COSMIC
Software

IDEA

Integrated Development Environment
for COSMIC Software
C Compilers and ZAP Debuggers

Quick Start Guide

PC/Windows 95/98/NT

Document Version V1.2 July 1999

This page intentionally left blank.

IDEA Quick Start Guide

- ◆ Overview
- ◆ Installing IDEA
- ◆ Running IDEA
- ◆ Managing a project
- ◆ Building a project
- ◆ Debugging a project

This page intentionally left blank.

Overview

Who is Cosmic Software?

Cosmic Software provides highly-optimized target support for Motorola microprocessors, including 68HC05, 68HC08, 6809, 68HC11, 68HC12, 68HC16, CPU32/CPU32+, and M680x0, with others in development.

The product line includes complete ANSI/ISO C language cross compilers, macro assemblers, linkers, utilities, ZAP C and assembler source-level cross debuggers, and the IDEA integrated development environment. These products are prepackaged and ready-to-run on PC/Windows and SUN SPARC/HP9000 UNIX workstations.

The compilers have been updated to Version 4 technology, which gives improved code optimization levels and improved language features for developers of embedded systems.

The ZAP debugger products are packaged to work off-the-shelf with popular debugging hardware configurations, such as low-cost evaluation boards or In-Circuit Emulators; the simulator versions of ZAP allows application code to be debugged entirely on a PC without access to target hardware, and can therefore simplify the development effort by providing for a "software debugging" phase before hardware/software integration.

The IDEA integrated development environment products provide a Windows-based graphical user interface (GUI) for building and managing projects. IDEA is fully integrated with all Cosmic tools, including compilers, assemblers, linkers, utilities, and ZAP debuggers.

What is IDEA?

IDEA is an integrated development environment and editor for managing cross development projects using Cosmic tools. IDEA is supplied in a target specific version customized to the

Cosmic tools you are using. For example, IDEA12 is designed for use with the Cosmic cross compiler and ZAP debugger for the Motorola MC68HC12. In order to run IDEA you must have the matching cross compiler installed on your system; the ZAP debugger is optional.

With IDEA you can create and edit projects; compile, assemble and link C or assembler code; run an application Make or Build; and run a ZAP debugger session; all with a few simple mouse clicks in a user-friendly, graphical Windows interface.

Using IDEA

The IDEA GUI (graphical user interface) provides immediate access to all the tools you need to manage full projects.
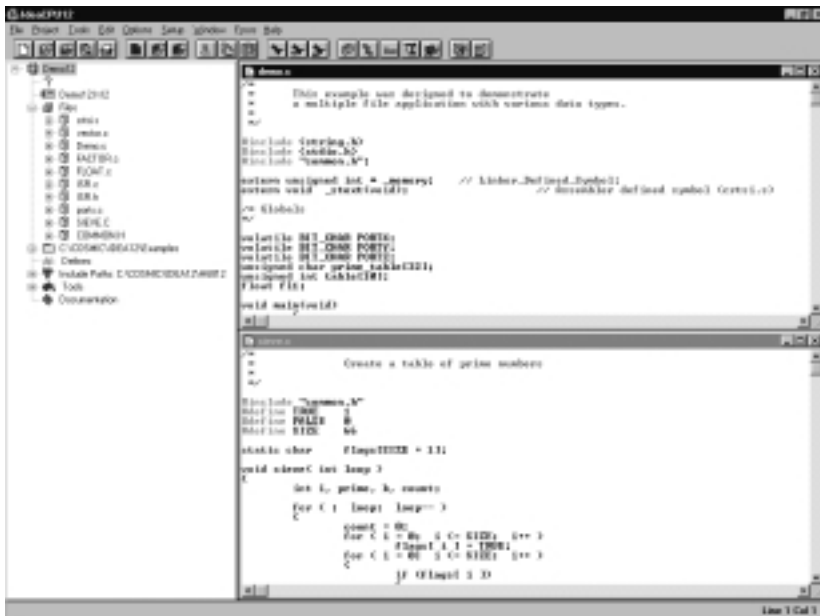
Figure 1-1: IDEA GUI

The **Project window** at the left provides a graphical, tree-structured view of your project. Using just the Project window, you can add or remove files from the project, set compiler options, configure build utilities, and much more.

The **File windows** at the right allow you to open project files for editing and compiling. IDEA provides color-coding of Comments, Preprocessor Keywords, C Keywords, and several other coding items so that you can easily edit source code files.

All IDEA functionality is available from the nine **drop-down menus** under the title bar. The most frequently used options are also available via a single click on the **Tool bar**. In addition, you can assign custom key bindings to any program option.

For detailed information on the IDEA GUI, refer to Chapter 4, *IDEA User Interface*, in the IDEA User's Manual.

Installing IDEA

Preparing for installation

Installation requirements

In order to run IDEA, your system **must** meet the following minimum hardware and software requirements:

- PC with an 80386 or better microprocessor
- Microsoft Windows 95/98 or Windows NT operating system
- 3 1/2", 1.44 Mb diskette drive
- Hard disk drive with at least 5 Mb of free space
- 8 Mb of RAM

Compiler requirements

IDEA is supplied in several different versions, with each version matched to a specific Cosmic Software C cross-compiler. In order to use IDEA, you **must** have the matching Cosmic Software C cross-compiler installed.

For example, IDEACPU12 requires the Cosmic MC68HC12 compiler. During installation, the software asks you to specify the path to the compiler.

Installation media

Your IDEA software package consists of the IDEA Integrated Development Environment program and installation script files, and is supplied on a single 3 1/2", 1.44 Mb floppy diskette. The diskette label identifies the product and the product version number.

Installation process

In the installation instructions that follow, we assume that your floppy disk drive is designated by the letter "A" and your hard disk partition by the letter "C". If your system uses different letter designations, change the installation instructions accordingly.

IDEA is installed by the Microsoft Windows Setup utility program. Throughout the installation procedure, there is an assumed default directory in which IDEA is installed. This directory is **C:\Cosmic\Idea***xx*, where *xx* stands for the Cosmic Software compiler that you are using (for example, **C:\Cosmic\Idea12** for the Cosmic MC68HC12 compiler).

If you install IDEA in a different directory or on a different hard disk drive, you must substitute your specified location wherever you see **C:\Cosmic\Idea***xx*.

Running the installation program

1. Insert the IDEA diskette into your floppy disk drive.

2. Open the **Windows Explorer** and in the left pane double click on "3 1/2 Floppy (A:)".

3. In the right pane, double click on **Setup.exe** to run the IDEA installation and setup program.

— NOTE —

As an alternative to Steps 2 and 3 you can use the RUN command from the Windows Start Menu and type a:\setup to run the installation program.

---

4. The **Welcome** screen appears.
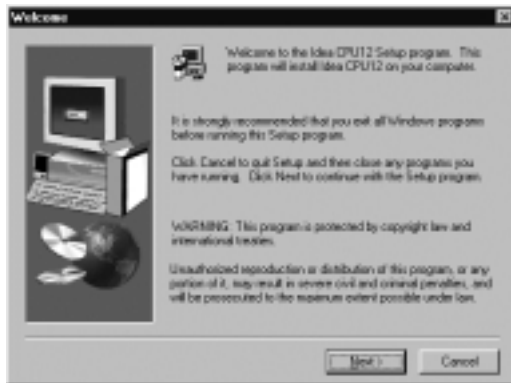
Figure 1-2: Welcome screen

Click on **Next>** when you are ready to proceed.

