MICROCHIP

AN2799

Using the Petit FAT File System Module with AVR® Microcontrollers

Introduction

Author: Elizabeth Roy, Microchip Technology Inc.

This application note describes the use of the Petit FAT file system library with a limited-memory AVR® device. Full SDC/MMC file system interfacing is normally not possible on devices with such small memories, but this compact library enables limited interfacing with a file system on such devices. The unneeded functions in the library can be disabled to minimize memory usage even further. The Petit FatFS module is a subset of the FatFS module, hence its limited functionality. It is platform independent for any ANSI C compliant compiler, separated completely from the disk I/O layer; this means device-specific functions for disk interfacing needs to be provided.

Features

- SDC/MMC File System Access with Limited Memory Consumption
- Device Independent, Ideal for AVR Applications Requiring Optimized Memory Usage
- FAT12, FAT16, and FAT32 Supported
- Single Volume and Single File Only
- Streaming File Read
- File Write with Some Restrictions

Table of Contents

1. Relevant Devices

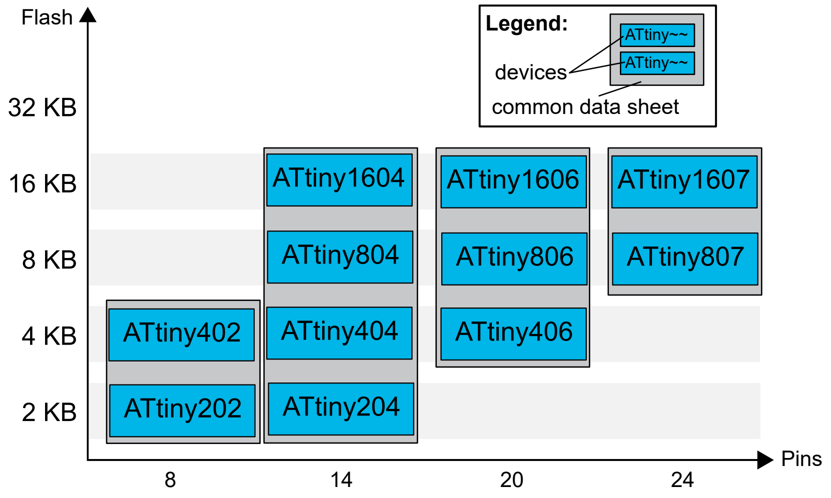
This chapter lists the relevant devices for this document.

1.1 tinyAVR® 0-series

The figure below shows the tinyAVR® 0-series, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin- and feature compatible.
- Horizontal migration to the left reduces the pin count and therefore, the available features.

Figure 1-1. tinyAVR® 0-series Overview



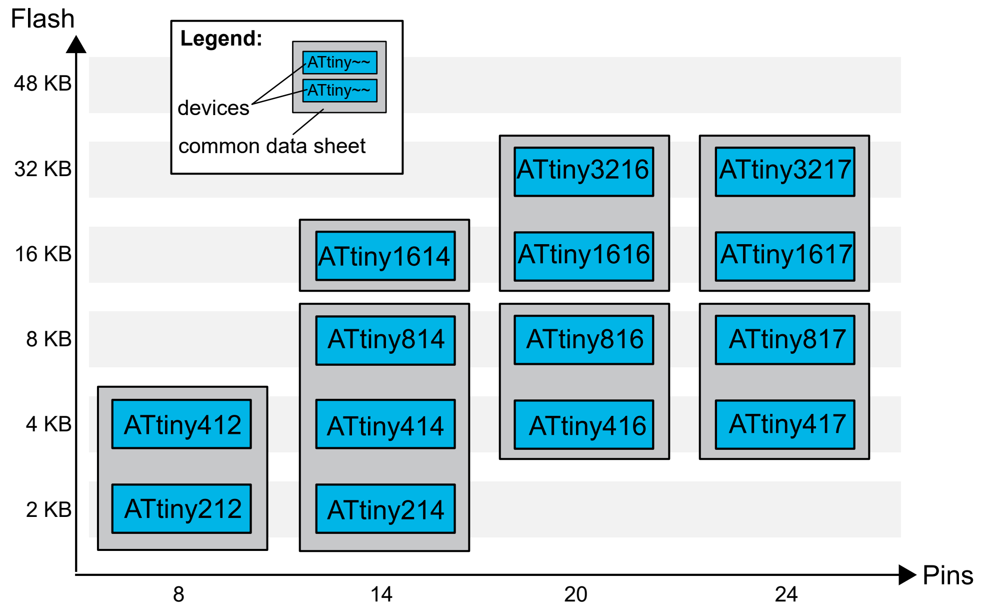
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR 1-series

