at this time. Customers who wish to implement an encryption scheme in their bootloader should contact Microchip for support in this regard.

Chapter 3. Hex File

3.1 INTEL® HEX FILE FORMAT

EXAMPLE 3-1: HEX FILE RECORD FORMAT

```
:BBAAAATTHHHH……HHCC
:10F90000E1022000E1023000C1121000C1524004D

:      Record Start Character
BB     two digit byte count specifying the number of data bytes in
       this record.
AAAA   Four digit starting address of this data record
TT     Two digit record type
       00 = data record
       01 = End of File record
       02 = Segment Address Record
       04 = Extended Linear Address record
HH     Data Bytes
CC     Two digit checksum calculated as 2's complement of all
       preceding bytes in data record except the colon.
```

3.2 PIC16F1XXX INTERPRETATIONS

The Intel® hex file is byte-oriented, while the PIC16 is word-oriented. The address in the hex file line is a byte address and must be divided by two to get the word address. As shown in Example 3-1 above, the address is 0x0F90. The word address is half that, 0x07C8.

Also, the word data is stored low byte first (little endian). Thus, the first two bytes make up the word at the first address. In Example 3-1 the bytes are 0x0E and 0x10. Thus, the word stored at 0x07C8 is 0x100E.

Each PIC device has several different memory regions:

- Program Memory is stored at its natural word address.
- ID locations are at 0x8000 through 0x8003, Configuration Words at 0x8007 and 0x8008.
- EEData is encoded at 0xF000.

EEData only uses the low-order byte of each two-byte pair.

EXAMPLE 3-2: EE DATA ENCODED IN PIC16F1XXX HEX FILE

```
:02000004 0001F9
:10E000 00 0800 09000A000B000C000D000E000F00B4
```

The first record sets the upper 16 bits of address to 0x0001.

The second record sets the lower 16 bits of the address. The resulting 32-bit address is 0x0001E000. When divided by 2, the PIC16F1XXX address of 0xF000 is obtained.

Only the low order byte of each pair is used for EEData. In this case, EEData address 0 would get 0x08, address 1 would get 0x09, address 2 would get 0x0A, etc.

3.3 PIC18 INTERPRETATIONS

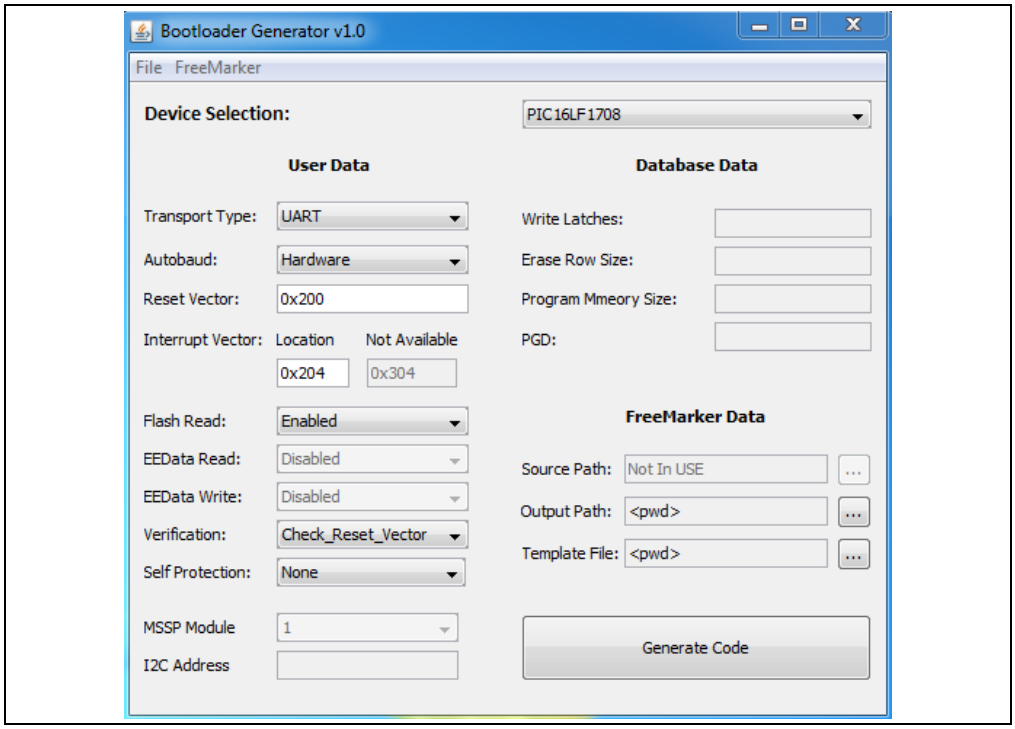
PIC18F devices are byte-oriented, so the address on the line does not need any correction. EEData is encoded in the hex file at 0xF00000 and one byte per address (no skipped bytes as with the PIC16F1XXX devices).

Chapter 4. Generator Tool

The main intent of the bootloader generator is to allow users to create a bootloader assembly source file depending on the selected device features.

The application is responsible for creating the `out_<Template File Name>` file by taking in user inputs and information from the device database.

FIGURE 4-1: BOOTLOADER GENERATOR USER INTERFACE



4.1 ELEMENTS IN USE

The application uses simple XML framework and FreeMarker to parse the input data and create an assembly file, respectively. Both of these are provided in the form of libraries (.jar) with the application.

4.2 OVERALL WORKING STRATEGY

The application requires the user to select a device and enter several parameters displayed on the GUI screen under **User Data** and **FreeMarker Data**.

After retrieving the necessary inputs, a data structure representing the parameters is created and fed to FreeMarker along with a template file. This allows FreeMarker to process and create the output assembly source file.