5. The **Software License Agreement** screen appears.

**Figure 1-3: Software License
Agreement screen**

After you read the **Software License Agreement**, click on **Yes** to
proceed.

6. The **User Information** screen appears.

Figure 1-4: User Information screen

Enter your name and company name and click on **Next>** to
proceed.

7. The **Choose Destination Folder** screen appears.

Figure 1-5: Choose Destination Folder dialog box

Select the destination folder where you want IDEA to be installed. By default, IDEA will be installed in the **C:\Cosmic \Idea***xx* folder, where *xx* stands for the Cosmic Software compiler that you are using (for example, **C:\Cosmic\Idea12** for the Cosmic MC68HC12 compiler).

You can select the default or click on the Browse button to specify a different location.

Click on **Next>** when you are ready to proceed.

8. The **Select Program Folder** screen appears.

Figure 1-6: Select Program Folder screen

Specify a program folder for IDEA, and click on **Next>** to proceed.

9. The **Compiler Directory** screen appears.

Figure 1-7: Compiler Directory screen

Select the folder where your C compiler is installed. You can select the default or click on the **Browse** button to specify a different location.

Click on **Next>** when you are ready to proceed.

10. The **Check Setup Information** screen appears.
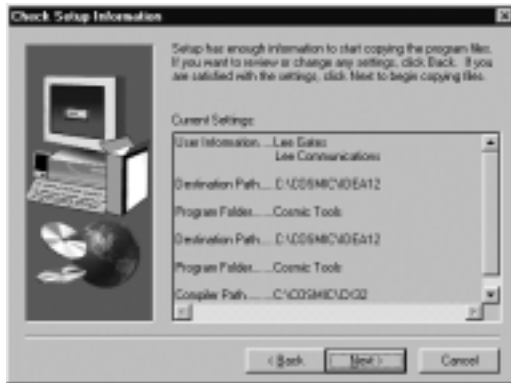
Figure 1-8: Check Setup Information screen

Check the setup information that you have provided. Click on **<Back** to make any changes. Click on **Next>** when to proceed.

11. The IDEA Setup program proceeds with the installation of IDEA. After the installation is complete, the **Setup Complete** screen appears.

Figure 1-9: Setup Complete screen

If you want to run IDEA immediately upon exiting the Setup program, select the checkbox "Do you wish to start IDEA now?".

Select **Finish** to complete the IDEA installation process.

Running IDEA

Starting IDEA

From the Windows Start menu, select **Programs > Cosmic Tools >
Idea***xx*, where *xx* stands for the Cosmic Software compiler that you
are using (for example, **Programs > Cosmic Tools > Idea12** if you
are using the Cosmic MC68HC12 compiler).

The IDEA main window appears:
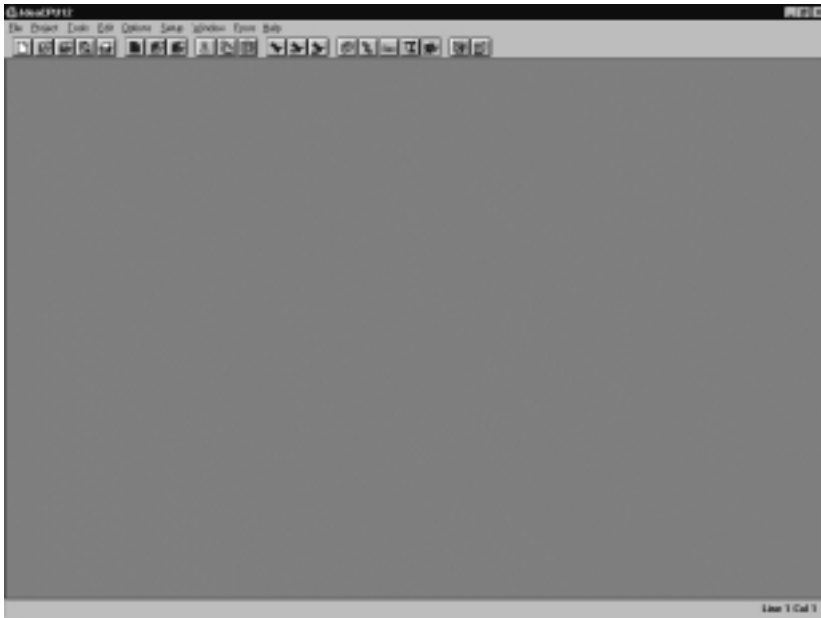
Figure 1-10: IDEA main window

After you open a project and some files within the project, the IDEA
main window appears as in the following Figure.

Figure 1-11: IDEA main window with Project and File windows open

The IDEA main window is the principal graphical user interface (GUI) for the program.

For complete details on the components of the IDEA main window, refer to Chapter 4, *IDEA User Interface* in the IDEA User's Guide.

Getting Help

The **Help** drop-down menu provides help on the C language and on the C Library. You can also view IDEA version information.

Open the **Help** drop-down menu by clicking on **Help** in the Main menu. Alternatively, type **Alt+H**.

Starting a new project

To start a new project, click on the **New Project** tool [image] on the Tool bar, or select **Project > New** from the Main menu.

The Project window appears with a new project opened.
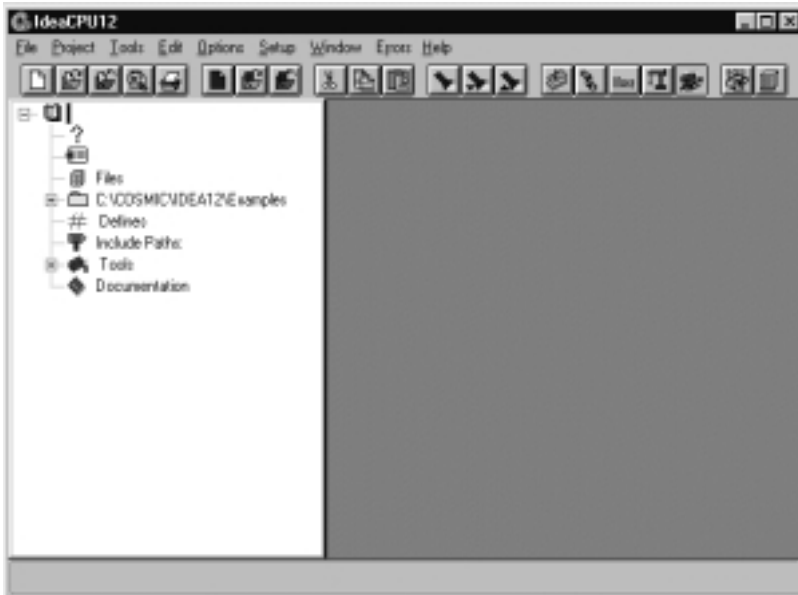
**Figure 1-12: IDEA Project window
with new project open**

The Project window displays the various project components as icons in a tree-structured format, similar to Windows Explorer. Each icon in the project tree represents a project component.