The following figure shows the tinyAVR 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-2. tinyAVR® 1-series Overview



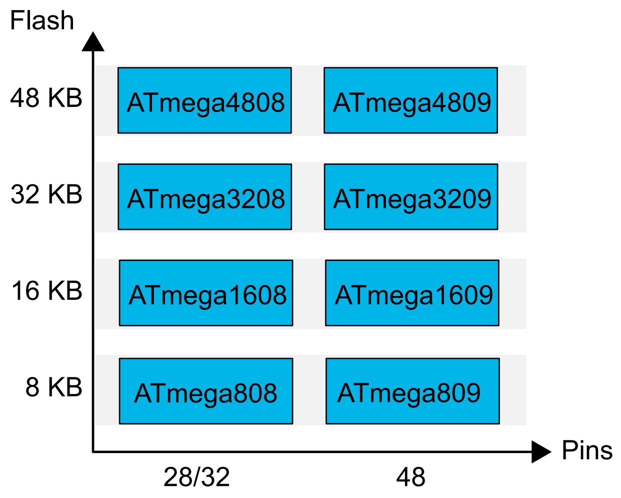
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-3. megaAVR® 0-series Overview



Devices with different Flash memory size typically also have different SRAM and EEPROM.

2. Petit FAT File System

An SD card is a very convenient way to store large amounts of nonvolatile data. A file system is generally used to allow the data to be portable and easily accessible from a PC. An industry standard, widely used file system is FAT. Many AVR applications require optimized usage of Flash memory and SRAM; when only limited interfacing is needed or insufficient resources are available, a normal FAT file system module may be unsuitable, hence the need for the subset Petit FAT file system module (Petit FatFs), available from http://elm-chan.org/fsw/ff/00index_p.html. It is specifically designed for 8-bit microcontrollers, such as AVR 8-bit microcontrollers, and provides a basic interface to a FAT file system formatted MMC or SD card while requiring limited resources. It is platform independent and so it requires a lower disk I/O layer to be functional, but sample drivers are available.

The functionality provided by the Petit FatFs library includes:

- Mounting a volume with a FAT file system
- Opening a file
- Read a file
- Write to a file (with some restrictions)
- Move R/W pointer
- Open a directory
- Read a directory item

Configuration regarding inclusion or exclusion of any function is defined in the configuration header file (`pffconf.h`), in order to minimize compile size.

3. Petit FatFs Limitations

Petit FatFs specifically use as little Flash and stack as possible. This, however, comes at the expense of some functionality. Files cannot be created or increased in size, and only one file can be accessed at a time. The mode of sector access is limited in terms of efficiency, and therefore there are significant periods where the SDC/MMC is busy. This busy period occurs for every partial sector access operation, so in applications where an entire sector cannot be read or written at once, the maximum access frequency further decreases. For more information, refer to AVR42776.

3.1 Write Function

The Petit FatFS module contains a write function with some restrictions, in order to minimize the required memory. These restrictions include:

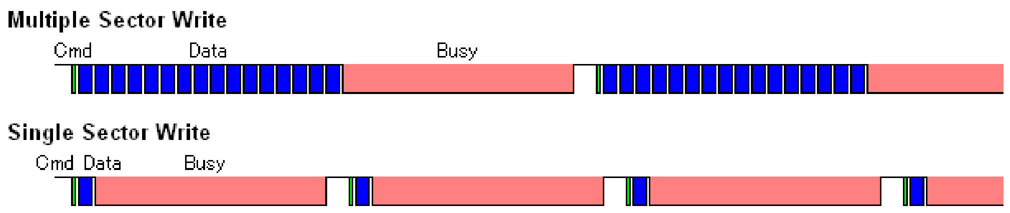
- Files cannot be created; only an existing file can be written to
- Files cannot be expanded in size
- The time stamp of a file cannot be updated
- A write operation will only start and stop on a sector boundary
- A read-only file attribute will not block a write operation

This means that in order to write to a file, it must already exist on the SDC/MMC, and have the required size allocated.