4.3 PREREQUISITES

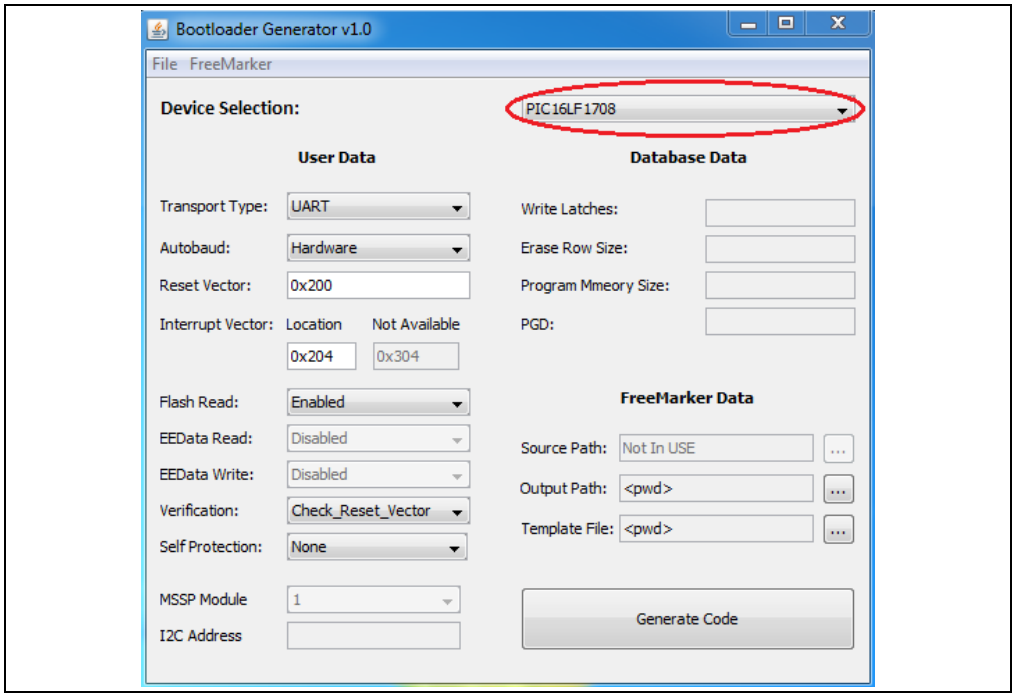
1. The application has to be launched from the `./dist` directory.
2. The MPLAB® X IDE or MPLAB X communication libraries must be installed on the machine. If starting the application is problematic, re-installing MPLAB X could solve this issue.
3. The internal application files should not be moved to different directories. Moving the files will affect the file paths and will cause a "*file not found*" exception.

4.4 DETAILED STEPS

Once the application starts up, the order of steps listed below has to be followed:

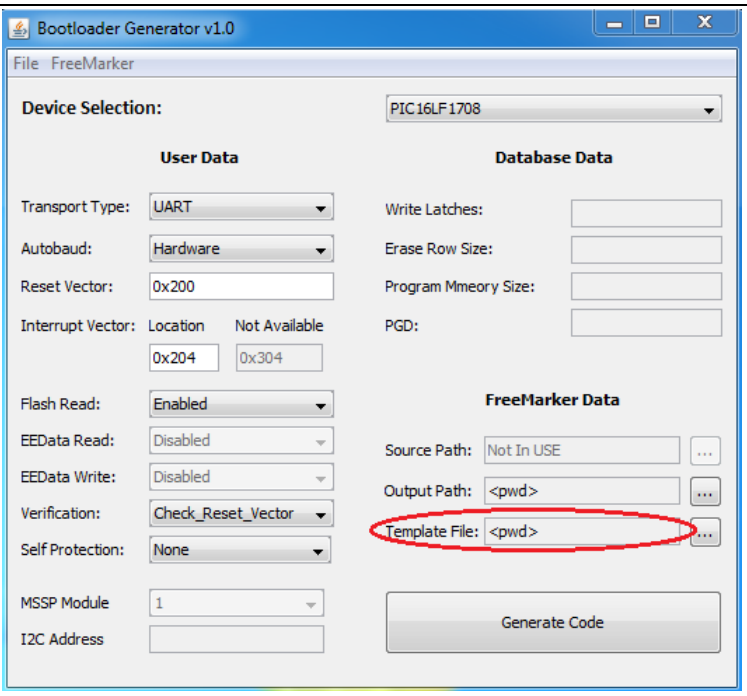
1. Select the device from the list (see Figure 4-2).

FIGURE 4-2: DEVICE SELECTION



2. Under **FreeMarker Data**, set the Template Path Directory (the folder that contains the file) (see Figure 4-3). If the Template Path Directory is not selected, the present working directory is chosen by default.

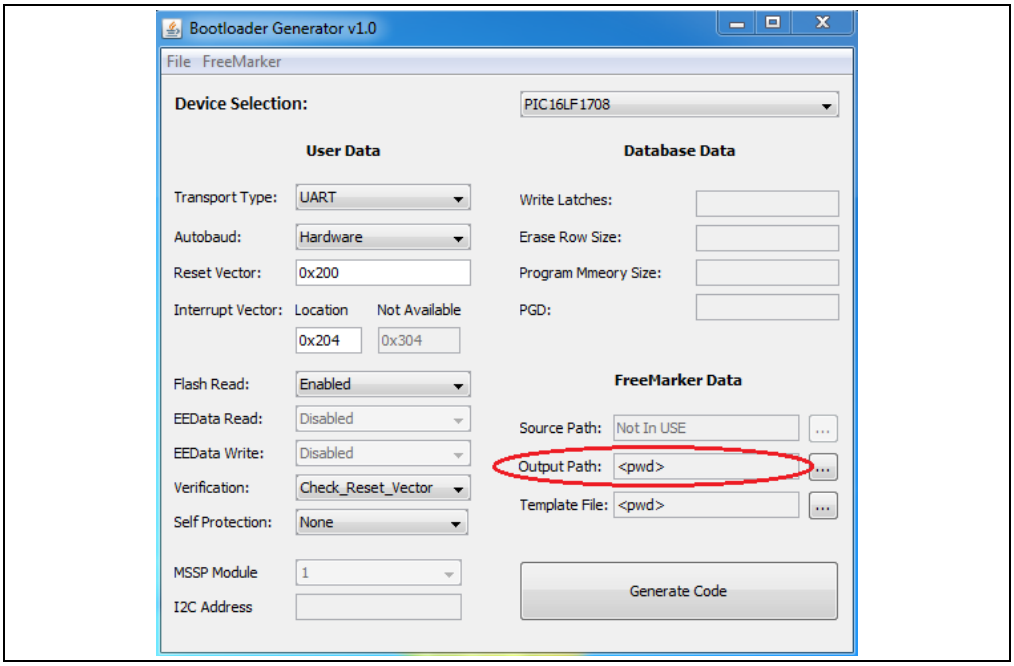
FIGURE 4-3: TEMPLATE FILE DIRECTORY SELECTION



