WORK BENCH

PICMICRO™ EMBEDDED WORKBENCH

Interface Guide

First edition: October 1998

Part no: EWPIC-1

WELCOME

Welcome to the IAR Embedded Workbench™, a powerful integrated Windows environment for developing applications in C or assembler for the PICmicro™ family of microcontrollers.

Before reading this guide refer to the *QuickStart Card*, or the chapter *Installation and documentation*, for information about installing the IAR Embedded Workbench, and an overview of the IAR Embedded Workbench documentation.

This guide explains how to configure and run the IAR Systems development tools from the IAR Embedded Workbench interface. It also includes complete reference information about the IAR Embedded Workbench commands and dialog boxes, and the IAR Embedded Workbench editor.

Refer to the *PICmicro™ C Compiler Programming Guide* and *PICmicro™ Assembler, Linker, and Librarian Programming Guide* for detailed information about working with the individual development tools incorporated in the Embedded Workbench.

If you are using the IAR C-SPY® Debugger refer to the *PICmicro™ C-SPY® User Guide*, for information about debugging with C-SPY.

ABOUT THIS GUIDE

This guide consists of the following chapters:

Installation and documentation explains how to install and run the Embedded Workbench or C-SPY, and gives an overview of the documentation supplied with the IAR Systems tools.

The *Introduction* provides a brief summary of the Workbench features, and describes how you would use it to develop a typical project.

The *Overview* explains how the Embedded Workbench organizes your work into a project, to help you keep track of the source files involved in a typical application. It explains how the configuration options relate to your project, and how you use the IAR Systems development tools to operate on the files within a project.

The *Tutorial* demonstrates how to use the most important features of the Embedded Workbench to develop and maintain a simple C project.

Reference provides complete reference information about the components of the Embedded Workbench window, and the commands on each of the Embedded Workbench menus.

ASSUMPTIONS AND CONVENTIONS

ASSUMPTIONS

This guide assumes that you already have a working knowledge of the following:

- The C programming language.
- The PICmicro™ microcontroller.
- The procedures for using menus, windows, and dialog boxes in a windows environment.

Note that the illustrations in this guide show the Embedded Workbench running in a Windows 95-style environment, and their appearance will be slightly different if you are using another platform.

CONVENTIONS

This user guide uses the following typographical conventions:

Style	*Used for*
`computer`	Text that you type in, or that appears on the screen.
`parameter`	A label representing the actual value you should type as part of a command.
`[option]`	An optional part of a command.
bold	Names of menus, menu commands, buttons, and dialog boxes that appear on the screen.
reference	A cross-reference to another part of this guide, or to another guide.

CONTENTS

INSTALLATION AND DOCUMENTATION

This chapter explains how to install and run the Embedded Workbench and command line versions of the IAR products, and gives an overview of the available documentation. It also describes the `iar` subdirectories and file types.

INCLUDED IN THIS PACKAGE

The PICmicro™ package contains the following items:

- CD-ROM or floppy disks.
- Product documentation:

  PICmicro™ Embedded Workbench Interface Guide

  PICmicro™ Command Line Interface Guide

  PICmicro™ Assembler, Linker, and Librarian Programming Guide

  PICmicro™ C Compiler Programming Guide

  PICmicro™ C-SPY User Guide, if you have purchased the IAR C-SPY debugger.
- Licence agreement including the *Product Registration Form*, which we urge you to fill out and send us to ensure that you receive the latest release of the IAR development tools for the PICmicro™ family of microcontrollers.

INSTALLING THE EMBEDDED WORKBENCH WITH C-SPY

This section explains how to install and run the Embedded Workbench with C-SPY.

WHAT YOU NEED

- Windows 95/98, or Windows NT 3.51 or later.
- At least 40 Mbytes of free disk space for the Embedded Workbench.
- 32 Mbytes of RAM recommended for the Embedded Workbench and the IAR C-SPY Debugger.

If you are using C-SPY you should install the Workbench before C-SPY.

INSTALLING FROM WINDOWS 95/98 OR NT 4.0

1 Insert the installation CD-ROM or the first installation disk.

If you install from a CD-ROM, follow the instructions on the screen. If you install from a floppy disk, follow the instructions below:

2 Click the **Start** button in the taskbar, then click **Settings** and **Control Panel**.

3 Double-click the **Add/Remove Programs** icon in the **Control Panel** folder.

4 Click **Install**, then follow the instructions on the screen.

RUNNING FROM WINDOWS 95/98 OR NT 4.0

1 Click the **Start** button in the taskbar, then click **Programs** and **IAR Embedded Workbench**.

2 Click the **IAR Embedded Workbench** program icon.

INSTALLING FROM WINDOWS NT 3.51

1 Insert the first installation disk or the installation CD-ROM.

If you install from a CD-ROM, follow the instructions on the screen. If you install from a floppy disk, follow the instructions below:

2 Double-click the **File Manager** icon in the **Main** program group.

3 Click the floppy disk icon in the **File Manager** toolbar.

4 Double-click the **setup.exe** icon, then follow the instructions on the screen.

RUNNING FROM WINDOWS NT 3.51

Go to the Program Manager and double-click the **IAR Embedded Workbench** icon.

RUNNING C-SPY

Either:

Start C-SPY in the same way as you start the Embedded Workbench (see above).

Or:

Choose **Debugger** from the Embedded Workbench **Project** menu.

INSTALLING THE COMMAND LINE TOOLS

This section describes how to install and run the command line versions of the IAR Systems tools. You should be familiar with your operating system.

WHAT YOU NEED

- Windows 95/98, or Windows NT 3.51 or later.
- At least 35 Mbytes of free disk space.
- 32 Mbytes of RAM recommended for the IAR applications.

INSTALLATION

1. Insert the first installation disk.
2. At the command line prompt type `a:\install` and press Enter.
3. Follow the instructions on the screen.

When the installation is complete:

4. Add the path to the IAR Systems command line executable files to the PATH variable. For a default installation you would add `c:\iar\exe`.

   Define the environment variables `APIC_INC`, `C_INCLUDE`, `QPICINFO` and `XLINK_DFLTDIR` specifying the paths to the `inc` and `lib` directories; for example:

   ```
   set APIC_INC=c:\iar\inc\
   set C_INCLUDE=c:\iar\inc\
   set QPICINFO=c:\iar\setup\
   set XLINK_DFLTDIR=c:\iar\lib\
   ```

RUNNING THE TOOLS

Type the appropriate command at the command line prompt.

For more information refer to the *PICmicro™ C Compiler Programming Guide*, and the *PICmicro™ Assembler, Linker, and Librarian Programming Guide*.

INSTALLED FILES

The installation procedure creates several directories to contain the different types of files used with the IAR Systems development tools. The following sections give a description of the files contained by default in each directory. During installation you have the option to specify other directories than the ones created by default.

DOCUMENTATION FILES

Your installation may include a number of ASCII-format text files (`*.txt`) containing recent additional information. It is recommended that you read all of these files before proceeding.

ASSEMBLER FILES

The `apic` subdirectory holds the document files and assembler include files for the PICmicro™ Assembler.

The `iar\ew`*`nn`*`\picmicro\apic\tutor` directory contains the files used for the PICmicro™ Assembler tutorials.

MISCELLANEOUS FILES

The `etc` subdirectory holds XLINK-related files.

EXECUTABLE FILES

The `iar` directory contains the `ew`*`nn`*`.exe` and `cw`*`nn`*`.exe` files. Other executable files are located in the `iar\ew`*`nn`*`\picmicro\bin` directory.

The `bin` subdirectory holds the executable program files.

The installation procedure also includes an addition to the `autoexec.bat PATH` statement, directing having the `bin` subdirectory searched for command files. This allows you to issue a command from any directory.

C COMPILER FILES

The `iccpic` subdirectory holds various source files for basic I/O library routines.

The `iar\ew`*`nn`*`\picmicro\iccpic\tutor` directory contains the files used for the PICmicro™ C Compiler tutorials.

INCLUDE FILES

The `inc` subdirectory holds include files, such as the header files for the standard C library, as well as a specific header file defining SFRs (special function registers) for each supported PICmicro™ derivative. These files are used by the C compiler.

The C compiler searches for include files in the directory specified by the `C_INCLUDE` environment variable. If you set this environment variable to the path of the include subdirectory, as suggested in the installation procedure, you can refer to `inc` header files simply by their base names.

The assembler has an equivalent environment variable, `APIC_INC`.

LIBRARY FILES

The `lib` subdirectory holds library modules used by the C compiler.

XLINK searches for library files in the directory specified by the `XLINK_DFLTDIR` environment variable. If you set this environment variable to the path of the `lib` subdirectory, you can refer to `lib` library modules simply by their basenames.

No library modules are installed; instead the modules should be built for the appropriate microcontroller configuration using the included Buildlib utility, see the chapter *General C library definitions* in the PICmicro™ *C Compiler Programming Guide*.

Pre-built library modules with standard configuration are supplied in the `\lib` directory on the CD. A library is supplied for each supported microcontroller and is named `cl`*`xxxxx`*`.r39`, where *`xxxxx`* corresponds to the microcontroller name.

LINKER COMMAND FILES

The `iccpic` subdirectory holds an example linker command file for each supported microcontroller.

FILE TYPES

The PICmicro™ versions of the IAR Systems development tools use the following default file extensions to identify the IAR-specific types of file:

Ext.	*Type of file*	*Output from*	*Input to*
`.a39`	Target program	XLINK	EPROM, C-SPY, etc
`.c`	C program source	Text editor	C compiler
`.d39`	Target program with debug information	XLINK	C-SPY, etc
`.h`	C header source	Text editor	C compiler `#include`
`.i39`	Compiler/debugger	Processor description file	–
`.inc`	Assembler header	Text editor	Assembler `#include` file
`.lst`	List	C compiler and assembler	–
`.mac`	C-SPY macro definition	Text editor	C-SPY
`.map`	XLINK map	XLINK	–
`.mem`	Target memory layout	Text editor	C-SPY
`.prj`	Embedded Workbench project	Embedded Workbench	Embedded Workbench
`.r39`	Object module	C compiler and assembler	XLINK and XLIB
`.s39`	Assembler program source	Text editor	Assembler
`.xcl`	Extended command line	Text editor	XLINK and C compiler

The default extension may be overridden by simply including an explicit extension when specifying a filename.

Note that, by default, XLINK listings (maps) will have the `.lst` extension, and this may overwrite the listing file generated by the compiler. It is recommended that you explicitly name XLINK map files, for example `demo1.map`.

Files with the extensions `.ini` and `.cfg` are created dynamically when you install and run the tools. These files contain information about your configuration and other settings.

DOCUMENTATION

THE OTHER GUIDES

The other guides provided with the Embedded Workbench are as follows:

PICmicro™ Command Line Interface Guide
This guide explains how to configure and run the IAR Systems development tools from the command line. It also includes reference information about the command line environment variables.

PICmicro™ C Compiler Programming Guide
This guide provides programming information about the PICmicro™ C Compiler. It includes reference information about the C library functions and language extensions, and provides information about support for the target-specific options such as memory models.

You should refer to this guide when you are setting up the C compiler configuration options in the Embedded Workbench, and for information about the C language when writing and debugging C source programs.

This guide also describes the diagnostic functions and lists the PICmicro™- specific warning and error messages.

PICmicro™ Assembler, Linker, and Librarian Programming Guide
This guide provides reference information about the PICmicro™ Assembler, XLINK Linker, and XLIB Librarian for use with the Embedded Workbench.

The assembler programming sections include details of the assembler source format, and reference information about the assembler operators, directives, and mnemonics.

The XLINK Linker programming reference sections provide information about the XLINK Linker commands and output formats.

The XLIB Librarian programming sections provide information about the XLIB Librarian commands.

Finally, the guide includes a list of diagnostic messages for each of these tools.

PICmicro™ C-SPY User Guide
This optional guide describes how to use C-SPY Debugger for the PICmicro™ series of microcontrollers, and provides reference information about the features of C-SPY.

In addition to the information contained in the PICmicro™ guides, also online information is available.

ONLINE HELP

From the **Help** menu in the Embedded Workbench and the IAR C-SPY Debugger, you can access the PICmicro™ online information. It contains complete reference information for the PICmicro™ Embedded Workbench, C-SPY, C compiler, assembler, XLINK Linker, and XLIB Librarian.

READ-ME FILES

We recommend that you read the following `Read-Me` files for recent information that is not included in the guides:

```
apic.txt
cwpic.txt
ewpic.txt
iccpic.txt
xlink.txt
```

IAR ON THE WEB

The latest news from IAR Systems is available at the web site **www.iar.com**. You can access the IAR site directly from the Embedded Workbench **Help** menu and receive information about:

- Product announcements.
- Special offerings.
- Evaluation copies of the IAR products.
- Technical Support including FAQs (frequently asked questions).
- Links to chip manufacturers and other interesting sites.
- Distributor information.

INTRODUCTION

The IAR Systems Embedded Workbench is a flexible integrated environment for developing applications for a variety of different target processors. It provides a convenient windows interface for rapid development and debugging.

Support for a number of different target processors can be added to the Embedded Workbench, and users developing projects for different target processors can specify the target on a project by project basis. For more information on supported target processors please contact your local IAR distributor.

The tools include a fast compiler, an efficient linker, a librarian, a syntax highlighting text editor, an automatic Make facility, and an optional C-SPY debugger.

EMBEDDED WORKBENCH

The IAR Systems Workbench provides the following features:

GENERAL

- Runs under Windows 95/98 or Windows NT 3.51 or later.
- Hierarchical project representation.
- Intuitive user interface, taking advantage of Windows 95/98 features.
- The Make utility recompiles, reassembles, and links files only when necessary.
- Full integration between the Workbench tools and editor.
- Support of drag and drop features.
- Comprehensive hypertext help.

EMBEDDED WORKBENCH EDITOR

- Syntax of C programs shown using text styles and colors.
- Powerful search and replace commands, including multi-file search.
- Direct jump to context from error listing.

- Parenthesis matching.
- Automatic indentation.
- Multi-level undo and redo for each window.

C COMPILER AND ASSEMBLER

- Projects build in the background under Windows 95/98 or Windows NT, allowing simultaneous editing.
- Options can be set globally, on groups of source files, or on individual source files.

C COMPILER

The IAR Systems C Compiler for the PICmicro™ family of microcontrollers offers the standard features of the C language, plus many extensions designed to take advantage of the PICmicro™-specific facilities. The compiler is supplied with the IAR Systems Assembler for the PICmicro™, with which it is integrated, and shares linker and librarian manager tools.

It provides the following features:

LANGUAGE FACILITIES

- Conformance to the ANSI specification for free-standing C in accordance to what the controller design allows.
- Standard library of functions applicable to embedded systems, with source available.
- IEEE-compatible floating-point arithmetic.
- Powerful extensions for PIC-specific features, including efficient I/O.
- LINT-like checking of program source.
- Linkage of user code with assembly routines.
- Long identifiers – up to 255 significant characters.
- Up to 32000 external symbols.
- Maximum compatibility with other IAR Systems C compilers.