3.2 File Access Efficiency

The efficiency of a read or write operation is increased with the number of sectors able to be accessed in one operation. Due to the necessity to decrease the required memory, Petit FatFS uses single sector access, meaning the access performance is decreased when compared with multiple sector access. An example can be seen in the figure below, which depicts a card's busy time associated with both a multiple sector write and a single sector write. This means the maximum read and write frequencies are decreased substantially.

Figure 3-1. Multiple vs Single Sector Write



It should also be noted that SDC/MMC storage devices are not byte addressable, so for every partial sector access operation (one byte up to the sector size of 512 bytes), the entire sector needs to be accessed. For example, when reading a sector byte-by-byte, the sector will be read 512 times. The minimum busy time achievable with Petit FatFS is by reading or writing 512 bytes at a time (the size of a sector), so that the sector is only accessed once. Therefore, data may be written or read in as long of a block as possible.

4. Functions

Mount a Volume

```
FRESULT pf_mount (
  FATFS*  fs/* [IN] Pointer to the work area */
);
```

The `pf_mount()` function should be used prior to attempting any interfacing. It mounts the storage volume and gives a work area to the Petit FatFS module.

- Parameters:
  – `fs`: Pointer to the file system object (work area) to be registered.
- Return Values:
  – `FR_OK (0)`: Function success.
  – `FR_NOT_READY`: The volume could not be mounted, due to a hard error or no volume present.
  – `FR_DISK_ERR`: An error occurred in the disk read function.
  – `FR_NO_FILESYSTEM`: No valid FAT partition on the drive.

Open a File

```
FRESULT pf_open (
  const char* path /* [IN] Pointer to the file name */
);
```

The `pf_open()` function should be used prior to writing to or reading from a file. The open file is valid until the open function is used again for another file. Files cannot be created.

- Parameters:
  – `path`: Pointer to a null-terminated string specifying the file name of the file to open.
- Return Values:
  – `FR_OK (0)`: Function success.
  – `FR_NO_FILE`: The file or path could not be found.
  – `FR_DISK_ERR`: The function failed due to a hard error, wrong FAT structure or an internal error.
  – `FR_NOT_ENABLED`: The volume has not been mounted.

Read from a File

```
FRESULT pf_read (
  void* buff,  /* [OUT] Pointer to the read buffer */
  UINT btr,    /* [IN]  Number of bytes to read */
  UINT* br     /* [OUT] Number of bytes read */
);
```

The `pf_read()` function is used to read data from an open file. It is available when `_USE_READ` is written to '`1`' in the `pffconf.h` file. The file pointer (`fptr`) in the file system object increases by the number of bytes read. Upon function success, `*br` should be checked to detect end of file; if `*br` is less than `btr` at the end of the function, the end of the file was reached during the read operation. If the argument buff is given as NULL, the read data are forwarded to the outgoing stream rather than stored in a memory buffer. The streaming function depends on each project and will typically be implemented in the `disk_readp()` function.

- Parameters:
  - `buff`: Pointer to the buffer where read data should be stored. A null pointer will result in the read data being forwarded to the output stream.
  - `btr`: Number of bytes to read.
  - `br`: Pointer to a variable which will indicate number of bytes read upon return.
- Return Values:
  - `FR_OK (0)`: Function success.
  - `FR_DISK_ERR`: The function failed due to a hard error, wrong FAT structure or an internal error.
  - `FR_NOT_OPENED`: The file has not been opened.
  - `FR_NOT_ENABLED`: The volume has not been mounted.