The template file is a file required by FreeMarker and is the skeleton of the assembly file to be generated. This file consists of areas that instruct FreeMarker to insert the parameter values depending on the device selected.

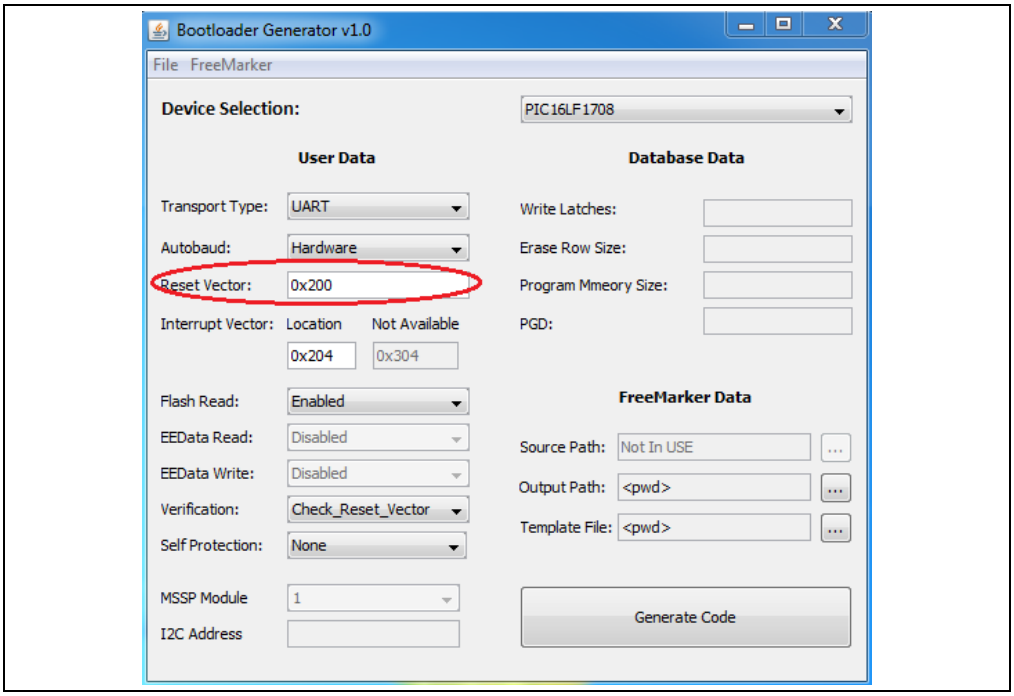
3. Under **FreeMarker Data**, set the Output Path Directory (where the user wants the output file to be generated) (see Figure 4-4). If the Output Path Directory is not selected, the present working directory is chosen by default.

FIGURE 4-4: OUTPUT FILE DIRECTORY SELECTION



4. Configure the Reset Vector (see Figure 4-5). The Reset vector is a location where the Embedded Application code starts on the PIC device (target device). Once the target device is reset, the application starts to execute from this Reset vector location (the program counter is set to the Reset vector on Reset).

FIGURE 4-5: RESET VECTOR CONFIGURATION



5. Configure the Interrupt Vector.

PIC16 devices have only one interrupt vector (see Figure 4-6).

Usually, this interrupt vector is at location 0x004 in the Program Memory map. However, since we have the bootloader at the start of the Program Memory, we need to remap the interrupt vector to a new location, for example 0x204.

PIC18 devices have two interrupt vectors, namely higher priority and lower priority. Both of these need to be configured (see Figure 4-7).

As soon as the user selects a device, depending on the device family the interrupt vector options switch on the GUI.

FIGURE 4-6: INTERRUPT VECTOR PIC16

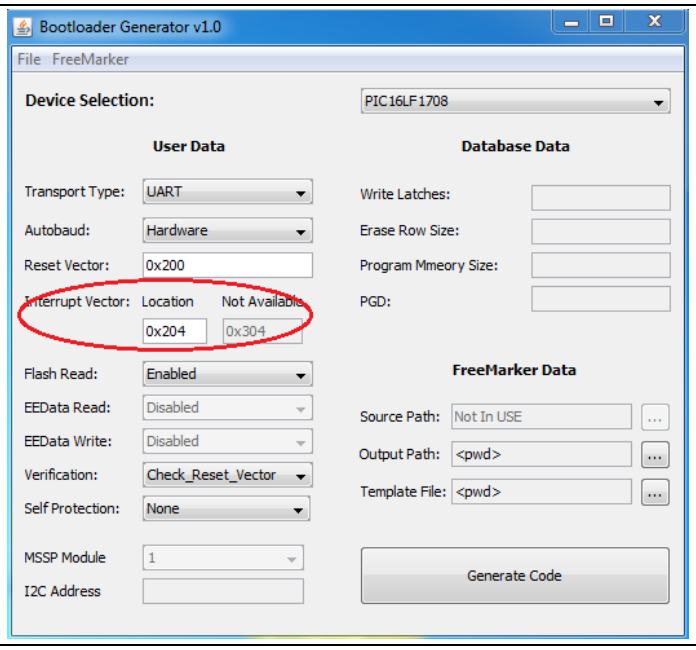
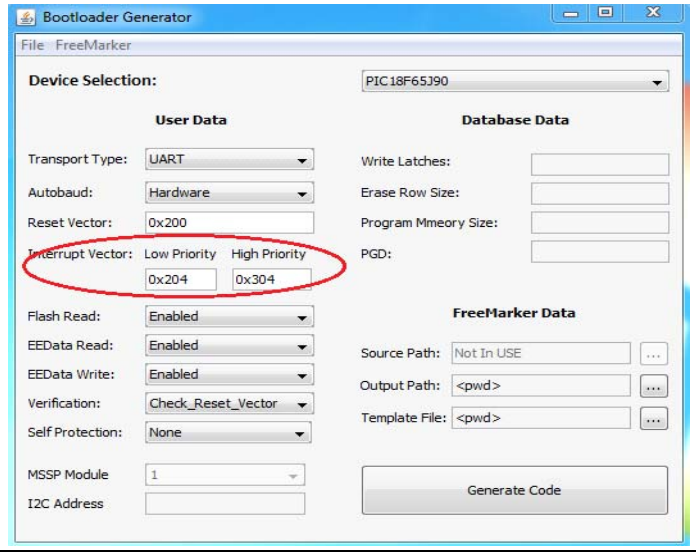