PERFORMANCE

- Fast compilation.
- Memory-based design which avoids temporary files or overlays.
- Rigorous type checking at compile time.
- Rigorous module interface type checking at link time.

CODE GENERATION

- Selectable optimization for code speed or size.
- Comprehensive output options, including relocatable binary, ASM, ASM + C, XREF, etc.
- Easy-to-understand error and warning messages.
- Compatibility with the C-SPY high-level debugger.

TARGET SUPPORT

- Flexible variable allocation.
- Interrupt functions.
- A `#pragma` directive to maintain portability while using processor-specific extensions.

DOCUMENTATION

The PICmicro™ C Compiler is documented in the *PICmicro™ C Compiler Programming Guide.*

ASSEMBLER

The IAR Systems PICmicro™ Assembler is a powerful relocating macro assembler with a versatile set of directives.

The assembler incorporates a high degree of compatibility with the processor manufacturer's own assemblers, to ensure that software originally developed using them can be transferred to the IAR Systems Assembler with little or no modification.

It provides the following features:

GENERAL

- One pass assembly, for faster execution.
- Integration with the XLINK Linker and XLIB Librarian.
- Integration with other IAR Systems software.
- Self-explanatory error messages.

ASSEMBLER FEATURES

- Support for the PIC family of microcontrollers.
- Up to 256 relocatable segments per module.
- 32-bit arithmetic and IEEE floating-point constants.
- 255 significant characters in symbols.
- Powerful recursive macro facilities.
- Number of symbols and program size limited only by available memory.
- Support for complex expressions with external references.
- Forward references allowed to any depth.
- Support for C language pre-processor directives.
- Macros in Intel/Motorola style.

DOCUMENTATION

The PICmicro™ Assembler is documented in the *PICmicro™ Assembler, Linker, and Librarian Programming Guide.*

XLINK LINKER

The IAR Systems XLINK Linker converts one or more relocatable object files produced by the IAR Systems Assembler or C Compiler to machine code for a specified target processor. It supports a wide range of industry-standard loader formats, in addition to the IAR Systems debug format used by the C-SPY high level debugger.

XLINK supports user libraries, and will load only those modules that are actually needed by the program you are linking.

The final output produced by XLINK is an absolute, target-executable object file that can be programmed into an EPROM, downloaded to a hardware emulator, or run directly on the host using the IAR Systems C-SPY debugger.

XLINK offers the following important features:

FEATURES OF XLINK

- Unlimited number of input files.
- Searches user-defined library files and loads only those modules needed by the application.
- Symbols may be up to 255 characters long with all characters being significant. Both upper and lower case may be used.
- Global symbols can be defined at link time.
- Flexible segment commands allow full control of the locations of relocatable code and data in memory.
- Support for over 30 output formats.

DOCUMENTATION

The XLINK Linker is documented in the *PICmicro™ Assembler, Linker, and Librarian Programming Guide*.

XLIB LIBRARIAN

The IAR Systems XLIB Librarian enables you to manipulate the relocatable object files produced by the IAR Systems Assembler and C Compiler.

XLIB provides the following features:

FEATURES OF XLIB

- Support for modular programming.
- Modules can be listed, added, inserted, replaced, deleted, or renamed.
- Segments can be listed and renamed.
- Symbols can be listed and renamed.
- Modules can be changed between program and library type.

- Interactive or batch mode operation.
- A full set of library listing operations.

DOCUMENTATION

The XLIB Librarian is documented in the *PICmicro™ Assembler, Linker, and Librarian Programming Guide.*

C-SPY DEBUGGER

An optional C-SPY debugger can be added to the Embedded Workbench, to run and debug PICmicro™ object code programs, and if present will be accessible from the menus and toolbar.

DOCUMENTATION

The C-SPY debugger is documented in the *PICmicro™ C-SPY User Guide.*

OVERVIEW

The IAR Embedded Workbench provides a powerful environment for developing projects with a range of different target processors, and a selection of tools for each target processor.

This chapter gives a brief discussion of the project model used by the Embedded Workbench, and explains how you use it to develop typical applications.

HOW PROJECTS ARE ORGANIZED

The Embedded Workbench has been specially designed to fit in with the way that software development projects are typically organized. For example, you may need to develop related versions of an application for different versions of the target hardware, and you may also want to include debugging routines into the early versions, but not in the final code.

Versions of your applications for different target hardware will often have source files in common, and you want to be able to maintain a unique copy of these files, so that improvements are automatically carried through to each version of the application. There will also be source files that differ between different version of the application, such as those dealing with hardware-dependent aspects of the application, and so these will need to be maintained separately for each target version.

The Embedded Workbench addresses these requirements, and provides a powerful environment for maintaining the source files used to build all versions of an application. It allows you to organize projects in a hierarchical tree structure showing the dependency between files at a glance.

TARGETS

At the highest level of the structure you specify the different target versions of your application that you want to build. For a simple application you might need just two targets, called **Debug** and **Release**. A more complex project might include additional targets for each of the different processor variants that the application is to run on.

GROUPS

Each target in turn contains one or more groups, which collect together related sets of source files. A group can be unique to a particular target, or it can be present in two or more targets. For example, you might create a group called **Debugging routines** which would be present only in the **Debug target**, and another group called **Common sources** which would be present in all targets.

SOURCE FILES

Each group is used to group together one or more related source files. For maximum flexibility each group can be included in one or more targets.

When you are working with a project you always have a current target selected, and only the groups that are members of that target, along with their enclosed files, are visible in the Project window. Only these files will actually be built and linked into the output code.

SETTING OPTIONS

For each target you set global assembler and compiler options at the target level, to specify how that target should be built. At this level you typically define the memory model you are using, and the processor variant.

You can also set local compiler and assembler options on individual groups and source files. These local options are specific to the context of a target and override override any corresponding global options set at the target level, and are specific to that target. A group can be included in two different targets and have different options set for it in each target. For example, you might set optimization high for a group containing source files that you have already debugged, but remove optimization from another group containing source files that you are still developing.

BUILDING A PROJECT

The **Compile** command on the Embedded Workbench **Project** menu allows you to compile or assemble the files of a project individually. The Embedded Workbench automatically determines whether a source file should be compiled or assembled by its file name extension.

Alternatively, you can build the entire project, automatically compiling and assembling all the component files, using the **Make** command. This identifies when the files have changed, and only recompiles or assembles files as necessary before re-linking the project, based on whether they have changed, and their dependencies on other files.

A **Build All** option is also provided, which unconditionally regenerates all files.

When running the Embedded Workbench on Windows NT or Windows 95/98, the **Compile**, **Make**, **Link**, and **Build** commands all run in the background so that you can continue editing or working with the Embedded Workbench while your project is being built.

TESTING THE CODE

The compiler and assembler are fully integrated with the development environment, so that if there are errors in your source code you can jump directly from the error listing to the correct position in the appropriate source file, to allow you to locate and correct the error.

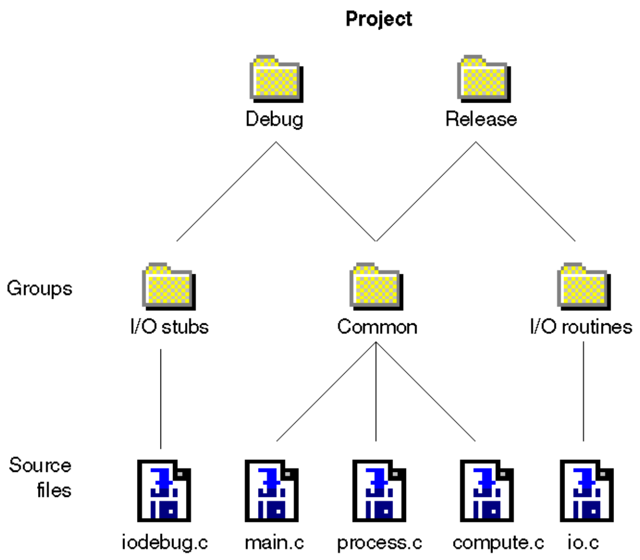
After you have resolved any compile-time errors you can switch directly to the C-SPY debugger to test the resulting code at source level. The C-SPY debugger runs in a separate window, so that you can make changes to the original source files to correct problems as you identify them in C-SPY.

SAMPLE APPLICATIONS

The following examples describe two sample applications to illustrate how you would use the Embedded Workbench in typical development projects.

A SIMPLE APPLICATION

In a simple application which you are developing for one version of the target hardware you would manage with the two default targets, **Release** and **Debug**, as shown in the following diagram:



Both targets share a common group containing the project's core source files. Each target also contains a group containing the source files specific to that target: **I/O routines**, contains the source files for the input/output routines to be used in the final release code, and **I/O stubs** which contains input/output stubs to allow the I/O to be debugged with a debugger such as C-SPY.

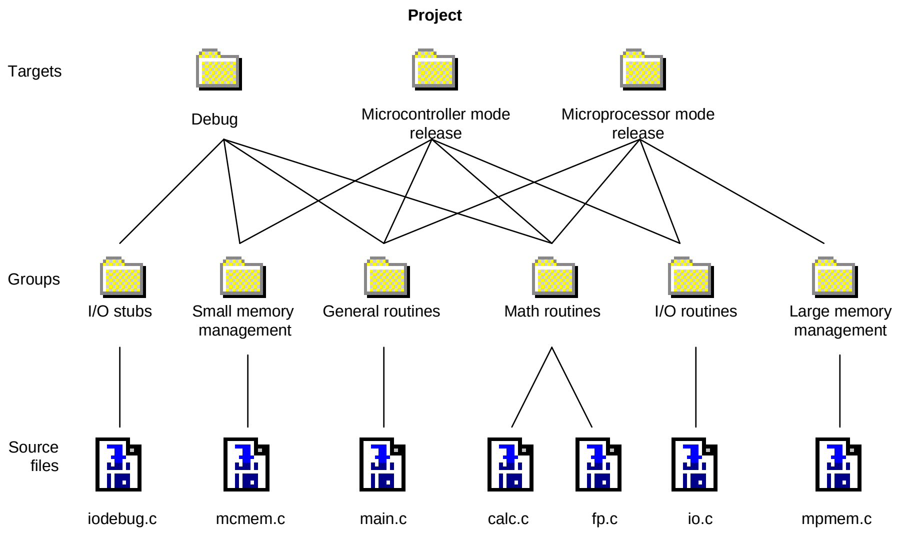
The release and debug targets would typically have different compiler options set for them; for example, you could compile the **Debug** version with trace, assertions, etc, and the **Release** version without it.

A MORE COMPLEX PROJECT

In the following more complex project an application is being developed for several different pieces of target hardware, containing different variants of the PICmicro™ processor, and different I/O ports and memory configurations. The project therefore includes a debug target, and a release target for each of the different sets of target hardware.

The source files that are common to all the targets are collected together, for convenience, into groups which are included in each of the targets. The names of these groups reflect the areas in the application that the source code deals with; for example **Math, I/O** etc.

Areas of the application that depend on the target hardware, such as the memory management, are included in a number of separate groups, one per target. Finally, as before, debugging routines are provided for the **Debug** target.



When working with large projects such as this the Embedded Workbench minimizes your development time by helping you to keep track of the structure of your project, and optimizes the development cycle by assembling and compiling the minimum set of source files necessary to keep the object code completely up to date after changes.

TUTORIAL

This tutorial illustrates how you might use the Embedded Workbench to develop a simple C program, compile it, and run it using the C-SPY debugger.

Before reading this chapter you should:

- ◆ Have installed the Embedded Workbench software, as described in the *QuickStart Card* or the chapter *Installation and documentation*.
- ◆ Be familiar with the architecture and instruction set of the PICmicro™ microcontrollers.

USING C-SPY

This tutorial assumes that you are using the C-SPY debugger with the Embedded Workbench, and describes how to use C-SPY to run the programs you are developing. If your installation does not include C-SPY, you can use the Embedded Workbench editor to examine the listing files.

GETTING STARTED

The files you are working on in the Embedded Workbench are organized into projects.

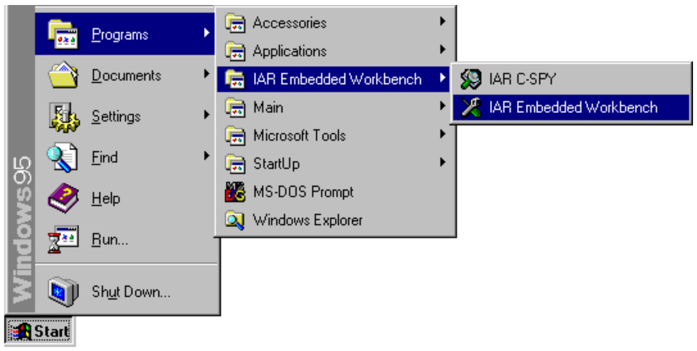
We recommend that you create a directory where you keep the project files. In this tutorial we assume that there is a directory called `projects` on the `c:` drive.

Copy the files `tutor.c`, `tutor.h`, `common.c`, and `common.h` that are provided in the `iar\ew`*`nn`*`\picmicro\iccpic\tutor` directory to `c:\projects\`.

The first step in using the Embedded Workbench is to create a new project, to specify which target processor you are working on, and to include a list of the files contained in the project.

RUNNING THE EMBEDDED WORKBENCH

Click the **Start** button in the taskbar, then click **Programs** and **IAR Embedded Workbench**.

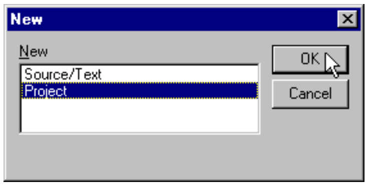


Then click the **IAR Embedded Workbench** program icon, and the Embedded Workbench window will be displayed.

CREATING A NEW PROJECT

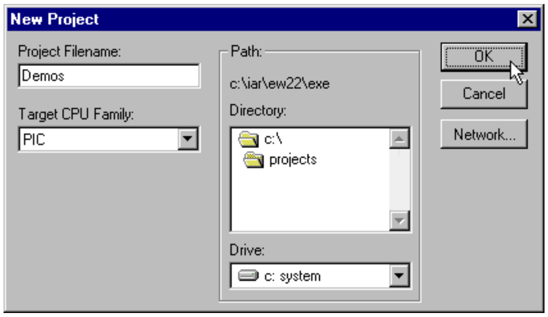
Create a project for the tutorial as follows.

Choose **New**… from the **File** menu to display the following dialog box:



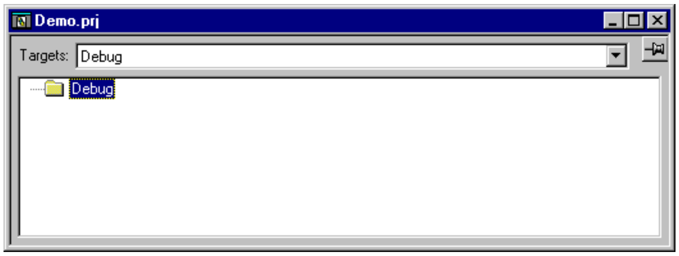
Select **Project** and choose **OK** to display the **New Project** dialog box.