Table 1-1: Project Components

📖	Project Name
?	Project Description
📇	Project Target File Name
📑	Project Source Files
🗀	Project Directory
#	Project Defines
🚩	Project Include Paths
🔧	Project Tools
◆	Project Documentation

A ⊞ sign next to a component icon means that sub-components are hidden below the icon. Click on the ⊞ sign or double click on the icon to display the sub-components.

A ⊟ sign next to a component icon means that the first level of sub-components below the icon is displayed. Click on the ⊟ sign or double click on the icon to hide the sub-components.

For additional details on the Project window, refer to Chapter 4, *IDEA User Interface* in the IDEA User's Guide.

For details on project management, refer to Chapter 7, *Managing an IDEA Project* in the IDEA User's Guide.

Exiting IDEA

To exit IDEA, click on **Exit** in the **File** menu. Alternatively, type
Alt+**F**+**X**.

If you have selected **Auto Save before C/asm** in the **Options** drop-
down menu (**Alt**+**O**+**A**), all changed files are saved prior to exiting. If
you have not selected **Auto Save before C/asm**, a dialog box appears
in turn for each changed file and lets you select whether to save the
file or not.

Managing a Project

Opening the project

IDEA is supplied with an example project called "**demoxx.prj"**,
where **xx** stands for the Cosmic Software compiler that you are using
(for example, "**demo12.prj"** for the Cosmic MC68HC12 compiler).
You can use this example project to become familiar with the
principles of managing an IDEA project.

If you opened a new project earlier, save it by clicking on the **Save**

Project tool ![icon] on the Tool bar. Close the project by selecting
Project > Close from the Main menu.

To open the example project, click on the **Open Project** tool ![icon] on
the Tool bar. In the dialog box that appears, select demo12.prj from
the Examples folder. The Project window appears with the
demo12.prj project opened.
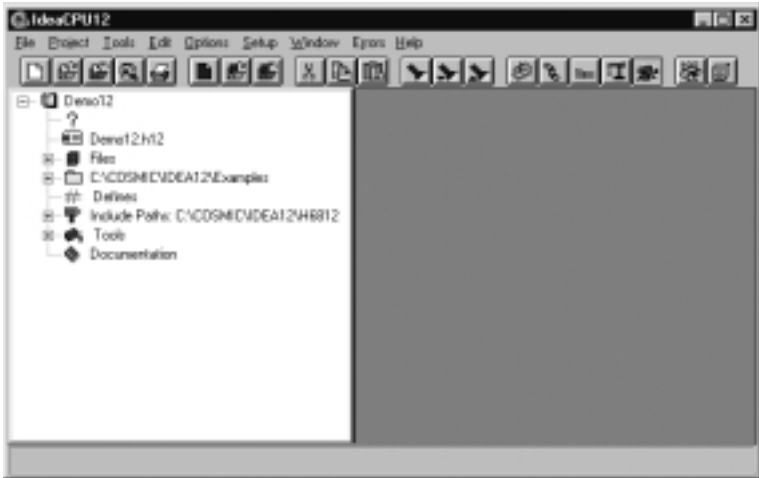
**Figure 1-13: Project pane with
demo12.prj project opened**

Naming the project

The **Project Name** component lets you specify a name for the
project. It also represents the parent component for all project sub-
components.

To specify a name for the project, click on the **Project Name**

icon ▯. A text cursor appears to the right of the icon. Click on the
text cursor to open a text box and enter a project name.

Right click on the **Project Name** icon to view a menu of project commands. These commands are shown in the following Table.

Table 1-2: Project commands

Add File	Adds a source file to the project.
Save	Saves the project.
Save As	Saves the project with a new name.
Make	Checks source file up-to-date status and dependencies. Then selectively compiles or assembles any out-of-date files and runs the Linker. The icons in the **Project Source Files** folder are colored yellow.
Build	Performs a **Make** as described above and then runs any utilities selected in the **Builder Configuration** dialog box. To have the **Build** rebuild all files regardless of their up-to-date status, right click on the project name, select **Mark All**, and then run the Builder.
Mark All	Marks all project source files for recompile/assemble without changing the file time/date stamp. The icons in the **Project Source Files** folder are colored orange.
Touch All	Marks all project source files for recompile/assemble and updates all project source files with the current system date and time stamp. The icons in the **Project Source Files** folder are colored red.
Documentation	Adds a document file to the project.

Describing the project

The **Project Description** component lets you specify a short description for the project.

To specify a description for the project, click on the **Project Description** icon ?. A text cursor appears to the right of the icon. Click on the text cursor to open a text box and enter a short project description.

Naming the project target file

The **Project Target File Name** component lets you specify a target file name for the project (for example, **demo12.h12**). This name is used as the root name for the linked executable.

To specify a project target file name, click on the **Project Target File Name** icon .

A text cursor appears to the right of the icon. Click on the text cursor to open a text box and enter a target file name. Be sure to include the target file name extension; for example, "**.h12**" for the Cosmic 68HC12 compiler.

Right click on the **Project Target File Name** icon to view a menu containing target file commands. These commands are shown in the following Table.

Table 1-3: Target File commands

Inspect Object	Runs the **Object Inspector** utility (*cobj*) on the target file.
Show Debug	Runs the **Debug Info Examiner** utility (*cprd*) and opens the project debug file in read-only mode.
Produce Hex Records	Runs the **Hex Converter** utility (*chex*), which translates executable images produced by the *clnk* linker to one of several hexadecimal interchange formats.
Produce Absolute Listings	Runs the **Absolute Lister** utility (*clabs*) to generate absolute listings.
Produce IEEE Output	Runs the **IEEE695 Converter** utility (*cv695*) to generate IEEE695 debug format.
Debug File	Runs the selected ZAP debugger and loads the linked executable.
Delete	Deletes the project target file. A pop-up dialog box asks you to confirm the deletion

Managing project source files

The **Project Source Files** component lets you specify the C and Assembly language source files to be included in the project.

Right click on the **Project Source Files** icon 🗐 to view a menu containing source file management commands. These commands are shown in the following Table.

Table 1-4: Source file management commands

Add File	Adds a source file to the project.
Touch All	Updates all project source files with the current system date and time stamp and marks them for recompile/assemble when a **Make** or **Build** is executed. The icons in the **Project Source Files** folder are colored red.
Mark All	Marks all project source files for recompile/assemble when **Make** or **Build** is executed. This option does not change the time-date stamp of the files. The icons in the **Project Source Files** folder are colored orange.

Adding source files to the project

You can add source files to the project using the **Add File**
command or the **Windows Explorer**.

- To add a source file using the **Add File** command, right
  click on the **Project Source Files** icon, select **Add File**,
  and select the file to add from the **Add File** dialog box.

- To add a source file using Windows Explorer, select the

  Windows Explorer tool 🔲 on the Tool bar. Windows
  Explorer appears next to the Project window. Select a file
  from Windows Explorer and drag it to the Project window.

Using either method, you can select more than one source file
at a time using standard Windows conventions for selecting and
grouping files.

Working with individual source files

Each source file included in the project is listed next to a

Source File icon 🔳. The **Source File** icon lets you view the
source file and its attributes.

Right click on the **Source File** icon to view a menu containing
source file commands. These commands are shown in the
following Table.

Table 1-5: Source file commands

Load (read only)	Opens the source file in read-only mode.
Open	Opens the source file for editing.
Remove	Removes the source file from the project.
Mark	Marks the source file for rebuilding. The **Source File** icon is colored orange.