Write to a File

```
FRESULT pf_write (
  const void* buff, /* [IN]  Pointer to the data to be written */
  UINT btw,         /* [IN]  Number of bytes to write */
  UINT* bw          /* [OUT] Pointer to the variable to return number of bytes written */
);
```

The `pf_write()` function is used to write data to an open file. It is available when `_USE_WRITE` is written to '`1`' in the `pffconf.h` file. The file pointer (`fptr`) in the file system object increases by the number of bytes written. Upon function success, `*bw` should be checked to detect end of file; if `*bw` is less than `btw` at the end of the function, the end of the file was reached during the write operation.

Restrictions:

- Cannot create files
- Cannot increase the size of existing files
- Cannot update timestamp of the file, i.e. "Last Modified" attribute
- A write operation can only start and stop on a sector boundary
- A write operation will not be blocked by a file having a read-only attribute

File write must be completed in the following sequence:

1. `pf_lseek(ofs);` The read/write pointer must be moved to a sector boundary prior to initiating a write operation. If this is not done, the pointer will be rounded down to the closest sector boundary.
2. `pf_write(buff, btw, &bw);` Write data to the file. This function can be called repeatedly if necessary i.e., data are being produced continually. However, no data will be fully committed until the write is finalized in the next step.
3. `pf_write(0, 0, &bw);` Finalize the write operation. If the read/write pointer is not on the sector boundary, the remaining bytes in the sector will be filled with zeros. This step must be completed to commit all written data.

- Parameters:
  - `buff`: Pointer to the buffer where the data to be written are stored. A null pointer will indicate that a write operation might be finalized.
  - `btw`: Number of bytes to write.
  - `bw`: Pointer to a variable, which will indicate number of bytes written upon return.

- Return Values:
  – `FR_OK (0)`: Function success.
  – `FR_DISK_ERR`: The function failed due to a hard error, wrong FAT structure or an internal error.
  – `FR_NOT_OPENED`: The file has not been opened.
  – `FR_NOT_ENABLED`: The volume has not been mounted.

Move the File Pointer

```
FRESULT pf_lseek (
  DWORD ofs        /* [IN] File offset in unit of byte */
);
```

The `pf_lseek()` function is used to move the read/write pointer within an open file. It is available when _USE_LSEEK is written to '`1`' in the `pffconf.h` file. The offset should be specified relative to the top of the file. The file pointer (`fptr`) in the file system object will reflect the change.

- Parameters:
  – `ofs`: Where the read/write pointer should be moved, specified in bytes from the start of the file.
- Return Values:
  – `FR_OK (0)`: Function success.
  – `FR_DISK_ERR`: The function failed due to a hard error, wrong FAT structure, or an internal error.
  – `FR_NOT_OPENED`: The file has not been opened.

Open a Directory

```
FRESULT pf_opendir (
  DIR* dp,          /* [OUT] Pointer to the blank directory object structure */
  const char* path  /* [IN]  Pointer to the directory name */
);
```

The `pf_opendir()` function is used to open an existing directory, and create a directory object for future reference that can be discarded at any time without a specific procedure. It is available when `_USE_DIR` is written to '`1`' in the `pffconf.h` file.

- Parameters:
  – `dp`: Pointer to the blank directory object to be created.
  – `path`: Pointer to a null-terminated string specifying the directory name of the directory to open.
- Return Values:
  – `FR_OK (0)`: Function success.
  – `FR_NO_FILE`: The path could not be found.
  – `FR_NOT_READY`: No volume.
  – `FR_DISK_ERR`: The function failed due to a hard error, wrong FAT structure or an internal error.
  – `FR_NOT_ENABLED`: The volume has not been mounted.

Read a Directory

```
FRESULT pf_readdir (
  DIR* dp,      /* [IN]  Pointer to the open directory object */
  FILINFO* fno  /* [OUT] Pointer to the file information structure */
);
```

The `pf_readdir()` function reads directory entries in sequence. It is available when `_USE_DIR` is written to '`1`' in the `pffconf.h` file. All items in the directory can be read by calling this function repeatedly. When all directory entries have been read, the function puts a null string into member `f_name[]` in the file information structure without returning an error. When `fno` is given as a null pointer, the read index of the directory object will be rewinded.

