

```
{  
    FILE* sfile;  
    int count = 0;  
  
    sfile = fopen("file1", "r");  
  
    if( sfile == NULL )  
    {  
        return -1;  
    }  
  
    while (1)  
    {  
        char c;  
        c = fgetc(sfile);  
        if(c == EOF)  
        {  
            break;  
        }  
        else  
        {  
            count++;  
        }  
    }  
  
    return count;  
}
```

68K/ColdFire v10.0

Getting Started Manual

A publication of
Altium BV
Documentation Department
Copyright © 1997–2003 Altium BV

All rights reserved. Reproduction in whole or part is prohibited
without the written consent of the copyright owner.

TASKING is a brand name of Altium Limited.

The following trademarks are acknowledged:

FLEXlm is a registered trademark of Globetrotter Software, Inc.

HP and HP-UX are trademarks of Hewlett-Packard Co.

Intel is a trademark of Intel Corporation.

Motorola is a registered trademark of Motorola, Inc.

MS-DOS and Windows are registered trademarks of Microsoft Corporation.

SUN is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of X/Open Company, Ltd.

All other trademarks are property of their respective owners.

Data subject to alteration without notice.

<http://www.tasking.com>

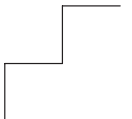
<http://www.altium.com>

The information in this document has been carefully reviewed and is believed to be accurate and reliable. However, Altium assumes no liabilities for inaccuracies in this document. Furthermore, the delivery of this information does not convey to the recipient any license to use or copy the software or documentation, except as provided in an executed license agreement covering the software and documentation.

Altium reserves the right to change specifications embodied in this document without prior notice.

CONTENTS

TABLE OF CONTENTS



CONTENTS

INTRODUCTION **1-1**

1.1	Overview	1-3
1.2	Documentation	1-3
1.2.1	How to Use This Documentation Set	1-4
1.3	The Development System	1-4
1.3.1	The Compiler	1-6
1.3.2	The Optimizer	1-6
1.3.3	The Run-Time Library	1-6
1.3.4	The Assembler	1-7
1.3.5	Utilities	1-7
1.3.6	The Linking Locator	1-7
1.3.7	The Formatters	1-8
1.3.8	The Librarian	1-9
1.3.9	The Global Symbol Mapper	1-9
1.3.10	The Object Size List Utility	1-9
1.3.11	The Symbol List Utility	1-9
1.3.12	CrossView Pro Debugger	1-10
1.4	Before You Start	1-10
1.4.1	Usage Conventions	1-10
1.4.2	Tool Versions	1-10
1.4.3	Driver Options	1-11
1.4.4	Invocation Conventions	1-11
1.4.5	Error Message Output (PC only)	1-12
1.5	Additional Help	1-12
1.5.1	Tutorial	1-12
1.5.2	On-line Help	1-12

INSTALLATION GUIDE **2-1**

2.1	Introduction	2-3
2.2	Software Installation	2-3
2.2.1	Installation for Windows	2-3
2.2.2	Installation for UNIX Hosts	2-4
2.3	Software Configuration	2-5
2.3.1	Configuring the Embedded Development Environment	2-5

2.3.2 Configuring the Command Line Environment 2-7

2.4 Licensing TASKING Products 2-10

2.4.1 Obtaining License Information 2-10

2.4.2 Installing Node-Locked Licenses 2-11

2.4.3 Installing Floating Licenses 2-12

2.4.4 Starting the License Daemon 2-14

2.4.5 Setting Up the License Daemon to Run Automatically . 2-15

2.4.6 Modifying the License File Location 2-16

2.4.7 How to Determine the Hostid 2-17

2.4.8 How to Determine the Hostname 2-17

TUTORIAL **3-1**

3.1 Introduction 3-3

3.2 Finding the Programs and Setting Up the Path 3-3

3.2.1 bin Directory 3-4

3.2.2 rtlbbs Directory 3-6

3.2.3 examples Directory 3-7

3.2.4 Derivatives Overview 3-8

3.3 Invoking the Tools 3-10

3.3.1 Invoking the Tools from EDE 3-10

3.3.1.1 Using the Sample Projects in EDE 3-12

3.3.1.2 Create a New Project Space with a Project 3-13

3.3.1.3 Set Options for the Tools in the Toolchain 3-17

3.3.1.4 Build your Application 3-19

3.3.2 Invoking the Tools Using Command Line 3-20

3.4 Tutorial Examples 3-21

3.4.1 Example 1: Building Your First Application Executable 3-21

3.4.2 Example 2: Listings and Non-Default Output Files . . . 3-23

3.4.3 Example 3: Non-Default Memory Models and
Linking Options 3-30

3.4.4 Example 4: Locator Options 3-34

3.4.5 Example 5: Formatting Options and Saving Symbol
Information 3-38

3.5 Introduction to System Building Concepts 3-41

3.5.1	System Initialization	3-41
3.5.2	A5-Relative vs. Separate Data Addressing	3-42
3.5.3	Linking and Locating	3-42
3.5.4	Linking C and Assembly	3-50
3.6	Tutorial Conclusion	3-53

FLEXIBLE LICENSE MANAGER (FLEXlm)

A-1

1	Introduction	A-3
2	License Administration	A-3
2.1	Overview	A-3
2.2	Providing For Uninterrupted FLEXlm Operation	A-5
2.3	Daemon Options File	A-7
3	License Administration Tools	A-8
3.1	lmcksum	A-10
3.2	lmdiag (Windows only)	A-11
3.3	lmdown	A-12
3.4	lmgrd	A-13
3.5	lmhostid	A-15
3.6	lmremove	A-16
3.7	lmreread	A-17
3.8	lmstat	A-18
3.9	lmswitchr (Windows only)	A-20
3.10	lmver	A-21
3.11	License Administration Tools for Windows	A-22
3.11.1	LMTOOLS for Windows	A-22
3.11.2	FLEXlm License Manager for Windows	A-23
4	The Daemon Log File	A-25
4.1	Informational Messages	A-26
4.2	Configuration Problem Messages	A-29
4.3	Daemon Software Error Messages	A-31
5	FLEXlm License Errors	A-33
6	Frequently Asked Questions (FAQs)	A-37
6.1	License File Questions	A-37
6.2	FLEXlm Version	A-37



6.3 Windows Questions A-38

6.4 TASKING Questions A-39

6.5 Using FLEXlm for Floating Licenses A-41

INDEX

MANUAL PURPOSE AND STRUCTURE

PURPOSE

This manual introduces the TASKING 68K/ColdFire toolchain to the new user. This Getting Started manual allows you to start using the tools right away.

MANUAL STRUCTURE

Related Publications

Conventions Used In This Manual

1. Introduction
Introduces the documentation conventions and organization. Gives an overview of the TASKING 68K/ColdFire toolchain.
2. Installation Guide
Describes how to install the 68K/ColdFire C Compiler/Assembler on your system.
3. Tutorial
Guides you through a brief tutorial to get you started using the tools.

APPENDICES

- A. Flexible License Manager (FLEXlm)
Contains a description of the Flexible License Manager.

INDEX

RELATED PUBLICATIONS

- TASKING 68K/ColdFire C Compiler/Assembler User's Manual
- TASKING 68K/ColdFire C Compiler/Assembler Reference Manual

CONVENTIONS USED IN THIS MANUAL

The notation used to describe the format of call lines is given below:

{ } Items shown inside curly braces enclose a list from which you must choose an item.

[] Items shown inside square brackets enclose items that are optional.

| The vertical bar separates items in a list. It can be read as OR.

italics Items shown in italic letters mean that you have to substitute the item. If italic items are inside square brackets, they are optional. For example:

filename

means: type the name of your file in place of the word *filename*.

... An ellipsis indicates that you can repeat the preceding item zero or more times.

screen font Represents input examples and screen output examples.

bold font Represents a command name, an option or a complete command line which you can enter.

For example

command [*option*]... *filename*

This line could be written in plain English as: execute the command *command* with the optional options *option* and with the file *filename*.

Illustrations

The following illustrations are used in this manual:



This is a note. It gives you extra information.



This is a warning. Read the information carefully.



This illustration indicates actions you can perform with the mouse.



This illustration indicates keyboard input.



This illustration can be read as “See also”. It contains a reference to another command, option or section.

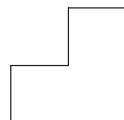
CHAPTER

1

INTRODUCTION



TASKING



1

CHAPTER

1.1 OVERVIEW

This introduction consists of a brief summary of the software documentation package, how to use the package, conventions used in the documentation, an explanation of each of the tools in the 68K/ColdFire cross-development system, and technical information for reference before you start using the tools.

1.2 DOCUMENTATION

Three manuals make up the 68K/ColdFire documentation: the *Getting Started Manual*, the *C Compiler/Assembler User's Manual* and the *C Compiler/Assembler Reference Manual*.

The *Introduction* chapter in this manual contains a summary of the development system, and valuable technical information about the software in the *Before You Start* section.

The *Installation Guide* chapter in this manual is primarily for the initial installation of the software on your computer system.

The *Tutorial* chapter in this manual contains sample code and exercises which lead you step-by-step through the powerful features of each software tool. This material is useful for investigating a particular function using the sample code provided in the `examples\tutor` directory.

The *C Compiler/Assembler User's Manual* includes invocation, options, and usage summaries, along with examples for each of the tools and definitions of special terminology and functions. This manual also contains additional information in the appendices on run-time and naming conventions, C language extensions, and object module formats.

The *C Compiler/Assembler Reference Manual* provides information on the run-time libraries and the information necessary to write programs in assembly language. It contains sections on source program coding, assembler directives, macro operations, structured control statements, and position-independent code, as well as a summary of the character set.

1.2.1 HOW TO USE THIS DOCUMENTATION SET

The documentation is organized to be as flexible and useful as possible. We have considered the varying levels of our users in organizing and writing this manual set. For the inexperienced user of the development system, we have included an *Tutorial* with exercises and sample code illustrating the basic features of the tools. By “inexperienced users,” we mean those who have a working knowledge of the C programming language, assembly language, and the host computer system they are using, but have limited experience using cross compilers, assemblers, and debuggers.

For the experienced user, the *User’s Manual* offers detailed explanations of the tools and their use. By “experienced users,” we mean those who are conversant with the processes of compiling, assembling and debugging for embedded systems software applications. Of course, beginning users will find the *User’s Manual* useful for exploring the features of the tools once the *Tutorial* has been mastered.

1.3 THE DEVELOPMENT SYSTEM

The TASKING 68K/ColdFire toolset is a fully integrated development system. The tools work seamlessly together and share a common object language. The tools produce object language files in ASCII format, while the formatter can produce either binary or ASCII files. Source code and TASKING object modules are portable from host to host, and are interchangeable between all supported hosts. The flexibility of the TASKING solution allows quick and efficient hardware upgrades.

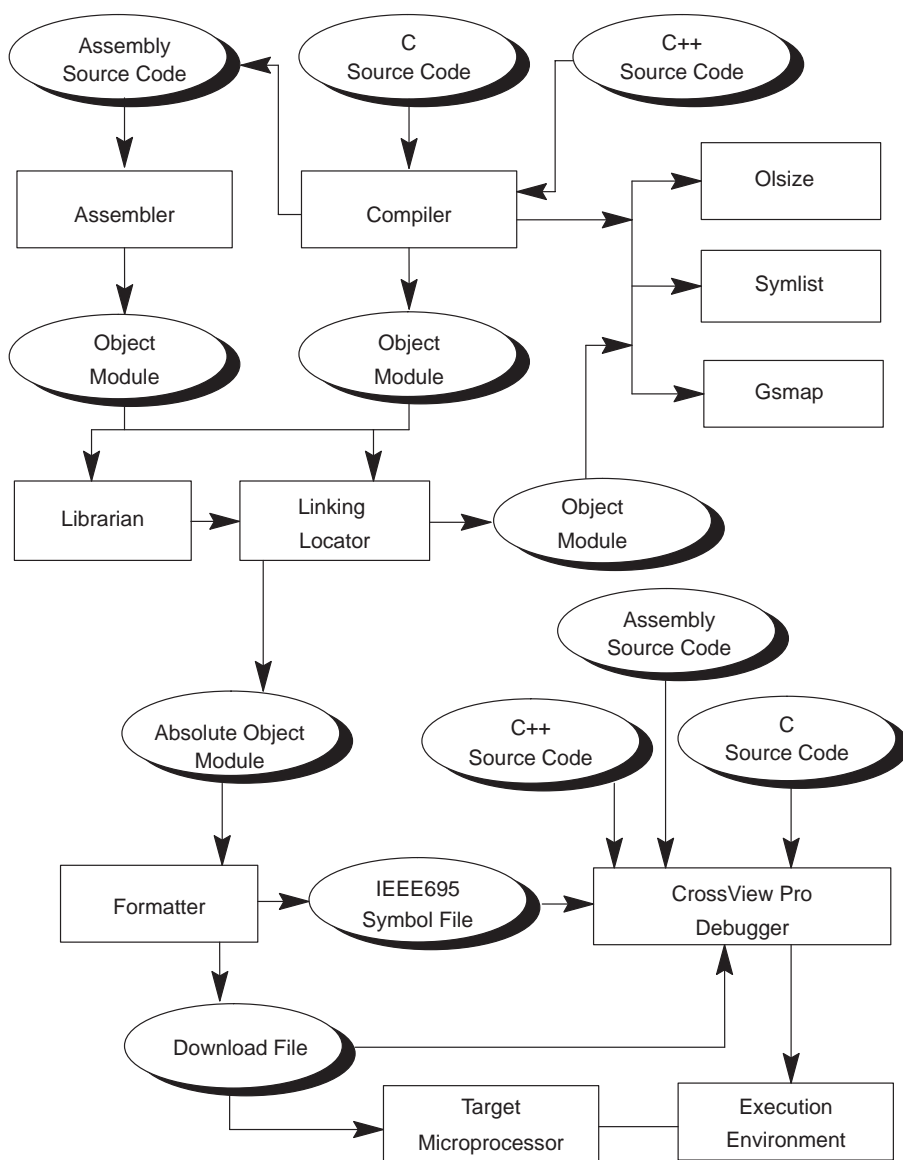


Figure 1-1: Toolchain overview

1.3.1 THE COMPILER

The compiler translates C source into machine instructions for the target microprocessor. The input is one or more source programs. The C language implemented conforms to the ANSI C standard ANSI/ISO 9899-1990.

Compiler output is an object module suitable for linking with other modules. These object modules can also be catalogued in a library using the librarian utility. The compiler has optional listings which show interleaved source and generated machine instructions, along with cross-reference listings. A pseudo-assembly listing is also available to allow you to view code emitted by the compiler at the assembly language level.

1.3.2 THE OPTIMIZER

The TASKING global optimizer improves the speed and reduces the size of the code generated by the TASKING compiler. The global optimizer runs after the compiler front end and before the code generator. It is completely integrated into the compiler system, but its use is entirely optional. Code compiled with the global optimizer can be freely combined with non-optimized code.

The global optimizer performs a variety of optimizations. These optimizations include automatic register allocation, loop optimization, code hoisting, loop rotation, and common subexpression elimination. Please refer to the *C Compiler* chapter in the *User's Manual* for a complete description of usage and options.

1.3.3 THE RUN-TIME LIBRARY

The TASKING 68K/ColdFire toolset includes full run-time libraries: math functions, memory allocation functions, standard I/O functions, string manipulation functions, and floating point routines.

See the *Run-Time Library* chapter in the *Reference Manual* for listings and more detailed information about integrating run-time library modifications.

1.3.4 THE ASSEMBLER

The TASKING 68K/ColdFire toolset includes a macro assembler. The source format is manufacturer-compatible. That is, existing manufacturer-compatible assembly code is easily reassembled using the TASKING assembler. Minor changes may be needed if the assembled modules are to be invoked by compiled modules. Refer to the *Linking C and Assembly* application note in the *User's Manual* for more information.

The input to the assembler is one or more source programs. The output is a corresponding number of object modules suitable for linking to other modules. The object modules can be catalogued in a library. Assembler object modules are compatible with C compiler object modules. Source, cross-reference, and symbol table listings are available from the assembler.

1.3.5 UTILITIES

The TASKING compiler and assembler software includes a full set of utilities. These tools increase programming productivity by reducing the time spent on repetitive software building tasks. A brief description of the utilities is given below. The linking locator and formatter must be run in order to produce a download module. These utilities are described first, followed by optional utilities. Each utility is explained in more detail in the appropriate chapter of the *User's Manual*.

1.3.6 THE LINKING LOCATOR

The linking locator integrates the results of separate compilations and assemblies into a single absolute module. This is done in three separate steps, any or all of which can be performed in a single invocation of the linking locator. The first step, called "linking," consists of combining separate object modules into a composite module by resolving references. Usually these object modules are produced by the assembler and/or compiler, but pre-linked object modules may also be used as input. The linking locator searches libraries to satisfy any unresolved references in the module it is constructing.

The second (optional) step, called ROM processing, consists of building initialization segments used to initialize read-write data. All ROM-based systems must execute code to initialize their read-write data, since the initial values cannot be maintained in RAM (random-access memory), and read-write data cannot be allocated in ROM (read only memory). This data could be initialized by large numbers of assignment statements, but it is more convenient and efficient to employ ROM processing instead. Unlike the read-write data, the initialization segment is suitable for placement in ROM. The initial data values are copied from ROM to RAM at the time of initialization by the library routine `rcopy`.

The final step, called locating, consists of assigning absolute target-memory locations to relocatable segments and resolving address references. The linking locator gives you complete control over placement of all code and data, but it also has the capacity to automatically locate collections of segments in bounded areas of the target memory. The output is an object module with absolute addresses substituted where appropriate. A completely located module contains all the information necessary to load and execute the code on the target microprocessor. The linking locator can resolve the problem of storing a program into a fragmented memory space consisting of ROM, RAM, and I/O mapped device addresses.

1.3.7 THE FORMATTERS

form and **form695** convert the contents of an absolute object module into one of the industry standard formats, in either an ASCII hex or a binary format. The formats provide for loading of object text, that is, code and data, into the memory of the target processor using a loader. The loader is generally provided by an emulator or other instrumentation system, or by a ROM-resident monitor program. The formatter offers many different formats in order to be compatible with a wide range of loaders.

The input is a module from the linking locator and the output is a formatted load file. The formats may also be used as input to a PROM burner to program read-only memory. See the *Formatter* chapter in the *User's Manual* for a list of supported formats. The *Object Module Formats* appendix in the *User's Manual* gives detailed information on each of the supported formats.

1.3.8 THE LIBRARIAN

The librarian is a tool for managing libraries of program modules at the pre-link or post-link phase of development. The librarian creates, maintains, and selectively lists library index files. A library index file is a text file defining an indexing structure which describes a collection of object modules. It consists of a series of index entries, one for each object module. The librarian's input is taken from the library and/or object modules named on the command line or through options specified on the command line. The object modules named on the command line or in a file are added to the library. Libraries simplify the task of linking modules, since the linking locator can automatically search libraries for required modules.

1.3.9 THE GLOBAL SYMBOL MAPPER

The global symbol mapper (gsmmap) displays global symbols either alphabetically or by address. Gsmmap can be used before or after linking or locating to list external names and the definitions of global symbols. The gsmmap listing shows an absolute address (after locating), length, class, and alignment for each segment.

1.3.10 THE OBJECT SIZE LIST UTILITY

The object size list utility (olsize) lists the total number of words of code, data, and constant data in an object module.

1.3.11 THE SYMBOL LIST UTILITY

The symbol list utility (symlist) produces a listing of all global and local symbols. When the debugger option, **(-d)**, is used in compilation or assembly, target locations for source lines of input code are included in the listing. (See the *C Compiler* and *Assembler* chapters in the *User's Manual* for details on compiler and assembler options.) The input may be any combination of unlinked object modules, linked object modules, and absolute modules. The symlist listing is composed of three parts: a table of executable line numbers and code addresses, a listing of all symbols and their attributes, and an alphabetical list of all symbols with pointers to each symbol's definition and attributes.

1.3.12 CROSSVIEW PRO DEBUGGER

The source-level debugger, CrossView Pro provides you with a means of monitoring and controlling execution of the embedded software using the same terms, definitions, and structures found in the original source program. CrossView Pro has complete access to the symbol tables produced by the compiler, and also knows the compiler's register, parameter passing and run-time stack layout conventions. This means that any data, including structured and dynamic data, can be viewed or set. Other features include breakpoint and assertion modes with which you can control the debugging of the target program. A "transparency" mode allows direct communication with the target system without exiting CrossView Pro. As a Windows debugger it gives you full control of window placement and size. You can customize the interface to suit your requirements.

1.4 BEFORE YOU START

This section contains technical information that you may wish to review before installing and starting to use the TASKING software.

1.4.1 USAGE CONVENTIONS

The common conventions for the TASKING 68K/ColdFire toolset are described here to avoid duplication in subsequent sections.

In this documentation set, we use M68000 to refer to any microprocessor in the 68K/ColdFire family. The supported targets within this microprocessor family are listed in section *Derivatives Overview* of the *Tutorial* chapter.