Enter `Demo` in the **Project Filename** box, and set the **Target CPU Family** to PIC:

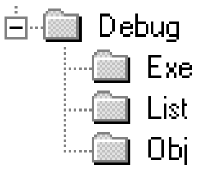


Locate the directory where you want to store your project. In this example we choose `c:\projects`. Then choose **OK** to create the new project.

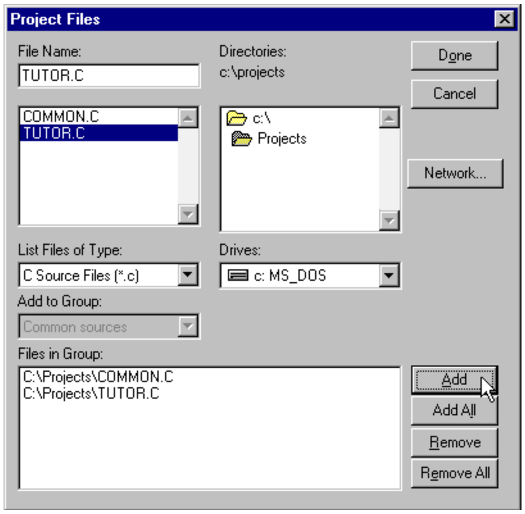
The Project window will be displayed. Make sure that **Debug** is selected from the **Targets** drop-down list box:



A directory structure is created in the project directory, with separate directories for executable files, list files, and object files:



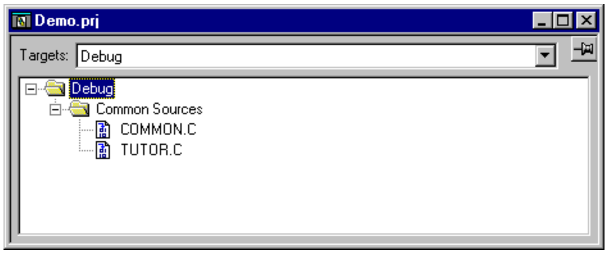
Choose **Files…** from the **Project** menu to display the **Project Files** dialog box. Locate the files `tutor.c` and `common.c` in the file selection list in the upper half of the dialog box, and choose **Add** to add them to the new project:



C:\Projects\COMMON.C
C:\Projects\TUTOR.C

Then click **Done** to close the **Project Files** dialog box.

Click the ⊞ symbol to display the files in the Project window tree display:

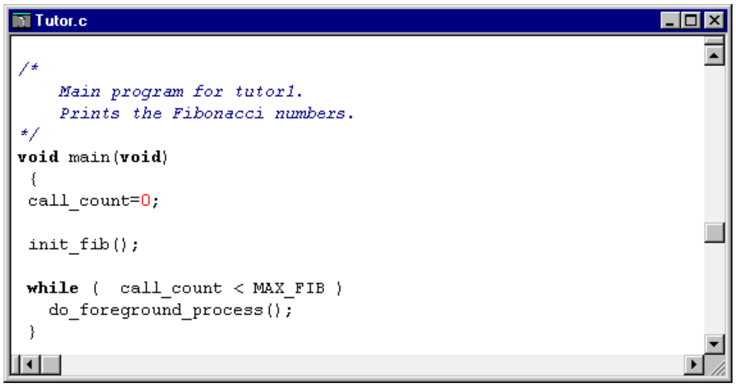


Since we did not create any group in our project the Embedded Workbench created the default group **Common Sources**.

You can add files to a project at a later date, or remove files, using the **Files…** command on the **Project** menu.

EDITING A FILE

To edit one of the files in a project you simply double-click its name in the Project window. For example, double-click the file `tutor.c`. The file is displayed in an editor window:



```
/*
    Main program for tutor1.
    Prints the Fibonacci numbers.
*/
void main(void)
 {
 call_count=0;

 init_fib();

 while (  call_count < MAX_FIB )
   do_foreground_process();
 }
```

Note that the Embedded Workbench editor provides a number of useful features to help you enter programs correctly, and to provide immediate syntax checking as you type.

For example, the following components of a program are identified:

Item	*Highlight*
Keyword	Black bold
Text string	Blue
Preprocessor directives	Green
Numeric constants	Red/magenta/blue
Comments	Dark blue italics
Other program constructions	Black

You can configure these settings; for more information see *Colors and Fonts*, page 71.

We will use the editor to introduce an error into the program so that we can see the error handling features provided by the Embedded Workbench.

Change the `put_fib( fib )` on line 22 to `put_fib( fibb )` and save the file by choosing **Save** from the **File** menu.

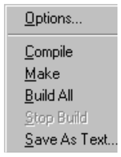
COMPILING THE PROJECT

SETTING COMPILER OPTIONS

The Embedded Workbench allows you to set options for the whole target, for a group of files, or for a single source file.

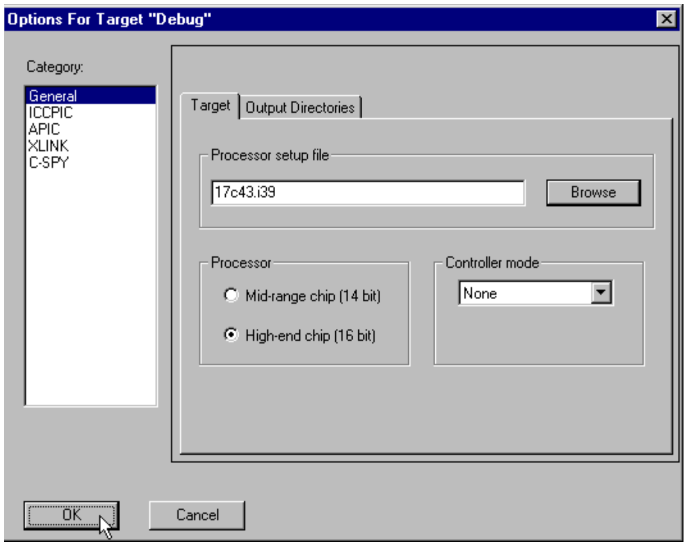
For this tutorial we will set options for the whole **Debug** target, as we do not need individual options for the groups or files in the target.

Select the **Debug** folder icon in the Project window, to specify which options are to be set, and choose **Options...** from the **Project** menu.



Alternatively, click the right mouse button on the **Debug** folder and choose **Options...** from the pop-up menu.

The **Options** dialog box is displayed. Then select **General** in the **Category** list and click the **Target** tab to display the target options for the Embedded Workbench tools:



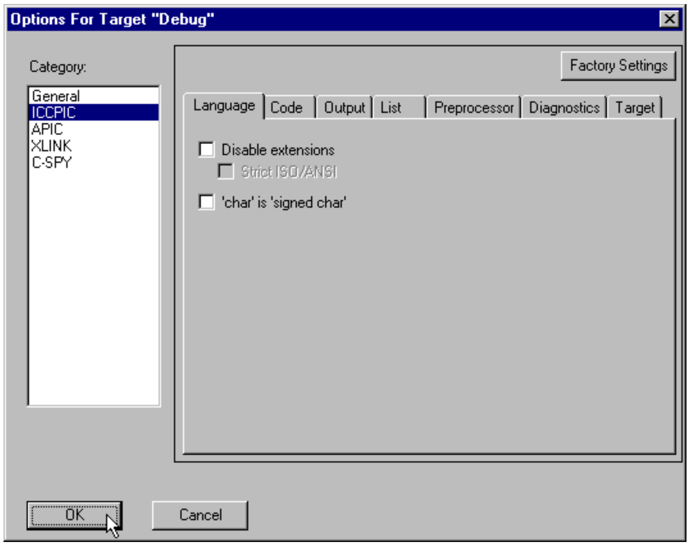
Processor setup file: 17c43.i39

Processor: High-end chip (16 bit)

Controller mode: None

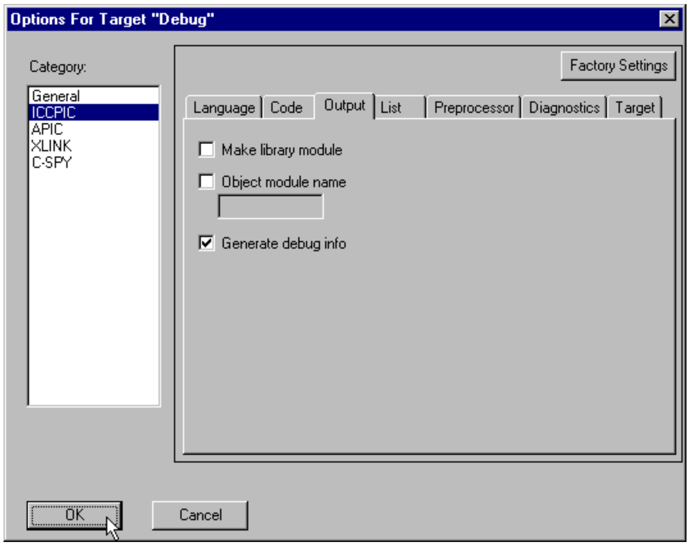
Set the **Processor setup file** to **17C43** by selecting the file `17c43.i39` in the `...\setup\` directory. Then select the **High-end (16 bit)** processor.

Next select **ICCPIC** in the **Category** list to display the pages of C compiler options:



You can display any page by clicking the appropriate tab at the top of the page.

Click **Output** to display the C compiler options and check that **Generate debug information** is selected, to create an output file for debugging with C-SPY:



- [ ] Make library module
- [ ] Object module name
- [x] Generate debug info

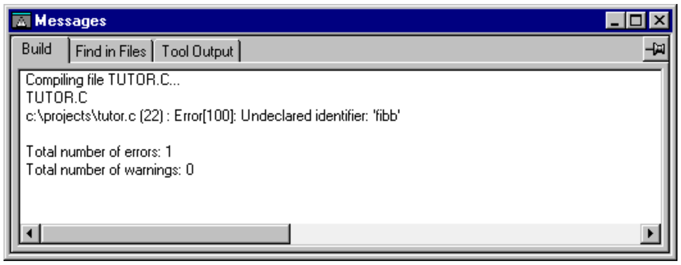
Choose **OK** to save the options you have specified.

COMPILING A FILE

To compile a source file you are editing choose **Compile** from the **Project** menu, or click the **Compile** button in the toolbar.

You can also compile a source file by selecting it in the Project window and choosing **Compile**.

The progress, and any error messages, will be displayed in the Messages window:



```
Compiling file TUTOR.C...
TUTOR.C
c:\projects\tutor.c (22) : Error[100]: Undeclared identifier: 'fibb'

Total number of errors: 1
Total number of warnings: 0
```

Here there is one error corresponding to the bug we inserted.

Double-click the error message in the Messages window. The cursor will be moved directly to the appropriate line in the program and you can simply correct the error.

For example, in this case change `put_fib( fibb )` to the correct version `put_fib( fib )`.

Then recompile as described earlier. It should compile this time without an error, and you can then close the `tutor.c` source file by choosing **Close** from the **File** menu.

Compile the file `common.c` in the same way.

The object files are created in the project's `debug\obj` subdirectory.

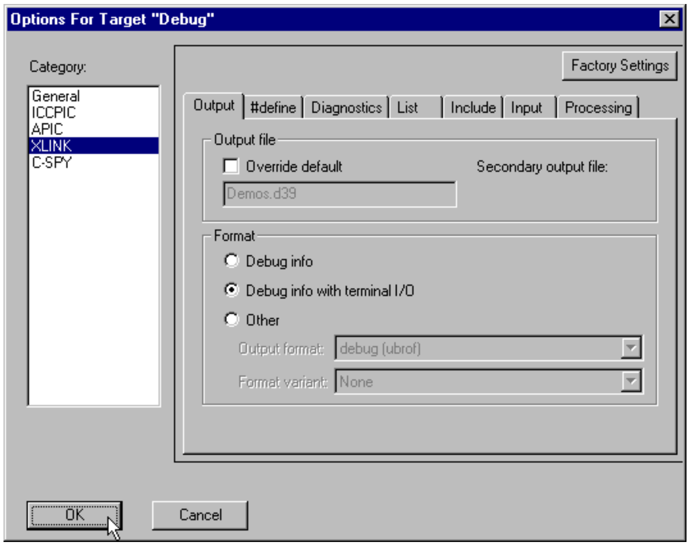
LINKING THE PROJECT

Before linking the program you need to set up the linker options for the project.

SETTING THE LINKER OPTIONS

Select the **Debug** folder in the Project window. Then choose **Options...** from the **Project** menu, and select **XLINK** in the **Category** list to display the linker option pages.

View the options on the **Output** page. Check that the **Format** option is set to **Debug info with terminal I/O**, to generate a file for debugging with C-SPY.



Output file

☐ Override default

Demos.d39

Secondary output file:

Format

○ Debug info

◉ Debug info with terminal I/O

○ Other

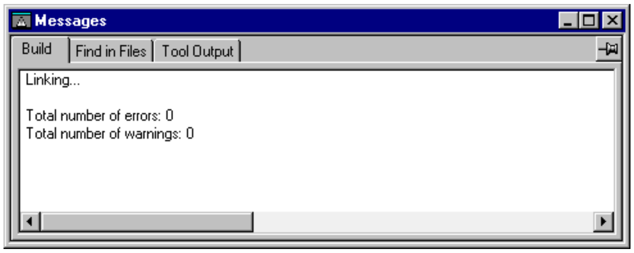
Output format: debug (ubrof)

Format variant: None

Then choose **OK** to close the dialog box and save your settings.

LINKING THE FILES

To link the project choose **Link** from the **Project** menu. The files will then be linked, and the Messages window will show progress during linking:



```
Linking...

Total number of errors: 0
Total number of warnings: 0
```

Providing there were no errors, the output file `demo.d39` will be generated for use with the C-SPY Simulator.

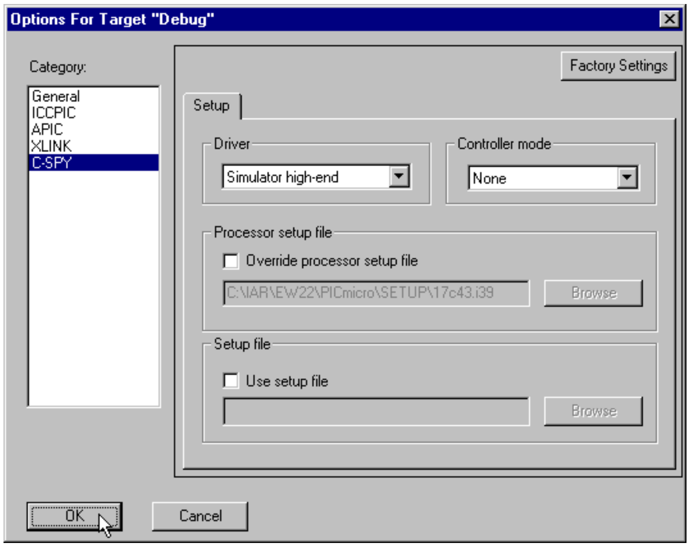
The executable files will be created in the project's `debug\exe` subdirectory.

DEBUGGING THE PROJECT

If you have the C-SPY debugger you can run the object code using C-SPY.