Table 1-5: Source file commands

Touch	Updates the source file with the current system date and time stamp and marks it for rebuilding. The **Source File** icon is colored red and the **Source File Time Stamp** icon is updated with the new date and time.
Compile	Compiles or assembles the source file. The source file icon is colored yellow if the **Compile** is successful.
Options	Opens the **Compiler** (or **Assembler**) **Options for Source File** dialog box, where you can specify options for the source file.
Defines	Opens the **#defines** dialog box, where you can specify compiler define options for the source file.
Documen-tation	Adds a document file for the source file.

Each icon in the source file tree represents a source file component. The source file components are described in the following Table.

Table 1-6: Source File Components

	Source File Time Stamp
	Source File Documentation
	Source File Options
	Source File Defines
	Source File Dependencies
	Source File Functions Appears only if **Project Analysis** option is selected in **Options** sub-menu.
	Source File Variables Appears only if **Project Analysis** option is selected in **Options** sub-menu.

Source File Time Stamp

The **Source File Time Stamp** component and icon 🖫
shows the day, date, and time that the file was last saved
or "touched".

Source File Documentation

The **Source File Documentation** component shows all
documents that are associated with the source file.

Right click on the **Documentation** icon ◆ and select
Add Doc to associate a documentation file with the
source file.

The **Document** component lets you view, edit, or
remove a document associated with a source file. The

appearance of the **Document** icon 🖾 varies, depending
on the type of document.

Right-click on the **Document** icon to view a menu
containing documentation file commands. The
documentation file commands are described in the
following Table.

Table 1-7: Documentation file commands

Load (read only)	Opens the document in read-only mode.
Open	Opens the document for editing using the appropriate Windows-registered application.
Remove	Removes the document from the project.

Source File Options

The **Source File Options** component lets you specify compiler or assembler options for the source file. These options override the default project compiler or assembler options.

Right-click on the **Source File Options** icon 📧 to open the **Compiler** (or **Assembler**) **Options for Source File** dialog box.
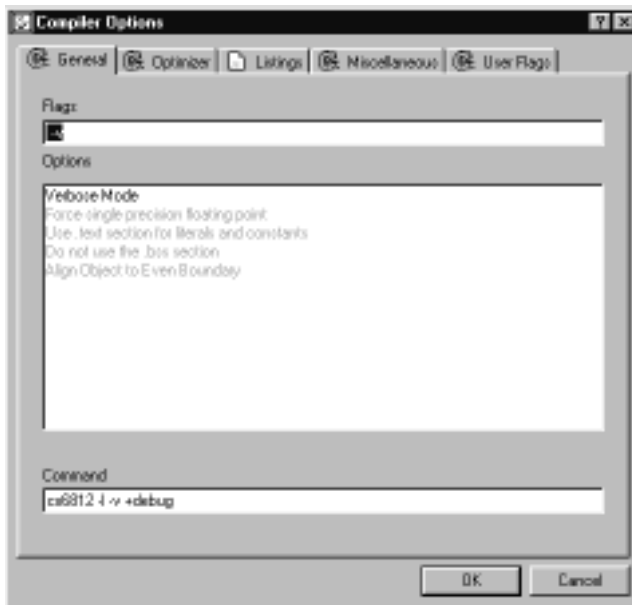
**Figure 1-14: Compiler Options
for Source File dialog box**

The **Compiler** (**Assembler**) **Options** dialog box has five tabs:

- **General options**
- **Optimizer options**
- **Listings options**
- **Miscellaneous options**
- **User Flags**

Choose a tab and select the desired options. Selected options are displayed in bold and unselected options are greyed out. To select an option simply click on the option description. To deselect an option click on the option again.

For a detailed explanation of compiler and assembler options, refer to Chapter 8, *IDEA Command Reference* in the IDEA User's Guide.

The source file compiler or assembler options will override the default project compiler or assembler options.

Refer to "Setting default compiler options" on page 1-37 or "Setting default assembler options" on page 1-38 for details.

Source File Defines

The **Source File Defines** component lets you specify compiler #define options for a source file.

Right-click on the **Source File Defines** icon # to open the **#defines** dialog box and specify up to twenty user-defined preprocessor symbols.

**Figure 1-15: Source File
#defines dialog box**

To add a symbol to the list, enter the symbol in the **Item** field and click on **Add**. To remove a symbol from the list, select the symbol and click on **Remove**.

You can also add project #defines to the source file #define list by clicking on **Add Project Defines**.

Refer to "Specifying Project Defines" on page 1-34 for details.

After you add #defines, they appear as individual sub-components in the **Defines** list, each one after a **Define** icon ᴰᴱꜰᴵɴᴱ. The define symbol is shown to the right of the icon. In addition, the day, date, and time that the #defines were last updated is shown next to the **Source File Defines** icon.

Source File Dependencies

The **Source File Dependencies** component and icon ⚓ let you view the files that are named in the source file #includes.

The **File** icon 📄 shows a file that is named in the source file #includes. If the file name is enclosed in brackets, it is a system include file and is not typically modified in each project.

If the file name is enclosed in quotes, it is a user include file and can be modified. Right-click on the **File** icon to display a menu include file commands. The include file commands are described in the following Table.

Table 1-8: Include file commands

Load (read only)	Opens the file in read-only mode.
Open	Opens the file for editing.
Touch	Updates the file with the current system date and time and marks it for rebuilding.

Source File Functions

The **Source File Functions** component and icon 📖 let you view the functions in the source file. The **Function**

icon ⟨⟩ shows a function in the source file and lists all of the variables local to that function. Right-click on the **Function** icon to open the source file at the function.

Source File Variables

The **Source File Variables** component and icon 📖 let you view the variables that are local to the source file.

The **Variable** icon 🎛 shows a variable declared with the source file.

```
————————— NOTE —————————

Functions and Variables appear in the Project
window only if Project Analysis is selected
from the Options menu.
```

Specifying the project directory

The **Project Directory** component is used to set the working directory for the project. This is typically where the source code files for the project are located.

Right-click on the **Project Directory** icon 📁 to open the **Path Editor** dialog box and set the path to the source files.
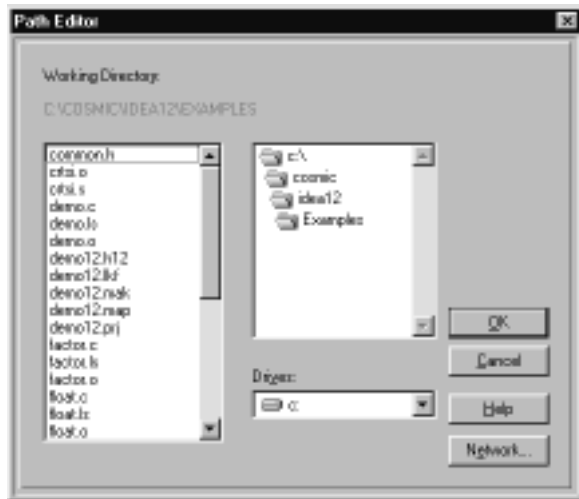
Figure 1-16: Path Editor dialog box

The **Folder** icon ▢ shows folders in the project directory.