1.4.2 TOOL VERSIONS

Every TASKING executable has a version number. To display the version information, invoke the program with the **-v** option.

The Customer Support department keeps a record of the version numbers of all the executables which have been shipped to you. However, if you report a problem, a support engineer may ask you to run the program with this option to verify that you are in fact executing the latest version you were sent.

1.4.3 DRIVER OPTIONS

There are some additional driver options used primarily for customer support. Please see the *Compiler/Assembler Driver* appendix in the *User's Manual* for details.

1.4.4 INVOCATION CONVENTIONS

All TASKING programs accept two kinds of arguments: primary arguments and options.

All options are preceded by a hyphen ('-').



All primary arguments must precede all options and options are case sensitive.

For example,

```
form695 xx.ab -o xx.abs
```

is correct, but

```
form695 -o xx.abs xx.ab
```

is not.

Furthermore, options cannot be combined as one option. For example, the following is correct:

```
gsmmap object.ln -a -n
```

This is not:

```
gsmmap object.ln -an
```


1.4.5 ERROR MESSAGE OUTPUT (PC ONLY)

MS-DOS does not provide a mechanism for redirecting error output from the command line. The following two options are accepted by all TASKING programs on the PC:

- err** [*file*] Write error messages to file *file*. If *file* does not exist, it will be created. If *file* does exist, it will be overwritten. If *file* is omitted, then error output will be redirected to standard output.
- err+** [*file*] Just like **-err**, except output will be appended if *file* exists.

1.5 ADDITIONAL HELP

The TASKING system provides several additional sources for further information on the toolkit, including on-line help. What follows is a summary of each of these sources with references to more detailed information provided in other sections of the documentation.

1.5.1 TUTORIAL

The tutorial introduces you to the compiler, assembler and utilities. Sample C and assembly source files are included with this release. By following the tutorial examples while invoking the tools on sample code, you generate listings and learn various options.

1.5.2 ON-LINE HELP

From Command Line

An on-line reference function produces a detailed listing of options available for the compilers and assemblers. For example, if you type:

c68000

function: compile one or more C programs

usage: c68000 prog.c [prog2.c ...] [options]

Options Summary

Listing Options

-a	expanded source listing, including #include files
-i	interleaved (pseudo-)assembly listing
-l	put all listings in optional list file name, e.g. -l filename
-nf	narrow format pseudo-assembly listing
-p	pseudo-assembly listing
-q	real-assembly listing
-s	basic source listing
-x	cross-reference listing

Include Options

-I	specifies user include directories, e.g. -I dir1 [dir2 ...]
-S	specifies system include directories, e.g. -S dir1 [dir2 ...]

--Hit <RETURN> to continue; q to quit--

From EDE

All on-line manuals have a corresponding icon located on the right side of the menu bar. Click on the appropriate icon to access.

Please refer to this manual for help with the software, if the on-line help does not answer your questions.



INTRODUCTION

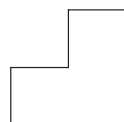
CHAPTER

2

INSTALLATION GUIDE



TASKING



2

CHAPTER

2.1 INTRODUCTION

This chapter guides you through the procedures to install the software on a Windows system or on a UNIX host.

The software for Windows has two faces: a graphical interface (Embedded Development Environment) and a command line interface. The UNIX software has only a command line interface.

After the installation, it is explained how to configure the software and how to install the license information that is needed to actually use the software.

2.2 SOFTWARE INSTALLATION

2.2.1 INSTALLATION FOR WINDOWS

1. Start Windows 95/98/XP/NT/2000, if you have not already done so.
2. Insert the CD-ROM into the CD-ROM drive.

If the TASKING Showroom dialog box appears, proceed with Step 5.

3. Click the **Start** button and select **Run...**
4. In the dialog box type **d:\setup** (substitute the correct drive letter for your CD-ROM drive) and click on the **OK** button.

The TASKING Showroom dialog box appears.

5. Select a product and click on the **Install** button.
6. Follow the instructions that appear on your screen.



You can find your serial number on the *Start-up kit envelope*, delivered with the product.

7. License the software product as explained in section 2.4, *Licensing TASKING Products*.

2.2.2 INSTALLATION FOR UNIX HOSTS

1. Login as a user.

Be sure you have read, write and execute permissions in the installation directory. Otherwise, login as "root" or use the **su** command.

If you are a first time user, decide where you want to install the product. By default it will be installed in `/usr/local`.

2. Insert the CD-ROM into the CD-ROM drive and mount the CD-ROM on a directory, for example `/cdrom`.

Be sure to use an ISO 9660 file system with Rock Ridge extensions enabled. See the UNIX manual pages about **mount** for details.

3. Go to the directory on which the CD-ROM is mounted:

```
cd /cdrom
```

4. Run the installation script:

```
sh install
```

Follow the instructions appearing on your screen.

First a question appears about where to install the software. The default answer is `/usr/local`.

On some hosts the installation script asks if you want to install SW000098, the Flexible License Manager (FLEXlm). If you do not already have FLEXlm on your system, you must install it otherwise the product will not work on those hosts. See section 2.4, *Licensing TASKING Products*.

If the script detects that the software has been installed before, the following messages appear on the screen:

```
*** WARNING ***
SWxxxxxx xxxx.xxxx already installed.
Do you want to REINSTALL? [y,n]
```

Answering **n** (no) to this question causes installation to abort and the following message being displayed:

```
=> Installation stopped on user request <=
```

Answer **y** (yes) to continue with the installation. The last message will be:

Installation of SWxxxxxxx xxxx.xxxx completed.

5. If you purchased a protected TASKING product, license the software product as explained in section 2.4, *Licensing TASKING Products*.

2.3 SOFTWARE CONFIGURATION

Now you have installed the software, you can configure both the Embedded Development Environment and the command line environment for Windows and UNIX.

2.3.1 CONFIGURING THE EMBEDDED DEVELOPMENT ENVIRONMENT

After installation, the Embedded Development Environment is automatically configured with default search paths to find the executables, include files and libraries. In most cases you can use these settings. To change the default settings, follow the next steps:

1. Double-click on the EDE icon on your desktop to start the Embedded Development Environment (EDE).
2. From the **Project** menu, select **Directories...**

The Directories dialog box appears.

3. Fill in the following fields:
 - In the **Executable Files Path** field, type the pathname of the directory where the executables are located. The default directory is `$(PRODDIR)\bin`. Where `$(PRODDIR)` refers to your installation directory (default `c:\Program Files\TASKING\c68k vx.y`).
 - In the **Include Files Path** field, add the pathnames of the directories where the compiler and assembler should look for include files. The default directory is `$(PRODDIR)\rtlibs\$(LIBSUBDIR)\inc`. Separate pathnames with a semicolon (;).

The first path in the list is the first path where the compiler and assembler look for include files. To change the search order, simply change the order of pathnames.

- In the **Library Files Path** field, add the pathname of the directory where the linker should look for library files. The default directory is `$(PRODDIR)\rtlibs`.



Instead of typing the pathnames, you can click on the **Configure...** button.

A dialog box appears in which you can select and add directories, remove them again and change their order.

2.3.2 CONFIGURING THE COMMAND LINE ENVIRONMENT

To facilitate the invocation of the tools from the command line (either using a Windows command prompt or using UNIX), you can set *environment variables*.

You can set the following variables:

Environment Variable	Description
PATH	With this variable you specify the directory in which the executables reside (default: <code>c:\c68k\bin</code>). This allows you to call the executables when you are not in the <code>bin</code> directory. Usually your system already uses the <code>PATH</code> variable for other purposes. To keep these settings, you need to add (rather than replace) the path. Use a semicolon (;) to separate pathnames.
INCLUDE I2INCLUDE	With this variable you specify one or more additional directories in which the C compiler looks for include files. The compiler looks in these directories after the <code>-S</code> directories, and then in the default <code>c:\c68k\include</code> directory. You can also use <code>I2INCLUDE</code> to avoid conflicts with other programs.
LIB I2LIB	With this variable you specify one or more alternative directories in which the linker looks for library files. The linker first looks in these directories, then always looks in the default <code>rtlibs</code> directory. You can also use <code>I2LIB</code> to avoid conflicts with other programs.
LM_LICENSE_FILE	With this variable you specify the location of the license data file. You only need to specify this variable if your host uses the FLEXlm licence manager.
TMP	With this variable you specify the location where programs can create temporary files. Usually your system already uses this variable. In this case you do not need to change it.

Table 2-1: Environment variables

The following examples show how to set an environment variable using the `PATH` variable as an example.

Example for Windows 95/98

Add the following line to your `autoexec.bat` file:

```
set PATH=%path%;c:\installation directory\bin
```



You can also type this line in a Command Prompt window but you will lose this setting after you close the window.

Example for Windows NT

1. Right-click on the My Computer icon on your desktop and select **Properties** from the menu.

The System Properties dialog appears.

2. Select the **Environment** tab.
3. In the list of **System Variables** select **Path**.
4. In the **Value** field, add the path where the executables are located to the existing path information. Separate pathnames with a semicolon (;). For example: `c:\c68k\bin`.
5. Click on the **Set** button, then click **OK**.

Example for Windows XP / 2000

1. Right-click on the My Computer icon on your desktop and select **Properties** from the menu.

The System Properties dialog appears.

2. Select the **Advanced** tab.
3. Click on the **Environment Variables** button.

The Environment Variables dialog appears.

4. In the list of **System variables** select **Path**.
5. Click on the **Edit** button.

The Edit System Variable dialog appears.

6. In the **Variable value** field, add the path where the executables are located to the existing path information. Separate pathnames with a semicolon (;). For example: `c:\c68k\bin`.

7. Click on the **OK** button to accept the changes and close the dialogs.

Example for UNIX

Enter the following line (C-shell):

```
setenv PATH $PATH:/usr/local/c68k/bin
```

2.4 LICENSING TASKING PRODUCTS

TASKING products are protected with license management software (FLEXlm). To use a TASKING product, you must install the licensing information provided by TASKING for the type of license purchased.

You can run TASKING products with a node-locked license or with a floating license. When you order a TASKING product determine which type of license you need (UNIX products only have a floating license).

Node-locked license (PC only)

This license type locks the software to one specific PC so you can use the product on that particular PC only.

Floating license

This license type manages the use of TASKING product licenses among users at one site. This license type does not lock the software to one specific PC or workstation but it requires a network. The software can then be used on any computer in the network. The license specifies the number of users who can use the software simultaneously. A system allocating floating licenses is called a **license server**. A license manager running on the license server keeps track of the number of users.



See Appendix A, *Flexible License Manager (FLEXlm)*, for more information.

2.4.1 OBTAINING LICENSE INFORMATION

Before you can install a software license you must have a "License Information Form" containing the license information for your software product. If you have not received such a form follow the steps below to obtain one. Otherwise, you can install the license.

Node-locked license (PC only)

1. If you need a node-locked license, you must determine the hostid of the computer where you will be using the product. See section 2.4.7, *How to Determine the Hostid*.
2. When you order a TASKING product, provide the hostid to your local TASKING sales representative. The License Information Form which contains your license key information will be sent to you with the software product.

Floating license

1. If you need a floating license, you must determine the hostid and hostname of the computer where you want to use the license manager. Also decide how many users will be using the product. See section 2.4.7, *How to Determine the Hostid* and section 2.4.8, *How to Determine the Hostname*.
2. When you order a TASKING product, provide the hostid, hostname and number of users to your local TASKING sales representative. The License Information Form which contains your license key information will be sent to you with the software product.

2.4.2 INSTALLING NODE-LOCKED LICENSES

Keep your "License Information Form" ready. If you do not have such a form read section 2.4.1, *Obtaining License Information*, before continuing.

Step 1

Install the TASKING software product following the installation procedure described in section 2.2.1, *Installation for Windows*.

Step 2

Create a file called "license.dat" in the c:\flexlm directory, using an ASCII editor and insert the license information contained in the "License Information Form" in this file. This file is called the "license file". If the directory c:\flexlm does not exist, create the directory.



If you wish to install the license file in a different directory, see section 2.4.6, *Modifying the License File Location*.



If you already have a license file, add the license information to the existing license file. If the license file already contains any SERVER lines, you must use another license file. See section 2.4.6, *Modifying the License File Location*, for additional information.

The software product and license file are now properly installed.



See Appendix A, *Flexible License Manager (FLEXlm)*, for more information.

2.4.3 INSTALLING FLOATING LICENSES

Keep your "License Information Form" ready. If you do not have such a form read section 2.4.1, *Obtaining License Information*, before continuing.

Step 1

Install the TASKING software product following the installation procedure described earlier in this chapter on the computer or workstation where you will use the software product.

As a result of this installation two additional files for FLEXlm will be present in the `flexlm` subdirectory of the toolchain:

Tasking	The Tasking daemon (vendor daemon).
license.dat	A template license file.

Step 2

If you already have installed FLEXlm v6.1 or higher for Windows or v2.4 or higher for UNIX (for example as part of another product) you can skip this step and continue with step 3. Otherwise, install SW000098, the Flexible License Manager (FLEXlm), on the license server where you want to use the license manager.

The installation of the license manager on Windows also sets up the license daemon to run automatically whenever a license server reboots. On UNIX you have to perform the steps as described in section 2.4.5, *Setting Up the License Daemon to Run Automatically*.



It is not recommended to run a license manager on a Windows 95 or Windows 98 machine. Use Windows NT instead (or UNIX).

Step 3

If FLEXlm has already been installed as part of a non-TASKING product you have to make sure that the `bin` directory of the FLEXlm product contains a copy of the **Tasking** daemon (see step 1).

Step 4

Insert the license information contained in the "License Information Form" in the license file, which is being used by the license server. This file is usually called `license.dat`. The default location of the license file is in directory `c:\flexlm` for Windows and in `/usr/local/flexlm/licenses` for UNIX.



If you wish to install the license file in a different directory, see section 2.4.6, *Modifying the License File Location*.

If the license file does not exist, you have to create it using an ASCII editor. You can use the license file `license.dat` from the toolchain's `flexlm` subdirectory as a template.



If you already have a license file, add the license information to the existing license file. If the `SERVER` lines in the license file are the same as the `SERVER` lines in the License Information Form, you do not need to add this same information again. If the `SERVER` lines are not the same, you must use another license file. See section 2.4.6, *Modifying the License File Location*, for additional information.

Step 5

On each PC or workstation where you will use the TASKING software product the location of the license file must be known. If it differs from the default location (`c:\flexlm\license.dat` for Windows, `/usr/local/flexlm/licenses/license.dat` for UNIX), then you must set the environment variable **LM_LICENSE_FILE**. See section 2.4.6, *Modifying the License File Location*, for more information.

Step 6

Now all license information is entered, the license manager must be started (see section section 2.4.4). Or, if it is already running you must notify the license manager that the license file has changed by entering the command (located in the `flexlm bin` directory):

lmreread

On Windows you can also use the graphical FLEXlm Tools (**lmtools**): Start **lmtools** (if you have used the defaults this can be done by selecting **Start -> Programs -> TASKING FLEXlm -> FLEXlm Tools**), fill in the current license file location if this field is empty, click on the **Reread** button and then on **OK**. Another option is to reboot your PC.

The software product and license file are now properly installed.

Where to go from here?

The license manager (daemon) must always be up and running. Read section 2.4.4 on how to start the daemon and read section 2.4.5 for information how to set up the license daemon to run automatically.

If the license manager is running, you can now start using the TASKING product.



See Appendix A, *Flexible License Manager (FLEXlm)*, for more information.

2.4.4 STARTING THE LICENSE DAEMON

The license manager (daemon) must always be up and running. To start the daemon complete the following steps on each license server:

Windows

1. From the Windows **Start** menu, select **Programs -> TASKING FLEXlm -> FLEXlm License Manager**.

The license manager tool appears.

2. In the **Control** tab, click on the **Start** button.
3. Close the program by clicking on the **OK** button.

UNIX

1. Log in as the operating system administrator (usually root).
2. Change to the FLEXlm installation directory (default /usr/local/flexlm):

```
cd /usr/local/flexlm
```

3. For C shell users, start the license daemon by typing the following:

```
bin/lmgrd -2 -p -c licenses/license.dat >>& \
/var/tmp/license.log &
```

Or, for Bourne shell users, start the license daemon by typing the following:

```
bin/lmgrd -2 -p -c licenses/license.dat >> \
/var/tmp/license.log 2>&1 &
```

In these two commands, the **-2** and **-p** options restrict the use of the **lmdown** and **lmremove** license administration tools to the license administrator. You omit these options if you want. Refer to the usage of **lmgrd** in Appendix A, *Flexible License Manager (FLEXlm)*, for more information.

2.4.5 SETTING UP THE LICENSE DAEMON TO RUN AUTOMATICALLY

To set up the license daemon so that it runs automatically whenever a license server reboots, follow the instructions below that are appropriate for your platform. steps on each license server:

Windows

1. From the Windows **Start** menu, select **Programs -> TASKING FLEXlm -> FLEXlm License Manager**.

The license manager tool appears.

2. In the **Setup** tab, enable the **Start Server at Power-Up** check box.
3. Close the program by clicking on the **OK** button. If a question appears, answer **Yes** to save your settings.

UNIX



In performing any of the procedures below, keep in mind the following:

- Before you edit any system file, make a backup copy.

SunOS5 (Solaris 2)

1. Log in as the operating system administrator (usually root).
2. In the directory `/etc/init.d` create a file named `rc.lmgrd` with the following contents. Replace `FLEXLMDIR` by the FLEXlm installation directory (default `/usr/local/flexlm`):

```
#!/bin/sh
FLEXLMDIR/bin/lmgrd -2 -p -c FLEXLMDIR/licenses/license.dat >> \
/var/tmp/license.log 2>&1 &
```

3. Make it executable:

```
chmod u+x rc.lmgrd
```

4. Create an 'S' link in the `/etc/rc3.d` directory to this file and create 'K' links in the other `/etc/rc?.d` directories:

```
ln /etc/init.d/rc.lmgrd /etc/rc3.d/Snumrc.lmgrd
ln /etc/init.d/rc.lmgrd /etc/rc?.d/Knumrc.lmgrd
```

`num` must be an appropriate sequence number. Refer to your operating system documentation for more information.

2.4.6 MODIFYING THE LICENSE FILE LOCATION

The default location for the license file on Windows is:

```
c:\flexlm\license.dat
```

On UNIX this is:

```
/usr/local/flexlm/licenses/license.dat
```

If you want to use another name or directory for the license file, each user must define the environment variable **LM_LICENSE_FILE**.

If you have more than one product using the FLEXlm license manager you can specify multiple license files to the **LM_LICENSE_FILE** environment variable by separating each pathname (*lppath*) with a ';' (on UNIX also ':'):

Example Windows:

```
set LM_LICENSE_FILE=c:\flexlm\license.dat;c:\license.txt
```

Example UNIX:

```
setenv LM_LICENSE_FILE  
/usr/local/flexlm/licenses/license.dat:/myprod/license.txt
```

If the license file is not available on these hosts, you must set **LM_LICENSE_FILE** to *port@host*; where *host* is the host name of the system which runs the FLEXlm license manager and *port* is the TCP/IP port number on which the license manager listens.

To obtain the port number, look in the license file at *host* for a line starting with "SERVER". The fourth field on this line specifies the TCP/IP port number on which the license server listens. For example:

```
setenv LM_LICENSE_FILE 7594@elliott
```



See Appendix A, *Flexible License Manager (FLEXlm)*, for more information.

2.4.7 HOW TO DETERMINE THE HOSTID

The hostid depends on the platform of the machine. Please use one of the methods listed below to determine the hostid.

Platform	Tool to retrieve hostid	Example hostid
SunOS/Solaris	hostid	170a3472
Windows	tkhostid (or use lmhostid)	0800200055327

Table 2-2: Determine the hostid



If you do not have the program **tkhostid** you can download it from our Web site at: <http://www.tasking.com/support/flexlm/tkhostid.zip> . It is also on every product CD that includes FLEXlm.

2.4.8 HOW TO DETERMINE THE HOSTNAME

To retrieve the hostname of a machine, use one of the following methods.

Platform	Method
SunOS/Solaris	hostname
Windows 95/98	Go to the Control Panel, open "Network", click on "Identification". Look for "Computer name".
Windows NT	Go to the Control Panel, open "Network". In the "Identification" tab look for "Computer Name".
Windows XP/2000	Go to the Control Panel, open "System". In the "Computer Name" tab look for "Full computer name".

Table 2-3: Determine the hostname

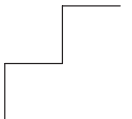


INSTALLATION

CHAPTER

3

TUTORIAL



3

CHAPTER

3.1 INTRODUCTION

This tutorial contains a step-by-step series of examples and exercises designed to teach you how to use your Toolkit. The examples vary in difficulty from simple to advanced, allowing you to progress to more advanced functions. The last section of this chapter gives an introduction to system building concepts so that new users can get started quickly.