First set up the options for the C-SPY debugger. Select the **Debug** folder icon in the **Project** window, choose **Options…** from the **Project** menu, and select **C-SPY** in the **Category** list to display the C-SPY options pages.

Select the **Simulator high-end** driver in the **Driver** list:



Category: General, ICCPIC, APIC, XLINK, C-SPY

Driver: Simulator high-end

Controller mode: None

Processor setup file

☐ Override processor setup file

C:\IAR\EW22\PICmicro\SETUP\17c43.i39

Setup file

☐ Use setup file

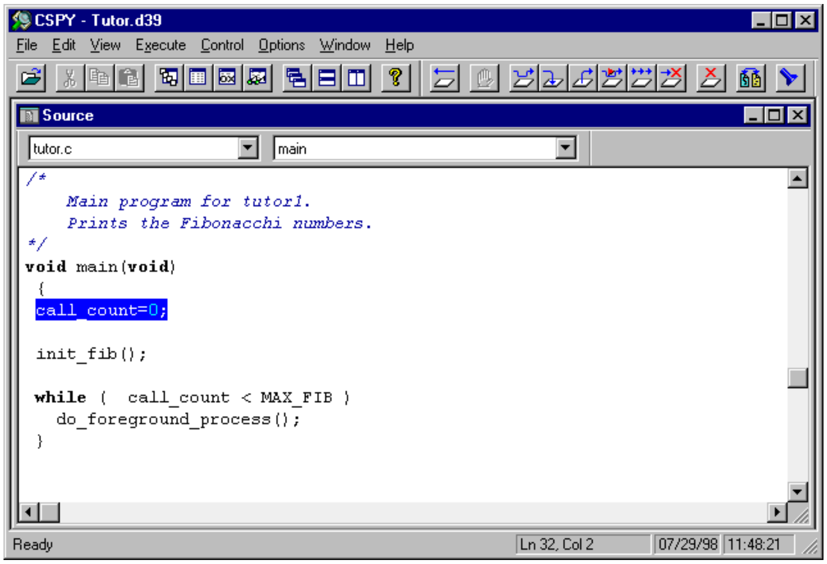
Then choose **OK** to save the C-SPY options.

Choose **Debugger** from the **Project** menu, or click the **C-SPY** button in the project bar.

If necessary, the project will be relinked with debugging information for use by the simulator, and then C-SPY will be run automatically.

Choose **Step** from the **Execute** menu, or click the **Step** button in the debug bar, to start executing the source code.

The source will be displayed on the screen, with the first executable statement highlighted.



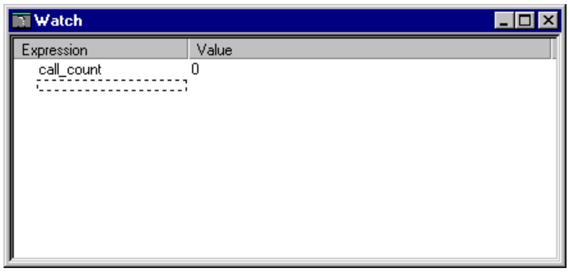
```
/*
    Main program for tutor1.
    Prints the Fibonacchi numbers.
*/
void main(void)
 {
 call_count=0;

 init_fib();

 while (  call_count < MAX_FIB )
   do_foreground_process();
 }
```

WATCHING VARIABLES

To keep track of a variable you can set a watchpoint on it.

For example, to watch the values of the variable `call_count` as you step through the program, first open the Watch window by choosing **Watch** from the **Window** menu.

Select the dotted rectangle, then click and briefly hold the left mouse button, and type `call_count`↵ to add the variable to the Watch window.

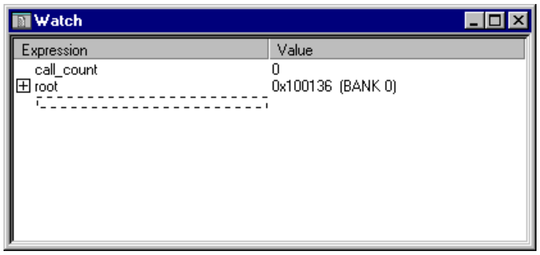


Expression	Value
call_count	0

Choose **Step** from the **Execute** menu, or click the **Step** button in the debug bar.

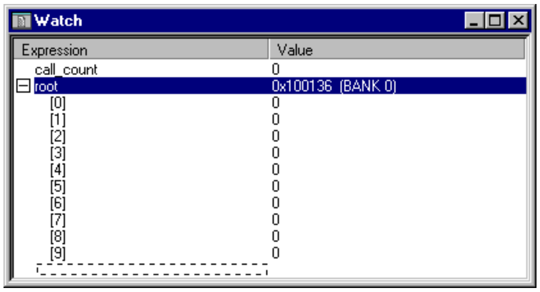
Then choose **Step Into** from the **Execute** menu or click the **Step Into** button in the debug bar to step into the function `init_fib()` which will initialize the `root` array. We can view the initialization of the array called `root` by placing it in the Watch window.

Double-click the `root` variable in the beginning of the `init_fib` function. Drag and drop `root` in the Watch window. The `root` should now appear just under the `call_count` variable.



```
Watch
Expression              Value
   call_count           0
⊞ root                  0x100136  (BANK 0)
```

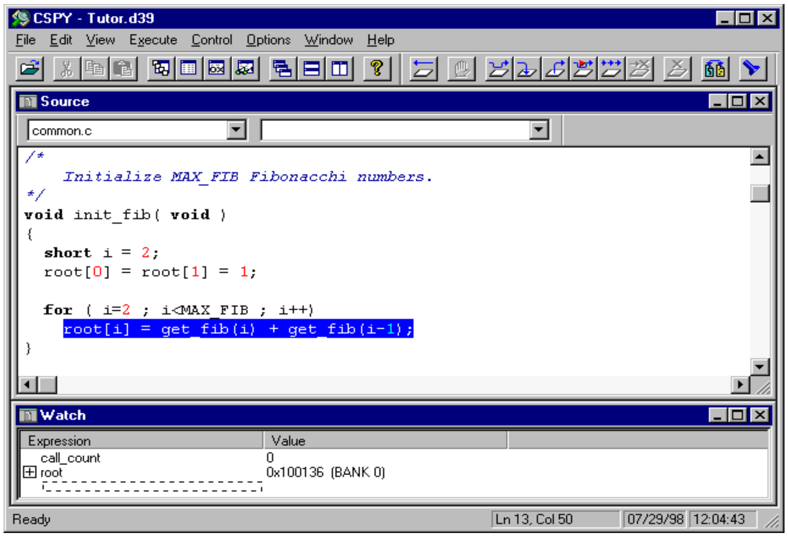
Just to the left of the `root` name there is a ⊞ symbol allowing us to view the contents of the array. (This can be done also for complex types such as `struct` and `union`).



```
Watch
Expression              Value
   call_count           0
⊟ root                  0x100136  (BANK 0)
     [0]                0
     [1]                0
     [2]                0
     [3]                0
     [4]                0
     [5]                0
     [6]                0
     [7]                0
     [8]                0
     [9]                0
```

Position the Source and Watch windows conveniently on the screen before proceeding.

Step to the `for` loop in the `init_fib` function. As you step through the `for` loop you can watch the `root` array being initialized. You will also notice that C-SPY will step through every statement inside the `for` loop.



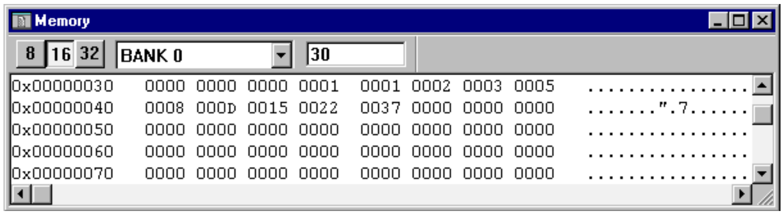
```
/*
    Initialize MAX_FIB Fibonacchi numbers.
*/
void init_fib( void )
{
  short i = 2;
  root[0] = root[1] = 1;

  for ( i=2 ; i<MAX_FIB ; i++)
    root[i] = get_fib(i) + get_fib(i-1);
}
```

```
Expression        Value
  call_count      0
⊞ root            0x100136 (BANK 0)
```

```
Ready                                Ln 13, Col 50   07/29/98  12:04:43
```

We can also view the memory where `root` is located. Open the Memory window by choosing **Memory** from the Window menu. In the Memory window you can select the size of the data that you are watching. In this case the `root` is an array of integer. Click the **16** button to view the 16-bit size of the memory window.

Mark `root` again and drag and drop it in the Memory window. As you step through the `for` loop the memory location for `root` will be updated with the initialization values:



```
0x00000030   0000 0000 0000 0001  0001 0002 0003 0005   ................
0x00000040   0008 000D 0015 0022  0037 0000 0000 0000   .......".7......
0x00000050   0000 0000 0000 0000  0000 0000 0000 0000   ................
0x00000060   0000 0000 0000 0000  0000 0000 0000 0000   ................
0x00000070   0000 0000 0000 0000  0000 0000 0000 0000   ................
```

Step out of the `init_fib` funtion by choosing **Go out** from the Execute menu or by stepping through the `init_fib` funtion until it returns to the main function.

Close the Memory window by clicking its close box.

SETTING A BREAKPOINT

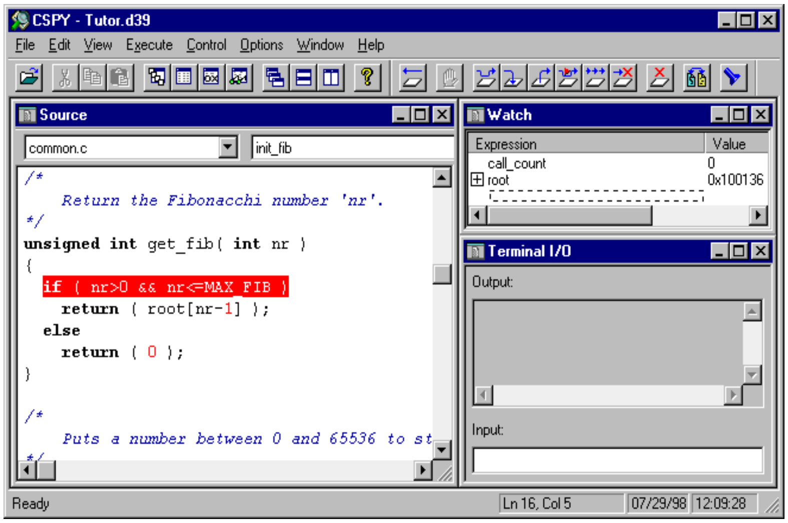
You can execute a program up to a specific statement by setting a breakpoint at that statement.

First open the Terminal I/O window, by choosing **Terminal I/O** from the **Window** menu, to display output from the program. Position the Source, Watch, and Terminal I/O windows conveniently on the screen.

Select the file `common.c` in the file box to the upper left in the Source window. Then select the function `get_fib` in the function box to the right of the file box.

Place the cursor at the `if` statement. In the **Control** menu choose **Toggle Breakpoint** to set a breakpoint on that statement.

The statement will be highlighted in red to show the breakpoint:



```
/*
   Return the Fibonacchi number 'nr'.
*/
unsigned int get_fib( int nr )
{
  if ( nr>0 && nr<=MAX_FIB )
    return ( root[nr-1] );
  else
    return ( 0 );
}

/*
   Puts a number between 0 and 65536 to st
```

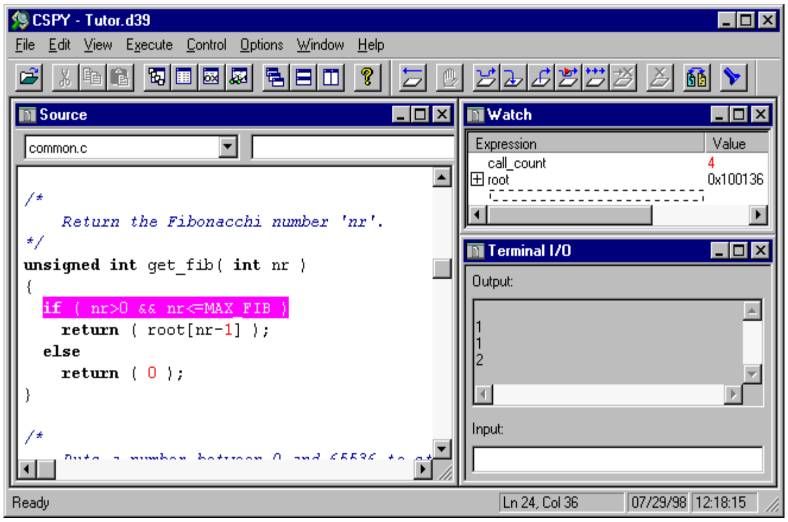
Expression	Value
call_count	0
⊞ root	0x100136

Ready Ln 16, Col 5 07/29/98 12:09:28

Then choose **Go** from the **Execute** menu or click the **Go** button in the debug bar, to execute up to the breakpoint.

Choose **Go** three times to see the variable `call_count` change.

The output will be displayed in the Terminal I/O window:



```
/*
   Return the Fibonacchi number 'nr'.
*/
unsigned int get_fib( int nr )
{
  if ( nr>0 && nr<=MAX_FIB )
    return ( root[nr-1] );
  else
    return ( 0 );
}

/*
   Puts a number between 0 and 65536 to st
```

Expression	Value
call_count	4
root	0x100136

```
1
1
2
```

Continue to step or choose **Go** from the **Execute** menu to step out of the program.

To exit from C-SPY and return to the Embedded Workbench, choose **Exit** from the **File** menu.

For more information on how to debug a program see the *PICmicro™ C-SPY User Guide*.

USING MAKE

Instead of compiling and assembling the files in a project individually, and then linking them, you can automatically bring the project up-to-date using the **Make** command.

The Embedded Workbench maintains a list of all the files in the project, and the include files they depend on. When you run **Make**, the Embedded Workbench checks the dependency files, and recompiles or reassembles as necessary to keep the project up to date.

Note that you do not need to add the include files to the project. They are automatically added to the list of dependency files when you reference them in an `#include` statement.

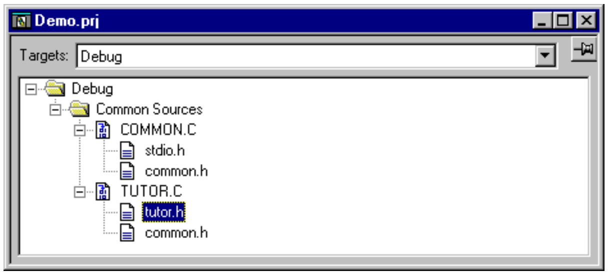
EDITING AN INCLUDE FILE

The following example illustrates how **Make** automatically detects that a dependent file has been changed.