The **File** icon 🗐 shows files in the project directory.

Right-click on the **File** icon to display a menu with file commands.
The file commands are described in the following Table.

Table 1-9: File commands

Load (read only)	Opens the file in read-only mode.
Open	Opens the file for editing.
Touch	Updates the file with the current system date and time and marks it for rebuilding.

Specifying Project Defines

The **Project Defines** component lets you specify #define options for the project.

Right-click on the **Project Defines** icon # to open the #defines dialog box and specify up to twenty preprocessor symbols.

Figure 1-17: Project #defines dialog box

To add a symbol to the list, enter the symbol in the **Item** field and click on **Add**. To remove a symbol, select it and click on **Remove**.

Symbols defined as project #defines can be imported into source file #defines. Refer to "Source File Defines" on page 1-29 for details.

After you add project #defines, they appear as individual sub-components in the **Defines** list, each one after a **Define** icon ^{DEFINE}.

The define symbol is shown to the right of the icon. In addition, the day, date, and time that the project #defines were last updated is shown next to the **Project Defines** icon.

Specifying project include paths

The **Project Include Paths** component lets you specify include paths for the compiler (**-i >** option).

Right-click on the **Project Include Paths** icon 🚩 to open the **Include Path Editor** dialog box and specify up to twenty include paths for the project.
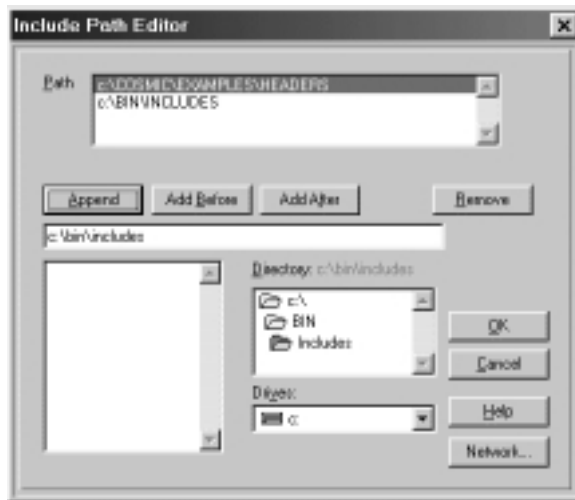
Figure 1-18: Include Path Editor

You can specify paths in any desired order. The paths are searched from top to bottom by the compiler.

You can use the **Drives** and **Directory** fields to specify the include path. Click on **Append** to add the path to the bottom of the list in the **Path** field.

To position the new path before or after the selected path, select an include path in the **Path** field and click on **Add Before** or **Add After**.

After you add include paths, they appear in order next to the **Project Include Paths** icon and as components in the **Project Include Paths** list, each one after a **Folder** icon.

Include path folders and files

The **Include Path Folder** icon 🗀 shows folders for include

file paths in the project directory. The **Folder** icon 🗀 shows
folders in an include file path.

The **File** icon 🖹 shows files in an include file path. Right-click
on the **File** icon to display a menu with the following
commands:

Table 1-10: Include file commands

Load (read only)	Opens the file in read-only mode.
Open	Opens the file for editing.
Touch	Updates the file with the current system date and time and marks it for rebuilding.

Configuring project tools

The **Project Tools** 🐾on lets you set project default options for:
- **Compiler**
- **Assembler**
- **Linker**
- **Builder**
- **Debugger**

The **Builder** component lets you configure build utilities for the
project, including:
- **Object Inspector (*cobj*)**
- **Hex Converter (*chex*)**
- **Debug Info Examiner (*cprd*)**
- **Absolute Lister (*clabs*)**
- **IEEE-695 Converter (*cv695*)**

Setting default compiler options

The **Compiler** component lets you set the default compiler options that are used to compile all C code (**.c**) files in a project.

Right click on the **Compiler** icon 🐾 to open the **Compiler Options** dialog box. You can also double-click on the

Compiler icon to display the **Compiler Options** icon 📧 and then right-click on the **Compiler Options** icon to open the **Compiler Options** dialog box.
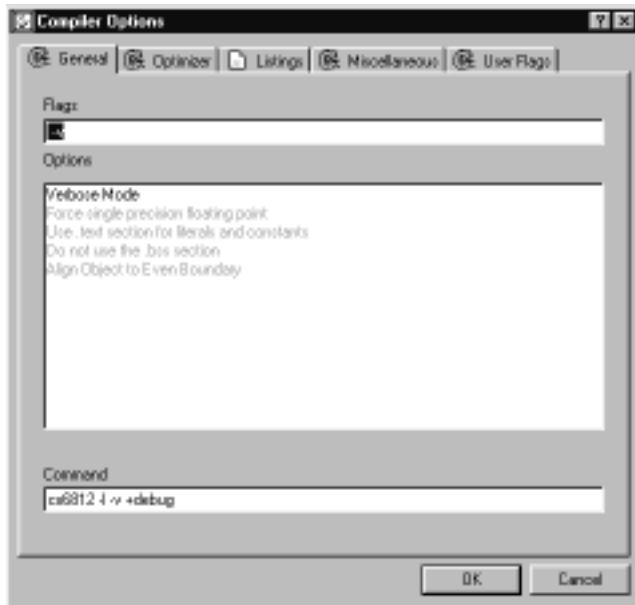

Figure 1-19: Compiler Options dialog box

The **Compiler Options** dialog box has five tabs:

- **General options**
- **Optimizer options**
- **Listings options**
- **Miscellaneous options**
- **User Flags**

Choose a tab and select the desired options. Selected options are displayed in bold and unselected options are greyed out. To select an option, simply click on the option description. To deselect an option, click on the option again.

For a detailed description of compiler options, refer to Chapter 8, *IDEA Command Reference* in the IDEA User's Guide.
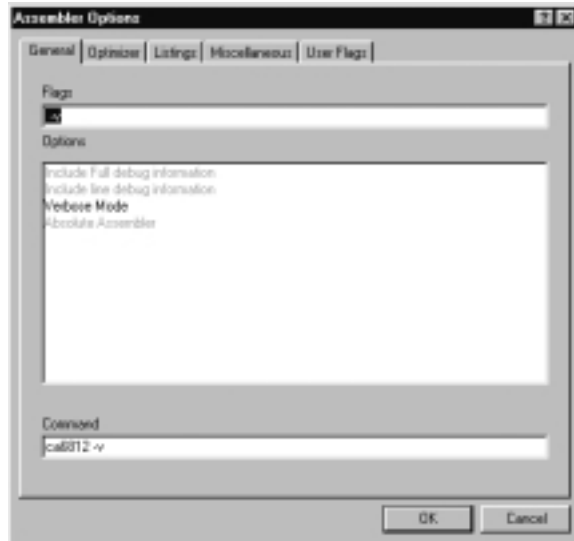
The default compiler options can be overridden by setting compiler options for the individual source files. Refer to "Source File Options" on page 1-28 for details.

Setting default assembler options

The **Assembler** component lets you set the default assembler options that are used to assemble all assembly language (**.s**) files in a project.

Right click on the **Assembler** icon Asm to open the **Assembler Options** dialog box. You can also double-click on the

Assembler icon to display the **Assembler Options** icon and then right-click on the **Assembler Options** icon to open the **Assembler Options** dialog box.