Our goal in this tutorial is to teach you how to use the 68K/ColdFire package to build an executable program from your C source and/or assembly language programs. The programs which comprise the toolset are described in the *The Development System* section in the *Introduction* chapter of this manual. We approach the tutorial by guiding you through examples with the tools, using sample programs included with the TASKING software. In the later examples we introduce a few advanced topics that will help you understand this manual. Throughout the tutorial, we follow document conventions described in the front of the manual.

In this tutorial, we will be compiling and assembling files for the 68K/ColdFire family. You will also be introduced to TASKING's graphical user interface, EDE, which gives you point-and-click control over the whole development process. We realize that you may have only one target available to you. The example output is included in such a way as to be useful for a variety of 68K/ColdFire family targets. We encourage you to run as many examples as are applicable, substituting your own target in place of the target listed in each example. Instructions for invoking the compiler and assembler for each target are described below. You can find additional details throughout this manual.

The first part of the tutorial emphasizes the use of the toolchain components. The second part of this chapter introduces important topics such as system initialization, memory management, and linking C and assembly language.

3.2 FINDING THE PROGRAMS AND SETTING UP THE PATH

The package contains many different files. Some are executables: files that make up the compiler, assembler, and utilities. Others contain sample programs, which we use in this tutorial to teach the development system. The files are organized into directories and subdirectories for easy reference. Before you can begin to use the product, you must know where the directories are located and what is contained in each.



This getting started manual includes some information about C++. For more information, consult your *C++ Compiler User's Manual*.

For discussion in this manual, we assume that the product was loaded on drive C: in the directory Program Files\TASKING\c68k for Windows hosts or in /usr/local/c68k for Unix hosts. If so, the executables will probably be in the bin subdirectory. You may want to modify your search path to find the TASKING programs. You can do this by setting the **Executable Files Path** in EDE or use the PATH environment variable. This is described in section 2.3, *Software Configuration* in chapter *Installation Guide*.

You will find at least three subdirectories under the installation directory:

```
bin
examples
rtlibs
```

if your system administrator has not renamed any of the subdirectories. For the purpose of this tutorial, we will assume that none of the directories has been renamed.

Any additional directories or executables are for installation only, and will not be discussed in this tutorial. The directories and their contents are described in the next sections.

3.2.1 BIN DIRECTORY

The directory bin contains the executables, or programs which run the compiler, assembler and utilities. Some of these executables are user-invoked: you must specify that the file be executed. Others are called automatically. You never enter these automatically invoked files in any commands.

A brief description of each file in this directory can be found in the following sections. On the PC, executable files have a .exe extension.

Assembler files

as68k (68K)
ascf (ColdFire)

Assembler executable. This file is invoked automatically when you invoke one of the target-specific assemblers. You do not invoke this executable file directly.

asmtarget

Target specific assembler driver. For example **asm68000** for the MC68000, **asmec040** for the MC68EC040 or **asm5204** for the MCF5204.

C compiler files

ctarget

Target specific C compiler driver. For example **c68000** for the MC68000, **cec040** for the MC68EC040 or **c5204** for the MCF5204.

The following executables are invoked automatically by the C compiler drivers above:

bc68000 (68K)
becf (ColdFire)

Compiler back end.

cpf

C compiler preprocessor and front end.

flow

Global optimizer.

interl

Used to produce interleaved assembly listings.

merge

Used to produce debugging symbols.

xref

Used to produce cross-reference listings.

C++ compiler files

cptarget

Target specific C++ compiler executables. For example **cp68000** for the MC68000, **cpec040** for the MC68EC040 or **cp5204** for the MCF5204.

Utilities

form
form695

Object module formatters.

gsmmap
symlist

Utilities which generate various listings of symbol information.

libr	Librarian utility.
llink	Linking locator.
olsize	Utility which lists the total size of a set of object modules.

C++ Utilities

ldriver	C++ Linker.
edg_munch	Linker Utility.
edg_prelink	Linker Utility.
edg_decode	C++ Name Demangling.

3.2.2 RTLIBS DIRECTORY

The compiler run-time libraries are contained in the directory `rtlibs`. The run-time libraries are necessary for linking, and contain source files, include files, and compiled or assembled run-time library routines, which resolve external references.

The `rtlibs` directory contains the following subdirectories:

```
lib000
lib020h
lib020s
lib040h
lib040s
lib060h
lib060s
lib5206
lib5206e
```

Below each of these directories on the PC are the following subdirectories (if your system administrator has not renamed them):

```
inc
lib
src
```

The `inc` directory contains the library include files. The `lib` directory contains the library index files and object files. The `src` directory contains the library source files.

The C++ compiler contains the additional directories `cppinc`, `cpplib` and `cppstl`.

3.2.3 EXAMPLES DIRECTORY

The `examples` directory contains amongst several sample projects a `tutor` directory with three subdirectories, `fact`, `main` and `cfile`. These directories contain files of sample C programs and assembly language routines which you use throughout the tutorial.

The list below summarizes the source files contained in the `tutor\fact` subdirectory:

<code>adexp.68k</code>	We will use this assembly language program to demonstrate use of the assembler.
<code>circle.c</code>	This program contains a function to compute the area of a circle given its radius. This file demonstrates floating-point features.
<code>fact.c</code>	This program includes a function to compute the factorial of a number. It has external references to the routines in <code>circle.c</code> and <code>adexp.68k</code> . Also, it contains external references to be resolved by linking to the run-time library.
<code>fpneg.68k</code>	This assembly language routine resolves an external reference in <code>fact.c</code> . We use the program to demonstrate the linking locator's function of resolving references.
<code>inc</code>	The <code>inc</code> directory contains the include files <code>dargstac.h</code> , <code>def.h</code> , and <code>ret_doub.h</code> as included source. In the examples, you will specify this directory in order to find files named in <code>#include</code> directives.
<code>link.lst</code>	We will use this file to demonstrate a linking feature of the linking locator.
<code>loc.lc</code>	This is a locator command file which demonstrates a locating feature of the linking locator utility.

Before you proceed with the tutorial, we recommend that you make your own copy of the `tutor` subdirectory, its contents, and its subdirectories. TASKING programs create files as you use the tools, and you may want to keep the tutorial intact for future use.

For the remainder of the tutorial, we assume that you have set your working directory to your copy of the `tutor` subdirectory.

3.2.4 DERIVATIVES OVERVIEW

The following table contains an overview of the supported derivatives with the corresponding *target* to identify the C compilers (*ctarget*), C++ compilers (*cptarget*) and assemblers (*asmtarget*). It also shows the corresponding library and in which `rtlibs\libdir\lib` directory this library is present.

Target Processor	target	Library Directory	Library
MC68000	68000	lib000	lib000
MC68HC000	68000	lib000	lib000
MC68HC001	68000	lib000	lib000
MC68EC000	68000	lib000	lib000
MC68SEC000	68000	lib000	lib000
MC68008	68000	lib000	lib000
MC68010	68010	lib000	lib010
MC68020 (sw fp)	68020	lib020s	lib020s
MC68020 (hw fp)	68020	lib020h	lib020h
MC68EC020 (sw fp)	68020	lib020s	lib020s
MC68EC020 (hw fp)	68020	lib020h	lib020h
MC68030 (sw fp)	68030	lib030s	lib030s
MC68030 (hw fp)	68030	lib030h	lib030h
MC68EC030 (sw fp)	ec030	lib020s	libe30s
MC68EC030 (hw fp)	ec030	lib020h	libe30h
MC68040	68040	lib040h	lib040

Target Processor	<i>target</i>	Library Directory	Library
MC68EC040	ec040	lib040s	libe40
MC68LC040	lc040	lib040s	libe40
MC68V040	lc040	lib040s	libe40
MC68060	68060	lib060h	lib060
MC68EC060	ec060	lib060s	libe60
MC68LC060	lc060	lib060s	libe60
MC68302	68302	lib000	lib302
MC68302 (ADS parallel I/O)	68302	lib000	lib302ap
MC68302 (ADS trap I/O)	68302	lib000	lib302at
MC68306	68302	lib000	lib302
MC68328	68000	lib000	lib000
MC68EZ328	68000	lib000	lib000
MC68VZ328	68000	lib000	lib000
MC68SZ328	68000	lib000	lib000
MC68330	68332	lib020s	lib332
MC68331	68332	lib020s	lib332
MC68332	68332	lib020s	lib332
MC68336	68332	lib020s	lib332
MC68340	68340	lib020s	lib340
MC68340 (BBC)	68340	lib020s	lib340b
MC68360	68360	lib020s	lib360
MC68360 (QUADS)	68360	lib020s	lib360b
MC68F375	68332	lib020s	lib332
MC68376	68332	lib020s	lib332
MCF5204	5204	lib5206	lib5206
MCF5206	5206	lib5206	lib5206
MCF5206E	5206e	lib5206e	lib5206e
MCF5249	5249	lib5206e	lib5206e

Target Processor	<i>target</i>	Library Directory	Library
MCF5249L	5249	lib5206e	lib5206e
MCF5272	5272	lib5206e	lib5206e
MCF5280	5280	lib5206e	lib5206e
MCF5282	5282	lib5206e	lib5206e
MCF5307	5307	lib5206e	lib5206e

3.3 INVOKING THE TOOLS

3.3.1 INVOKING THE TOOLS FROM EDE

EDE is a complete project environment in which you can create and maintain project spaces and projects. EDE gives you direct access to the tools and features you need to create an application from your project.

A *project space* holds a set of projects and must always contain at least one project. Before you can create a project you have to setup a project space. All information of a project space is saved in a *project space file* (.psp):

- a list of projects in the project space
- history information

Within a project space you can create *projects*. Projects are bound to a target! You can create, add or edit files in the project which together form your application. All information of a project is saved in a *project file* (.pjf):

- the target for which the project is created
- a list of the source files in the project
- the options for the compiler, assembler, linker and debugger
- the default directories for the include files, libraries and executables
- the build options
- history information

When you build your project, EDE handles file dependencies and the exact sequence of operations required to build your application. When you click the **Build** button, EDE generates a makefile, including all dependencies, and builds your application.

Overview of steps to create and build an application from EDE

1. Create a project space
2. Add one or more projects to the project space
3. Add files to the project
4. Edit the files
5. Set development tool options
6. Build the application

Start EDE

- Double-click on the EDE shortcut on your desktop.
 - or –

Launch EDE via the program folder created by the installation program. Select **Start -> Programs -> TASKING toolchain -> EDE**.



Figure 3-1: EDE icon

The EDE screen contains a menu bar, a toolbar with command buttons, one or more windows (default, a window to edit source files, a project window and an output window) and a status bar.

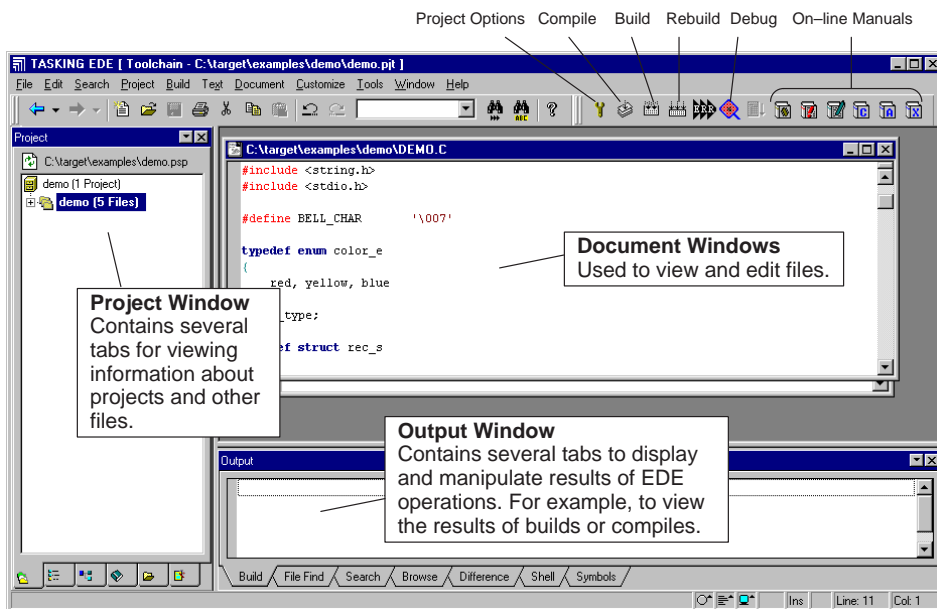


Figure 3-2: EDE desktop

Using the sample projects in EDE

3.3.1.1 USING THE SAMPLE PROJECTS IN EDE

When you start EDE for the first time, EDE opens with a ready defined project space (68K-ColdFire Example.psp) that contains several sample projects. Each project has its own subdirectory in the examples directory. Each directory contains a file `readme.txt` with information about the example. The default project is called `queens.pjt` and contains an eight queens chess problem example.

Select a sample project

To select a project from the list of projects in a project space:

1. In the Project Window, right-click on the project you want to open.

A menu appears.

2. Select **Set as Current Project**.

The selected project opens.

3. Read the file `readme.txt` for more information about the selected sample project.

Building a sample project

To build the currently active sample project:

- Click on the **Execute 'Make' command** button.



*Once the files have been processed you can inspect the generated messages in the **Build** tab of the **Output** window.*

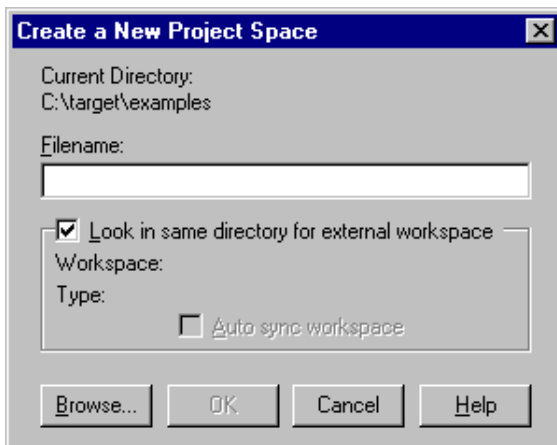
3.3.1.2 CREATE A NEW PROJECT SPACE WITH A PROJECT

Creating a project space is in fact nothing more than creating a project space file (`.psp`) in an existing or new directory.

Create a new project space

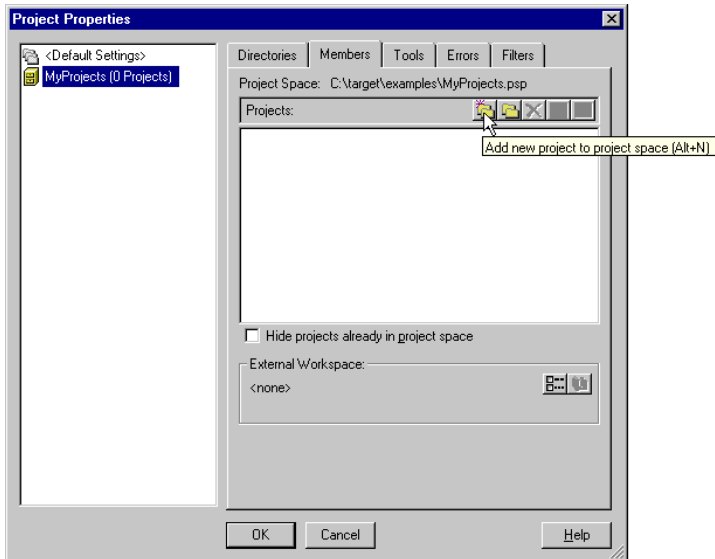
1. From the **File** menu, select **New Project Space...**

The Create a New Project Space dialog appears.



2. In the the **Filename** field, enter a name for your project space (for example **MyProjects**). Click the **Browse** button to select a directory first and enter a filename.
3. Check the directory and filename and click **OK** to create the `.psp` file in the directory shown in the dialog.

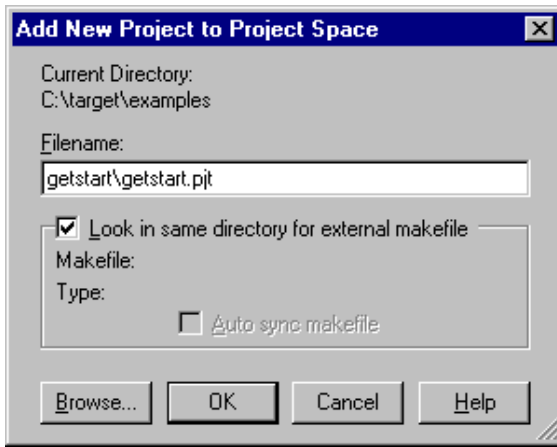
A project space information file with the name `MyProjects.psp` is created and the Project Properties dialog box appears with the project space selected.



Add a new project to the project space

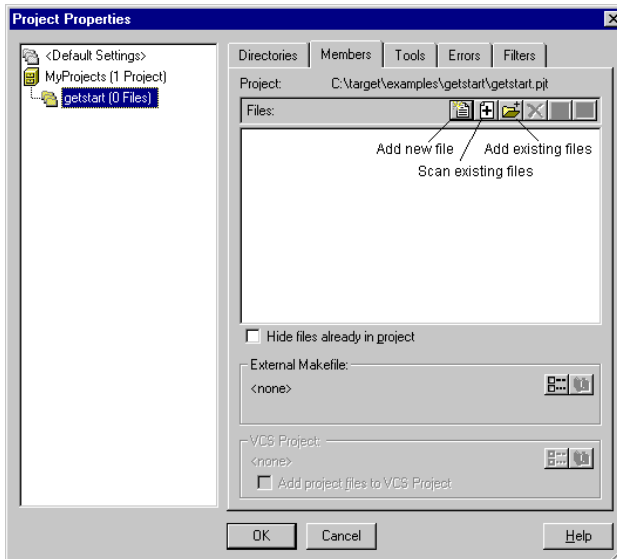
4. In the Project Properties dialog, click on the **Add new project to project space** button (see previous figure).

The Add New Project to Project Space dialog appears.



5. Give your project a name, for example `getstart\getstart.pjt` (a directory name to hold your project files is optional) and click **OK**.

A project file with the name `getstart.pjt` is created in the directory `getstart`, which is also created. The Project Properties dialog box appears with the project selected.

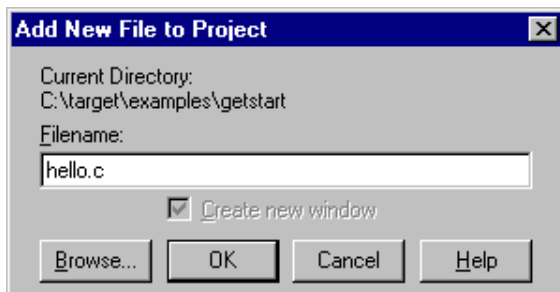


Add new files to the project

Now you can add all the files you want to be part of your project.

6. Click on the **Add new file to project** button.

The Add New File to Project dialog appears.



7. Enter a new filename (for example `hello.c`) and click **OK**.

A new empty file is created and added to the project. Repeat steps 6 and 7 if you want to add more files.

8. Click **OK**.

The new project is now open. EDE loads the new file(s) in the editor in separate document windows.

EDE automatically creates a *makefile* for the project (in this case `getstart.mak`). This file contains the rules to build your application. EDE updates the makefile every time you modify your project.

Edit your files

9. As an example, type the following C source in the `hello.c` document window:

```
#include <stdio.h>

void main(void)
{
    printf("Hello World!\n");
}
```

10. Click on the **Save the changed file <Ctrl-S>** button.



EDE saves the file.

3.3.1.3 SET OPTIONS FOR THE TOOLS IN THE TOOLCHAIN

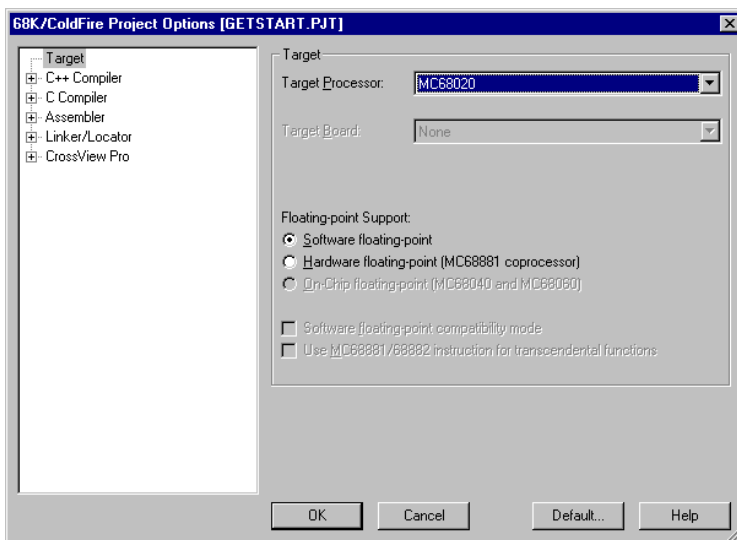
The next step in the process of building your application is to select a target processor and specify the options for the different parts of the toolchain, such as the C compiler, assembler, linker/locator and debugger.

Select a target processor

1. From the **Project** menu, select **Project Options...**

The Project Options dialog appears.

2. Select **Target**.



3. In the **Target processor** list select (for example) **MC68020**.
4. In the **Floating-point support** section select the option that is appropriate for your processor.