The file `tutor.c` contains the following `#include` statement:

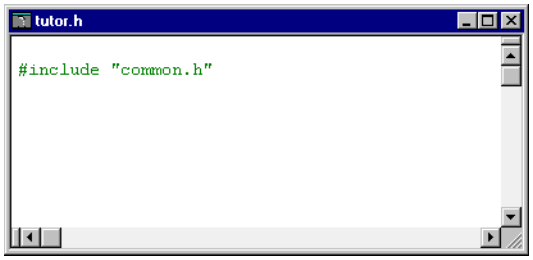
```
#include "tutor.h"
```

Once you have compiled a file the Project window shows any include files it references. Click the ⊞ symbol next to the source file to expand the tree display and show include files:



```
Debug
  Common Sources
    COMMON.C
      stdio.h
      common.h
    TUTOR.C
      tutor.h
      common.h
```

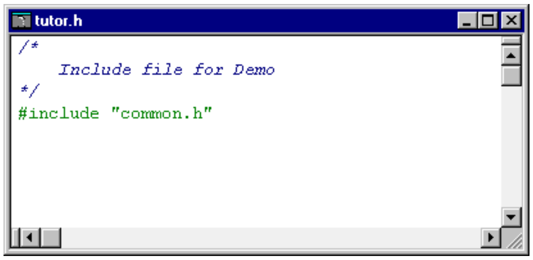
Open the file `tutor.h` by double-clicking its name in the Project window:



```
#include "common.h"
```

Add a comment, for example:

```
/*
Include file for Demo
*/
```



```
/*
    Include file for Demo
*/
#include "common.h"
```

Save and close the file.

MAKING THE PROJECT

To bring the project up to date choose **Make** from the **Project** menu, or click the **Make** button in the toolbar.

The source file `tutor.c` will be recompiled, because the include file it references has been changed, and the whole project will then be linked.

The source file `common.c` will not be recompiled since it does not depend on the `tutor.h` header file.

WHAT NEXT?

That completes this brief guided tour of the Embedded Workbench.

For more detailed information about using the Embedded Workbench and Embedded Workbench editor, refer to the *Reference* chapter.

For more information about using the Embedded Workbench tools refer to the *PICmicro™ C Compiler Programming Guide* and the *PICmicro™ Assembler, Linker, and Librarian Programming Guide*.

Reference

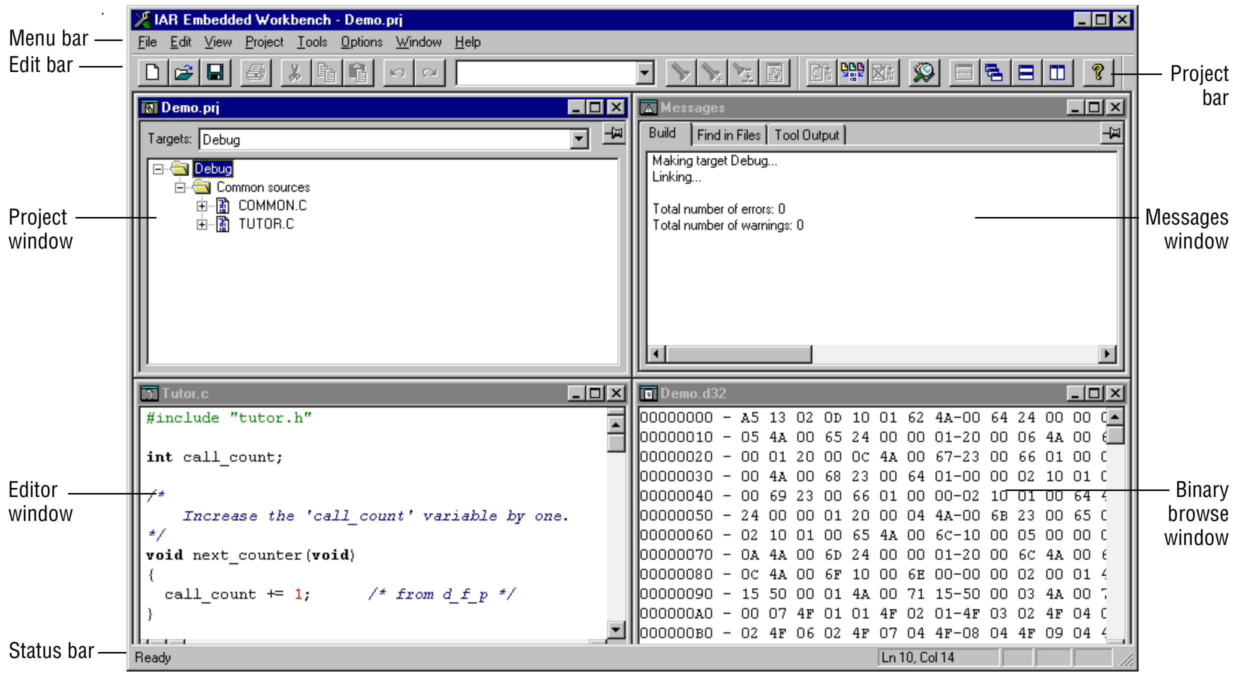
This chapter provides complete reference information about the PICmicro™ Embedded Workbench.

It first gives information about the components of the Workbench window, and each of the different types of window it encloses.

It then gives details of the menus, and the commands on each menu.

The Embedded Workbench window

The following illustration shows the different components of the Embedded Workbench window.



```
Making target Debug...
Linking...

Total number of errors: 0
Total number of warnings: 0
```

```
#include "tutor.h"

int call_count;

/*
    Increase the 'call_count' variable by one.
*/
void next_counter(void)
{
  call_count += 1;       /* from d_f_p */
}
```

```
00000000 - A5 13 02 0D 10 01 62 4A-00 64 24 00 00
00000010 - 05 4A 00 65 24 00 00 01-20 00 06 4A 00
00000020 - 00 01 20 00 0C 4A 00 67-23 00 66 01 00
00000030 - 00 4A 00 68 23 00 64 01-00 00 02 10 01
00000040 - 00 69 23 00 66 01 00 00-02 10 01 00 64
00000050 - 24 00 00 01 20 00 04 4A-00 6B 23 00 65
00000060 - 02 10 01 00 65 4A 00 6C-10 00 05 00 00
00000070 - 0A 4A 00 6D 24 00 00 01-20 00 6C 4A 00
00000080 - 0C 4A 00 6F 10 00 6E 00-00 00 02 00 01
00000090 - 15 50 00 01 4A 00 71 15-50 00 03 4A 00
000000A0 - 00 07 4F 01 01 4F 02 01-4F 03 02 4F 04
000000B0 - 02 4F 06 02 4F 07 04 4F-08 04 4F 09 04
```

These components are explained in greater detail in the following sections.

MENU BAR

Gives access to the Embedded Workbench menus.

Menu	*Description*
File	The **File** menu provides commands for opening projects and source files, saving and printing, and exiting from the Embedded Workbench.
Edit	The **Edit** menu provides commands for editing and searching in editor windows.
View	The commands on the **View** menu allow you to change the information displayed in the Embedded Workbench window.
Project	The **Project** menu provides commands for adding files to a project, creating groups, and running the IAR tools on the current project.
Tools	The **Tools** menu is a user-configurable menu to which you can add tools for use with the Embedded Workbench.
Options	The **Options** menu allows you to customize the Embedded Workbench to your requirements.
Window	The commands on the **Window** menu allow you to manipulate the Embedded Workbench windows and change their arrangements on the screen.
Help	The commands on the **Help** menu provides help about the Embedded Workbench.

The menus are described in greater detail on the following pages.

TOOLBARS

The Embedded Workbench window contains two toolbars:

- The edit bar.
- The project bar.

The edit bar provides buttons for the most useful commands on the Embedded Workbench menus, and a text box for entering a string to do a toolbar search.

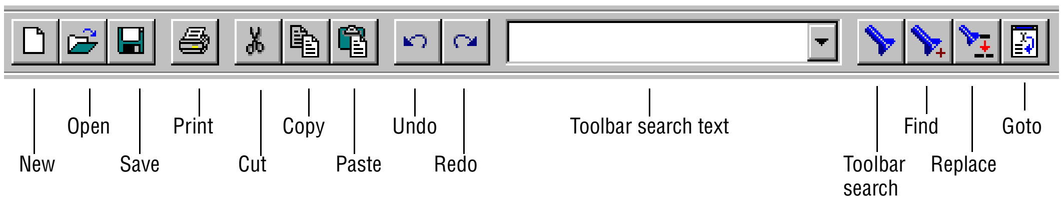
The project bar provides buttons for the build and debug options on the **Project** menu.

You can move either toolbar to a different position in the Embedded Workbench window, or convert it to a floating palette, by dragging it with the mouse.

You can display a description of any button by pointing to it with the mouse button. When a command is not available the corresponding toolbar button will be grayed out, and you will not be able to select it.

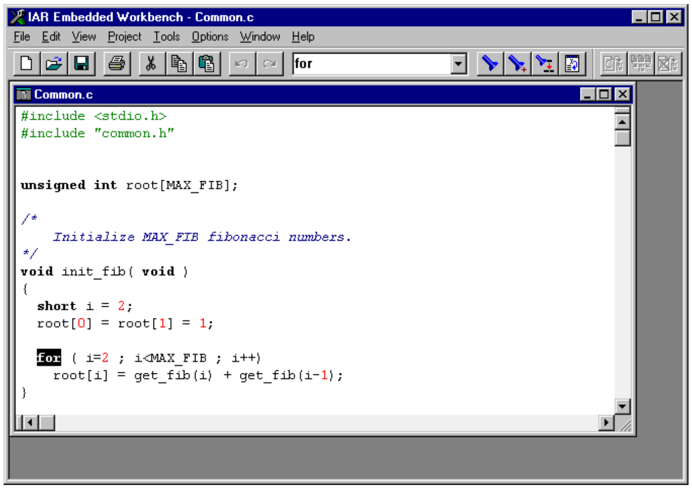
Edit bar

The following illustration shows the menu commands corresponding to each of the edit bar buttons:



Toolbar search

To search for text in the frontmost editor window enter the text in the **Toolbar search** text box, and press ↵ or click the **Toolbar search** button.


IAR Embedded Workbench - Common.c
File Edit View Project Tools Options Window Help
for
Common.c

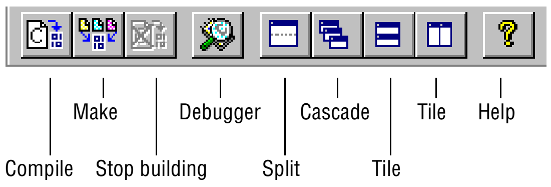
```
#include <stdio.h>
#include "common.h"


unsigned int root[MAX_FIB];

/*
    Initialize MAX_FIB fibonacci numbers.
*/
void init_fib( void )
{
  short i = 2;
  root[0] = root[1] = 1;

  for ( i=2 ; i<MAX_FIB ; i++)
    root[i] = get_fib(i) + get_fib(i-1);
}
```

Alternatively, you can select a string you have previously searched for from the drop-down list box.

You can choose whether or not the edit bar is displayed using the **Edit Bar** command on the **View** menu.

Project bar

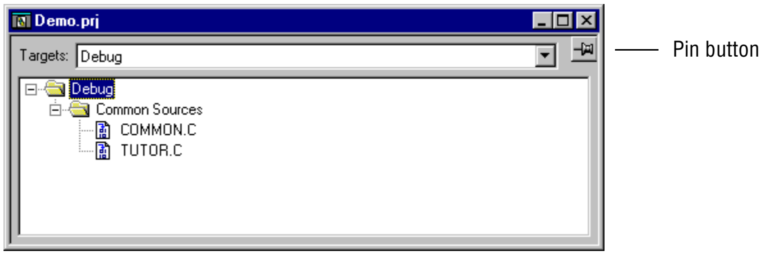
The following illustration shows the menu command corresponding to each of the project bar buttons:

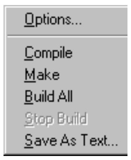


You can choose whether or not the project bar is displayed using the **Project Bar** command on the **View** menu.

PROJECT WINDOW

The Project window shows the name of the current project and a tree representation of the groups and files included in the project.





Pressing the right mouse button in the Project window displays a pop-up menu which gives you convenient access to several useful commands. **Save As Text...** allows you to save a textual description of the project, including all options that you have specified.

Pin button

The **Pin** button, in the top right corner of the Project window, allows you to pin the window to the desktop so that it is not affected by the **Tile** or **Cascade** commands on the **Window** menu.

Targets

The top node in the tree shows the current target. You can change the target by choosing a different target from the **Targets** drop-down list box at the top of the Project window. Each target corresponds to a different version of your project that you want to compile or assemble. For example, you might have a target called **Debug**, which includes debugging code, and one called **Release**, with the debugging code omitted.

You can expand the tree by double-clicking on the target icon, or by clicking on the ⊞ symbol, to display the groups included in this target.

Groups

Groups are used to collect together related source files. Each group may be included in one or more targets, and a source file can be present in one or more groups.

Source files
You can expand each group by double-clicking on its icon, or by clicking on the ⊞ symbol, to show the list of source files it contains.

Once a project has been successfully built any include files are displayed in the structure below the source file that included them. Note that the include files associated with a particular source file may depend on which target the source file appears in, since preprocessor or directory options may affect which include files are associated with a particular source file.

Editing a file
To edit a source or include file, double-click its icon in the Project window tree display.

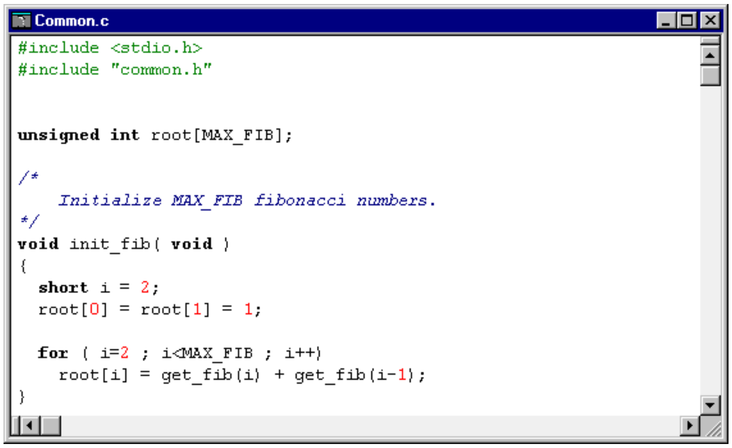
Moving a source file between groups
You can move a source file between two groups by dragging its icon between the group icons in the Project window tree display.

Removing items from a project
To remove an item from a project, click on it to select it, and then press Delete.

To remove a file from a project you can also use the **Project Files** dialog box, displayed by choosing **Files…** from the **Project** menu.

EDITOR WINDOW

Source files are displayed in the editor window. The Embedded Workbench editor automatically recognizes the syntax of C programs, and displays the different components of the C program in different text styles.



```
#include <stdio.h>
#include "common.h"


unsigned int root[MAX_FIB];

/*
    Initialize MAX_FIB fibonacci numbers.
*/
void init_fib( void )
{
  short i = 2;
  root[0] = root[1] = 1;

  for ( i=2 ; i<MAX_FIB ; i++)
    root[i] = get_fib(i) + get_fib(i-1);
}
```

The following table shows the default styles used for each component of a C program:

Item	*Style*
Default	Black plain
C Keyword	Black bold
Strings	Blue
Preprocessor	Green
Integer (dec)	Red
Integer (oct)	Magenta
Integer (hex)	Magenta
Real	Blue
C + +	Dark blue italic
Comment	Dark blue italic

To change these styles choose **Settings...** from the **Options** menu, and then select the **Colors and Fonts** page in the **Settings** dialog box, see *Colors and Fonts*, page 71.