FIGURE 4-7: INTERRUPT VECTOR PIC18

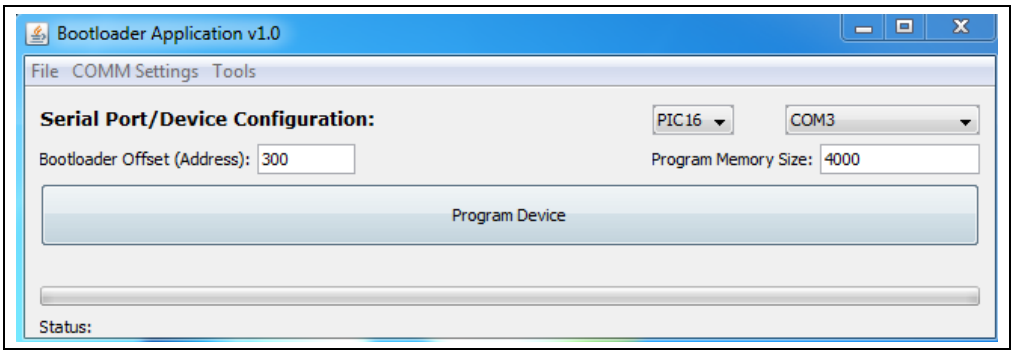


6. All other options under user data can be configured and changed. Set them as needed.
7. Click **Generate Code**.

Chapter 5. Bootloader Host Application

The bootloader host application is responsible for transferring an embedded application program (the .hex file) from the host machine (development host) to a target device (a device running the embedded application. It talks to the bootloader present on the target device by sending appropriate commands, and it transfers data (the embedded application program) using UART communication. It can be used to program the Flash memory for PIC16 and PIC18 devices. EEPROM writes are also supported.

FIGURE 5-1: BOOTLOADER HOST APPLICATION



5.1 EXTERNAL DEPENDENCIES

The application uses a hex file parser and MPLAB X communication to parse and store the hex file to be programmed and to communicate to the target device, respectively. Both of these are provided in the form of libraries (.jar) with the application.

5.2 OVERALL WORKING STRATEGY

The application requires the user to select the device family and the COM port used for communication. Once this is selected, the user has to configure the Program Memory and offset values on the GUI screen. As soon as the user clicks the **Program Device** button, the application prompts the user to select a .hex file, after which it tries to write (transfer) the file to the target device with the fastest possible speed (this depends massively on the baud rate). Once the file is written successfully, the application calculates a checksum over the entire .hex file and verifies data integrity. A checksum match resets the target device and the embedded application program starts to execute.

In case of a communication failure, the application retries three times after which it relinquishes control over the COM Port and stops the communication giving an error.

5.3 PREREQUISITES

Make sure that the MPLAB X IDE or MPLAB communication libraries are installed on the machine. If any problems arise while starting the application, try re-installing MPLAB X.

The values to be entered in the text fields for the "Program Memory size" and "Bootloader offset" must be in hex format. By default, the application is set to have an offset of 300 (hex) with Program Memory size as 4000 (hex).

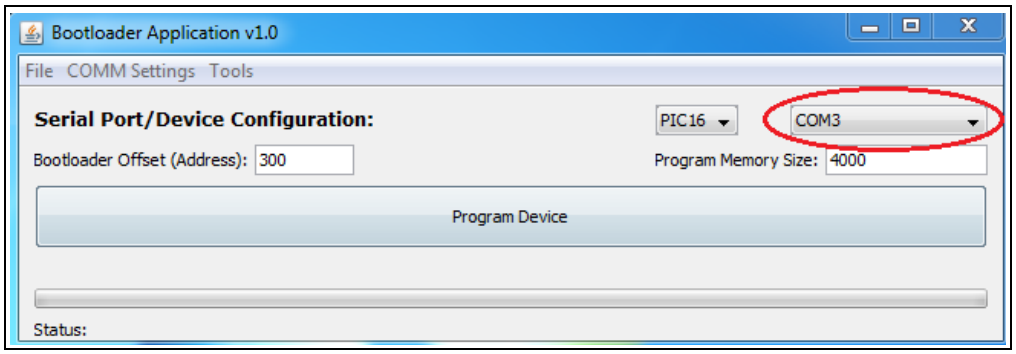
Version 1.0 of the application performs "Program Memory and EEPROM" writes only with UART interface.

5.4 DETAILED STEPS

As soon as the application is launched, there are several inputs that need to be provided, as listed below.