5. Click **OK** to accept the new project settings.

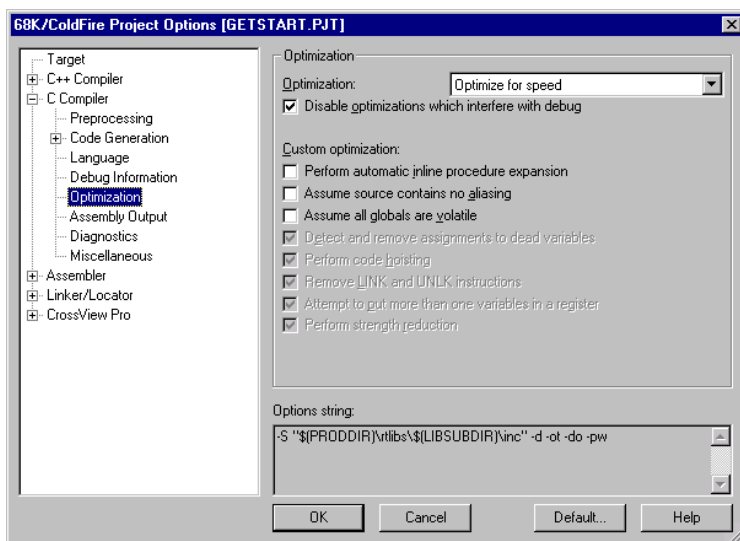
Set tool options

1. From the **Project** menu, select **Project Options...**

*The Project Options dialog appears. Here you can specify options that are valid for the entire project. To overrule the project options for the currently active file instead, from the **Project** menu select **Current File Options...***

2. Expand the **C Compiler** entry.

The C Compiler entry contains several pages where you can specify C compiler settings.



3. For each page make your changes. If you have made all changes click **OK**.



The **Cancel** button closes the dialog without saving your changes. With the **Default...** button you can restore the default project options (for the current page, or all pages in the dialog).

4. Make your changes for all other entries (Assembler, Linker/Locator, CrossView Pro) of the Project Options dialog in a similar way as described above for the C compiler.



If available, the **Options string** field shows the command line options that correspond to your graphical selections.

3.3.1.4 BUILD YOUR APPLICATION

If you have set all options, you can actually compile the file(s). This results in an absolute IEEE-695 object file which is ready to be debugged.

Build your Application

To build the currently active project:

- Click on the **Execute 'Make' command** button.



The file is compiled, assembled, linked and located. The resulting file is `getstart.abs`.



The build process only builds files that are out-of-date. So, if you click **Make** again in this example nothing is done, because all files are up-to-date.

Viewing the Results of a Build

Once the files have been processed, you can see which commands have been executed (and inspect generated messages) by the build process in the **Build** tab of the **Output** window.

This window is normally open, but if it is closed you can open it by selecting the **Output** menu item in the **Window** menu.

Compiling a Single File

1. Select the window (document) containing the file you want to compile or assemble.
2. Click on the **Execute 'Compile' command** button. The following button is the execute Compile button which is located in the toolbar.



If you selected the file `hello.c`, this results in the compiled and assembled file `hello.o1`.

Rebuild your Entire Application

If you want to compile, assemble and link/locate all files of your project from scratch (regardless of their date/time stamp), you can perform a rebuild.

- Click on the **Execute 'Rebuild' command** button. The following button is the execute Rebuild button which is located in the toolbar.



3.3.2 INVOKING THE TOOLS USING COMMAND LINE

All of the TASKING 68K/ColdFire programs follow a general three-part syntax. First, name the program you want to run. Next, name the primary arguments. Finally, append options.

The program named first on the command line is the command that acts upon the input. For instance, the command **c68000** compiles the C source code in the input file(s), creating one or more object modules for the MC68000 target. The command **c68020** compiles source code for the MC68020 target. All the examples shown in this tutorial use the MC68020 target. You should substitute your own target on the command line (for examples that are applicable to your target) when trying the sample commands described below.

The primary arguments are usually names of input files. The files may be source, object modules, or absolute files, depending on the executable. We will discuss each of these files in the examples.

To generate listings or specify particular operations of the program, append the necessary options to the invocation line. Options always begin with a hyphen, “-”. While you can append multiple options, you must include the hyphen for each. For example, the following syntax is correct:

```
c68020 fact.c -s -l
```

The combination below is incorrect:

```
c68020 fact.c -sl
```



Due to the page width limitations of this manual, we may present a single invocation line as two or more lines. You should, however, type the invocation all on one line.

3.4 TUTORIAL EXAMPLES

The remainder of this tutorial is dedicated to running the TASKING 68K/ColdFire programs on the sample source programs provided with the release. The examples cover the following topics:

Example 1: Building Your First Application Executable

Example 2: Listings and Non-Default Output Files

Example 3: Non-Default Memory Models and Linking Options

Example 4: Locator Options

Example 5: Formatting Options and Saving Symbol Information



The remaining examples all use PC-style directory pathnames; that is, they use backslashes (\). If you used the default installation directory on a PC the product is installed in `c:\Program Files\TASKING\c68k version\`. In the examples we use `\c68k` instead. If your host is Unix, you should use forward slashes (/) and directory names starting with something like `/usr/local/c68k` instead.

For all examples, we assume that your current working directory is set to your local copy of the `examples\tutor\fact` directory.

3.4.1 EXAMPLE 1: BUILDING YOUR FIRST APPLICATION EXECUTABLE

In this example you will do the following:

- Compile a C source program.
- Link, ROM process and locate the compiled object module.
- Format the absolute module in the default format for the 68K/ColdFire family.

Step 1

Compile a C source program, `circle.c`. In our example, we compile for the MC68020 target. Type:

```
c68020 circle.c
```

Step 2

Link, ROM process and locate the object module `circle.o1`. Search the run-time library specified by the library index file `\c68k\rtlibs\lib020s\lib\lib020s` to resolve references. Write output to a file with the default suffix. Type:

```
llink circle.o1 -L \c68k\rtlibs\lib020s\lib\lib020s -o
```

Step 3

Format the absolute module `circle.ab` using the IEEE-695 format:

```
form695 circle.ab -o circle.abs
```

Explanation

In Step 1 we compiled `circle.c` and created an object module for the MC68020 target. The command **c68020** compiles for the MC68020 target. (You can substitute a different compiler command name for your own target if you are trying this example.) You named `circle.c` as the input file. By default the compiler writes the object module to output to a file with the suffix `.o1`. In this example, the compiler wrote the object module `circle.o1`. To verify this, list the directory. You should see `circle.o1` among the files.

In Step 2 we linked, ROM processed and located the object module `circle.o1`. The executable `llink` invokes the linking locator.

The **-L** option specifies the library or libraries to be searched to resolve references during linking. In this case, we specified the library `\c68k\rtlibs\lib020s\lib\lib020s`, for the software floating-point MC68020 library. This is the appropriate library for the options we chose, since by default the compiler generates instructions for software floating-point operations. Some targets do not have separate software and hardware floating-point libraries. The organization of the compiler run-time library is described fully in the *Linking Locator* chapter in the *68K/ColdFire User's Manual*. If you are trying this example, substitute the name of the appropriate run-time library for your target.

When locating this object module, the linking locator assigns MC68020 memory locations to code and data in the object module. By default, it assigns code and data segments to consecutive addresses, beginning at zero.

The **-o** option directs the linking locator to write output to a file, rather than the terminal. The output is called absolute because the addresses are fixed, or completely located, and is therefore named with the `.ab` suffix. You should see the file `circle.ab` in your directory listing. If we had not appended the **-o** option, the linking locator would have directed the output to the screen rather than saving it in a file.

In Step 3 we formatted `circle.ab` using the IEEE-695 format. The executable `form695` formats the named absolute module `circle.ab`, and the **-o** option specifies the output file `circle.abs`. You should see the file `circle.abs` in your directory list.

This IEEE-696 file is the final result of the development processing chain. It can be loaded into your target system in a variety of ways and then executed. Refer to the *Downloading* application note in the *68K/ColdFire User's Manual* for more details.

3.4.2 EXAMPLE 2: LISTINGS AND NON-DEFAULT OUTPUT FILES

In this example you will do the following:

- Generate a source listing.
- Generate and write the following listings to the file `fact.mny`: assembly interleaved with C source code, cross-referenced.
- Assemble two assembly language source files and generate a listing for one file which shows both primary and included source, and expanded macros.
- Link, ROM process and locate the compiled and assembled object modules.
- Format the resulting absolute `.ab` file, and direct the output to a non-default file.

Step 1

Compile C source code in `circle.c` for the MC68020 target and generate a source listing. Type:

```
c68020 circle.c -s
```

Step 2

Compile C source code in `fact.c` for the MC68020 target and generate an assembly listing interleaved with source code, and a cross-reference listing. Write both listings to `fact.mny`. Type:

```
c68020 fact.c -q -i -x -l fact.mny
```

Step 3

Assemble the assembly language routines `adexp.68k` and `fpneg.68k` for the MC68020 target. Search the `inc` directory for include files. Generate a listing for `adexp.68k` which shows included source, and expanded macros. Type:

```
asm68020 adexp.68k -a -m -I inc  
asm68020 fpneg.68k
```

Step 4

Link, ROM process and locate the object modules `fact.ol`, `circle.ol`, `fpneg.ol` and `adexp.ol`. Search the library `\c68k\rtlibs\lib020s\lib\lib020s` to resolve references. Write output to the default file. Type:

```
llink fact.ol circle.ol adexp.ol fpneg.ol -L  
\c68k\rtlibs\lib020s\lib\lib020s -rs idata -o
```



The entire `llink` command above should be typed on one line.

Step 5

Format the absolute file `fact.ab` in the default format, packed Motorola. Write output to a non-default file. Type:

```
form fact.ab -o output.hex
```

Explanation

In Step 1 we compiled the C source program `circle.c` for the MC68020 target. By appending the `-s` option to the invocation line, we generated a source listing with the default suffix `.lis`. The listing `circle.lis` appears below:

```
circle.c                               Oct  8 2003  12:23:49
                                       PAGE      1

1  /******
2  **
3  **  VERSION:      @(#)circle.c      version
4  **
5  **  IN PACKAGE:   68K/ColdFire
6  **
7  **  COPYRIGHT:    Copyright year Altium BV
8  **
9  **  DESCRIPTION:  This program calculates the
                    area of a circle
10 **
11 *****/
12
13 extern double pow(double, double);
14
15 float pi = 3.1416;
16
17 float circle (double radius)
18 {
19     float answer;
20
21     /* pow is a function supplied */
22     /* in the C run time library. */
23
24     answer = pi * pow(radius,2.0);
25
26     return (answer);
27 }
```

In Step 2 we compiled the C program `fact.c` for the MC68020 target. Because the TASKING MC68020 C compiler converts source code directly to object code, it normally bypasses generating an assembly language representation. To see the C source program as it would appear in assembly language, you can generate an assembly listing. This assembly listing can also show the C source interleaved with the generated code. To generate an interleaved listing, we appended the `-q -i` options. A portion of the interleaved listing from `fact.mny` appears below:

```
*57          sum = 0;
          CLR.L    -20(A6)
*58
*59          for (loopvar = 1; loopvar < 8; ++loopvar) {
          MOVEQ.L  #1,D2
*(code hoisted from following statement)
          LEA.L    -14(A6),A1
L20003
*60          table[loopvar] = factorial(loopvar);
          MOVE     D2,-(A7)
          JSR      _factorial
          MOVE     D0,(A1)
*61          sum += table[loopvar];
          MOVE.L   -20(A6),-(A7)
          MOVE     (A1)+,-(A7)
          JSR      __Itof
          ADDQ.L   #2,A7
          MOVE.L   D0,-(A7)
          JSR      __Fadd
          MOVE.L   D0,-20(A6)
*(see line 59)
          ADDQ     #1,D2
          CMPI     #8,D2
          ADDA     #10,A7
          BLT.S    L20003
*62          }
```



The ColdFire compilers always generate assembly. Use the `-i` option to see the interleaved assembly listing.

A cross-reference listing is a table which shows all user-defined types, variables and constants, the line numbers in the source code where they are defined, and any line numbers where they are referenced. The **-x** option appended to the invocation line generated this listing. Part of the cross-reference listing portion of `fact.mny` appears below:

```
Oct  8 2003  12:38:42  CROSS-REFERENCE  :  fact.c    PAGE    1

0
  Def :  fact.c           31
  Ref :  fact.c           31

1
  Def :  fact.c           32
  Ref :  fact.c           32

2
  Def :  fact.c           33
  Ref :  fact.c           33

ENUMTYPE
  Def :  fact.c           14
  Ref :  fact.c           20

FPNEG
  Def :  * undefined *
  Ref :  fact.c           32      76

RECTYPE
  Def :  fact.c           21
  Ref :  fact.c           24      25

a
  Def :  fact.c           17
  Ref :  fact.c           69

b
  Def :  fact.c           18

blue
  Def :  fact.c           14
  Ref :  fact.c           24      68
```

To write output to a non-default file, we appended the **-l** option with the desired file name, `fact.mny` as an argument. The compiler wrote each listing to `fact.mny` consecutively.

In Step 3 we assembled the assembly language routines `adexp.68k` and `fpneg.68k` for the MC68020 target. The **asm68020** executable assembled, for the MC68020 target, the input files `adexp.68k` and `fpneg.68k`. You can substitute a different assembler executable name to assemble for your target.

The source program `adexp.68k` names another file to be included upon assembly. By default, the assembler searches the working directory for include files. In this tutorial, the named include file resides in the `inc` subdirectory. To search outside the working directory, we appended the **-I** option with the directory `inc` as an argument.

The **-a** option generates an assembler listing that contains primary and included source. By default, the listing file has the `.lis` suffix. A portion of the listing `adexp.lis` appears below. Note that lines #17 and #18 name `dargstac.h` and `ret_doub.h` as included source. The following portion of `adexp.lis` shows an assembler listing that contains the expansion of the first include file source.

```

13      0      | *      registers not being modified when it is called.
14      0      | *
15      0      | *****/
16      0      |
17      0      |      include 'dargstac.h'
1      1 0      | *****/
2      1 0      | *      include file dargstac.h for tutor directory
3      1 0      | *
4      1 0      | *      @(#)dargstac.h      1.1      03/03/26
5      1 0      | *****/
6      1 0      | *
7      1 0      | regw  set      4      register width
8      1 0      | regnum set     14     number of registers saved
9      1 0      | regspc set     regw*regnum amount of space
          |                  taken by saved regs
10     1 0      | [$0]  numbyt  set     0      init numbyt
11     1 0      |
12     1 0      | pusharg MACRO      macro to copy next argument to stack

```

The `-m` option causes the assembler to include macro expansions in the listing file. Macros provide a shorthand means to invoke a series of assembly language source statements that appear a number of times throughout a program. Rather than writing the consecutive lines of code at every appropriate point in the program, you can name the consecutive lines as a “macro,” then invoke the macro with a single line of code. Note that line #12 of the `dargstac.h` included source file defines a macro. The macro ends at line #23. The macro is invoked in the file `adexp.68k`, which includes `dargstac.h`. For more information on macros, please refer to the *68K/ColdFire Reference Manual*. Macro expansions show the macro contents read by the assembler.

In Step 4 we linked, ROM processed and located the compiled and assembled `.ol` files `fact.ol`, `circle.ol` and `adexp.ol`. We named each input object module, separating file names with a space.

The `-rs idata` option causes the linking locator to perform ROM processing. ROM processing is described fully in the *Linking Locator* chapter in the *68K/ColdFire User's Manual*. Briefly, ROM processing provides a way for the program to initialize its global data at run-time. Initialization is done by copying from a ROM-resident initialization segment (created by the ROM processor) into the segment. By default the TASKING 68K/ColdFire compiler allocates all initialized data in the `idata` segment. In this example, by supplying the `idata` argument to the `-rs` option, we create a new segment with the initialization values from the `idata` segment.

To resolve external references, we first searched the library file `\c68k\rtlibs\lib020s\lib\lib020s` and then we located by default.

The linking locator wrote output to the default file `fact.ab`, using the first file name listed, `fact.ol`, to form the root, and appended the default `.ab` suffix.

In Step 5, we entered the following invocation:

```
form fact.ab -o output.hex
```

This invocation formatted the absolute file `fact.ab` in the format packed Motorola, and wrote the output to a non-default file, `output.hex`, by appending the `-o` option with the argument `output.hex`.

3.4.3 EXAMPLE 3: NON-DEFAULT MEMORY MODELS AND LINKING OPTIONS

This example illustrates various options which affect the compiler's choice of generated code. Due to the many approaches, we have divided the example into two sections: floating-point compilations, and compilations with the long integer data type option. We will do the following:

- Compare compilations for the MC68020 target using hardware and software floating-point.
- Link and ROM process multiple files named in a single file.
- Compile using the long integer data type option.
- Link and locate using a long integer library.
- Format the absolute files.

Floating-Point Compilations

Step 1

Compile and generate an interleaved assembly and source listing for the MC68020 target using software floating-point. Write the listing to `circle.sw`. Type:

```
c68020 circle.c -q -i -l circle.sw
```

Step 2

Compile and generate an interleaved assembly and source listing for the MC68020 target using hardware floating-point. Type:

```
c68020 circle.c -q -i -h
```

Step 3

Compile `fact.c` for the MC68020 target using hardware floating-point and assemble `fpneg.68k`. Type:

```
c68020 fact.c -h  
asm68020 fpneg.68k
```

Step 4

Link only the three `.ol` files `fact.ol`, `circle.ol` and `fpneg.ol` named in a single file, `link.lst`.

Link with the MC68020 hardware floating-point run-time library. Type:

```
llink -i link.lst -rs idata
      -L \c68k\rtlibs\lib020h\lib\lib020h -lo -o
```

Explanation

In Step 1, we compiled `circle.c` for the MC68020 target using software floating-point. The assembly and source code listing from file `circle.sw` appears below. Note the code size shown at the bottom:

```
*21          /* pow is a function supplied */
*22          /* in the C run time library. */
*23
*24          answer = pi * pow(radius,2.0);
          SUBA    #16,A7
          MOVE.L  _pi-data(A5),-(A7)
          PEA.L   4(A7)
          JSR     __Ftod
          CLR.L   -(A7)
          MOVE.L  #:40000000,-(A7)
          MOVE.L  12(A6),-(A7)
          MOVE.L  8(A6),-(A7)
          PEA.L   16(A7)
          JSR     _pow
          ADDA    #20,A7
          PEA.L   16(A7)
          JSR     __Dmul
          ADDA    #20,A7
          JSR     __Dtof
          MOVE.L  D0,-4(A6)
*25
*26          return (answer);
*27      }
          UNLK    A6
          RTS
* Function size = 80
* bytes of code = 80
* bytes of idata = 4
* bytes of udata = 0
* bytes of sdata = 0
          XREF    __Ftod
          XREF    _pow
          XREF    __Dmul
          XREF    __Dtof
          _dgroup data
          END
```

In Step 2 we compiled using hardware floating-point and generated an interleaved listing. The `-h` option directs the compiler to use MC68881 instructions to perform floating-point operations. If your target is not equipped with the MC68881, do not use the `-h` option.

By default, the compiler names the interleaved listing by appending the `.s` suffix. A portion of `circle.s` appears below:

```
*21          /* pow is a function supplied */
*22          /* in the C run time library. */
*23
*24          answer = pi * pow(radius,2.0);
          FMOVE.S _pi-data(A5),FP4
          FMOVE.X FP4,-(A7)
          CLR.L   -(A7)
          MOVE.L  #:40000000,-(A7)
          MOVE.L  (___R1+32)(A7),-(A7)
          MOVE.L  (___R1+32)(A7),-(A7)
          JSR     _pow
          ADDA    #16,A7
          FMOVE.X (A7)+,FP4
          FMUL.X  FP4,FP0
          FMOVE.X FP0,FP1

*25
*26          return (answer);
          FMOVE.X FP1,FP0
*27          }
          FMOVE.X (A7)+,FP1
          RTS

* Function size = 62
* bytes of code = 62
* bytes of idata = 4
* bytes of udata = 0
* bytes of sdata = 0
          XREF    _pow
          _dgroup data
          END
```



The figure of bytes of code shown near the end of the listing for comparison with the software floating-point example.

In Step 3, we compiled `fact.c` using hardware floating-point. To do so, we used the hardware floating-point option, `-h`.

In this step, we also assembled `fpneg.68k`. Due to the nature of assembly language, there is no need to append an option to specify floating-point type, so we assembled `fpneg.68k` normally for the MC68020 target, by using the `asm68020` command.

In Step 4, we linked the three object modules, but rather than listing each file name separately, we listed the names of the object language files in a single file, `link.lst`, included with this release. By appending the `-i` option and file name, we directed the linking locator to read the files named in `link.lst`. If you list the contents of `link.lst`, you will see the three file names.