Note that the Embedded Workbench editor can also be used for editing plain text files, such as Read-Me files.

Auto indent

The editor automatically indents a line to the same indent as the previous line, making it easy to lay out programs in a structured way.

Matching brackets

When the cursor is next to a bracket you can automatically find the matching bracket by choosing **Match Brackets** from the **Edit** menu.

Read-only and modification indicators

The name of the open source file is displayed in the editor window title bar.

When a file has been modified after it was last saved, an asterisk appears after the the window title, for example `Common *`.

If a file is read-only, the text `(Read Only)` appears after the file name, for example `Common (Read Only)`.

Editor options

The Embedded Workbench editor provides a number of special features, each of which can be enabled or disabled independently in the **Editor** page of the **Settings** dialog box. For more information see *Editor*, page 67.

Editor key summary

The following tables summarize the editor's keyboard commands:

Use the following keys and key combinations for moving the insertion point:

To move the insertion point	*Press*
One character left	←
One character right	→
One word left	Ctrl ←
One word right	Ctrl →
One line up	↑
One line down	↓
To the start of the line	Home
To the end of the line	End
To the first line in the file	Ctrl Home
To the last line in the file	Ctrl End

Use the following keys and key combinations for scrolling text:

To scroll	*Press*
Up one line	Ctrl ↑
Down one line	Ctrl ↓
Up one page	PgUp
Down one page	PgDn

Use the following key combinations for selecting text:

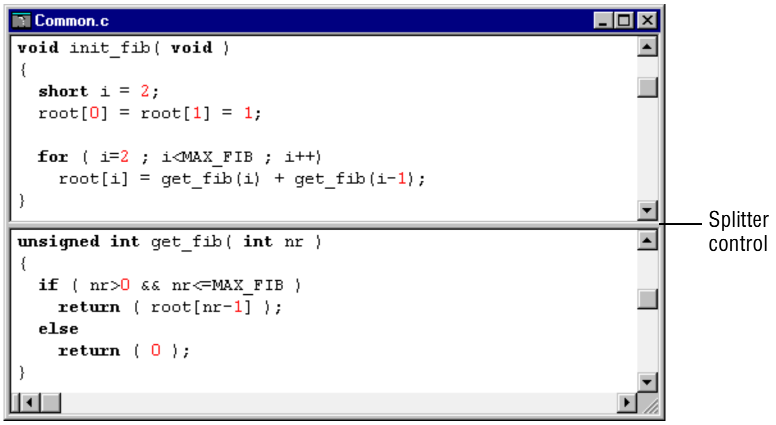
To select	*Press*
The character to the left	Shift ←
The character to the right	Shift →
One word to the left	Shift Ctrl ←
One word to the right	Shift Ctrl →

To select	*Press*
To the same position on the previous line	Shift ↑
To the same position on the next line	Shift ↓
To the start of the line	Shift Home
To the end of the line	Shift End
One screen up	Shift PgUp
One screen down	Shift PgDn
To the beginning of the file	Shift Ctrl Home
To the end of the file	Shift Ctrl End

Splitting the editor window into panes

You can split the editor window horizontally or vertically into multiple panes, to allow you to look at two different parts of the same source file at once, or cut and paste text between two different parts.

To split the window drag the appropriate **splitter** control to the middle of the window:



```
void init_fib( void )
{
  short i = 2;
  root[0] = root[1] = 1;

  for ( i=2 ; i<MAX_FIB ; i++)
    root[i] = get_fib(i) + get_fib(i-1);
}
```

```
unsigned int get_fib( int nr )
{
  if ( nr>0 && nr<=MAX_FIB )
    return ( root[nr-1] );
  else
    return ( 0 );
}
```

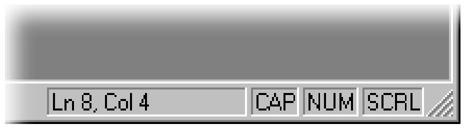
To revert to a single pane double-click the appropriate splitter control, or drag it back to the end of the scroll bar.

You can also split a window into panes using the **Split** command on the **Window** menu.

STATUS BAR

Displays the status of the Embedded Workbench, and the state of the modifier keys.

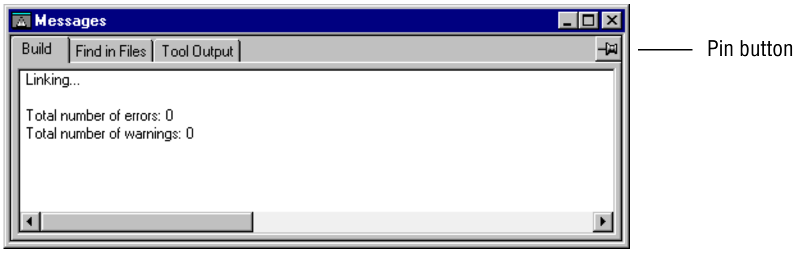
As you are editing in the editor window the status bar shows the current line and column number containing the cursor, and the CapsLock, NumLock, and ScrollLock status:



You can choose whether or not the status bar is displayed using the **Status Bar** command on the **View** menu.

MESSAGES WINDOW

The Messages window shows the output from different Embedded Workbench commands. The window is divided into multiple pages and you select the appropriate page by clicking on the corresponding tab.



```
Linking...

Total number of errors: 0
Total number of warnings: 0
```

Pressing the right mouse button in the Messages window displays a pop-up menu which allows you to save the contents of the window as a text file.

Pin button

The **Pin** button, in the top right corner of the Messages window, allows you to pin the window to the desktop so that it is not affected by the **Tile** or **Cascade** commands on the **Window** menu.

Build

Build shows the messages generated when building a project. Double-clicking a message in the Build panel opens the appropriate file for editing, with the cursor at the correct position.

Find in Files

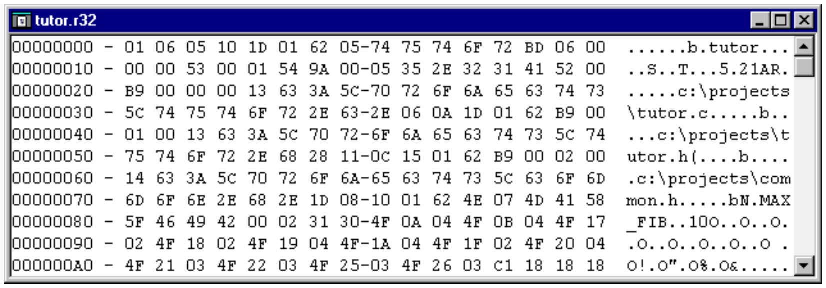
Find in Files displays the output from the **Find in Files…** command on the **Edit** menu. Double-clicking an entry in the panel opens the appropriate file with the cursor positioned at the correct location.

Tool Output

Tool Output displays any messages output by user-defined tools in the **Tools** menu.

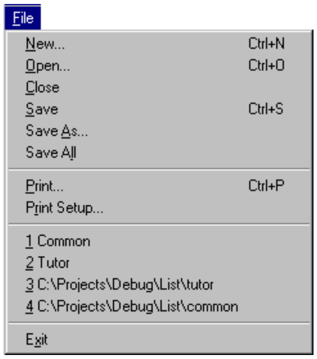
BINARY BROWSE WINDOW

The Binary Browse window displays the contents of a binary file as hexadecimal data, with its **ASCII** equivalent to the right of each line. To display binary data choose **Browse Binary Data…** from the **Tools** menu.



```
00000000 - 01 06 05 10 1D 01 62 05-74 75 74 6F 72 BD 06 00   ......b.tutor...
00000010 - 00 00 53 00 01 54 9A 00-05 35 2E 32 31 41 52 00   ..S..T...5.21AR.
00000020 - B9 00 00 00 13 63 3A 5C-70 72 6F 6A 65 63 74 73   .....c:\projects
00000030 - 5C 74 75 74 6F 72 2E 63-2E 06 0A 1D 01 62 B9 00   \tutor.c.....b..
00000040 - 01 00 13 63 3A 5C 70 72-6F 6A 65 63 74 73 5C 74   ...c:\projects\t
00000050 - 75 74 6F 72 2E 68 28 11-0C 15 01 62 B9 00 02 00   utor.h(....b....
00000060 - 14 63 3A 5C 70 72 6F 6A-65 63 74 73 5C 63 6F 6D   .c:\projects\com
00000070 - 6D 6F 6E 2E 68 2E 1D 08-10 01 62 4E 07 4D 41 58   mon.h.....bN.MAX
00000080 - 5F 46 49 42 00 02 31 30-4F 0A 04 4F 0B 04 4F 17   _FIB..10O..O..O.
00000090 - 02 4F 18 02 4F 19 04 4F-1A 04 4F 1F 02 4F 20 04   .O..O..O..O..O .
000000A0 - 4F 21 03 4F 22 03 4F 25-03 4F 26 03 C1 18 18 18   O!.O".O%.O&.....
```

FILE MENU

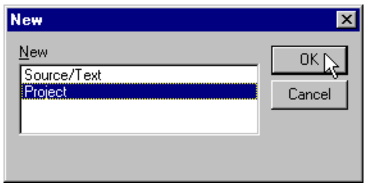


The File **menu** provides commands for opening projects and source files, saving and printing, and exiting from the Embedded Workbench.

The menu also includes a numbered list of the most recently opened files to allow you to open one by selecting its name from the menu.

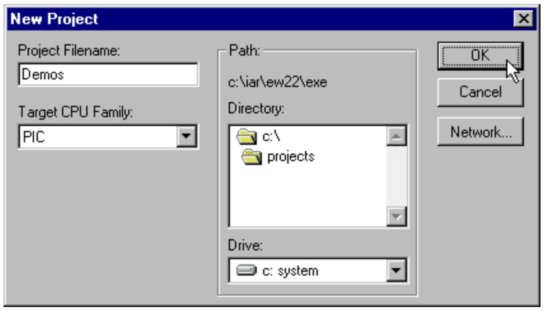
NEW...

Displays the following dialog box to allow you to specify whether you want to create a new project, or a new text file:



Choosing **Source/Text** opens a new Editor window to allow you to enter a text file.

Choosing **Project** displays the following dialog box to allow you to specify a name for the project and the target CPU family:



The project will then be displayed in a new Project window.

By default new projects are created with two targets, **Release** and **Debug**.

OPEN...

Displays a standard **Open** dialog box to allow you to select a text or project file to open. Opening a new project file automatically closes and saves any currently-open project.

CLOSE

Closes the active window.

You will be warned if a text document has changed since it was last saved, and given the opportunity to save it before closing. Projects are saved automatically.

SAVE

Saves the current text or project document.

SAVE AS…

Displays the standard **Save As** dialog box to allow you to save the active document with a different name

SAVE ALL

Saves all open text documents.

PRINT…

Displays the standard **Print** dialog box to allow you to print a text document.

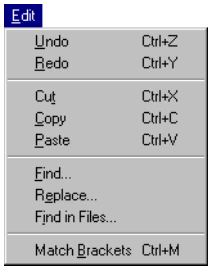
PRINT SETUP…

Displays the standard **Print Setup** dialog box to allow you to set up the printer before printing.

EXIT

Exits from the Embedded Workbench. You will be asked whether to save any changes to text windows before closing them. Changes to the project are saved automatically.

EDIT MENU



The **Edit** menu provides commands for editing and searching in editor windows.

UNDO

Undoes the last edit made to the current editor window.

REDO

Redoes the last **Undo** in the current editor window.

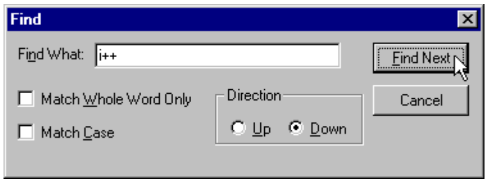
You can undo and redo an unlimited number of edits independently in each editor window.

CUT, COPY, PASTE

Provide the standard Windows functions for editing text within editor windows and dialog boxes.

FIND...

Displays the following dialog box to allow you to search for text within the current editor window:



Find What: i++

Enter the text to search for in the **Find What** text box.

Select **Match Whole Word Only** to find the specified text only if it occurs as a separate word. Otherwise `int` will also find `print`, `sprintf` etc.

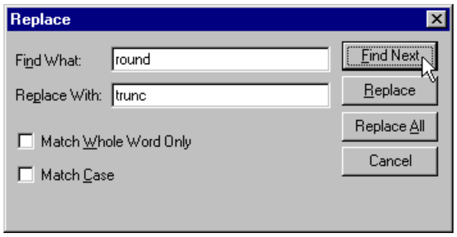
Select **Match Case** to find only occurrences that exactly match the case of the specified text. Otherwise specifying `int` will also find `INT` and `Int`.

Select **Up** or **Down** to specify the direction of the search.

Choose **Find Next** to find the next occurrence of the text you have specified.

REPLACE...

Allows you to search for a specified string and replace each occurrence with another string.

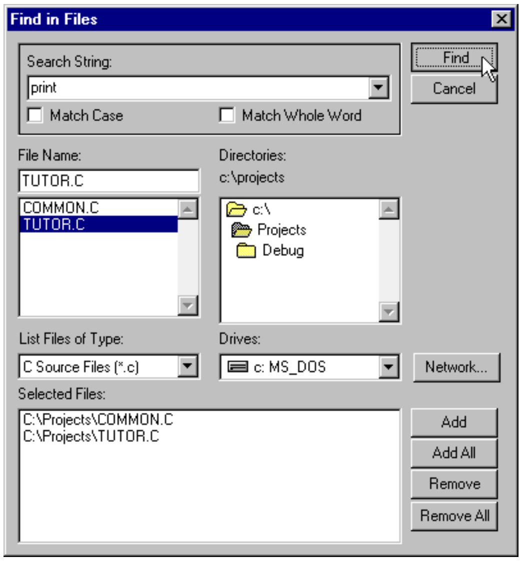


Enter the text to replace each found occurrence in the **Replace With** box. The other options are identical to those for **Find...**.

Choose **Find Next** to find the next occurrence, and **Replace** to replace it with the specified text. Alternatively choose **Replace All** to replace all occurrences in the current editor window.

FIND IN FILES...

Allows you to search for a specified string in multiple text files. The following dialog box allows you to specify the criteria for the search:



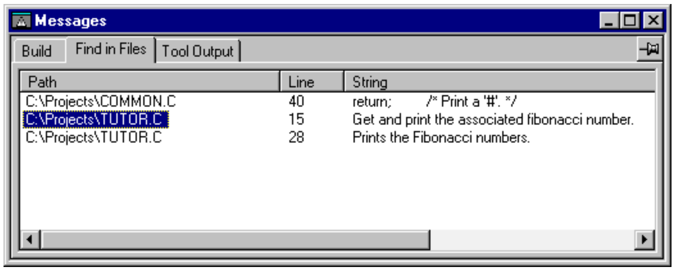
Specify the string you want to search for in the **Search String** text box, or select a string you have previously searched for from the drop-down list box.