**Figure 1-20: Assembler Options
dialog box**

The **Assembler Options** dialog box five tabs:

- **General options**
- **Optimizer options**
- **Listings options**
- **Miscellaneous options**
- **User Flags**

Choose a tab and select the desired options. Selected options
are displayed in bold and unselected options are greyed out. To
select an option, simply click on the option description. To
deselect an option, click on the option again.

For a description of assembler options, refer to Chapter 8,
*IDEA Command Reference i*n the IDEA User's Guide.

The default assembler options can be overridden by setting
assembler options for the individual source files. Refer to
"Source File Options" on page 1-28 for details.

Setting default linker options

The **Linker** component lets you set the default *clnk* **utility** options that are used to link all files in a project. You can also specify a linker command file and edit the file.

Right click on the **Linker** icon ✆ to view a menu containing linker commands. The linker commands are described in the following Table.

Table 1-11: Linker commands

Options	Opens the **Link Configuration** dialog box.
Edit Com-mand File	Opens the project link command file for editing.
Change Command File	Opens the **Select Linker Command File** dialog box.

You can also double-click on the **Linker** icon to display the

Linker Options icon 🖳 and the **Linker Command File**

icon 📄.

Setting the linker configuration

Select **Options** from the **Project Linker** menu (or right-click on the **Linker Options** icon) to open the **Link Configuration** dialog box.

The **Link Configuration** dialog box lets you specify:

• **Linker options**
• **Libraries path option**
• **Reporting mode option**
• **Memory banking option**

Figure 1-21: Link Configuration dialog box

Specifying linker options

The Link Configuration dialog box lets you specify *clnk* **utility** options. These options are described in the following Table.

Table 1-12: *Clnk* utility options

Output file	**(-o option)**: writes output to the specified file. This option is required and has no default value.
Command file (.lkf)	The linker command file. This option is required and has no default value.
Map file	**(-m option)**: produces map information for the program being built to the specified file.
Error file	**(-e option)**: logs errors in the text file specified instead of displaying the messages on the screen.

After you select any one of these files, you can click on the **Find** button to specify the file name and path.

Specifying the libraries path

Click on the **Libs Path** button to open the
Libraries Path Editor and set a path to the
compiler library (**-l >** option).
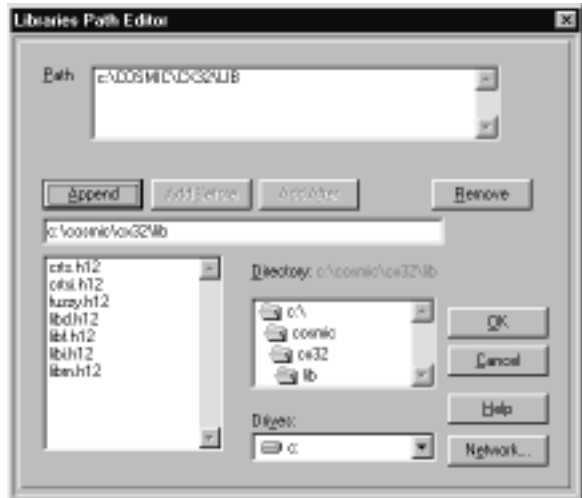
Figure 1-22: Libraries Path Editor

You can specify up to twenty library paths in any
order. The paths are searched from top to bottom.
After you add paths, they appear in order next to
the **Libs Path** button.

Other linker options you can set are described in
the following Table.

Table 1-13: Other *Clnk* utility options

Verbose	(**-v option**): be verbose.
Symbols Only	(**-s option**): create an output file containing only an absolute symbol table, but still with an object file format.
Memory Banking	(**-bs option**): enter the **size** of the page to be used. The size is translated to the correct **-bs** option for the linker. For example the default page size for 68HC12 paging is 0x4000 which translates to a -bs14. The default value for most processors is 0 (bank switching disabled).

Editing the linker command file

Before you can edit a linker command file, you must first check the **Command File** check box in the **Link Configuration** dialog box and then specify a linker command file name and path.

Select **Edit Command File** from the **Project Linker** menu (or right-click on the **Linker Command File** icon) to open the linker command file for editing.

**Figure 1-23: Linker command file
for demo12.prj**

To edit the linker command file, you can make changes
directly in the file using the options in the **Edit** menu.
Type **Alt+E** to view the editing options. You can also
right click to view a menu of editing options.

For complete details on linker command file editing
options, refer to Chapter 8, *IDEA Command Reference*
in the IDEA User's Guide.

Changing the linker command file

Select **Change Command File** from the **Project Linker**
menu to change the linker command file. The **Select
Linker Command File** dialog box lets you specify a file
name and path for the new linker command file.

After you select a new command file, the **Command
File** check box is checked in the **Link Configuration**
dialog box, and the linker command file name and path
are displayed.

Specifying project builder utilities

The **Project Builder** component lets you specify utilities for building the project.

Right click on the **Project Builder** icon 🏗 to open the **Builder Configuration** dialog box. You can also double-click on the **Project Builder** icon to display the **Builder Options**

icon 📧, and then right-click on the **Builder Options** icon to display the **Builder Configuration** dialog box.
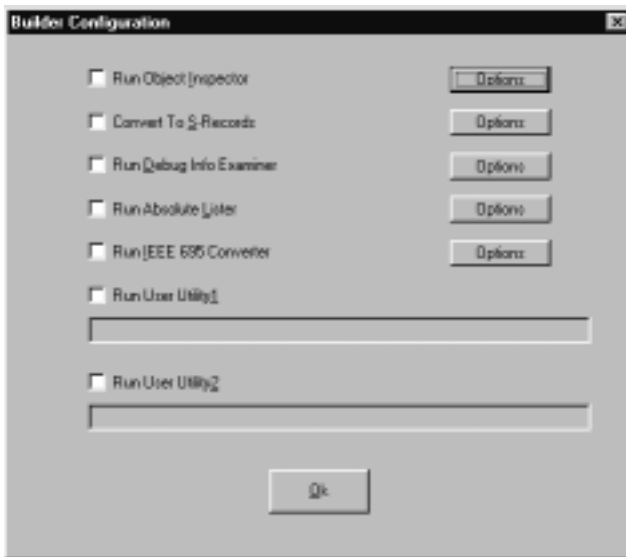
**Figure 1-24: Builder Configuration
dialog box**

The **Builder Configuration** dialog box contains check boxes that let you specify which builder utilities to run. The builder utilities are described in the following Table.

Table 1-14: Builder utilities

Run Object Inspector	Runs the *cobj* **utility** to examine object modules. If you select **Run Object Inspector** and then click on the **Options** button, the **Options** dialog box appears. Refer to "Configuring the Object Inspector utility" on page 1-48 for details.
Convert to S-Records	Runs the *chex* **utility** to translate object module format to hexadecimal format. If you select **Convert to S-Records** and then click on the **Options** button, the **CHEX Configuration** dialog box appears. Refer to "Configuring the Hex Converter utility" on page 1-50 for details.
Run Debug Info Examiner	Runs the *cprd* **utility** to print debugging information about functions and data objects. If you select **Run Debug Info Examiner** and then click on the **Options** button, the **CPRD Configuration** dialog box appears. Refer to "Configuring the Debug Info Examiner utility" on page 1-52 for details.
Run Absolute Lister	Runs the *clabs* **utility** to generate absolute listings. If you select **Run Absolute Lister** and then click on the **Options** button, the **CLABS Configuration** dialog box appears. Refer to "Configuring the Absolute Lister utility" on page 1-54 for details

Table 1-14: Builder utilities

Run IEEE 695 Converter	Runs the *cv695* **utility** to generate IEEE695 format. If you select **Run IEEE 695 Converter** and then click on the **Options** button, the **CLABS Configuration** dialog box appears. Refer to "Configuring the IEEE695 Converter utility" on page 1-56 for details.
Run User Utility 1	Runs the specified user utility. You can specify a path and filename for the utility.
Run User Utility 2	Runs the specified user utility. You can specify a path and filename for the utility.