Because the compiled object modules use hardware floating-point, we linked with the hardware floating-point library,
`\c68k\rtlibs\lib020h\lib\lib020h.`

The `-rs` option with the `idata` argument ROM processes the `idata` segment. To link only, we appended the `-lo` option. This option bypasses the locate step, leaving a relocatable object module. Linked and ROM processed output is named, by default, with the `.rmp` suffix. By using the root of the first object module listed, and appending the `.rmp` suffix, the linking locator wrote the linked output to `fact.rmp`.

The Long Integer Data Type Option

Compile for the MC68020 target using the long integer data type option. Link, locate and format. Type:

```
c68020  circle.c -L
llink   circle.ol -L
        \c68k\rtlibs\lib020s\lib\lib020s.l -o
form695 circle.ab -o circle.abs
```

Explanation

In the previous step, we compiled using the long integer option data type option. The long integer option, `-L`, causes the compiler to assign integers four bytes of memory, and short two bytes. The default is to assign integer two bytes of memory and shorts one byte. When invoking the linking locator, we linked with the long integer run-time library, which has an `l` in its extension. All object modules in a single link must use the same data type options. There are many other data type options which are discussed in the *C Compiler* chapter in the *68K/ColdFire User's Manual*.



The ColdFire compilers always use the long integer data type, so the option `-L` is not needed for these compilers. Also the run-time libraries do not have the `.l` extension.

3.4.4 EXAMPLE 4: LOCATOR OPTIONS

This example addresses locator options. We have divided the example into two sections: Separate Data and Locator Commands. We will do the following:

- Locate separate data
- Read locator commands from a file
- Produce a global symbol listing

Separate Data

Step 1

Generate a source listing for the source file `fact.c`, which contains an `#pragma separate` preprocessor directive. Write the listing to `fact.sep`. Type:

```
c68020 fact.c -s -l fact.sep
```

Explanation

In Step 1, we generated `fact.sep`. `fact.sep` is a source listing of `fact.c`, which contains a `#pragma separate` preprocessor directive. The line of the source listing that contains the directive appears below:

```
28 #pragma separate io_port
```

`#pragma separate` allows you to control the allocation of variables into segments. You may want to allocate a variable into its own segment to place it at a particular hardware address. If all data after linking is larger than 64K bytes, you must use `#pragma separate` to avoid exceeding the 64K limit on global data. To specify a separate segment in source code, use a `#pragma separate` directive, as shown above.

A common use of `#pragma separate` among embedded system developers is to accommodate memory mapped I/O. Memory mapped I/O refers to hardware built so that reading or writing a particular hardware address causes input or output to an external device. One way to accommodate memory mapped I/O is to declare a separate variable in your source code and locate the segment at the memory mapped address. This use of `#pragma separate` allows C code to manipulate memory mapped I/O without using assembly language routines.

Locator Commands

Step 2

Compile the two sample C programs and assemble the sample assembly language program for the MC68020 target. Invoke the linking locator to link and locate, reading locator instructions from the file `loc.lc`. Write output to the default file. Type:

```
c68020 fact.c circle.c -s
asm68020 fpneg.68k -I inc
llink -i link.lst -L \c68k\rtlibs\lib020s\lib\lib020s
      -c loc.lc -o
form695 fact.ab -o fact.abs
```

Now, generate a global symbol listing for the linked and located file `fact.ab`. Show all symbols in increasing order of memory address. Type:

```
gsmap fact.ab -n -o
```

Explanation

In Step 2, with the `-c` option, we directed the linking locator to read locator commands from the file `loc.lc`. By default, code and data segments are allocated in memory one after another, beginning at address 0. But in a real embedded system, memory areas are often not contiguous. Also, certain address ranges correspond to RAM, ROM or memory-mapped I/O. You must take care to place code and constant data in ROM and read-write data in RAM.

Default conventions may not result in optimal placement. You can control placement of code or data using locator commands. The tutorial files provided with this release include the file `loc.lc`, a file of commands to locate segments from the input modules. The `-c` option tells the link program to read commands from `loc.lc`. Its contents appear below:

```
--@(#)loc.lc      1.1 03/07/01
MEMORY (#10000);      --Total memory limited to 64K bytes
RESERVE ( #0100 TO #1000);  --Prevent locator from overwriting
RESERVE ( #8000 TO #8100);  --memory reserved for another program
LOCATE (S_io_port : #FF00);  --I/O port's memory location
LOCATE (libcode: AFTER #7000);--put runtime libraries at top of ROM
LOCATE ( {} {code} : #0100);  --Put other code at start of ROM
LOCATE ({data} : #8100);      --RAM area
```


Before we discuss locator commands, let's discuss the concepts of segment and class. A segment is the smallest piece of code or data that can be independently located in target memory. A collection of segments which share a common attribute define a class. Classes are defined implicitly by the compiler, and may be defined explicitly in assembly language or by using compiler directives and options. By convention, class names are delimited by curly braces, {}, in locator commands. For instance, the compiler defines a class named {code} which contains all compiler-generated segments containing machine instructions.

The file `loc.lc` uses the optional `MEMORY` command. The `MEMORY` command defines the true size of target memory. By default the linking locator assumes it can use the entire MC68020 address space.

The `RESERVE` command prevents the location of segments in specified areas. For example, if your system has a ROM-based monitor program, you may wish to avoid loading another program into its address range. Note that in the `TO` syntax, the lower bound is included in the reserve area; the upper bound is not.

The `LOCATE` command actually places code and data in memory areas. You can locate individual segments or whole classes with one command. For example, we located the class {data} after address #8100. The # sign indicates a hexadecimal value.

In the final step, we generated a global symbol listing in increasing address order. The global symbol mapper generates a listing of segments and the definitions of global symbols. It summarizes segment addresses (if you have located the module), length, class, alignment requirements, and combinability. You may require a global symbol table when writing locator commands. Because the symbolic information is derived from an object module, you can generate a global symbol listing either before or after linking and locating. The `gsmmap` executable generates a global symbol listing.

It may also prove useful to generate the listing in increasing order of address. With an address order, you can see a sequential listing of segments. The `-n` option causes gsmmap to sort the symbols in increasing order of address. The `-o` option writes gsmmap output to a file by appending the default `.map` suffix. A portion of `fact.map` follows:

```
Symbol Map for fact.ab                      Oct  8 2003  14:43:39      Page 1

Translator : link
Target     : 68020

Global      Address

__modf      00001000 (4096)
__atexit    000010b6 (4278)
__exit      000010da (4314)
__main      0000110c (4364)
__putc      0000116c (4460)
__getc      0000119e (4510)
__ungetc    000011ea (4586)
__pow       00001218 (4632)
__circle    000013a4 (5028)
__factorial 000013f4 (5108)
__main      0000141c (5148)
.
.
.

Group      Size Limit      Align  Member Segments

data       0000d0 (208)     hword  idata      udata

Segment    Address          Length      Class      Align  Combine

init@0     00000000 (0)             000008 (8)  <null>     byte   private
S___libcdata 00000008 (8)           000009 (9)  constant   hword   private
S__modf     00001000 (4096)        0000b6 (182) code       hword   private
S__atexit   000010b6 (4278)        000056 (86) code       hword   private
init        0000110c (4364)        000060 (96) code       hword   private
S__putc     0000116c (4460)        000032 (50) code       hword   private
S__getc     0000119e (4510)        00007a (122) code       hword   private
S__pow      00001218 (4632)        00018c (396) code       hword   private
S__circle   000013a4 (5028)        000050 (80) code       hword   private
S__factorial 000013f4 (5108)        0000c8 (200) code       hword   private
libcode     00008100 (33024)       00146a (5226) code       hword   private
idata       0000956c (38252)       000040 (64) data       lword   private
udata       000095ac (38316)       000090 (144) data       hword   private
S__io_port  0000ff00 (65280)        000004 (4)  usep       lword   private
.
.
.

Statistics

Segments   : 14
Globals    : 79
Groups     : 1
Sum of class "code" segments : 00001926 (6438)
Sum of class "data" segments : 000000d0 (208)
Sum of all other segments    : 00000015 (21)
```

```
Total size of all segments      : 00001a0b (6667)
```

```
User Start Address = #110c
```

3.4.5 **EXAMPLE 5: FORMATTING OPTIONS AND SAVING SYMBOL INFORMATION**

This example addresses formatting options and saving symbol information. The three sections are: Creating Debugging Symbols, Formatting for Multiple ROMs, and Non-Default Formatting. We will do the following:

- Suppress all optimizations that may interfere with debugging.
- Save symbol information for later use by the CrossView Pro debugger.
- Generate a symbol table listing during assembly.
- Use the formatter window and bias options.
- Format using a non-default format.

Creating Debugging Symbols

Step 1

Compile, assemble and generate a symbol table listing, llink and format. When compiling, suppress all optimizations that may interfere with source-level debugging. At each invocation, save symbol information for CrossView Pro, the TASKING source level cross debugger. Type:

```
c68020 fact.c circle.c -d -do
asm68020 fpneg.68k -d -b -l table.asm
llink -i link.lst -L \c68k\rtlibs\lib020s\lib\lib020s
      -rs idata -x -o
form695 fact.ab -o fact.abs
```

Explanation

We used the `-do` option to suppress optimizations that may interfere with source-level debugging. For more information on optimizations and their possible effects on debugging, please refer to the *C Compiler* chapter in the *68K/ColdFire User's Manual*. We also saved symbol information and prepared for running CrossView Pro, the TASKING source level debugger. At each invocation step, we appended the necessary option: `-d` for the compiler and assembler to save symbols, `-x` for the linking locator to link in two small debugging routines, and `-o` for the formatter to produce the necessary `.abs` symbol table file.

When we assembled `fpneg.68k`, we generated an assembly listing and a symbol table listing by appending the `-b` option. We directed output to `table.asm` with the `-l` option. Portions of `table.asm` appear below:

```

38  1A          |
39  1A          | * exponent is all 1's; if mantissa is
                    non-zero, then the val is Nan
40  1A  2200    |      MOVE.L D0,D1
41  1C  0281007FFFFF |      ANDI.L #$007FFFFF,D1
                    ; Check if the value of the mantissa is zero
42  22  66000008 |      BNE FPN000
                    ; Mantissa is non-zero, NaN
43  26          |
44  26  0A80800000000 |      FPN010 EORI.L #$80000000,D0 ; Flip sign bit
45  2C          |
46  2C  2F00    |      FPN000 MOVE.L D0,-(SP)
                    ; Push value (negated or not) back on the stack
47  2E  4ED0    |      JMP (A0) ; Return
48  30          |      END
.
.
.
```

----- Symbol Table -----

FPN000	type: RELOCATABLE SYMBOL	value : \$2C + libcode
FPN010	type: RELOCATABLE SYMBOL	value : \$26 + libcode
_FPNEG	type: EXTERNAL SYMBOL	value : \$0 + libcode
libcode	type: RELOCATABLE SECTION	size : \$30

Formatting for Multiple ROMs

Step 2

Use the formatter bias and window options to create hex files for multiple ROMs each with 16K of memory. Write the output to non-default files. Type:

```

c68020 fact.c circle.c
asm68020 fpneg.68k
llink -i link.lst -L \c68k\rtlibs\lib020s\lib\lib020s
      -rs idata -o
form fact.ab -w 4000 -o fact.1hx
form fact.ab -w 4000 -a 4000 -o fact.2hx
form fact.ab -w 4000 -a 8000 -o fact.3hx
```

Explanation

Suppose your system uses 48K bytes of memory, and you plan to burn your system into three 16K ROM chips, one for each 16K of memory space. You will want to generate three hex files for input to your PROM burner. Each hex file will contain 16K bytes starting at address 0, each extracted in successive 16K byte chunks from the absolute object module (.ab file). This example shows how to create those three hex files. We used our small `fact.ab` file, although it does not really contain 48K bytes of code and data.

Here, we use the `-w` and `-a` options to format three windows of 16K bytes each, starting at hex addresses 0, 4000 and 8000 in succession. The `-w` (windowing) option chooses the highest address in a hex file after biasing. The `-a` (biasing) option subtracts a constant hex value from each address in the object module.

In our example, the first formatter invocation creates a hex file that contains hex addresses 0 through 3FFF. The second invocation places the next 16K of addresses, from 4000 to 7FFF, into the second hex file, `fact.2hx`, where 4000 hex has been subtracted from each address. In the third invocation, the formatter creates a hex file that contains the highest 16K of addresses. After these three formatter invocations, we have three hex files (`fact.1hx`, `fact.2hx`, and `fact.3hx`), each of which is suitable for loading into a 16K ROM.

Non-Default Formatting

Step 3

Format using a non-default format. Type:

```
c68020 fact.c circle.c
asm68020 fpneg.68k
llink -i link.lst -rs idata
      -L \c68k\rtlibs\lib020s\lib\lib020s -o
form fact.ab -f bt
```

Explanation

Non-default formatting may be required to meet specific requirements of your target system. In Step 3 we invoked the formatter to produce a download file in Binary Tekhex format. To specify a non-default format, append the `-f` option and the desired format. The switch to specify Binary Tekhex is `bt`.

3.5 INTRODUCTION TO SYSTEM BUILDING CONCEPTS

These notes are designed to be an extension to the normal tutorial. They describe things you must consider for system initialization, loading code, and linking C with assembly language in order to build an application.

3.5.1 SYSTEM INITIALIZATION

The TASKING 68K/ColdFire run-time library comes with a set of system initialization templates called either `pmain.68k` or `pmnxxx.68k` for 68K derivatives, where the `xxx` refers to the target board (i.e., `pmn332.68k` or `pmn302a.68k`), or `pmain.asm` for ColdFire derivatives. On the PC, these files are in the `\c68k\rtlibs\libxxx\src` directory. On Unix hosts, these files are in `/usr/local/c68k/rtlibs/libxxx/src`. When compiling a C module containing the procedure `main`, the compiler automatically generates an unresolved reference to a symbol called `__main` (double underscore). This symbol is defined in all of the `pmain` modules as the start of the initialization routine. Thus, when you link your source modules together, the linker will **automatically** link in the appropriate `pmain` module in order to satisfy the unresolved reference to `__main`.

The initialization clears registers, sets up the stack, and performs other required power-on initializations such as enabling I/O. The libraries provide examples which were developed for specific boards. The initialization code must be customized for the actual environment in which it will run. Therefore, it is important that you look at the source code for your particular `pmain`, and change it, if desired. Note that the `pmain` code automatically sets up the first two reset vectors (the `ORG 0` at the bottom), and also sets up `A5` to point to the global data area (the `LEA data,A5` — don't change it if you want `A5`-relative code !).

Once you have customized the `pmain.68k` module for your particular target board, the next step is to make sure that the modified `pmain` is the one that gets used by your application. There are two ways to accomplish this. The first method is to assemble your modified `pmain.68k` file and then explicitly link it with all of your other source files. This will cause the reference to `__main` to be satisfied (by your `pmain` module), and thus the linker will never look in the library. This is probably the preferable solution if you think that your system initialization routine might undergo further changes. The second method is to add your modified `pmain` to the library, following the procedure in the *Modifying the Libraries* section of the *Run-Time Library* appendix in the *68K/ColdFire Reference Manual*.

3.5.2 A5-RELATIVE VS. SEPARATE DATA ADDRESSING

The TASKING 68K/ColdFire compiler by default uses the A5 register as a pointer to the global data area. This is accomplished via the `LEA data, A5` instruction in `pmain`. Setting up this pointer means that when the compiler manipulates global variables it can now generate instructions using the “address register indirect with offset” addressing mode, which will produce smaller and faster code than the direct addressing mode that the compiler would otherwise have to use. The `data` symbol is set up by the linker to hold the address to be stored in A5. For example,

```
i = 2; /* i is a global integer */
```

With A5-relative code:

```
move #2, _i-data(A5) 33fc0002xxxx
```

`xxxx` is the 16-bit offset from A5. On a 68000 this takes 16 cycles plus 3 reads and a write.

With direct addressing:

```
move #2, _i 3b7c0002xxxxxxxx
```

`xxxxxxxx` is the absolute address of `_i`. On a 68000 this takes 20 cycles plus 4 reads and a write.

The A5-relative instruction allows for a 16-bit offset to be specified, giving the compiler a total global data area of 64K to work with. Any global data that your application has in excess of 64K must be declared as separate using the `#pragma separate` directive (see the *Pragma Separate (Option Separate)* application note for details).

3.5.3 LINKING AND LOCATING

The `llink` step combines three important functions: linking, locating, and ROM processing.

The linking step involves:

1. Telling the linker which modules to link together.
2. Giving the linker access to the libraries it needs to resolve any unresolved references.

The first part is accomplished by listing your `.o1` files (that you got from compiling/assembling your source files) on the command line or in a separate file (using the `-i` option).

The second part is accomplished by specifying the path to a library index file using the `-L` option. A library index file is essentially a look-up table of symbols and the modules that define those symbols. If the linker finds a symbol that is unresolved in your source files (e.g., `printf`), the linker searches the library index file for the symbol, and upon finding it links in the appropriate library module automatically.

It is important to make sure that you specify the right library index file when linking. If you look in your `rtlibs\lib000\lib` or `rtlibs/lib000` (or substitute another target for `lib000`) directory, you will find four distinct library index files: `lib000`, `lib000.l`, `lib000.nf`, and `lib000.lnf`. Library index files with an `“l”` extension assume that the source files have been compiled with `-L` option (68K compilers only), which changes the default sizes of shorts and ints to 2 and 4 bytes respectively. Library index files with a `“nf”` extension assume no floating-point operations. Thus, if you did not compile with `-L` and used floating-point operations, you would use `lib000` as your index file, and so on.



The ColdFire compilers always use the long integer data type, so the option `-L` is not needed for these compilers. Also the run-time libraries do not have the `.l` extension.

The locating step involves telling the linker where to locate your code and data through the use of a locator command file. The things that you will actually be locating are segments and classes. A segment is a contiguous section of memory containing code or data that has had a name assigned to it either by you or the compiler. A class is a larger classification which contains any number of “member” segments. For example, the development system has created the class `data` to represent all global data which is A5-relative. The class `data` has two member segments, `idata` and `udata`, which contain the initialized and uninitialized global data respectively. The tables which summarize the segments and classes used in toolchain are in the *Linking Locator* chapter of the *68K/ColdFire User's Manual*. Below is a table which essentially retranslates this information, but with more emphasis on the segment-to-class relationship:

A	Segment	Class
global variable	idata if initialized udata if uninitialized	{data}
string literal	sdata	{constant}
const qualified variable	cdata if you compile with -cs, it is treated as a regular global variable otherwise	{constant} if you compile with -cs, {data} otherwise
separate variable, no user-defined segment name or class name used	S-variable name	{isep} if initialized variable, {usep} if uninitialized variable
separate variable, with user-defined segment name ONLY	whatever segment name you specified	{separate}
ROM processing segment that you have created	whatever you specified with the -b linker option	{constant}
C procedure	S_fname, where fname is the first function in the file containing the <code>proc</code>	{code}, unless you change it with the -cc compiler option
assembly language routine	whatever you specified the segment name to be w/ the SECTION directive. For example, <code>SECTION foo,,"bar"</code> will create a segment <code>foo</code> and a class <code>{bar}</code>	whatever you specified the class to be w/ the SECTION directive, or in the NULL class (<code>{ }</code>), if you didn't use the SECTION directive
assembly language that has been absolutely located with <code>ORG</code>	name@address, where name is the SECTION name containing the <code>ORG</code> and address is the absolute address (e.g., <code>init@0</code>)	NULL, although the segment will be located according to the address in the <code>ORG</code> statement regardless of where the null class is located

See the *ORG – Absolute Origin* and *SECTION – Relocatable Program Section* sections of the *Assembler Directives* chapter in the *68K/ColdFire Reference Manual* for more information on the `ORG` and `SECTION` directives.

Once you know what segments and classes are being created by your compilations, the actual locating process is fairly straight-forward. Within the locator command file, the `LOCATE` directive specifies the segments and/or classes to be located, followed by the absolute address to which they are to be located. Note that class names which are specified in locator command files should be enclosed in curly braces (e.g., `{code}` and `{data}`). If you mention more than one segment or class in a particular locate statement they are located in the order you mention them, one right after the other. Segments are considered to be word-aligned, so you must locate segments and classes at even addresses (for separate variables this restriction can be avoided, see the linking and locating example below).

The last thing to consider in the `link` step is ROM processing. ROM processing is a procedure for storing initialized read-write data in ROM (so that the initial values are not lost when power is off), with the intention of copying this data to RAM at startup (so that the values can be changed). The general procedure works as follows:

1. Establish the identity and contents of a “ROM processing segment”. The ROM processing segment is the area of memory in ROM that contains the copies of everything that will be copied to RAM at startup.

The name of the ROM processing segment is established through the use of the `-b` option (e.g., `-b _myrompsseg`). The catch is that linker symbol names (such as the ROM processing segment name you create with `-b`) are formed by prepending an underscore to the C symbol name. Since you will need to refer to the ROM processing segment's name in C, you must give the ROM processing segment a name that starts with an underscore, so that you can “strip” off the underscore when you refer to it in C.