Select **Match Whole Word** or **Match Case** to restrict the search to the occurrences that match as a whole word or match exactly in case, respectively.

Select each file you want to search in the **File Name** list, and choose **Add** to add it to the **Selected Files** list.

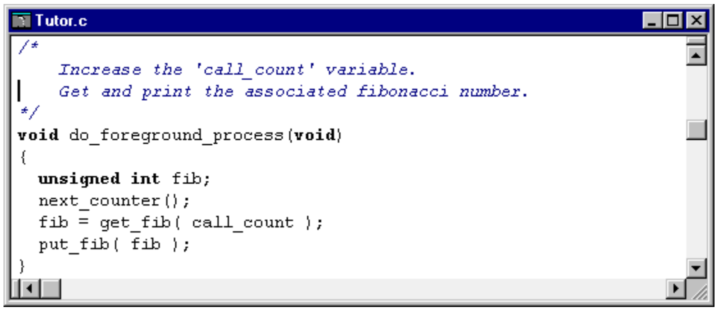
You can add all the files in the **File Name** list by choosing **Add All**, or you can select multiple files using the Shift and Ctrl keys and choose **Add** to add the files you have selected. Likewise you can remove files from the **Selected Files** list using the **Remove** and **Remove All** buttons.

When you have selected the files you want to search choose **Find** to proceed with the search. All the matching occurrences are listed in the Messages window. You can then very simply edit each occurrence by double-clicking it:



Path	Line	String
C:\Projects\COMMON.C	40	return; /* Print a '#'. */
C:\Projects\TUTOR.C	15	Get and print the associated fibonacci number.
C:\Projects\TUTOR.C	28	Prints the Fibonacci numbers.

This opens the corresponding file in an editor window with the cursor positioned at the start of the line containing the specified text:

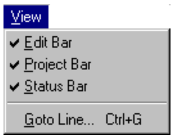


```
/*
    Increase the 'call_count' variable.
    Get and print the associated fibonacci number.
*/
void do_foreground_process(void)
{
  unsigned int fib;
  next_counter();
  fib = get_fib( call_count );
  put_fib( fib );
}
```

MATCH BRACKETS

If the cursor is positioned next to a bracket this command moves the cursor to the matching bracket, or beeps if there is no matching bracket.

VIEW MENU



The commands on the **View** menu allows you to change the information displayed in the Embedded Workbench window.

EDIT BAR

Toggles the edit bar on and off.

PROJECT BAR

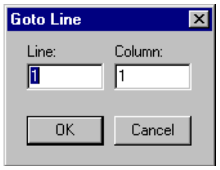
Toggles the project bar on and off.

STATUS BAR

Toggles the status bar on and off.

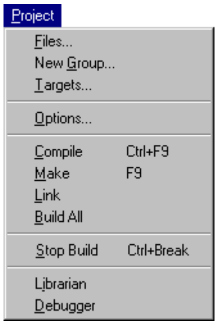
GOTO LINE...

Displays the following dialog box to allow you to move the cursor to a specified line and column in the current editor window:



Line: 1
Column: 1

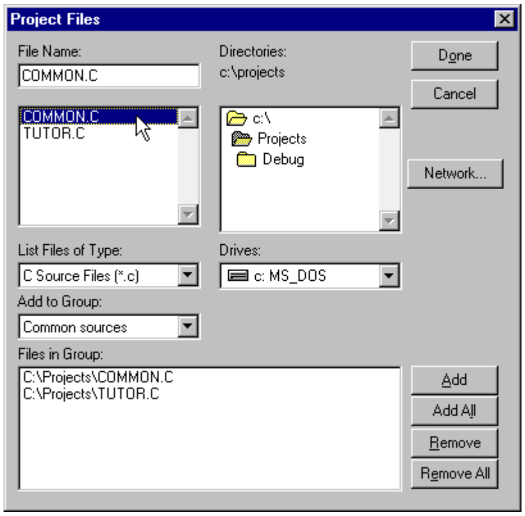
PROJECT MENU



The **Project** menu provides commands for adding files to a project, creating groups, and running the IAR tools on the current project.

FILES…

Displays the following dialog box to allow you to edit the contents of the current project:



File Name: COMMON.C

Directories: c:\projects

COMMON.C
TUTOR.C

c:\
Projects
Debug

List Files of Type: C Source Files (*.c)

Drives: c: MS_DOS

Add to Group: Common sources

Files in Group:
C:\Projects\COMMON.C
C:\Projects\TUTOR.C

The **Add to Group** drop-down list box shows all the groups included in the current target. Select the one you want to edit, and the files currently in that group are displayed in the **Files in Group** list at the bottom of the dialog box.

The upper part of the **Project Files** dialog box is a standard file dialog box, to allow you to locate and select the files you want to add to each particular group.

Adding files to a group

To add files to the currently displayed group select them using the standard file controls in the upper half of the dialog box and choose the **Add** button, or choose **Add All** to add all the files in the **File Name** list box.

Removing files from a group

To remove files from the currently displayed group select them in the **Files in Group** list and choose **Remove**, or choose **Remove All** to remove all the files from the group.

You can use the **Project Files** dialog box to make changes to several groups. Choosing **Done** will then apply all the changes to the project. Alternatively, choosing **Cancel** will discard all the changes and leave the project unaffected.

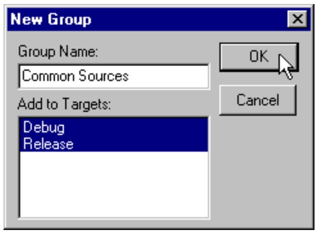
Source file paths

The Embedded Workbench supports relative source file paths to a certain degree.

If a source file is located in the project file directory or in any subdirectory of the project file directory, the Embedded Workbench will use a path relative to the project file when accessing the source file.

NEW GROUP...

Displays the following dialog box to allow you to create a new group:


New Group

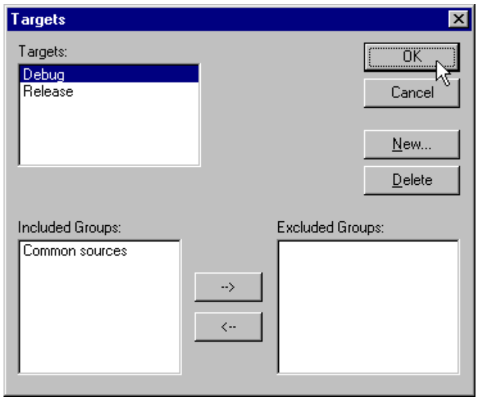
Group Name: Common Sources

Add to Targets: Debug, Release

Specify the name of the group you want to create in the **Group Name** text box. Select the targets to which you want to add the new group in the **Add to Targets** list. By default the group is added to all targets.

TARGETS…

Displays the following dialog box to allow you to create new targets, and display or change the groups included in each target:



To create a new target click **New…** and enter a name for the new target.

To delete a target select it and click **Delete**.

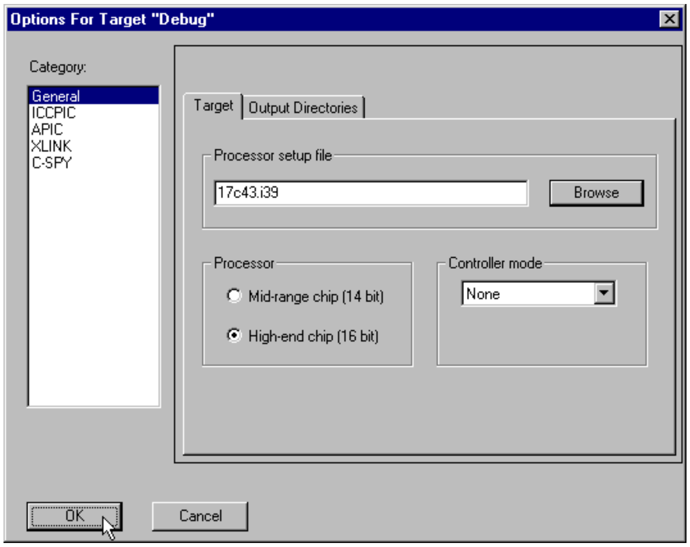
To view the groups included in a target select it in the **Targets** list.

The groups are shown in the **Included Groups** list, and you can add or remove groups using the arrow buttons:

OPTIONS...

Displays the **Options** dialog box to allow you to set directory and compiler options on the selected item in the Project window.

You can set options on the entire target, on a group of files, or on an individual file.



Processor setup file: 17c43.i39

Processor: Mid-range chip (14 bit) [✓ High-end chip (16 bit)]

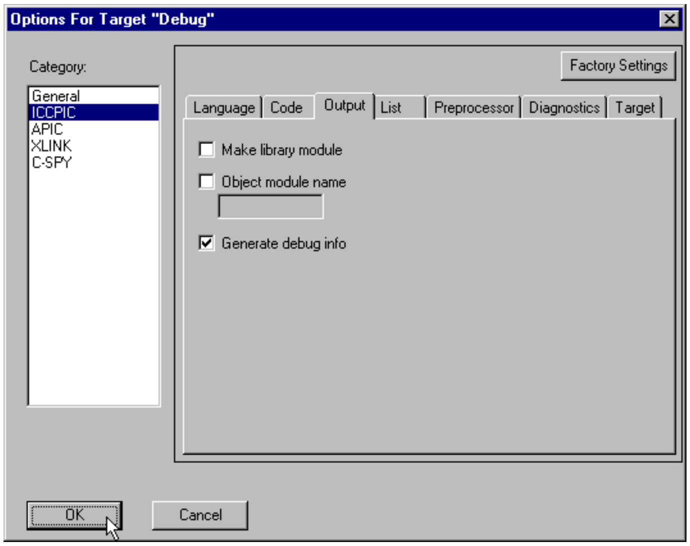
Controller mode: None

The **Category** list allows you to select which set of options you want to edit. The options available in the **Category** list will depend on the tools installed in your Embedded Workbench, and will typically include the following options:

Category	*Description*
General	General options.
ICCPIC	PICmicro™ C Compiler options.
APIC	PICmicro™ Assembler options.
XLINK	XLINK Linker options.
C-SPY (optional)	C-SPY options.

For more detailed information about the tools installed see the *PICmicro™ C Compiler Programming Guide, PICmicro™ Assembler, Linker, and Librarian Programming Guide* or the *PICmicro™ C-SPY User Guide.*

Selecting a category then displays one or more pages of options for that component of the Embedded Workbench:



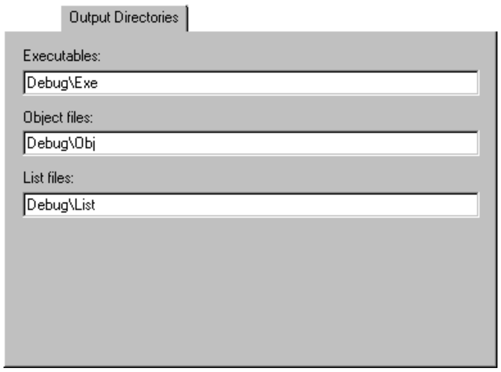
☐ Make library module
☐ Object module name
☑ Generate debug info

Click the appropriate tab to display the corresponding page of options.

Note that for all categories except **General** a **Factory Settings** button is available which resets all options for the selected category to the default settings for release and debug targets, respectively.

General options

The **General** category provides an **Output Directories** page, to allow you to specify the paths for executable, object, and list files:



Executables: Debug\Exe

Object files: Debug\Obj

List files: Debug\List

Other options

For details of the options available in the other categories refer to the *PICmicro™ C Compiler Programming Guide*, *PICmicro™ Assembler, Linker, and Librarian Programming Guide*, or the *PICmicro™ C-SPY User Guide*.

COMPILE

Compiles or assembles the currently active file as appropriate.

You can compile a file by selecting its icon in the Project window and choosing **Compile**. Alternatively, you can compile the file in the currently active editor window provided it is a member of the current target.

MAKE

Brings the current target up to date by compiling, assembling, and linking only those files as necessary.

LINK

Explicitly re-links the current target.

BUILD ALL

Marks all the components of the current target as having changed, and then runs Make to re-build and re-link all files in the current target.

STOP BUILD

Stops the current build operation.

LIBRARIAN

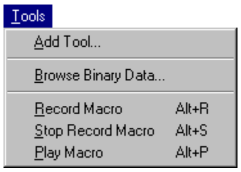
Runs the XLIB Librarian to allow you to perform operations on library modules in library files.

DEBUGGER

Runs the optional C-SPY for Windows debugger so that you can debug the project object file.

You can specify the version of C-SPY to run in the **Debug** options for the target. If necessary a Make will be performed before running C-SPY to ensure that the project is up to date.

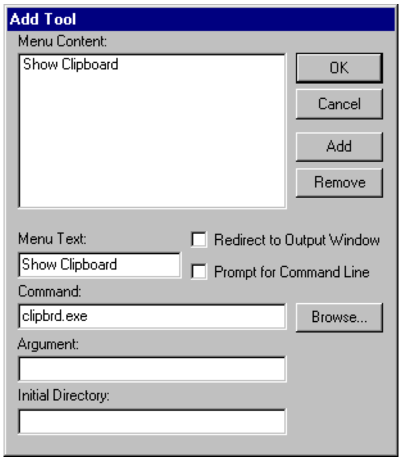
TOOLS MENU



The **Tools** menu is a user-configurable menu to which you can add tools for use with the Embedded Workbench.

ADD TOOL...

Add Tool... displays the following dialog box to allow you to specify a user-defined tool to add to the menu:


Add Tool

Menu Content: Show Clipboard

Menu Text: Show Clipboard

Command: clipbrd.exe

Argument:

Initial Directory:

Specify the text for the menu item in the **Menu Text** box, and the command to be run when you select the item in the **Command** text box. Alternatively, choose **Browse** to display a standard file dialog box to allow you to locate an executable file on disk and add its path to the **Command** text box.

Specify the argument for the command in the **Argument** text box, or select **Prompt for Command Line** to display a prompt for the command line argument when the command is selected from the **Tools** menu.

Variables can be used in the arguments, allowing you to set up useful tools like interfacing to a command-line revision control system, or running an external tool on the selected file.

The following variables can be used:

Variable	*Description*
`$FILE_PATH$`	Full path of active file (in editor, project, or message window)
`$FILE_FNAME$`	File name of active file without path
`$FILE_DIR$`	Directory of active file, no file name
`$CUR_LINE$`	Current line
`$TARGET_PATH$`	Full path of primary output file
`$TARGET_FNAME$`	Filename without path of primary output file
`$TARGET_DIR$`	Directory of primary output file
`$EXE_DIR$`	Directory for exe output
`$OBJ_DIR$`	Directory for obj output
`$LIST_DIR$`	Directory for list output
`$PROJ_PATH$`	Full path of project file
`$PROJ_FNAME$`	Project file name without path
`$PROJ_DIR$`	Project directory
`$CUR_DIR$`	Current directory

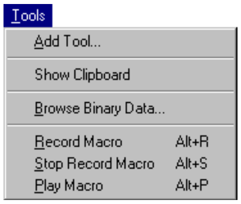
The **Initial Directory** text box allows you to specify an initial working directory for the tool.