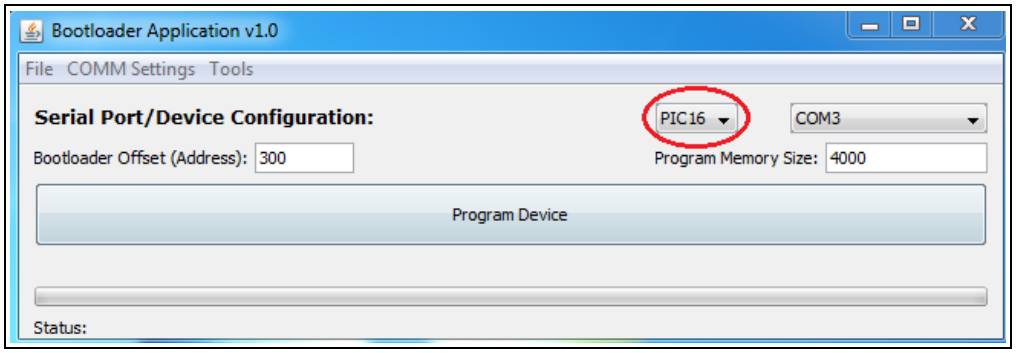
1. Select the COM port on which the target device is connected from the drop-down box (see Figure 5-2).

FIGURE 5-2: COM PORT SELECTION



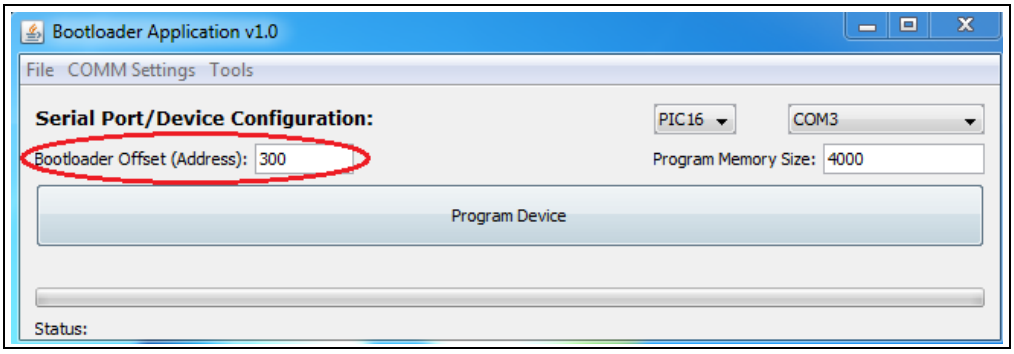
2. Pick the device family (PIC16 or PIC18) from the other drop-down menu (see Figure 5-3).

FIGURE 5-3: DEVICE FAMILY SELECTION



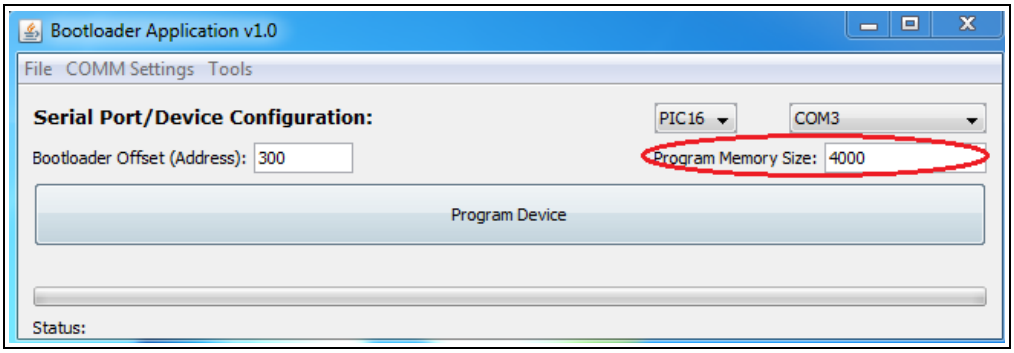
3. Set the bootloader offset address for the device (see Figure 5-4). This is typically known when the bootloader is designed for a particular target device.

FIGURE 5-4: BOOTLOADER OFFSET CONFIGURATION



4. Set the Program Memory size depending on the target device (see Figure 5-5). Be aware that the Program Memory size is the number of locations. For example, if the target device has Program Memory starting from 0-3FFF, then 4000 should be entered in the text field. The size of every location depends on the target device. Some devices have word-addressable Flash and others have it byte-addressable.