The contents of the ROM processing segment are defined via the `-rc` (for classes) and `-rs` (for segments) options. Any classes or segments that are specified will be copied into the segment named via the `-b` option. In this fashion you can copy any number of segments and classes into the same ROM processing segment.

2. Having established the contents and name of the ROM processing segment, locate it at an appropriate address in ROM. You can do this by locating the segment name explicitly:

```
locate (_myrompseg: #2000);
```

or by the class name constant;

```
locate ({constant}: #2000);
```

3. Locate the segments/classes that you are ROM processing (i.e., the ones you specified with `-rs` or `-rc`) at addresses in RAM.
4. Modify your `main()` routine to call the library routine `rcopy`. Given that you have used `-b` to define a ROM processing segment called `_myrompseg`, the C file containing `main()` would be modified as follows:

```
#pragma separate myrompseg /* no underscore here, we
strip it when we're in C */
extern int myrompseg;      /* #pragma separate is
necessary to make this work */
#include <rcopy.h>
...
...
main()
{
    rcopy(&myrompseg); /* should be the first executable
line of main() */
    ...
}
```

Linking/Locating example:

For this example assume a “mythical” 68K target, with ROM located at address \$0 (exception vector table occupies \$0 through \$3ff) and RAM at address \$10000. This target also has a bank of 8 LEDs located at address \$9001 (each triggered on a different bit in the byte), and two hex displays at address \$8000 (each displays a hex digit, and both are enabled upon writing a byte to \$8000). The application program (`main.c` as present in the `examples\tutor\main` directory) continuously increments the displayed value after a specified delay; when the display reads FF the bank of LEDs flashes on and off. The program also performs the set-up required for ROM processing in the `l1link` step.

```

/* begin main.c */

char full = 0xff;
char unused;

#pragma sep_on segment ledbank
char dummy; /* A segment must be word-aligned, so we must add a
dummy */
char leds; /* variable to put leds at an odd address */
#pragma sep_off

#pragma separate display
char display;

#pragma separate myromp /* dummy variable for ROM processing
segment */
extern int myromp;

#include <rcopy.h>

void main(void);
void delay(void);

void main(void)
{
    rcopy(&myromp); /* call to rcopy to do ROM processing */
    leds = display = 0;
    while(1)
    {
        delay();
        leds = 0;
        display++;
        if (display == full) leds = full;
    }
}

void delay(void)
{
    return; /* At some point we could insert an
appropriate delay routine */
}
/* end main.c */

```

Now, we compile this file (make the examples\tutor\main directory the current working directory):

```
c68332 main.c -S \c68k\rtlibs\lib020s\inc
```

The segments and classes that are created by this compilation are as follows:

- Segment `idata`, which has class `{data}`, contains the one initialized non-separate global variable in the program (`full`).

- Segment `udata`, which has class `{data}`, contains the one uninitialized non-separate global variable in the program (unused).
- Segment `ledbank` contains the separate variables `dummy` and `leds`. Since the `#pragma` directive specifies a segment name (`ledbank`) but no class, the class defaults to `{separate}`.
- Segment `S_display` contains the variable `display` and has class `{usep}`.
- Segment `S_main` contains the code for the routines `main()` and `delay()`, and has class `{code}`.

Moving on to the `llink` step, we want to accomplish the following:

1. Memory map the variables `leds` and `display` so that they occupy hex addresses `$9001` and `$8000` respectively.
2. ROM process the segment `idata`, locate the ROM processing segment in ROM and reserve space for `idata` (where it will be copied to) in RAM.
3. Locate all code and constant stuff in ROM, and all read-write data in RAM.

This is accomplished by the following **llink** command line:

```
llink main.ol -L \c68k\rtlibs\lib020s\lib\lib332
-c loc.lc -rs idata -b _myromp -o
```

and locator command file, `loc.lc`:

```
locate({code})){constant}: #2000); -- locate code/constant/null
                                classes in ROM
locate(S_display: #8000); -- map hex displays to $8000
locate(ledbank: #9000); -- satisfies linker's desire for
                        word-alignment
locate({data}: #10000); -- locate the data (both idata and udata)
                        in RAM
```

After the `llink`, try running `gsmap` on `main.ab`:

```
gsmap main.ab -o
```

Look at the file `main.map`. In addition to the list of global variables at the beginning, you should also find the following chart of segments:

Segment	Address	Length	Class	Align	Combine
S_atexit	00002002 (8194)	000056 (86)	code	hword	private
S_display	00008000 (32768)	000001 (1)	usep	hword	private
S_main	0000218e (8590)	00003c (60)	code	hword	private
S_mem-set	00002058 (8280)	000086 (134)	code	hword	private
S_rcopy	00002134 (8500)	00005a (90)	code	hword	private
_myromp	000021ca (8650)	000014 (20)	constant	hword	private
idata	00010000 (65536)	000004 (4)	data	hword	private
init	000020de (8414)	000056 (86)	code	hword	private
init@0	00000000 (0)	000008 (8)	<null>	byte	private
ledbank	00009000 (36864)	000002 (2)	separate	hword	private
libcode	00002000 (8192)	000002 (2)	code	hword	private
udata	00010004 (65540)	000082 (130)	data	hword	private



All of the segments generated as a result of the compilation of `main.c` are present, and that new segments have appeared as a result of the linking process. The `init` and `init@0` segments were created by the linking in of the module `pmn332.ln`, which contains the default system initialization code. This happened automatically due to the existence of a routine named `main()`. The `init@0` segment represents the data explicitly located at address 0 due to the `ORG 0` at the bottom of the `pmn332.68k` file. The `_myromp` segment is the ROM processing segment that we created by invoking the linker with `-rs idata -b _myromp`. It has class constant by default, a fact that we used in the `loc.lc` file (by locating {constant} instead of `_myromp`). The `S_memset` and `S_rcopy` segments were generated by the external references to the `rcopy` routine.

3.5.4 LINKING C AND ASSEMBLY

To link C and assembly there really is only one trick and that is:

OBEY THE RUN-TIME AND NAMING CONVENTIONS OF THE COMPILER. Following is a list of important conventions:

- Each symbol referenced in a C program must have an underscore prepended to it when you reference it in assembly language. So, if you declare a variable “`int i;`” in C and then want to access that variable in assembly language, you would do something like `move #2, _i`. Likewise, a function called `main` in C will be called `_main` in assembly.

- The compiler, when calling a function with parameters, will push them onto the stack in reverse order. Also, parameters that are 8 bits in size will be pushed onto the stack as 16 bits (with the high order byte undefined). So, as an example, let's say you have an external function called `foo` that you call in C like so:

```
extern void foo(char,int,long);
...
char c;                | "e" (low word)          |
int d;      8(A7)->    | "e" (high word)       |
long e;     6(A7)->    | "d"                   |
...         4(A7)->    | garbage | "c"         |
foo(c,d,e);                | return addr (low word) |
                        A7-> | return addr (high word) |
```

The compiler will push `e` on the stack first, followed by `d` and `c` (which will be pushed as a word), followed by the return address. If `foo` were actually defined in assembly language (as `_foo`), you would expect parameter `c` to be at location `5(a7)` (not `4(a7)`), since the high-order byte is junk), followed by `d` at `6(a7)`, and then `e` at `8(a7)`.

- The compiler will expect integer return values to be placed in `D0`, and pointer return values to be placed in `A0`. For hardware floating-point, floats and doubles are returned in `FP0`. Otherwise, floats are returned in `D0` and doubles are returned in a temporary stack location.

- The compiler considers D0, D1, A0, and A4 (FP0 and FP4 also for hardware floating-points) to be scratch registers. As such, you never need to worry about saving and restoring these registers in your assembly language programs. A5, A6, and A7 are used by the compiler as the pointer to the global data area, the frame pointer, and stack pointer respectively, and you should not load values into these registers unless you are using a compiler option which suppresses the default use of these registers (i.e., `-n5` or `-n6`). All other registers (D2–D7, A1–A3) must be saved and restored if your assembly language routine writes into them. The compiler will be following the same convention, so you **cannot** leave a “live” value in any of the scratch registers prior to making a function call, because the compiler will not be saving and restoring the scratch registers either, and will happily overwrite any useful data that might exist in these registers. Likewise, you **can** leave useful data in any of the preserved registers (D2–D7, A1–A3) prior to calling a C function from assembly language, since the compiler will save and restore these registers if it uses them.

Here is an example to illustrate some of the above concepts. Consider a function `foo`, which takes two parameters (`p1` and `p2`, both `int`'s), and returns the value $(2 * p1 + 2 * p2)$. The function `foo` is to be written in assembly language and it calls another function `bar`, which is written in C. The `bar` function takes an `int` parameter and returns $2 * \text{parameter}$.

In file `cfile.c` (in the `examples\tutor\cfile` directory):

```
extern int foo(int,int);
int bar(int);
int a, b, c;

void t(void)
{
    c = foo(a,b);
}

int bar(int p1)
{
    return(2*p1);
}
```

In file `asmfile.68k`:

```

XDEF _foo          ; make _foo public
XREF _bar          ; "extern" the _bar symbol so you can use it
SECTION fooseg, "code"
_foo
    MOVE.L D2,-(A7)    ; save D2, we'll be using it
    MOVE    8(A7),-(A7) ; get value of a, push on stack for call to bar
    JSR     _bar       ; call bar(a)
    ADDQ.L #2,A7       ; clean up stack (clear off a)
    MOVE    D0,D2      ; D0 has return value from bar(a), save it since * it
                        ; is a scratch register and our second call to bar will clobber it
    MOVE    10(A7),-(A7) ; get value of b, push on stack for call to bar
    JSR     _bar       ; call bar(b)
    ADDQ.L #2,A7       ; clean up stack (clear off b)
    ADD     D2,D0      ; D0 has return value from bar(b) this time,
    * we add in the value of bar(a) (still safe and sound in D2,
    * since the compiler will save and restore D2 if bar uses it)
    MOVE.L (A7)+,D2    ; restore D2 before exiting
    RTS           ; now we can return, since the return value
    * from foo, currently resides in D0, which is what the compiler expects

```

3.6 TUTORIAL CONCLUSION

The examples above address hypothetical but common needs of the embedded system developer. The tutorial addresses basic topics and may be helpful for periodic review and reference. With an understanding of terminology and practice using the tutorial files, you can approach this manual with fundamental knowledge for detailed applications.



APPENDIX

A

FLEXIBLE LICENSE MANAGER (FLEXlm)



A

APPENDIX

1 INTRODUCTION

This appendix discusses Globetrotter Software's Flexible License Manager and how it is integrated into the TASKING toolchain. It also contains descriptions of the Flexible License Manager license administration tools that are included with the package, the daemon log file and its contents, and the use of daemon options files to customize your use of the TASKING toolchain.

2 LICENSE ADMINISTRATION

2.1 OVERVIEW

The Flexible License Manager (FLEXlm) is a set of utilities that, when incorporated into software such as the TASKING toolchain, provides for managing access to the software.

The following terms are used to describe FLEXlm concepts and software components:

feature	A feature could be any of the following: <ul style="list-style-type: none">• A TASKING software product.• A software product from another vendor.
license	The right to use a feature. FLEXlm restricts licenses for features by counting the number of licenses for features in use when new requests are made by the application software.
client	A TASKING application program.
daemon	A process that "serves" clients. Sometimes referred to as a <i>server</i> .
vendor daemon	The daemon that dispenses licenses for the requested features. This daemon is built by an application's vendor, and contains the vendor's personal encryption code. Tasking is the vendor daemon for the TASKING software.

license daemon

The daemon process that sends client processes to the correct vendor daemon on the correct machine. The same license daemon is used by all applications from all vendors, as this daemon neither performs encryption nor dispenses licenses. The license daemon processes no user requests on its own, but forwards these requests to other daemons (the vendor daemons).

server node A computer system that is running both the license and vendor daemon software. The server node will contain all the dynamic information regarding the usage of all the features.

license file An end-user specific file that contains descriptions of the server nodes that can run the license daemons, the various vendor daemons, and the restrictions for all the licensed features.

The TASKING software is granted permission to run by FLEXlm daemons; the daemons are started when the TASKING toolchain is installed and run continuously thereafter. Information needed by the FLEXlm daemons to perform access management is contained in a license data file that is created during the toolchain installation process. As part of their normal operation, the daemons log their actions in a daemon log file, which can be used to monitor usage of the TASKING toolchain.

The following sections discuss:

- Installation of the FLEXlm daemons to provide for access to the TASKING toolchain.
- Customizing your use of the toolchain through the use of a daemon options file.
- Utilities that are provided to assist you in performing license administration functions.
- The daemon log file and its contents.

For additional information regarding the use of FLEXlm, refer to the chapter *Installation Guide*.

2.2 PROVIDING FOR UNINTERRUPTED FLEXLM OPERATION

TASKING products licensed through FLEXlm contain a number of utilities for managing licenses. These utilities are bundled in the form of an extra product under the name SW000098. TASKING products themselves contain two additional files for FLEXlm in a *flexlm* subdirectory:

Tasking	The Tasking daemon (vendor daemon).
license.dat	A template license file.

If you have already installed FLEXlm (e.g. as part of another product) then it is not needed to install the bundled SW000098. After installing SW000098 on UNIX, the directory `/usr/local/flexlm` will contain two subdirectories, `bin` and `licenses`. After installing SW000098 on Windows the directory `c:\flexlm` will contain the subdirectory `bin`. The exact location may differ if FLEXlm has already been installed as part of a non-TASKING product but in general there will be a directory for executables such as `bin`. That directory must contain a copy of the **Tasking** daemon shipped with every TASKING product. It also contains the files:

lmgrd	The FLEXlm daemon (license daemon).
lm*	A group of FLEXlm license administration utilities.

Next to it, a license file must be present containing the information of all licenses. This file is usually called `license.dat`. The default location of the license file is in directory `c:\flexlm` for Windows and in `/usr/local/flexlm/licenses` for UNIX. If you did install SW000098 then the `licenses` directory on UNIX will be empty, and on Windows the file `license.dat` will be empty. In that case you can copy the `license.dat` file from the product to the `licenses` directory after filling in the data from your "License Information Form".



Be very careful not to overwrite an existing `license.dat` file because it contains valuable data.

Example `license.dat`:

```
SERVER HOSTNAME HOSTID PORT
DAEMON Tasking /usr/local/flexlm/bin/Tasking
FEATURE SW008002-32 Tasking 3.000 EXPDATE NUSERS PASSWORD SERIAL
```


After modifications from a license data sheet (example):

```
SERVER elliot 5100520c 7594
DAEMON Tasking /usr/local/flexlm/bin/Tasking
FEATURE SW008002-32 Tasking 3.000 1-jan-00 4 0B1810310210A6894 "123456"
```

If the `license.dat` file already exists then you should make sure that it contains the DAEMON and FEATURE lines from your license data sheet. An appropriate SERVER line should already be present in that case. You should only add a new SERVER line if no SERVER line is present. The third field of the DAEMON line is the pathname to the **Tasking** daemon and you may change it if necessary.

The default location for the license file on Windows is:

```
c:\flexlm\license.dat
```

On UNIX this is:

```
/usr/local/flexlm/licenses/license.dat
```

If the pathname of the resulting license file differs from this default location then you must set the environment variable **LM_LICENSE_FILE** to the correct pathname. If you have more than one product using the FLEXlm license manager you can specify multiple license files by separating each pathname (*lfp_{path}*) with a ';' (on UNIX also ':') :

Windows:

```
set LM_LICENSE_FILE=lfppath;lfppath...
```

UNIX:

```
setenv LM_LICENSE_FILE lfppath[:lfppath]...
```

If you are running the TASKING software on multiple nodes, you have three options for making your license file available on all the machines:

1. Place the license file in a partition which is available (via NFS on Unix systems) to all nodes in the network that need the license file.
2. Copy the license file to all of the nodes where it is needed.
3. Set LM_LICENSE_FILE to "*port@host*", where *host* and *port* come from the SERVER line in the license file.

When the main license daemon **lmgrd** already runs it is sufficient to type the command:

```
lmreread
```

for notifying the daemon that the `license.dat` file has been changed. Otherwise, you must type the command:

```
lmgrd >/usr/tmp/license.log &
```

Both commands reside in the flexlm bin directory mentioned before.

2.3 DAEMON OPTIONS FILE

It is possible to customize the use of TASKING software using a daemon options file. This options file allows you to reserve licenses for specified users or groups of users, to restrict access to the TASKING toolchain, and to set software timeouts. The following table lists the keywords that are recognized at the start of a line of a daemon options file.

Keywords	Function
RESERVE	Ensure that TASKING software will always be available to one or more users or on one or more host computer systems.
INCLUDE	Specify a list of users who are allowed exclusive access to the TASKING software.
EXCLUDE	Specify a list of users who are not allowed to use the TASKING software.
GROUP	Specify a group of users for use in the other commands.
TIMEOUT	Allow licenses that are idle for a specified time to be returned to the free pool, for use by someone else.
NOLOG	Causes messages of the specified type to be filtered out of the daemon's log output.

Table A-1: Daemon options file keywords

In order to use the daemon options capability, you must create a daemon options file and list its pathname as the fourth field on the **DAEMON** line for the **Tasking** daemon in the license file. For example, if the daemon options were in file `/usr/local/flexlm/Tasking.opt` (UNIX), then you would modify the license file **DAEMON** line as follows:

```
DAEMON Tasking /usr/local/Tasking /usr/local/flexlm/Tasking.opt
```

A daemon options file consists of lines in the following format:

```
RESERVE      number feature {USER | HOST | DISPLAY | GROUP} name
INCLUDE      feature {USER | HOST | DISPLAY | GROUP} name
EXCLUDE      feature {USER | HOST | DISPLAY | GROUP} name
GROUP        name <list_of_users>
TIMEOUT      feature timeout_in_seconds
NOLOG        {IN | OUT | DENIED | QUEUED}
REPORTLOG    file
```

Lines beginning with the sharp character (#) are ignored, and can be used as comments. For example, the following options file would reserve one copy of feature SWxxxxxx-xx for user “pat”, three copies for user “lee”, and one copy for anyone on a computer with the hostname of “terry”; and would cause QUEUED messages to be omitted from the log file. In addition, user “joe” and group “pinheads” would not be allowed to use the feature SWxxxxxx-xx:

```
GROUP        pinheads moe larry curley
RESERVE 1    SWxxxxxx-xx USER pat
RESERVE 3    SWxxxxxx-xx USER lee
RESERVE 1    SWxxxxxx-xx HOST terry
EXCLUDE      SWxxxxxx-xx USER joe
EXCLUDE      SWxxxxxx-xx GROUP pinheads
NOLOG        QUEUED
```

3 LICENSE ADMINISTRATION TOOLS

The following utilities are provided to facilitate license management by your system administrator. In certain cases, execution access to a utility is restricted to users with root privileges. Complete descriptions of these utilities are provided at the end of this section.

lmcksum

Prints license checksums.

lmdiag (Windows only)

Diagnoses license checkout problems.

lmdown

Gracefully shuts down all license daemons (both **lmgrd** all vendor daemons, such as **Tasking**) on the license server.

lmgrd

The main daemon program for FLEXlm.

lmbostid

Reports the hostid of a system.

lmremove

Removes a single user's license for a specified feature.

lmreread

Causes the license daemon to reread the license file and start any new vendor daemons.

lmstat

Helps you monitor the status of all network licensing activities.

lmswitchr

Switches the report log file.

lmver

Reports the FLEXlm version of a library or binary file.

lmtools (*Windows only*)

This is a graphical Windows version of the license administration tools.

3.1 LMCKSUM

Name

lmcksum – print license checksums

Synopsis

lmcksum [**-c** *license_file*] [**-k**]

Description

The **lmcksum** program will perform a checksum of a license file. This is useful to verify data entry errors at your location. **lmcksum** will print a line-by-line checksum for the file as well as an overall file checksum.

The following fields participate in the checksum:

- hostid on the SERVER lines
- daemon name on the DAEMON lines
- feature name, version, daemon name, expiration date, # of licenses, encryption code, vendor string and hostid on the FEATURE lines
- daemon name and encryption code on FEATURESET lines

Options

-c *license_file*

Use the specified *license_file*. If no **-c** option is specified, **lmcksum** looks for the environment variable LM_LICENSE_FILE in order to find the license file to use. If that environment variable is not set, **lmcksum** looks for the file c:\flexlm\license.dat (Windows), or /usr/local/flexlm/licenses/license.dat (UNIX).

-k

Case-sensitive checksum. If this option is specified, **lmcksum** will compute the checksum using the exact case of the FEATURE's and FEATURESET's encryption code.

3.2 LMDIAG (Windows only)

Name

lmdiag – diagnose license checkout problems

Synopsis

lmdiag [**-c** *license_file*] [**-n**] [*feature*]

Description

lmdiag (Windows only) allows you to diagnose problems when you cannot check out a license.

If no *feature* is specified, **lmdiag** will operate on all features in the license file(s) in your path. **lmdiag** will first print information about the license, then attempt to check out each license. If the checkout succeeds, **lmdiag** will indicate this. If the checkout fails, **lmdiag** will give you the reason for the failure. If the checkout fails because **lmdiag** cannot connect to the license server, then you have the option of running "extended connection diagnostics".