Configuring the Object Inspector utility

The *cobj* **utility** lets you inspect relocatable object files or executable output by the assembler or linker. The *cobj* **utility** can be used to check the size and configuration of relocatable object files or to output information from their symbol tables.

Right click on the **Object Inspector** icon $^{c}{}_{0}{}_{B}{}_{J}$ and select **Options** to open the **Options** dialog box. You can also double-click on the **Object Inspector** icon to display the **Options** icon and then right-click on it to display the **Options** dialog box.

**Figure 1-25:** *cobj* **utility Options dialog box**

Selected options are displayed in bold and unselected options are greyed out. To select an option simply click on the option description and it is added to the command line. To deselect an option click on the option again.

For complete details on the *cobj* **utility** options, refer to Chapter 8, *IDEA Command Reference* in the IDEA User's Guide.

You can also specify a path and file name to receive the **Object Inspector** output. This file may be in relocatable format or executable format.

Configuring the Hex Converter utility

The *chex* **utility** translates executable images produced by the
clnk **utility** to one of several hexadecimal interchange formats.

Right click on the **Hex Converter** icon $^{c}H_{Ex}$ and select **Options**
to open the **CHEX Configuration** dialog box. You can also
double-click on the **Hex Converter** icon to display the

Options icon 🗒 and then right-click on it to display the
CHEX Configuration dialog box.

Figure 1-26: CHEX Configuration
dialog box

The following Table describes the formats and options that are
available. For details on *chex* utility options, refer to Chapter 8,
IDEA Command Reference in the IDEA User's Guide.

Table 1-15: *chex* utility options

Motorola S Records format	(**-fm** option) - produces S1 and S2 records as needed.
Motorola S2 Records format	(**-f2** option) - produces S2 records only. This is the default.
Intel Hex format	(**-fi** option)
Absolute Start Address	(**-a** option) - the output address of the first byte.
Address Bias	(**-b** option) - subtract from any address before output.
Max Bytes per line	(**-m** option) - maximum data bytes per line. The default is 32 bytes per line.
Do not Output Header	(**-h** option)
Output Paged Addresses	(**-p** option)
Output by Increasing Addresses	(**-s** option)
Output to File	(**-o** option) - the default is STDOUT.

Table 1-15: *chex* utility options

Insert Header Sequence	(+**h** option)
Output named segments only	(-**n** option). Up to twenty different named segments can be specified. To add a named segment to the **Segments** field, enter the named segment in the **Item** field and click on the **Add** button. To remove a named segment from the **Segments** field, select the segment and click on the **Remove** button.

Configuring the Debug Info Examiner utility

The *cprd* **utility** extracts and prints information about functions and data objects from an object module or executable image that has been compiled with the +**debug** option.

Right click on the **Debug Info Examiner** icon ᶜᴾᴿᴰ and select **Options** to open the **CPRD Configuration** dialog box. You can also double-click on the **Debug Info Examiner** icon to display the **Options** icon 🗐 and then right-click on it to display the **CPRD Configuration** dialog box.
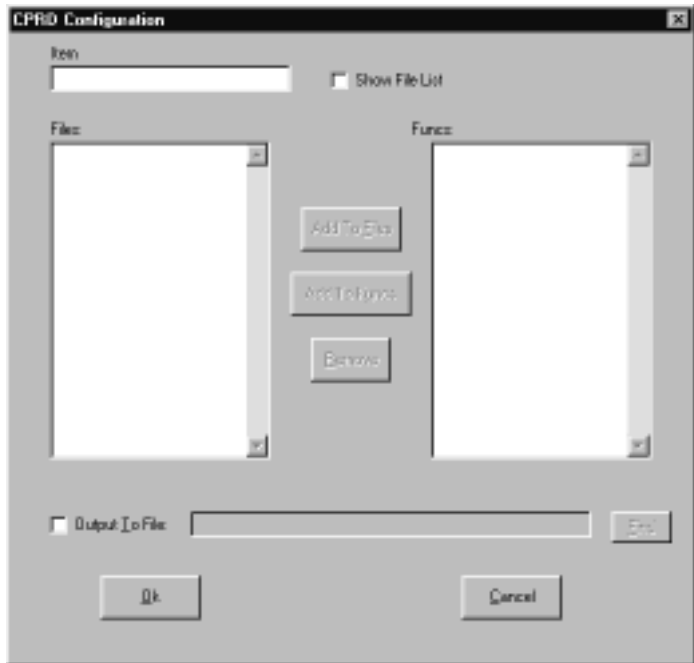
**Figure 1-27: CPRD Configuration
dialog box**

The **CPRD Configuration** dialog box lets you build a list of
files and functions for debugging purposes. Enter a file or
function name in the **Item** field, and then click on **Add to Files**
to add the item to the **Files** list or **Add to Funcs** to add the item
to the **Functions** list.

If you check the **Show File List** check box, the **Item** field
changes to a **File List** field, with a drop-down list of the files in
the project directory. Select a file from the list and then click on
the **Add to Files** button to add it to the **Files** list.

To remove an item from either list, select the item and then
click on the **Remove** button.

Each file in the **Files** list is processed with the **-fl** option, which prints debugging information about the file. By default, the *cprd* **utility** prints debugging information on all C source files.

Each function in the **Functions** list is processed with the **-fc** option, which prints information only about the function. By default, the *cprd* **utility** prints debugging information on all functions in a file.

You can also specify a path and file name to receive the debugger output. This is equivalent to the *cprd* **utility -o** option. By default, the *cprd* **utility** writes debugging information to the terminal screen.

Configuring the Absolute Lister utility

The *clabs* **utility** processes relocatable C and Assembly listing files with the associated executable file to produce absolute listings with updated code and address values.