Select **Redirect to Output Window** to display any console output from the tool in the Tools window. Note that tools that require user input or make special assumptions regarding the console that they execute in, will *not* work if you set this option.

When you have specified the command you want to add choose **Add** to add it to the **Menu Content** list. You can remove a command from the **Tools** menu by selecting it in this list and choosing **Remove**.

To confirm the changes you have made to the **Tools** menu and close the dialog box choose **OK**.

The menu items you have specified will then be displayed in the **Tools** menu:



Specifying MS-DOS commands or batch files
MS-DOS commands or batch files need to be run from a command shell, so to add these to the **Tools** menu you need to specify an appropriate command shell in the **Command** text box, and the MS-DOS command or batch file name in the **Argument** text box.

The command shells are specified as follows:

System	*Command shell*
Windows 95/98	`command.com`
Windows NT	`cmd.exe` (recommended) or `command.com`

The **Argument** text should be specified as:

`/C` *`name`*

where *`name`* is the name of the MS-DOS command or batch file you want to run.

The `/C` option terminates the shell after execution, to allow the Embedded Workbench to detect when the tool has completed.

For example, to add the command **Backup** to the **Tools** menu to make a copy of the entire `project` directory to a network drive, you would specify **Command** as `command` and **Argument** as:

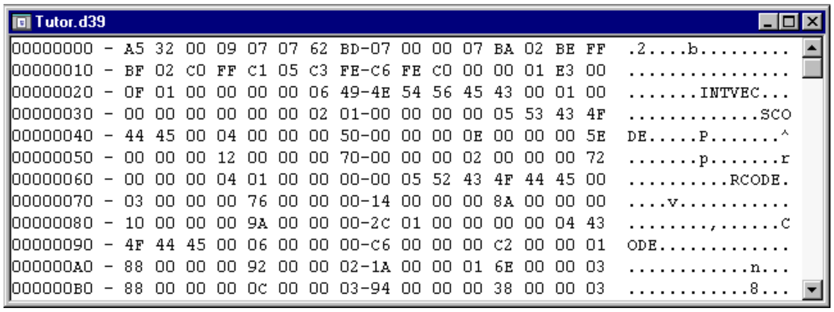
```
/C copy c:\project\*.* F:
```

or

```
/C copy $PROJ_DIR$\*.* F:
```

BROWSE BINARY DATA...

Lists a file in binary and ASCII format for browsing and debugging. It displays a standard file dialog box to allow you to select a file, and then displays it in a Browse window:



```
00000000 - A5 32 00 09 07 07 62 BD-07 00 00 07 BA 02 BE FF  .2....b.........
00000010 - BF 02 C0 FF C1 05 C3 FE-C6 FE C0 00 00 01 E3 00  ................
00000020 - 0F 01 00 00 00 00 06 49-4E 54 56 45 43 00 01 00  .......INTVEC...
00000030 - 00 00 00 00 00 00 02 01-00 00 00 00 05 53 43 4F  .............SCO
00000040 - 44 45 00 04 00 00 00 50-00 00 00 0E 00 00 00 5E  DE.....P.......^
00000050 - 00 00 00 12 00 00 00 70-00 00 00 02 00 00 00 72  .......p.......r
00000060 - 00 00 00 04 01 00 00 00-00 05 52 43 4F 44 45 00  ..........RCODE.
00000070 - 03 00 00 00 76 00 00 00-14 00 00 00 8A 00 00 00  ....v...........
00000080 - 10 00 00 00 9A 00 00 00-2C 01 00 00 00 00 04 43  ........,......C
00000090 - 4F 44 45 00 06 00 00 00-C6 00 00 00 C2 00 00 01  ODE.............
000000A0 - 88 00 00 00 92 00 00 02-1A 00 00 01 6E 00 00 03  ............n...
000000B0 - 88 00 00 00 0C 00 00 03-94 00 00 00 38 00 00 03  ............8...
```

Note that you cannot edit the contents of the Browse window.

RECORD MACRO

Allows you to record a sequence of editor keystrokes as a macro.

STOP RECORD MACRO

Ends the recording of a macro.

PLAY MACRO

Replays the macro you have recorded.

OPTIONS MENU

The **Options** menu **Settings…** command allows you to customize the Embedded Workbench to your own requirements.

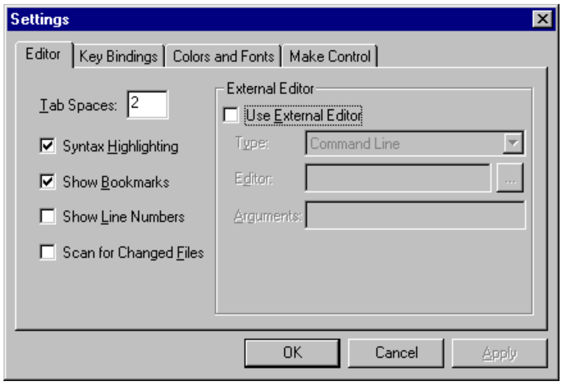
SETTINGS…

Displays the **Settings** dialog box to allow you to customize the Embedded Workbench.

Select the feature you want to customize by clicking the **Editor, Key Bindings, Colors and Fonts**, or **Make Control** tabs.

Editor

Allows you to change the editor options:



Tab Spaces: 2

[✓] Syntax Highlighting

[✓] Show Bookmarks

[ ] Show Line Numbers

[ ] Scan for Changed Files

[ ] Use External Editor

Type: Command Line

The **Editor** panel provides the following options:

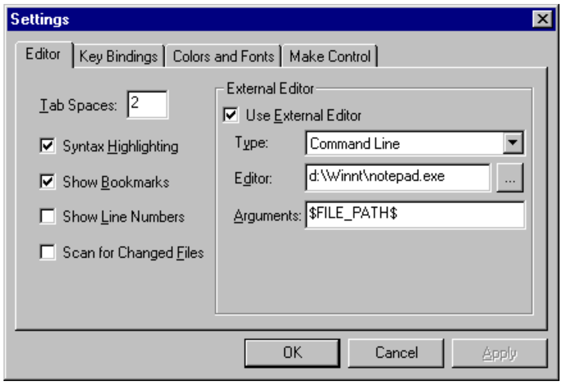
Options	*Descriptions*
Tab Spaces	Specifies the number of character spaces corresponding to each tab.
Syntax Highlighting	Displays the syntax of C programs in different text styles.
Show Bookmarks	Indicates compiler errors and **Find in Files…** search results.
Show Line Number	Displays line numbers in editor windows.

Options	*Descriptions*
Scan for Changed Files	The editor will check if files have been modified by some other tool and automatically reload them. If a file has been modified in the Embedded Workbench, you will be prompted first.
Use External Editor	Starts an external editor when your open a file in the project or Message window.

An external editor can be called either by passing command line parameters or by using DDE (Windows Dynamic Data Exchange).

Selecting **Type: Command Line** will call the external editor by passing command line parameters.

Provide the file name and path of your external editor in the **Editor** field. A browse button is available for your convenience. Then specify the command line to pass to the editor in the **Arguments** field, for example:



Tab Spaces: 2

[✓] Syntax Highlighting

[✓] Show Bookmarks

[ ] Show Line Numbers

[ ] Scan for Changed Files

External Editor

[✓] Use External Editor

Type: Command Line

Editor: d:\Winnt\notepad.exe

Arguments: $FILE_PATH$

Note that variables can be used in arguments, see page 64 for information about the argument variables available.

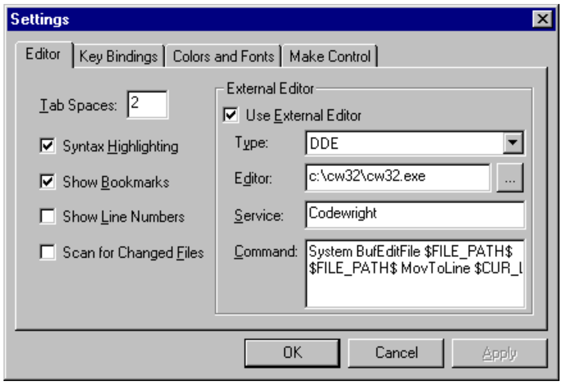
Selecting **Type: DDE** will call the external editor by using DDE.

Provide the file name and path of your external editor in the **Editor** field. A browse button is available for your convenience.

Specify the DDE service name used by the editor in the **Service** field. Then specify a sequence of command strings to send to the editor in the **Command** field. The command strings should be entered as:

```
DDE-Topic CommandString
DDE-Topic CommandString
```

as in the following example, which applies to Codewright:



Tab Spaces: 2
☑ Syntax Highlighting
☑ Show Bookmarks
☐ Show Line Numbers
☐ Scan for Changed Files

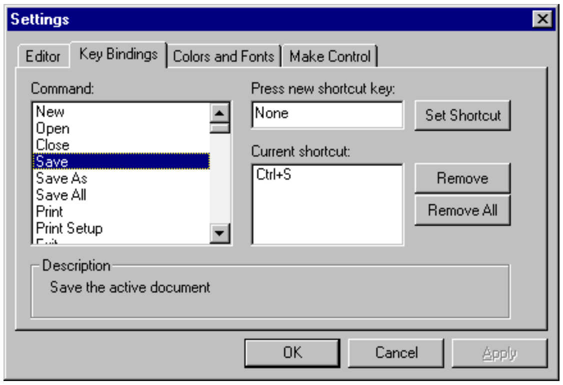
External Editor
☑ Use External Editor
Type: DDE
Editor: c:\cw32\cw32.exe
Service: Codewright
Command: System BufEditFile $FILE_PATH$ $FILE_PATH$ MovToLine $CUR_L

The service name and command strings depend on the external editor that you are using. Refer to the editor's user documentation to find the appropriate settings.

Note that variables can be used in the arguments, see page 64 for more information about the argument variables available.

Key Bindings

Displays the shortcut keys used for each of the menu options, and allows you to change them:



Command: New, Open, Close, Save, Save As, Save All, Print, Print Setup

Press new shortcut key: None

Current shortcut: Ctrl+S

Description: Save the active document

Select the command you want to edit in the **Command** list. Any currently defined shortcut keys are shown in the **Current shortcut** list.

To add a shortcut key to the command click in the **Press new shortcut key** box and type the key combination you want to use. Then click **Set Shortcut** to add it to the **Current shortcut** list. You will not be allowed to add it if it is already used by another command.

To remove a shortcut key select it in the **Current shortcut** list and click **Remove**, or click **Remove All** to remove all the command's shortcut keys.

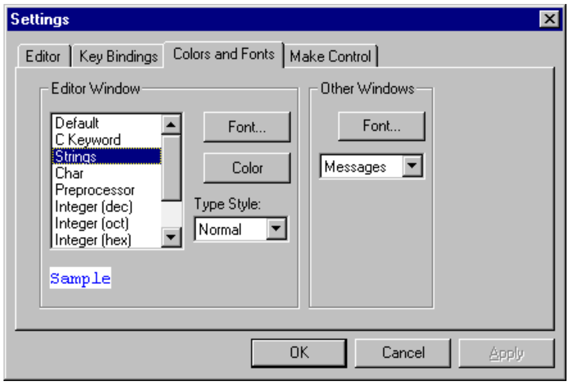
Then choose **OK** to use the new key bindings you have defined and the menus will be updated to show the shortcuts you have defined.

You can set up more than one shortcut for a command, but only one will be displayed in the menu.

Colors and Fonts

Allows you to specify the colors and fonts used for text in the editor windows, and the font used for text in the other windows.

The panel shows a list of the C syntax elements you can customize in the Editor window:



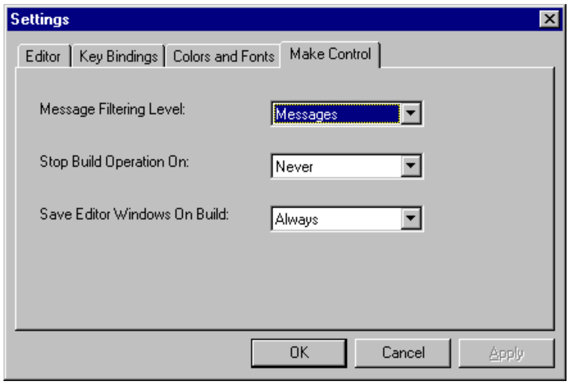
To specify the style used for each element of C syntax in the editor windows select the item you want to define from the **Editor Window** list. The current setting is shown in the **Sample** box below the list box.

You can choose a text color by clicking **Color**, and a font by clicking **Font...**. You can also choose the type style from the **Type Style** drop-down menu.

Then choose **OK** to use the new styles you have defined, or **Cancel** to revert to the previous styles.

Make Control

Allows you to set options for Make and Build:

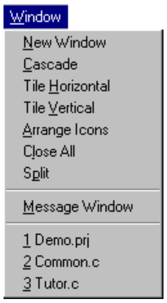


Message Filtering Level: Messages
Stop Build Operation On: Never
Save Editor Windows On Build: Always

The following table gives the options, and the alternative settings for each option:

Option	*Setting*
Message Filtering Level	All: Show all messages. Messages: Show message, warnings, and errors. Warnings: Show warnings and errors. Errors: Show errors only.
Stop Build Operation On	Never: Do not Stop. Warnings: Stop on warnings and errors. Errors: Stop on errors.
Save Editor Windows On Build	Always: Always save before Make or Build. Ask: Prompt before saving. Never: Do not save.

WINDOW MENU



The commands on the **Window** menu allow you to manipulate the Workbench windows and change their arrangement on the screen.

The last section of the **Window** menu lists the windows currently open on the screen, and allows you to activate one by selecting it.

NEW WINDOW

Opens a new window for the current file.

CASCADE, TILE HORIZONTAL, TILE VERTICAL

Provide the standard Windows functions for arranging the Embedded Workbench windows on the screen.

ARRANGE ICONS

Arranges minimized window icons neatly at the bottom of the Embedded Workbench window.

CLOSE ALL

Closes all open windows.

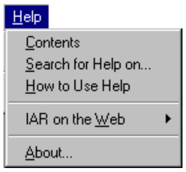
SPLIT

Allows you to split an editor window horizontally into two panes to allow you to see two parts of a file simultaneously.

MESSAGE WINDOW

Opens the Messages window which displays messages and text output from the Embedded Workbench commands.

HELP MENU



Provides help about the Embedded Workbench.

CONTENTS

Displays the Contents page for help about the Embedded Workbench.

SEARCH FOR HELP ON...

Allows you to search for help on a keyword.

HOW TO USE HELP

Displays help about using help.

IAR ON THE WEB

Allows you to browse the home page, news page, and FAQ page of the IAR web-site, and to contact IAR technical support.

ABOUT...

Displays the version number of the Embedded Workbench.

Symbols