FIGURE 5-5: PROGRAM MEMORY SIZE CONFIGURATION



5. Click the **Program Device** button (see Figure 5-6 through Figure 5-8).

FIGURE 5-6: CONNECTING TO THE TARGET DEVICE

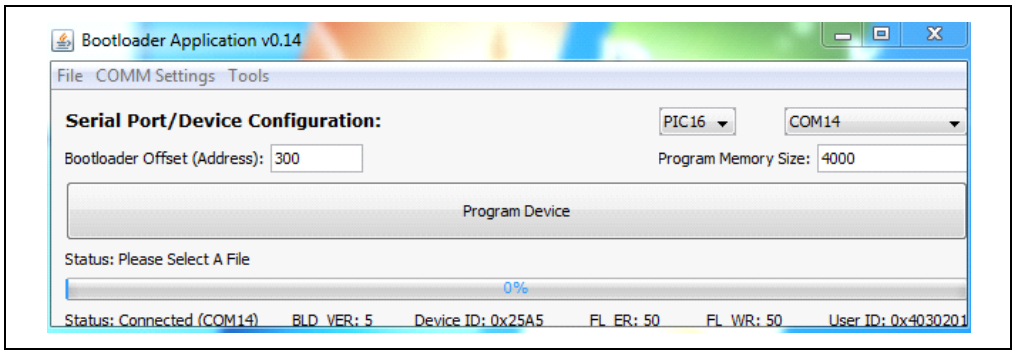


FIGURE 5-7: DISPLAYING A FILE CHOOSER DIALOG

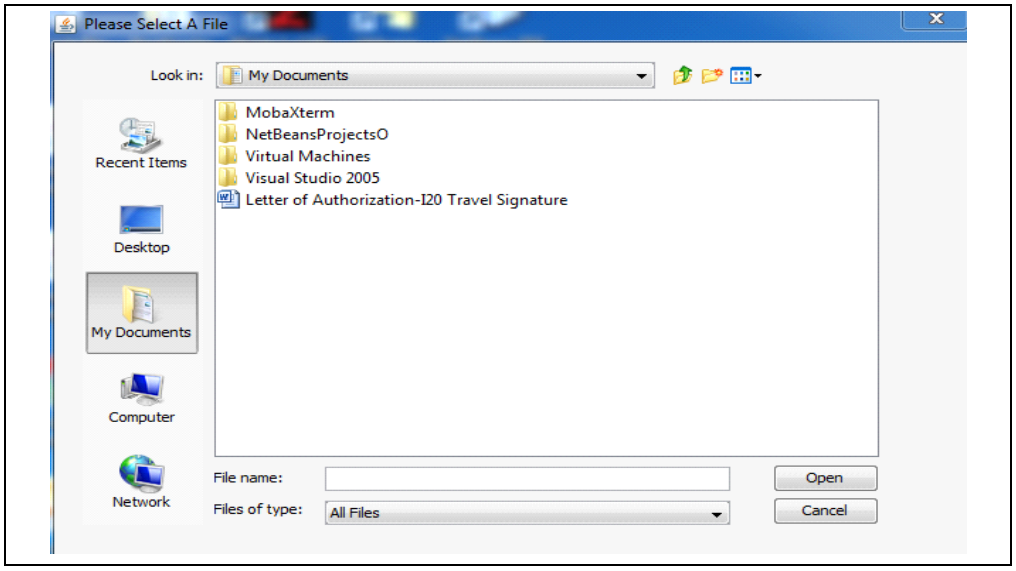
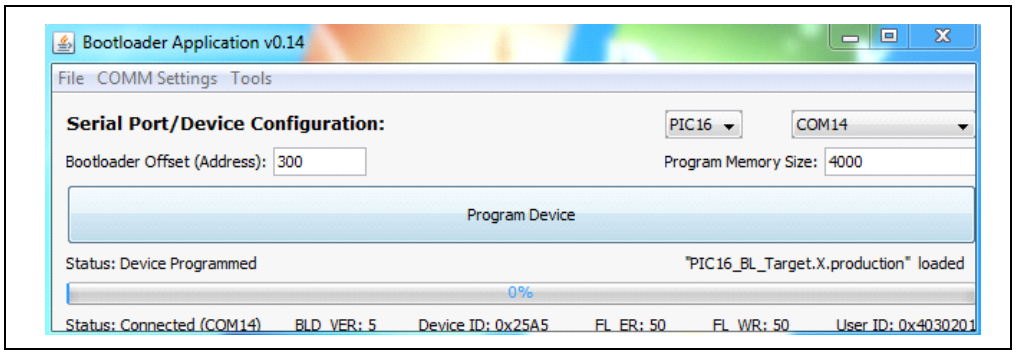


FIGURE 5-8: DEVICE PROGRAMMED



5.5 MENU BAR OPTIONS

There are additional options that can be used if needed. The **Program Device** button should take care of connecting to the target device and popping up a file selection dialog. However, if the user wants to connect or load the file externally, options have been provided in the menu. If the user needs to disconnect the application, this can be done under the menu.

Please ensure that the Console window is pulled up when programming a device. This window will provide step-by-step visibility into the bootloading process.

5.6 TROUBLESHOOTING

If the application fails to connect, close the application, unplug the target device, plug it back in and start the application again. If the device is connected while the application is running, click the **Refresh** option under the menu to pull up the COM port. Pull up the console window to check the nature of the error. There is a status label above the progress bar which displays messages appropriately.

Chapter 6. Manual Configuration Requirements

Inside `hardware_defs.inc` there is a `CONFIGURE_IO` macro. This may need to be customized to the part. There are too many possibilities of analog or alternate pin functions that could affect the pin that the generator cannot account for. Configuring the pin correctly is left to the developer.

Configuration bits are a particularly thorny issue. Good software engineering practice requires setting them in only one place. However, it is difficult to run the end application in a debugger with the boot loader present. The bootloader defaults to only programming the FOSC bits to select the internal oscillator. The other Configuration bits may be configured as needed for the end application.

If the Watchdog Timer is enabled, a `CLRWDT` instruction may need to be added in the bootloader main loop.

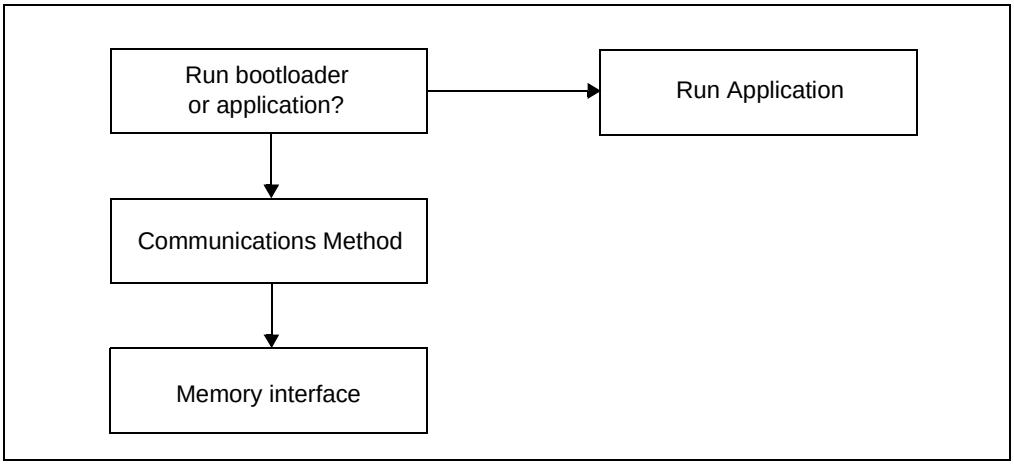
Another strategy would be to test and debug the generated bootloader with default Configuration Words. Once the bootloader is working, these Configuration Words should be commented out.

Also check the #define OSCCON_VALUE in the hardware-defs.inc file. This value is copied into OSCCON. The use of the SFR changes depending on the part in use. For robust communications, the clock speed should be set as fast as possible.