- Parameters:
  - `dp`: Pointer to the open directory object.
  - `fno`: Pointer to the file information structure in which to store the read item.
- Return Values:
  - `FR_OK (0)`: Function success.
  - `FR_DISK_ERR`: The function failed due to a hard error, wrong FAT structure or an internal error.
  - `FR_NOT_OPENED`: The directory object has not been opened.

5. Module Memory Usage and Configuration

The size of the Petit FatFS module when compiled using AVR-GCC varies between 1 kB-4 kB depending on configuration and optimization settings used. The table below shows example compile sizes with different configuration options, using Atmel Studio 7.0 with optimization for size. Default configuration is with `_USE_READ` and `_FS_FAT16` enabled, and all other options disabled. These approximations are for the Petit FatFs module only, and does not include the disk I/O layer.

Table 5-1. Example Memory Usage of the Petit FatFs Module

	Size [bytes]
Flash (`Default configuration`)	2064
Flash (`!_USE_READ`)	-416
Flash (`_USE_DIR`)	+688
Flash (`_USE_LSEEK`)	+556
Flash (`_USE_WRITE`)	+486
SRAM (`bss`)	1
SRAM (`data`)	42

Configuration can be customized according to what functions are required by the application. The following table shows which functions are removed for each configuration option, contained in the configuration header file (`pffconf.h`).

Table 5-2. Functions Removed by Configuration Options

Function	_USE_READ 0	_USE_DIR 0	_USE_LSEEK 0	_USE_WRITE 0
`pf_mount`				
`pf_open`				
`pf_read`	x			
`pf_lseek`			x	
`pf_opendir`		x		
`pf_readdir`		x		
`pf_write`				x

6. References

This module and information regarding it was sourced from ELM-ChaN at http://elm-chan.org/fsw/ff/00index_p.html.

7. Get Source Code from Atmel | START

The example code is available through Atmel | START, which is a web-based tool that enables configuration of application code through a Graphical User Interface (GUI). The code can be downloaded for both Atmel Studio and IAR Embedded Workbench® via the direct example code-link below or the *Browse examples* button on the Atmel | START front page.

Atmel | START web page: http://start.atmel.com/

Example Code

AVR42776 Petit FAT File System Module

- http://start.atmel.com/#example/Atmel:petitfs_example:1.0.0::Application:AVR42776_Petit_FatFs_Example:

Click *User guide* in Atmel | START for details and information about example projects. The *User guide* button can be found in the example browser, and by clicking the project name in the dashboard view within the Atmel | START project configurator.

Atmel Studio

Download the code as an `.atzip` file for Atmel Studio from the example browser in Atmel | START, by clicking *Download selected example*. To download the file from within Atmel | START, click *Export project* followed by *Download pack*.

Double click the downloaded `.atzip` file and the project will be imported to Atmel Studio 7.0.

IAR Embedded Workbench

For information on how to import the project in IAR Embedded Workbench, open the Atmel | START User Guide, select *Using Atmel Start Output in External Tools*, and *IAR Embedded Workbench*. A link to the Atmel | START User Guide can be found by clicking *Help* from the Atmel | START front page or *Help And Support* within the project configurator, both located in the upper right corner of the page.

8. Revision History

Doc. Rev.	Date	Comments
A	10/2018	• Updated document number and Microchip DS00002799A replaces Atmel 42776A • Updated chapter "Relevant Devices" to include tinyAVR® 0-series, tinyAVR® 1-series and megaAVR® 0-series devices • Minor editorial updates
42776A	10/2016	Initial document release

The Microchip Web Site

Microchip provides online support via our web site at http://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 (FAQ), 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

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 http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

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

Microchip Devices Code Protection Feature

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.

Legal Notice

Trademarks

ISBN: 978-1-5224-3660-7

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.

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

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
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

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

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

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

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

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

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

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

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

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

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

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

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

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

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

Finland - Espoo
Tel: 358-9-4520-820

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

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

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

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

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

Italy - Padova
Tel: 39-049-7625286

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

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

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

Sweden - Gothenberg
Tel: 46-31-704-60-40

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

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