These extended diagnostics attempt to connect to each port on the license server node, and can detect if the port number in the license file is incorrect. **lmdiag** will indicate each port number that is listening, and if it is an **lmgrd** process, **lmdiag** will indicate this as well. If **lmdiag** finds the vendor daemon for the feature being tested, then it will indicate the correct port number for the license file to correct the problem.

Parameters

feature Diagnose this feature only.

Options

-c *license_file*

Diagnose the specified *license_file*. If no **-c** option is specified, **lmdiag** looks for the environment variable `LM_LICENSE_FILE` in order to find the license file to use. If that environment variable is not set, **lmdiag** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

-n

Run in non-interactive mode; **lmdiag** will not prompt for any input in this mode. In this mode, extended connection diagnostics are not available.

3.3 **LMDOWN**

Name

lmdown – graceful shutdown of all license daemons

Synopsis

lmdown [**-c** *license_file*] [**-q**]

Description

The **lmdown** utility allows for the graceful shutdown of all license daemons (both **lmgrd** and all vendor daemons, such as **Tasking**) on all nodes. You may want to protect the execution of **lmdown**, since shutting down the servers causes users to lose their licenses. See the **-p** option in Section 3.4, **lmgrd**.

lmdown sends a message to every license daemon asking it to shut down. The license daemons write out their last messages to the log file, close the file, and exit. All licenses which have been given out by those daemons will be revoked, so that the next time a client program goes to verify his license, it will not be valid.

Options

-c *license_file*

Use the specified *license_file*. If no **-c** option is specified, **lmdown** looks for the environment variable **LM_LICENSE_FILE** in order to find the license file to use. If that environment variable is not set, **lmdown** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

-q

Quiet mode. If this switch is not specified, **lmdown** asks for confirmation before asking the license daemons to shut down. If this switch is specified, **lmdown** will not ask for confirmation.



lmgrd, lmstat, lmread

3.4 LMGRD

Name

lmgrd – flexible license manager daemon

Synopsis

lmgrd [**-c** *license_file*] [**-l** *logfile*] [**-2 -p**] [**-t** *timeout*] [**-s** *interval*]

Description

lmgrd is the main daemon program for the FLEXlm distributed license management system. When invoked, it looks for a license file containing all required information about vendors and features. On UNIX systems, it is strongly recommended that **lmgrd** be run as a non-privileged user (not root).

Options

-c *license_file*

Use the specified *license_file*. If no **-c** option is specified, **lmgrd** looks for the environment variable `LM_LICENSE_FILE` in order to find the license file to use. If that environment variable is not set, **lmgrd** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

-l *logfile*

Specifies the output log file to use. Instead of using the **-l** option you can use output redirection (**>** or **>>**) to specify the name of the output log file.

-2 -p

Restricts usage of **lmdown**, **lmreread**, and **lmremove** to a FLEXlm administrator who is by default root. If there is a UNIX group called "lmadmin" then use is restricted to only members of that group. If root is not a member of this group, then root does not have permission to use any of the above utilities.

-t *timeout*

Specifies the *timeout* interval, in seconds, during which the license daemon must complete its connection to other daemons if operating in multi-server mode. The default value is 10 seconds. A larger value may be desirable if the daemons are being run on busy systems or a very heavily loaded network.

-s *interval* Specifies the log file timestamp *interval*, in minutes. The default is 360 minutes. This means that every six hours **lmgrd** logs the time in the log file.



lmdown, lmstat

3.5 LMHOSTID

Name

lmhostid – report the hostid of a system

Synopsis

lmhostid

Description

lmhostid calls the FLEXlm version of `gethostid` and displays the results.

The output of **lmhostid** looks like this:

```
lmhostid - Copyright (C) 1989, 1999 Globetrotter Software, Inc.  
The FLEXlm host ID of this machine is "1200abcd"
```

Options

lmhostid has no command line options.

3.6 LMREMOVE

Name

lmremove – remove specific licenses and return them to license pool

Synopsis

lmremove [**-c** *license_file*] *feature user host* [*display*]

Description

The **lmremove** utility allows the system administrator to remove a single user's license for a specified feature. This could be required in the case where the licensed user was running the software on a node that subsequently crashed. This situation will sometimes cause the license to remain unusable. **lmremove** will allow the license to return to the pool of available licenses.

lmremove will remove all instances of “user” on node “host” on display “display” from usage of “feature”. If the optional **-c file** is specified, the indicated file will be used as the license file. Since removing a user's license can be disruptive, execution of **lmremove** is restricted to users with root privileges.

Options

-c *license_file*

Use the specified *license_file*. If no **-c** option is specified, **lmremove** looks for the environment variable `LM_LICENSE_FILE` in order to find the license file to use. If that environment variable is not set, **lmremove** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).



lmstat

3.7 LMREREAD

Name

lmreread – tells the license daemon to reread the license file

Synopsis

lmreread [**-c** *license_file*]

Description

lmreread allows the system administrator to tell the license daemon to reread the license file. This can be useful if the data in the license file has changed; the new data can be loaded into the license daemon without shutting down and restarting it.

The license administrator may want to protect the execution of **lmreread**. See the **-p** option in Section 3.4, *lmgrd* for details about securing access to **lmreread**.

lmreread uses the license file from the command line (or the default file, if none specified) only to find the license daemon to send it the command to reread the license file. The license daemon will always reread the file that it loaded from the original path. If you need to change the path to the license file read by the license daemon, then you must shut down the daemon and restart it with that new license file path.

You cannot use **lmreread** if the *SERVER* node names or port numbers have been changed in the license file. In this case, you must shut down the daemon and restart it in order for those changes to take effect.

lmreread does not change any option information specified in an options file. If the new license file specifies a different options file, that information is ignored. If you need to reread the options file, you must shut down (**lmdown**) the daemon and restart it.

Options

-c *license_file*

Use the specified *license_file*. If no **-c** option is specified, **lmreread** looks for the environment variable *LM_LICENSE_FILE* in order to find the license file to use. If that environment variable is not set, **lmreread** looks for the file *license.dat* in the default location.



lmdown

3.8 LMSTAT

Name

lmstat – report status on license manager daemons and feature usage

Synopsis

```
lmstat [ -a ] [ -A ] [ -c license_file ] [ -f feature ]
      [ -l regular_expression ] [ -s server ] [ -S daemon ] [ -t timeout ]
```

Description

License administration is simplified by the **lmstat** utility. **lmstat** allows you to instantly monitor the status of all network licensing activities.

lmstat allows a system administrator to monitor license management operations including:

- Which daemons are running
- Users of individual features
- Users of features served by a specific DAEMON

Options

-a Display all information.

-A List all active licenses.

-c *license_file*

Use the specified *license_file*. If no **-c** option is specified, **lmstat** looks for the environment variable LM_LICENSE_FILE in order to find the license file to use. If that environment variable is not set, **lmstat** looks for the file c:\flexlm\license.dat (Windows), or /usr/local/flexlm/licenses/license.dat (UNIX).

-f *feature* List all users of the specified *feature*(s).

-l *regular_expression*

List all users of the features matching the given *regular_expression*.

-s *server* Display the status of the specified *server* node(s).

-S *daemon* List all users of the specified *daemon*'s features.

- t *timeout*** Specifies the amount of time, in seconds, **lmstat** waits to establish contact with the servers. The default value is 10 seconds. A larger value may be desirable if the daemons are being run on busy systems or a very heavily loaded network.



lmgrd

3.9 LMSWITCHR (Windows only)

Name

lmswitchr – switch the report log file

Synopsis

lmswitchr [**-c** *license_file*] *feature new-file*

or:

lmswitchr [**-c** *license_file*] *vendor new-file*

Description

lmswitchr (Windows only) switches the report writer (REPORTLOG) log file. It will also start a new REPORTLOG file if one does not already exist.

Parameters

<i>feature</i>	Any feature this daemon supports.
<i>vendor</i>	The name of the vendor daemon (such as Tasking).
<i>new-file</i>	New file path.

Options

-c *license_file* Use the specified *license_file*. If no **-c** option is specified, **lmswitchr** looks for the environment variable LM_LICENSE_FILE in order to find the license file to use. If that environment variable is not set, **lmswitchr** looks for the file `c:\flexlm\license.dat` (Windows), or `/usr/local/flexlm/licenses/license.dat` (UNIX).

3.10 LMVER

Name

lmver – report the FLEXlm version of a library or binary file

Synopsis

lmver *filename*

Description

The **lmver** utility reports the FLEXlm version of a library or binary file.

Alternatively, on UNIX systems, you can use the following commands to get the FLEXlm version of a binary:

strings *file* | grep Copy

Parameters

filename Name of the executable of the product.

3.11 LICENSE ADMINISTRATION TOOLS FOR WINDOWS

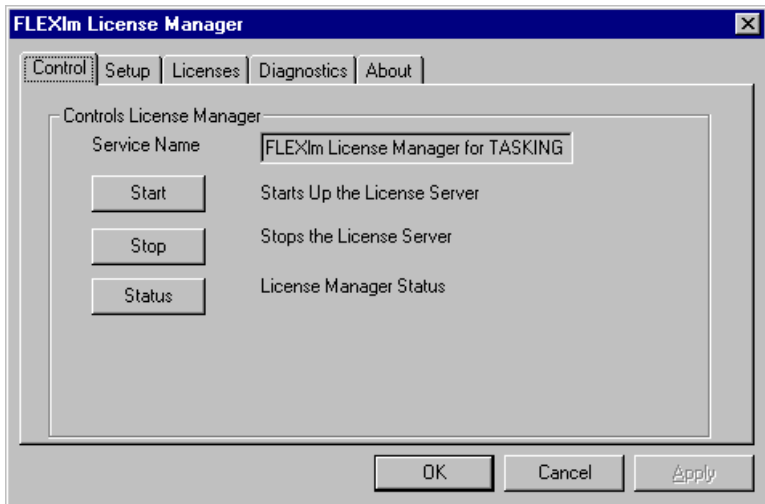
3.11.1 LMTOOLS FOR WINDOWS

For the 32 Bit Windows Platforms, an **lmtools.exe** Windows program is provided. It has the same functionality as listed in the previous sections but is graphically-oriented. Simply run the program (Start | Programs | TASKING FLEXlm | FLEXlm Tools) and choose a button for the functionality required. Refer to the previous sections for information about the options of each feature. The command line interface is replaced by pop-up dialogs that can be filled out. The central EDIT field is where the license file path is placed. This will be used for all other functions and replaces the "**-c** *license_file*" argument in the other utilities.

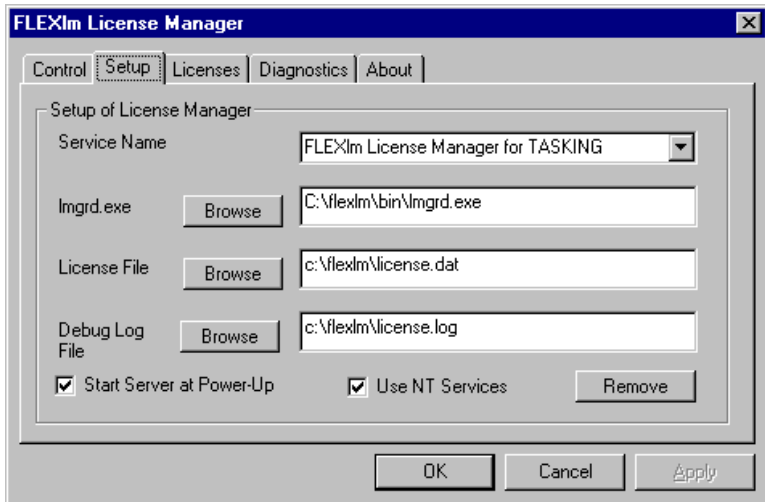
The HOSTID button displays the hostid's for the computer on which the program is running. The TIME button prints out the system's internal time settings, intended to diagnose any time zone problems. The TCP Settings button is intended to fix a bug in the Microsoft TCP protocol stack which has a symptom of very slow connections to computers. After pressing this button, the system will need to be rebooted for the settings to become effective.

3.11.2 FLEXLM LICENSE MANAGER FOR WINDOWS

lmgrd.exe can be run manually or using the graphical Windows tool. You can start this tool from the FLEXlm program folder. Click on Start | Programs | TASKING FLEXlm | FLEXlm Tools



From the Control tab you can start, stop, and check the status of your license server. Select the Setup tab to enter information about your license server.



Select the **Control** tab and click the **Start** button to start your license server. **lmgrd.exe** will be launched as a background application with the license file and debug log file locations passed as parameters.

If you want **lmgrd.exe** to start automatically on NT, select the **Use NT Services** check box and **lmgrd.exe** will be installed as an NT service. Next, select the **Start Server at Power-UP** check box.

The **Licenses** tab provides information about the license file and the **Advanced** tab allows you to perform diagnostics and check versions.

4 THE DAEMON LOG FILE

The FLEXlm daemons all generate log files containing messages in the following format:

mm/dd hh:mm (DAEMON name) message

Where:

mm/dd hh:mm Is the month/day hour:minute that the message was logged.

DAEMON name Either “license daemon” or the string from the DAEMON line that describes your daemon.

In the case where a single copy of the daemon cannot handle all of the requested licenses, an optional “_” followed by a number indicates that this message comes from a forked daemon.

message The text of the message.

The log files can be used to:

- Inform you when it may be necessary to update your application software licensing arrangement.
- Diagnose configuration problems.
- Diagnose daemon software errors.

The messages are grouped below into the above three categories, with each message followed by a brief description of its meaning.

4.1 INFORMATIONAL MESSAGES

Connected to node

This daemon is connected to its peer on node *node*.

CONNECTED, master is name

The license daemons log this message when a quorum is up and everyone has selected a master.

DEMO mode supports only one SERVER host!

An attempt was made to configure a demo version of the software for more than one server host.

DENIED: N feature to user (mm/dd/yy hh:mm)

user was denied access to *N* licenses of *feature*. This message may indicate a need to purchase more licenses.

EXITING DUE TO SIGNAL mm

EXITING with code mm

All daemons list the reason that the daemon has exited.

EXPIRED: feature

feature has passed its expiration date.

IN: feature by user (N licenses) (used: d:hh:mm:ss) (mm/dd/yy hh:mm)

user has checked back in *N* licenses of *feature* at *mm/dd/yy hh:mm*.

IN server died: feature by user (number licenses) (used: d:hh:mm:ss) (mm/dd/yy hh:mm)

user has checked in *N* licenses by virtue of the fact that his server died.

License Manager server started

The license daemon was started.

Lost connection to host

A daemon can no longer communicate with its peer on node *host*, which can cause the clients to have to reconnect, or cause the number of daemons to go below the minimum number, in which case clients may start exiting. If the license daemons lose the connection to the master, they will kill all the vendor daemons; vendor daemons will shut themselves down.

Lost quorum

The daemon lost quorum, so will process only connection requests from other daemons.

MASTER SERVER died due to signal mm

The license daemon received fatal signal *mm*.

MULTIPLE xxx servers running. Please kill, and restart license daemon

The license daemon has detected that multiple copies of vendor daemon *xxx* are running. The user should kill all *xxx* daemon processes and re-start the license daemon.

OUT: feature by user (N licenses) (mm/dd/yy hh:mm)

user has checked out *N* licenses of *feature* at *mm/dd/yy hh:mm*

Removing clients of children

The top-level daemon logs this message when one of the child daemons dies.

RESERVE feature for HOST name***RESERVE feature for USER name***

A license of *feature* is reserved for either user *name* or host *name*.

REStarted xxx (internet port mm)

Vendor daemon *xxx* was restarted at internet port *mm*.

Retrying socket bind (address in use)

The license servers try to bind their sockets for approximately 6 minutes if they detect *address in use* errors.

Selected (EXISTING) master node

This license daemon has selected an existing master (node) as the master.

SERVER shutdown requested

A daemon was requested to shut down via a user-generated kill command.

[NEW] Server started for: feature-list

A (possibly new) server was started for the features listed.

Shutting down xxx

The license daemon is shutting down the vendor daemon *xxx*.

SIGCHLD received. Killing child servers

A vendor daemon logs this message when a shutdown was requested by the license daemon.

Started name

The license daemon logs this message whenever it starts a new vendor daemon.

Trying connection to node

The daemon is attempting a connection to *node*.

4.2 CONFIGURATION PROBLEM MESSAGES

hostname: Not a valid server host, exiting

This daemon was run on an invalid hostname.

hostname: Wrong hostid, exiting

The hostid is wrong for *hostname*.

BAD CODE for feature-name

The specified feature name has a bad encryption code.

CANNOT OPEN options file “file”

The options file specified in the license file could not be opened.

Couldn't find a master

The daemons could not agree on a master.

license daemon: lost all connections

This message is logged when all the connections to a server are lost, which often indicates a network problem.

lost lock, exiting

Error closing lock file

Unable to re-open lock file

The vendor daemon has a problem with its lock file, usually because of an attempt to run more than one copy of the daemon on a single node. Locate the other daemon that is running via a **ps** command, and kill it with **kill -9**.

NO DAEMON line for daemon

The license file does not contain a DAEMON line for *daemon*.

No “license” service found

The TCP *license* service did not exist in `/etc/services`.

No license data for “feat”, feature unsupported

There is no feature line for *feat* in the license file.

No features to serve!

A vendor daemon found no features to serve. This could be caused by bad data in the license file.

UNSUPPORTED FEATURE request: feature by user

The *user* has requested a feature that this vendor daemon does not support. This can happen for a number of reasons: the license file is bad, the feature has expired, or the daemon is accessing the wrong license file.

Unknown host: hostname

The hostname specified on a `SERVER` line in the license file does not exist in the network database (probably `/etc/hosts`).

lm_server: lost all connections

This message is logged when all the connections to a server are lost. This probably indicates a network problem.

NO DAEMON lines, exiting

The license daemon logs this message if there are no `DAEMON` lines in the license file. Since there are no vendor daemons to start, there is nothing to do.

NO DAEMON line for name

A vendor daemon logs this error if it cannot find its own `DAEMON` name in the license file.

4.3 DAEMON SOFTWARE ERROR MESSAGES

accept: message

An error was detected in the accept system call.

ATTEMPT TO START VENDOR DAEMON xxx with NO MASTER

A vendor daemon was started with no master selected. This is an internal consistency error in the daemons.

BAD PID message from mm: pid: xxx (msg)

A top-level vendor daemon received an invalid PID message from one of its children (daemon number xxx).

BAD SCONNECT message: (message)

An invalid “server connect” message was received.

Cannot create pipes for server communication

The pipe call failed.

Can't allocate server table space

A malloc error. Check swap space.

Connection to node TIMED OUT

The daemon could not connect to *node*.

Error sending PID to master server

The vendor server could not send its PID to the top-level server in the hierarchy.

Illegal connection request to DAEMON

A connection request was made to DAEMON, but this vendor daemon is not DAEMON.

Illegal server connection request

A connection request came in from another server without a DAEMON name.

KILL of child failed, errno = mm

A daemon could not kill its child.

No internet port number specified

A vendor daemon was started without an internet port.

Not enough descriptors to re-create pipes

The “top-level” daemon detected one of its sub-daemon’s death. In trying to restart the chain of sub-daemons, it was unable to get the file descriptors to set up the pipes to communicate. This is a fatal error, and the daemons must be re-started.

read: error message

An error in a read system call was detected.

recycle_control BUT WE DIDN'T HAVE CONTROL

The hierarchy of vendor daemons has become confused over who holds the control token. This is an internal error.

return_reserved: can't find feature listhead

When a daemon is returning a reservation to the “free reservation” list, it could not find the listhead of features.

select: message

An error in a select system call was detected.

Server exiting

The server is exiting. This is normally due to an error.

SHELLO for wrong DAEMON

This vendor daemon was sent a “server hello” message that was destined for a different DAEMON.

Unsolicited msg from parent!

Normally, the top-level vendor daemon sends no unsolicited messages. If one arrives, this message is logged. This is a bug.

***WARNING: CORRUPTED options list (o->next == 0)
Options list TERMINATED at bad entry***

An internal inconsistency was detected in the daemon’s option list.

5 FLEXLM LICENSE ERRORS

FLEXlm license error, encryption code in license file is inconsistent

Check the contents of the license file using the license data sheet for the product. Correct the license file and run the **lmreread** command.

However, do not change the last (fourth) field of a SERVER line in the license file. This cannot have any effect on the error message but changing it will cause other problems.

license file does not support this version

If this is a first time install then follow the procedure for the error message:

```
FLEXlm license error, encryption code in license file is
inconsistent
```

because there may be a typo in the fourth field of a FEATURE line of your license file. In all other cases you need a new license because the current license is for an older version of the product.

Replace the FEATURE line for the old version of the product with a FEATURE line for the new version (it can be found on the new license data sheet). Run the **lmreread** command afterwards. You can have only one version of a feature (previous versions of the product will continue to work).

FLEXlm license error, cannot find license file

Make sure the license file exists. If the pathname printed on the line after the error message is incorrect, correct this by setting the `LM_LICENSE_FILE` environment variable to the full pathname of the license file.