Chapter 7. Bootloader Operation

On start-up, the bootloader checks to see if a valid application is loaded. If not, it runs the bootloader (see Figure 7-1).

FIGURE 7-1: BOOTLOADER STRUCTURE



The heart of the bootloader is the communication method and the memory interface. The communication method talks to the host, it receives commands from the host and it returns the results. The memory interface processes the commands and prepares the results for the communication method to send back to the host.

The interface between the communication method and memory interface is an array of bytes. The array consists of a 9-byte header with optional additional data. The length of the array is inferred by the length field in the header plus the 9-byte header.

Each command starts with a 9-byte header:

<Command 1 Byte><Length 2 Bytes><0x55><0xAA><address 4 Bytes><data>

Multiple byte fields are sent low byte first.

Length is the number of bytes in the data portion of the transfer (Write commands), or the number of bytes to return (Read commands).

Address is the actual address where the data is to be written to (or read from). For the PIC16F1XXX devices, this is a word address; for the PIC18F devices, it is a byte address.

The UART communication method adds an extra 0x55 to the front of the 9-byte packet. The UART uses this character to derive the baud rate, preserving communication reliability at high speeds while using the internal oscillator.

Appendix A. Protocol

A.1 BASIC COMMAND FORMAT

Each command starts with a 9-byte header as follows:

<Command> <Length (2 bytes)><0x55><0xAA><address (4 bytes)>

Command is one of the commands listed in Table A-1.

Length is the amount of data in bytes. For Write commands, it represents the additional data bytes after the 9-byte header. For Read commands, it represents the amount returned by the read. For Checksum commands, it is the number of bytes to include.

All nine bytes are sent whether they are used or not.

TABLE A-1: SUPPORTED COMMANDS

0x00	Get Version and other info
0x01	Read Flash(1)
0x02	Write Flash
0x03	Erase Flash
0x04	Read EE Data(1)
0x05	Write EE Data(1)
0x06	Read Configuration words
0x07	Write Configuration words
0x08	Calculate and return Flash checksum
0x09	Reset Device

Note 1: Command optional. May be omitted at code generation time. If the commands are omitted, bootloader responds with 0xFF (invalid command).

TABLE A-2: RETURN CODES

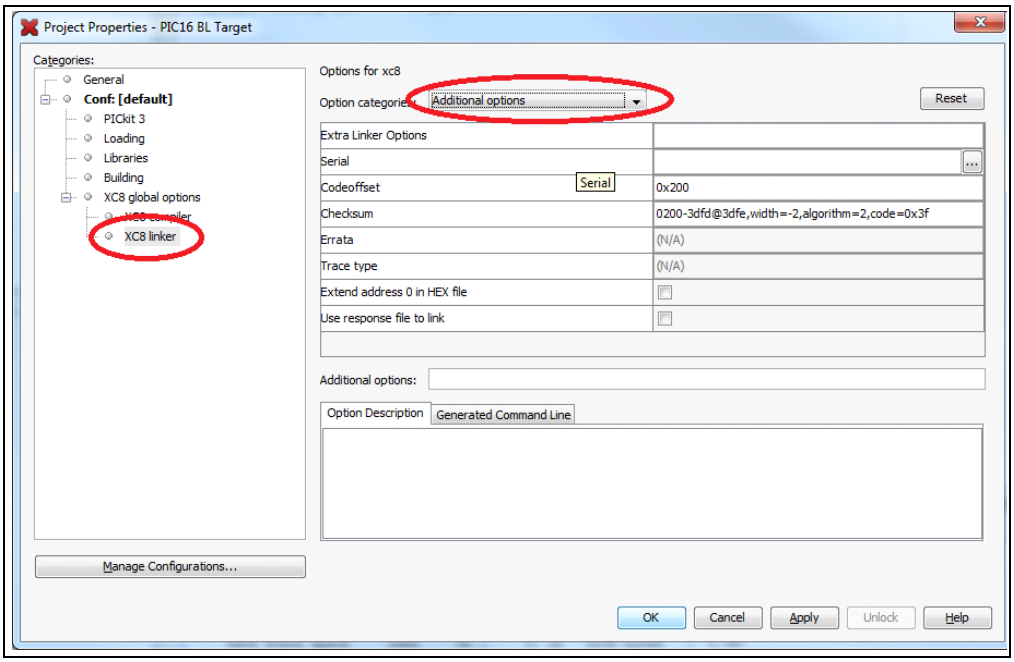
Code	Meaning
0x01	Success (Write commands)
0xFE	Address out of bounds (Write, Erase)
0xFF	Command Not Supported

Appendix B. How to Calculate and Embed a Checksum Using XC8

XC8 includes a program called HexMate, which can be used to manipulate hex files. MPLAB X provides a means to invoke HexMate during the build process. The procedure is as described below:

1. Bring up the Project Properties window. Select "XC8 linker" and "Additional options" fields (see Figure B-1)

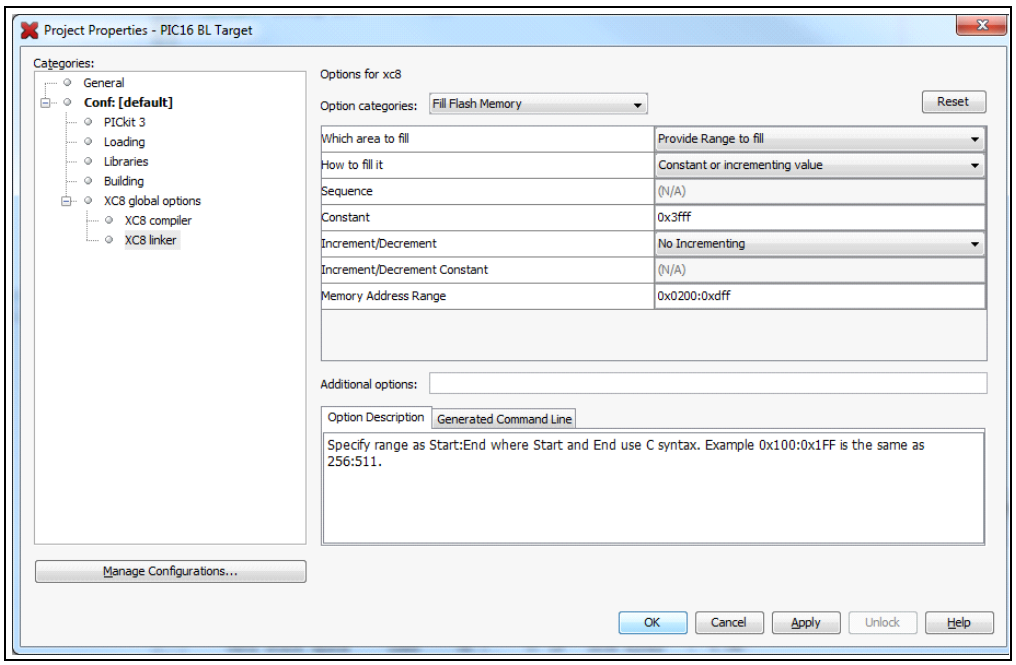
FIGURE B-1: CALCULATE AND EMBED CHECKSUM



2. Set the code offset to the new Reset vector. Then, on the checksum line, add the following "*0200-3dfd@3fde,width=-2,algorithm=2,code=0x3f*" (no spaces). This example calculates a 16-bit checksum from 0x200 to 0x3DFD and places it at 0x3DFE and 0x3DFF. The address range should be adjusted for your application and the device memory size. The *-2* on the width parameter specifies two bytes wide, low byte first. *Algorithm 2* specifies the checksum is calculated by 16-bit addition. *Code* specifies how the high order six bits are padded on each word. If left off, it will truncate the checksum to 14 bits. Adding this spreads the 16-byte checksum across two words.

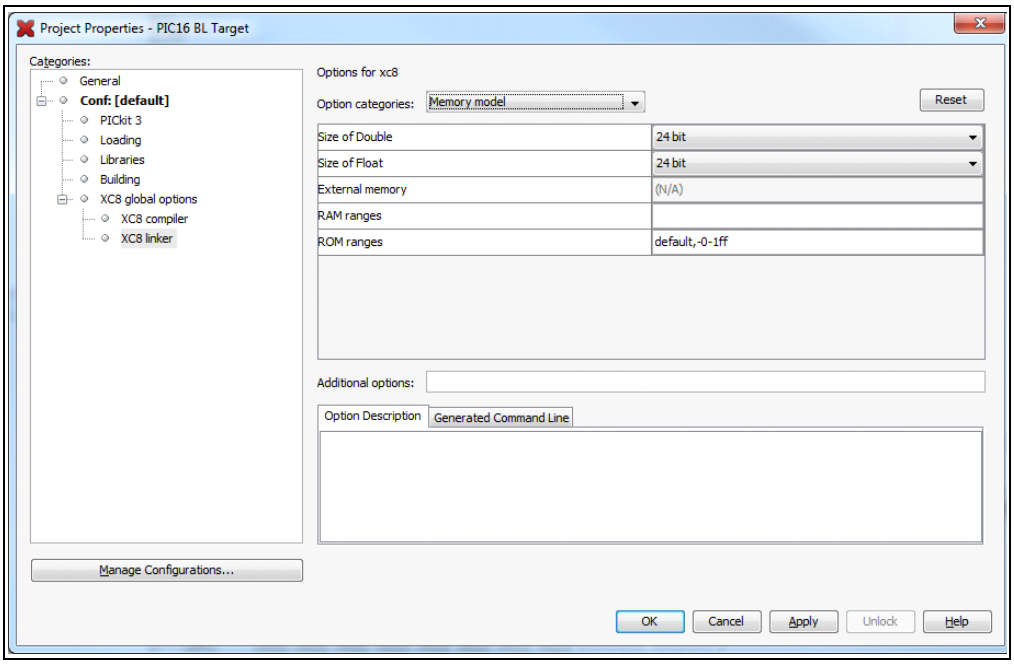
3. Fill unused memory. The compiler only generates code where it has instructions. To correctly calculate the checksum, the hex file needs to look like it will be in memory. This is on the **Fill Flash Memory** tab. Select "Provide range to fill". Use a constant or incrementing value for this. In the example provided in Figure B-2, the constant 0X3FFF has been used, which is the Erased condition of memory. Finally, the address range to fill is specified. 0x0001 could be a better fill memory choice for the PIC16F1XXX device family (i.e., the Reset instruction op code). If the code goes rogue and lands in unprogrammed memory, it will reset the device and recover gracefully. 0x00FF is the PIC18F Reset instruction.

FIGURE B-2: FILL UNUSED MEMORY



4. Specify where in ROM code is allowed. The bootloader will exist at the beginning of memory, so that space needs to be reserved. This is on the **Memory Model** tab. `default,-0-1ff` represents the default memory range, except for the first 200 words. For large programs and working with a debugger such as PICkit™ 3 or ICD3, the user may need to reserve space for the debug execution at the top of memory as well.

FIGURE B-3: RESERVE BOOTLOADER MEMORY



5. Click **Apply** and **OK**, then build normally.

Complete HEXMATE documentation is in the "*MPLAB® XC8 C Compiler User's Guide*" (DS50002053).

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

Austin, TX
Tel: 512-257-3370

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
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983
Indianapolis

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

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

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
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: 852-2943-5100
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 - Dongguan
Tel: 86-769-8702-9880

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
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: 86-755-8864-2200
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

ASIA/PACIFIC

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

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

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: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

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: 886-7-213-7828

Taiwan - Taipei
Tel: 886-2-2508-8600
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 - Dusseldorf
Tel: 49-2129-3766400

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

Germany - Pforzheim
Tel: 49-7231-424750

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

Italy - Venice
Tel: 39-049-7625286

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

Poland - Warsaw
Tel: 48-22-3325737

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

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

01/27/15