Right click on the **Absolute Lister** icon 📄 and select **Options** to open the **CLABS Configuration** dialog box.You can also double-click on the **Absolute Lister** icon to display the

Options icon 📰 and then right-click on it to display the **CLABS Configuration** dialog box.
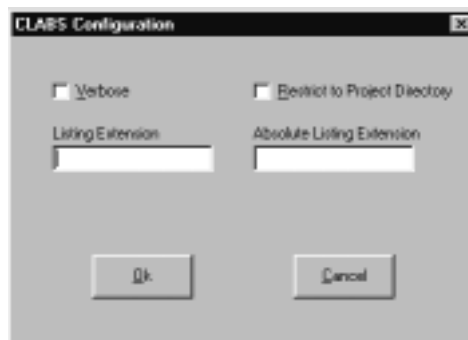
Figure 1-28: CLABS Configuration dialog box

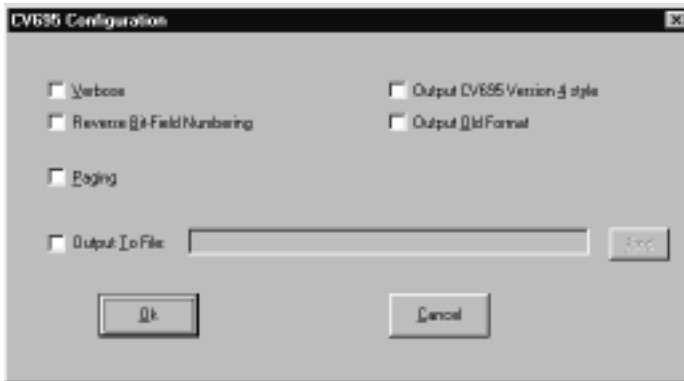The *clabs* utility options are described in the followingTable.

Table 1-16: *clabs* utility options

Verbose	**(-v option)** - the name of each module of the application is output to STDOUT.
Restrict to Project Directory	**(-l option)** - process files in the project directory only. The default is to process all files of the application.
Listing Extension	**(-r option)** - specify the input file extension. The default is "**.ls**".
Absolute Listing Extension	**(-s option)** - specify the output file extension. The default id "**.la**"

Configuring the IEEE695 Converter utility

The *cv695* **utility** converts a file produced by the linker into IEEE695 format.

Right click on the **IEEE695 Converter** icon $^{IE_{EE}}$ and select **Options** to open the **CV695 Configuration** dialog box. You can also double-click on the **IEEE695 Converter** icon to display the **Options** icon and then right-click on it to display the **CV695 Configuration** dialog box.

Figure 1-29: CV695 Configuration dialog box

The *cv695* utility options are described in the following Table.

Table 1-17: *cv695* utility options

Verbose	**(-v option) -** the *cv695* utility displays information about its activity.
Reverse Bit-Field Numbering	**(-rb option) -** reverses bitfield from left to right.
Paging	(+**page#** option) - this option is currently meaningful for the MC68HC12 only. This option specifies the address format for bank-switched code. If you check the **Paging** check box, three options appear to the right: **Physical (+page1)** - the application is banked and the *cv695* utility outputs physical addresses. This is the default if **Paging** is checked. **Logical (+page2)** - the application is banked and the *cv695* utility outputs addresses in paged mode: `<page><offset_in_page>`. This is equivalent to the old +**paged** flag. **data paging (+dpage)** - the application uses data paging.
Output to File	(**-o** option) - you can specify a path and file name to receive the *cv695* utility output. By default, thr *cv695* utility outputs to the file whose name is obtained from the input file by replacing the filename extension with "**.695**".

Specifying a project debugger

Right-click on the **Project Debugger** icon 🖼️ to open a dialog box that allows you to specify a debugger for the project.

After you select a debugger, the path and filename appears after the **Project Debugger** icon.

Once you have specified a debugger, you can double click on the **Debugger** icon to run the ZAP debugger with the project target file opened. You can also run the debugger by clicking on the **Debugger** tool 🖼️ in the Tool bar.

Specifying project documentation

The **Project Documentation** component shows all documents that are associated with the project.

Right click on the **Documentation** icon ◆ and select **Add Doc** to associate a documentation file with the project.

The **Document** icon 🖼️ lets you view, edit, or remove a document associated with the project. The appearance of the icon varies, depending on the type of document.

Right-click on the **Document** icon to view a menu containing documentation file commands. These commands are described in the following Table.

Table 1-18: Documentation file commands

Load (read only)	Opens the document in read-only mode.
Open	Opens the document for editing.
Remove	Removes the document from the project.

Building a Project

After a project is configured, you need to build the application. There are a three different ways to do this:

1. Right click on the **Project Name** icon ▥ in the Project window and select **Make** or **Build** from the pop-up menu.

2. Choose **Compile** (single, open file), **Make**, or **Build** from the **Project** pull-down menu.

3. Click on one of the following tools on the Tool bar:

Table 1-19: Tool bar tools for project building

▨	**Compile** tool - compiles (**.c** file) or assembles (**.s** file) an open project source file. Options are specified in the **Compiler** or **Assembler Options** dialog box.
▨	**Link** tool - runs the linker (and no other utilities) using the options specified for the project in the **Link Configuration** dialog box. Project source files are not checked for up-to-date status.
Make	**Make Project** tool - checks source file up-to-date status and dependencies. Selectively compiles or assembles any out-of-date files and runs the Linker.
▥	**Build Project** tool - performs a Make and then runs any utilities selected in the **Builder Configuration** dialog box. To have the Build rebuild all files regardless of their up-to-date status, right click on the project name, select **Mark All**, and then run the Builder.

For additional details on the project building tools, refer to Chapter 4, *IDEA User interface* in the IDEA User's Guide.

For additional details on building an IDEA project, refer to Chapter 7, *Building an IDEA Project* in the IDEA User's Guide.

Debugging a project

IDEA lets you use a Cosmic ZAP debugger to debug your project.

You can open the ZAP Debugger by clicking on the **Debugger**

tool ![debugger icon] in the Tool bar.

NOTE

Before you can use the ZAP debugger, you must first specify its location by right clicking on the **Debugger** tool ![debugger icon] in the **Tool Browser** (select **Tools** from the Main menu to open the **Tool Browser**). This opens a dialog box that allows you to specify the debugger for the project.
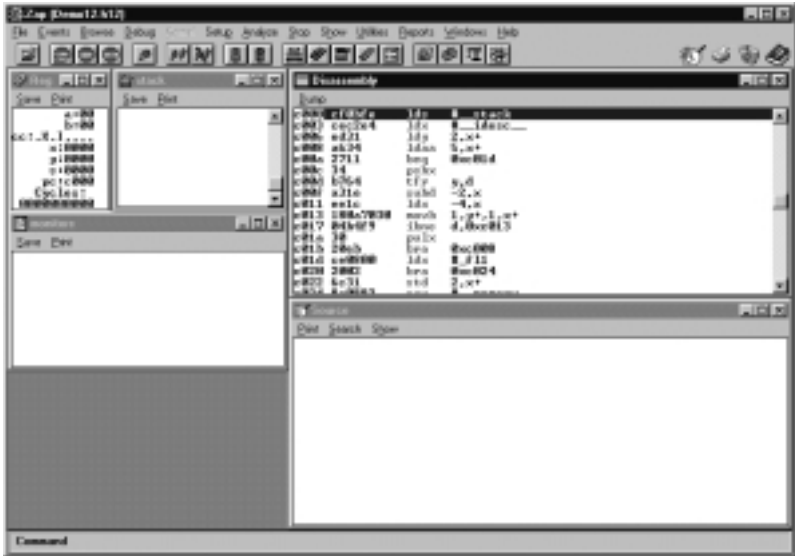
Figure 1-30: ZAP Debugger with
project target file open

When you run the ZAP Debugger from within IDEA, the ZAP Debugger automatically opens the target file for the currently loaded project (for example, **demo12.h12**).

For details on using the ZAP Debugger, refer to the ZAP User's Guide.

This page intentionally left blank.