FLEXlm license error, cannot read license file

Every user needs to have read access on the license file and at least execute access on every directory component in the pathname of the license file. Write access is never needed. Read access on directories is recommended.

FLEXlm license error, no such feature exists

Check the license file. There should be a line starting with:

```
FEATURE SWiiiiiii-jj
```

where "iiiiii" is a six digit software code and "jj" is a two digit host code for identifying a compatible host architecture. During product installations the product code is shown, e.g. SW008002, SW019002. The number in the software code is the same as the number in the product code except that the first number may contain an extra leading zero (it must be six digits long).

The line after the license error message describes the expected feature format and includes the host code.

Correct the license file using the license data sheet for the product and run the **lmreread** command. There is one catch: do not add extra SERVER lines or change existing SERVER lines in the license file.

FLEXlm license error, license server does not support this feature

If the LM_LICENSE_FILE variable has been set to the format *number@host* then see first the solution for the message:

```
FLEXlm license error, no such feature exists
```

Run the **lmreread** program to inform the license server about a changed license data file. If **lmreread** succeeds informing the license server but the error message persists, there are basically three possibilities:

1. The license key is incorrect. If this is the case then there must be an error message in the log file of **lmgrd**. Correct the key using the license data sheet for the product. Finally rerun **lmreread**. The log file of **lmgrd** is usually specified to **lmgrd** at startup with the **-l** option or with **>**.
2. Your network has more than one FLEXlm license server daemon and the default license file location for **lmreread** differs from the default assumed by the program. Also, there must be more than one license file. Try one of the following solutions on the same host which produced the error message:

- type:

```
lmreread -c /usr/local/flexlm/licenses/license.dat
```

- set LM_LICENSE_FILE to the license file location and retry the **lmreread** command.
- use the **lmreread** program supplied with the product SW000098, Flexible License Manager. SW000098 is bundled with all TASKING products.

3. There is a protocol version mismatch between **lmgrd** and the daemon with the name "Tasking" (the vendor daemon according to FLEXlm terminology) or there is some other internal error. These errors are always written to the log file of **lmgrd**. The solution is to upgrade the **lmgrd** daemon to the one supplied in SW000098, the bundled Flexible License Manager product.

On the other hand, if **lmreread** complains about not being able to connect to the license server then follow the procedure described in the next section for the error message "Cannot read license file data from server". The only difference with the current situation is that not the product but a license management utility shows a connect problem.

FLEXlm license error, Cannot read license file data from server

This indicates that the program could not connect to the license server daemon. This can have a number of causes. If the program did not immediately print the error message but waited for about 30 seconds (this can vary) then probably the license server host is down or unreachable. If the program responded immediately with the error message then check the following if the `LM_LICENSE_FILE` variable has been set to the format *number@host*:

- is the number correct? It should match the fourth field of a `SERVER` line in the license file on the license server host. Also, the host name on that `SERVER` line should be the same as the host name set in the `LM_LICENSE_FILE` variable. Correct `LM_LICENSE_FILE` if necessary.

In any case one should verify if the license server daemon is running. Type the following command on the host where the license server daemon (**lmgrd**) is supposed to run.

On SunOS 4.x:

```
ps wwax | grep lmgrd | grep -v grep
```

On HP-UX or SunOS 5.x (Solaris 2.x):

```
ps -ef | grep lmgrd | grep -v grep
```

If the command does not produce any output then the license server daemon is not running. See below for an example how to start **lmgrd**.

Make sure that both license server daemon (**lmgrd**) and the program are using the same license data. All TASKING products use the license file `/usr/local/flexlm/licenses/license.dat` unless overruled by the environment variable `LM_LICENSE_FILE`. However, not all existing **lmgrd** daemons may use the same default. In case of doubt, specify the license file pathname with the `-c` option when starting the license server daemon. For example:

```
lmgrd -c /usr/local/flexlm/licenses/license.dat \  
-l /usr/local/flexlm/licenses/license.log &
```

and set the `LM_LICENSE_FILE` environment variable to the `license.dat` pathname mentioned with the `-c` option of **lmgrd** before running any license based program (including **lmreread**, **lmstat**, **lmdown**). If **lmgrd** and the program run on different hosts, transparent access to the license file is assumed in the situation described above (e.g. NFS). If this is not the case, make a local copy of the license file (not recommended) or set `LM_LICENSE_FILE` to the form *number@host*, as described earlier.

If none of the above seems to apply (i.e. **lmgrd** was already running and `LM_LICENSE_FILE` has been set correctly) then it is very likely that there is a TCP port mismatch. The fourth field of a `SERVER` line in the license file specifies a TCP port number. That number can be changed without affecting any license. However, it must never be changed while the license server daemon is running. If it has been changed, change it back to the original value. If you do not know the original number anymore, restart the license server daemon after typing the following command on the license server host:

```
kill PID
```

where `PID` is the process id of **lmgrd**.

6 FREQUENTLY ASKED QUESTIONS (FAQS)

6.1 LICENSE FILE QUESTIONS

I've received FLEXlm license files from 2 different companies. Do I have to combine them?

You don't have to combine license files. Each license file that has any 'counted' lines (the 'number of licenses' field is >0) requires a server. It's perfectly OK to have any number of separate license files, with different **lmgrd** server processes supporting each file. Moreover, since **lmgrd** is a lightweight process, for sites without system administrators, this is often the simplest (and therefore recommended) way to proceed. With v6+ **lmgrd/lmdown/lmreread**, you can stop/reread/restart a single vendor daemon (of any FLEXlm version). This makes combining licenses more attractive than previously. Also, if the application is v6+, using 'dir/*.lic' for license file management behaves like combining licenses without physically combining them.

When is it recommended to combine license files?

Many system administrators, especially for larger sites, prefer to combine license files to ease administration of FLEXlm licenses. It's purely a matter of preference.

Does FLEXlm handle dates in the year 2000 and beyond?

Yes. The FLEXlm date format uses a 4-digit year. Dates in the 20th century (19xx) can be abbreviated to the last 2 digits of the year (xx), and use of this feature is quite widespread. Dates in the year 2000 and beyond must specify all 4 year digits.

6.2 FLEXLM VERSION

Which FLEXlm versions does TASKING deliver?

For Windows we deliver FLEXlm v6.1 and for UNIX we deliver v2.4.

I have products from several companies at various FLEXlm version levels. Do I have to worry about how these versions work together?

If you're not combining license files from different vendors, the simplest thing to do is make sure you use the tools (especially **lmgrd**) that are shipped by each vendor.

lmgrd will always correctly support older versions of vendor daemons and applications, so it's **always** safe to use the latest version of **lmgrd** and the other FLEXlm utilities. If you've combined license files from 2 vendors, you **must** use the latest version of **lmgrd**.

If you've received 2 versions of a product from the same vendor, you must use the latest vendor daemon they sent you. An older vendor daemon with a newer client will cause communication errors.

Please ignore letters appended to FLEXlm versions, i.e., v2.4d. The appended letter indicates a patch, and does NOT indicate any compatibility differences. In particular, some elements of FLEXlm didn't require certain patches, so a 2.4 **lmgrd** will work successfully with a 2.4b vendor daemon.

I've received a new copy of a product from a vendor, and it uses a new version of FLEXlm. Is my old license file still valid?

Yes. Older FLEXlm license files are always valid with newer versions of FLEXlm.

6.3 WINDOWS QUESTIONS

What Windows Host Platforms can be used as a server for Floating Licenses?

The system being used as the server (where the FLEXlm License Manager is running) for Floating licenses, must be Windows NT. The FLEXlm License Manager does not run properly with Windows 95/98.

Why do I need to include NWlink IPX/SPX on NT?

This is necessary for either obtaining the Ethernet card address, or to provide connectivity with a Netware License server.

6.4 TASKING QUESTIONS

How will the TASKING licensing/pricing model change with License Management (FLEXlm)?

TASKING will now offer the following types of licenses so you can purchase licenses based upon usage:

License	Description	Pricing
Node Locked	This license can only be used on a specific system. It cannot be moved to another system.	The pricing for this license will be the current product pricing.
Floating	This license requires a network (license server and a TCP/IP (or IPX/SPX) connection between clients and server) and can be used on any host system (using the same operating system) in the network.	The pricing for this license will be 50% higher than the node locked license.

How does FLEXlm affect future product ordering?

For all licenses, node locked or floating, you must provide information that is used to create a license key. For node locked licenses we must have the HOST ID. Floating licenses require the HOST ID and HOST NAME. The HOST ID is a unique identification of the machine, which is based upon different hardware depending upon host platform. The HOST NAME is the network name of the machine.



TASKING Logistics CANNOT ship ANY orders that do not include the HOST ID and/or HOST NAME information.

What if I do not know the information needed for the license key?

We have a software utility (**tkhostid.exe**) which will obtain and display the HOST ID so a customer can easily obtain this information. This utility is available from our web site, placed on all product CDs (which support FLEXlm), and from technical support. If you have already installed FLEXlm, you can also use **lmhostid**.

- In the case of a *Node locked license*, it is important that the customer runs this utility on the exact machine he intends to run the TASKING tools on.

- In the case of a *Floating License*, the **tkhostid.exe** (or **lmhostid**) utility should be run on the machine on which the FLEXlm license manager will be installed, e.g. the server. The HOST NAME information can be obtained from within the Windows Control Panel. Select "Network", click on "Identification", look for "Computer name".

How will the "locking" mechanism work?

- For node locked licenses, FLEXlm will first search for an ethernet card. If one exists, it will lock onto the number of the ethernet card. If an ethernet card does not exist, FLEXlm will lock onto the hard disk serial number.
- For floating licenses, the ethernet card number will be used.

What happens if I try to move my node locked license to another system?

The software will not run.

What does linger-time for floating licenses mean?

When the TASKING product starts to run, it will try to obtain a license from the license server. The license server keeps track of the number of licenses already issued, and grants or denies the request. When the software has finished running, the license is kept by the license server for a period of time known as the "linger-time". If the same user requests the TASKING product again within the linger-time, he is granted the license again. If another user requests a license during the linger-time, his request is denied until the linger-time has finished.

What is the length of the linger-time for floating licenses?

The length of the linger-time for both the PC and UNIX floating licenses is 5 minutes.

Can the linger-time be changed?

Yes. A customer can change the linger-time to be larger (but not shorter) than the time specified by TASKING.

What happens if my system crashes or I upgrade to a new system?

You will need to contact Technical Support for temporary license keys due to a system crash or to move from one system to another system. You will then need to work with your local sales representative to obtain a permanent new license key.

6.5 USING FLEXLM FOR FLOATING LICENSES

Does FLEXlm work across the internet?

Yes. A server on the internet will serve licenses to anyone else on the internet. This can be limited with the 'INTERNET=' attribute on the FEATURE line, which limits access to a range of internet addresses. You can also use the INCLUDE and EXCLUDE options in the daemon option file to allow (or deny) access to clients running on a range of internet addresses.

Does FLEXlm work with Internet firewalls?

Many firewalls require that port numbers be specified to the firewall. FLEXlm v5 **lmgrd** supports this.

If my client dies, does the server free the license?

Yes, unless the client's whole system crashes. Assuming communications is TCP, the license is automatically freed immediately. If communications are UDP, then the license is freed after the UDP timeout, which is set by each vendor, but defaults to 45 minutes. UDP communications is normally only set by the end-user, so TCP should be assumed. If the whole system crashes, then the license is not freed, and you should use '**lmremove**' to free the license.

What happens when the license server dies?

FLEXlm applications send periodic heartbeats to the server to discover if it has died. What happens when the server dies is then up to the application. Some will simply continue periodically attempting to re-checkout the license when the server comes back up. Some will attempt to re-checkout a license a few times, and then, presumably with some warning, exit. Some GUI applications will present pop-ups to the user periodically letting them know the server is down and needs to be re-started.

How do you tell if a port is already in use?

99.44% of the time, if it's in use, it's because **lmgrd** is already running on the port – or was recently killed, and the port isn't freed yet. Assuming this is not the case, then use '**telnet host port**' – if it says "*can't connect*", it's a free port.

Does FLEXlm require root permissions?

No. There is no part of FLEXlm, **lmgrd**, vendor daemon or application, that requires root permissions. In fact, it is strongly recommended that you do not run the license server (**lmgrd**) as root, since root processes can introduce security risks.

If **lmgrd** must be started from the root user (for example, in a system boot script), we recommend that you use the '**su**' command to run **lmgrd** as a non-privileged user:

```
su username -c"/path/lmgrd -c /path/license.dat \  
-l /path/log"
```

where *username* is a non-privileged user, and *path* is the correct paths to **lmgrd**, *license.dat* and debug log file. You will have to ensure that the vendor daemons listed in */path-to-license/license.dat* have execute permissions for *username*. The paths to all the vendor daemons in the license file are listed on each DAEMON line.

Is it ok to run lmgrd as 'root' (UNIX only)?

It is not prudent to run any command, particularly a daemon, as root on UNIX, as it may pose a security risk to the Operating System. Therefore, we recommend that **lmgrd** be run as a non-privileged user (not 'root'). If you are starting **lmgrd** from a boot script, we recommend that you use

```
su username -c"umask 022; /path/lmgrd \  
-c /path/license.dat -l /path/log"
```

to run **lmgrd** as a non-privileged user.

Does FLEXlm licensing impose a heavy load on the network?

No, but partly this depends on the application, and end-user's use. A typical checkout request requires 5 messages and responses between client and server, and each message is < 150 bytes.

When a server is not receiving requests, it requires virtually no CPU time. When an application, or **lmstat**, requests the list of current users, this can significantly increase the amount of networking FLEXlm uses, depending on the number of current users. Also, prior to FLEXlm v5, use of 'port@host' can increase network load, since the license file is down-loaded from the server to the client. 'port@host' should be, if possible, limited to small license files (say < 50 features). In v5, 'port@host' actually improves performance.

Does FLEXlm work with NFS?

Yes. FLEXlm has no direct interaction with NFS. FLEXlm uses an NFS-mounted file like any other application.

Does FLEXlm work with ATM, ISDN, Token-Ring, etc.?

In general, these have no impact on FLEXlm. FLEXlm requires TCP/IP or SPX (Novell Netware). So long as TCP/IP works, FLEXlm will work.

Does FLEXlm work with subnets, fully-qualified names, multiple domains, etc.?

Yes, although this behavior was improved in v3.0, and v6.0. When a license server and a client are located in different domains, fully-qualified host names have to be used. A fully-qualified hostname is of the form:

node.domain

where *node* is the local hostname (usually returned by the '**hostname**' command or '**uname -n**') *domain* is the internet domain name, e.g. 'globes.com'.

To ensure success with FLEXlm across domains, do the following:

1. Make the sure the fully-qualified hostname is the name on the SERVER line of the license file.
2. Make sure ALL client nodes, as well as the server node, are able to 'telnet' to that fully-qualified hostname. For example, if the host is locally called 'speedy', and the domain name is 'corp.com', local systems will be able to logon to speedy via 'telnet speedy'. But very often, 'telnet speedy.corp.com' will fail, locally.
Note that this telnet command will always succeed on hosts in other domains (assuming everything is configured correctly), since the network will resolve speedy.corp.com automatically.
3. Finally, there must be an 'alias' for speedy so it's also known locally as speedy.corp.com. This alias is added to the `/etc/hosts` file, or if NIS/Yellow Pages are being used, then it will have to be added to the NIS database. This requirement goes away in version 3.0 of FLEXlm.

If all components (application, **lmgrd** and vendor daemon) are v6.0 or higher, no aliases are required; the only requirement is that the fully-qualified domain name, or IP-address, is used as a hostname on the SERVER, or as a hostname in `LM_LICENSE_FILE port@host, or @host`.

Does FLEXlm work with NIS and DNS?

Yes. However, some sites have broken NIS or DNS, which will cause FLEXlm to fail. In v5 of FLEXlm, NIS and DNS can be avoided to solve this problem. In particular, sometimes DNS is configured for a server that's not current available (e.g., a dial-up connection from a PC). Again, if DNS is configured, but the server is not available, FLEXlm will fail.

In addition, some systems, particularly Sun, SGI, HP, require that applications be linked dynamically to support NIS or DNS. If a vendor links statically, this can cause the application to fail at a site that uses NIS or DNS. In these situations, the vendor will have to relink, or recompile with v5 FLEXlm. Vendors are strongly encouraged to use dynamic libraries for libc and networking libraries, since this tends to improve quality in general, as well as making NIS/DNS work.

On PCs, if a checkout seems to take 3 minutes and then fails, this is usually because the system is configured for a dial-up DNS server which is not currently available. The solution here is to turn off DNS.

Finally, hostnames must NOT have periods in the name. These are not legal hostnames, although PCs will allow you to enter them, and they will not work with DNS.

We're using FLEXlm over a wide-area network. What can we do to improve performance?

FLEXlm network traffic should be minimized. With the most common uses of FLEXlm, traffic is negligible. In particular, checkout, checkin and heartbeats use very little networking traffic. There are two items, however, which can send considerably more data and should be avoided or used sparingly:

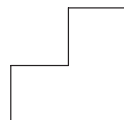
- **'lmstat -a'** should be used sparingly. **'lmstat -a'** should not be used more than, say, once every 15 minutes, and should be particularly avoided when there's a lot of features, or concurrent users, and therefore a lot of data to transmit; say, more than 20 concurrent users or features.
- Prior to FLEXlm v5, the 'port@host' mode of the LM_LICENSE_FILE environment variable should be avoided, especially when the license file has many features, or there are a lot of license files included in LM_LICENSE_FILE. The license file information is sent via the network, and can place a heavy load. Failures due to 'port@host' will generate the error LM_SERVNOREADLIC (-61).

INDEX

INDEX



TASKING



INDEX

B

bin directory, 3-4
 build, viewing results, 3-19
 Building an executable, 3-21

C

compile, 3-19
 configuration
 EDE directories, 2-5
 UNIX, 2-7
 creating a makefile, 3-16
 customer support, 1-11

D

data addressing, 3-42-3-46
 derivatives, 3-8
 directories, setting, 2-5, 2-7
 documentation, 1-3-1-14

E

EDE
 build an application, 3-19
 create a project, 3-14
 create a project space, 3-13
 Invoking tools from, 3-10
 rebuild an application, 3-20
 specify development tool options,
 3-17
 environment variable,
 LM_LICENSE_FILE, 2-16, A-6
 environment variables, 2-7
 LIB, 2-7
 INCLUDE, 2-7
 INCLUDE, 2-7

LIB, 2-7

LM_LICENSE_FILE, 2-7

PATH, 2-7

TMP, 2-7

errors, FLEXlm license, A-33
 examples, directory, 3-7

F

FAQ, FLEXlm, A-37
 Flexible License Manager, A-1
 FLEXlm, A-1
 daemon log file, A-25
 daemon options file, A-7
 FAQ, A-37
 frequently asked questions, A-37
 license administration tools, A-8
 for Windows, A-22
 license errors, A-33
 floating license, 2-10
 formatter, 1-8-1-12

G

global symbol mapper, 1-9-1-12
 gsmmap. *See* global symbol mapper

H

help, on-line, 1-12-1-14
 hostid, determining, 2-17
 hostname, determining, 2-17

I

include files, setting search directories,
 2-5, 2-7

installation
 licensing, 2-10
 UNIX, 2-4
 Windows 95/98/XP/NT/2000, 2-3
 Invoking tools, 3-10
 Invoking tools from command line,
 3-20
 Invoking tools from EDE, 3-10

L

librarian, 1-9-1-12
 libraries, setting search directories, 2-6,
 2-7
 license
 floating, 2-10
 node-locked, 2-10
 obtaining, 2-10
 license file
 default location, A-6
 location, 2-16
 setting search directory, 2-7
 licensing, 2-10
 linking C and assembly, 3-50-3-54
 Linking Locator, Rom Processing, 1-8
 LM_LICENSE_FILE, 2-16, A-6
 lmcksum, A-10
 lmdiag, A-11
 lmdown, A-12
 lmgrd, A-13
 lmhostid, A-15
 lmremove, A-16
 lmreread, A-17
 lmstat, A-18
 lmswitchr, A-20
 lmver, A-21

M

makefile
 automatic creation of, 3-16

updating, 3-16
 microprocessor family, 1-10

N

node-locked license, 2-10

O

on-line help, 1-12-1-14

P

Path, Setting, 3-3
 project, 3-10
 add new files, 3-16
 create, 3-14
 project file, 3-10
 project space, 3-10
 create, 3-13
 project space file, 3-10

R

RAM, 1-8
 rcopy, 1-8
 ROM processor, and the linking
 locator, 1-8
 run-time libraries, directory, 3-6

S

Setting , path, 3-3
 software installation
 UNIX, 2-4
 Windows 95/98/XP/NT/2000, 2-3
 support, customer, 1-11

symbol list utility, 1-9-1-12
symlist. *See* symbol list utility
system building concepts, 3-41-3-54
system initialization, 3-41-3-46

T

temporary files, setting directory, 2-7

toolchain, 1-5
Tools, invoking, 3-10

U

updating makefile, 3-16

