

Pro Fortran

User Guide

Mac OS X
With Intel Processors

absoft
development tools and languages

Pro Fortran

User Guide

Mac OS X

With Intel Processors

absoft
development tools and languages

2781 Bond Street
Rochester Hills, MI 48309
U.S.A.
Tel (248) 853-0095
Fax (248) 853-0108
support@absoft.com

All rights reserved. No part of this publication may be reproduced or used in any form by any means, without the prior written permission of Absoft Corporation.

THE INFORMATION CONTAINED IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE AND RELIABLE. HOWEVER, ABSOFT CORPORATION MAKES NO REPRESENTATION OF WARRANTIES WITH RESPECT TO THE PROGRAM MATERIAL DESCRIBED HEREIN AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. FURTHER, ABSOFT RESERVES THE RIGHT TO REVISE THE PROGRAM MATERIAL AND MAKE CHANGES THEREIN FROM TIME TO TIME WITHOUT OBLIGATION TO NOTIFY THE PURCHASER OF THE REVISION OR CHANGES. IN NO EVENT SHALL ABSOFT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE PURCHASER'S USE OF THE PROGRAM MATERIAL.

U.S. GOVERNMENT RESTRICTED RIGHTS — The software and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. The contractor is Absoft Corporation, 2781 Bond Street, Rochester Hills, Michigan 48309.

ABSOFT CORPORATION AND ITS LICENSOR(S) MAKE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. ABSOFT AND ITS LICENSOR(S) DO NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL ABSOFT, ITS DIRECTORS, OFFICERS, EMPLOYEES OR LICENSOR(S) BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF ABSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Absoft and its licensor(s) liability to you for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, tort, (including negligence), product liability or otherwise), will be limited to \$50.

Absoft, the Absoft logo, Fx, Fx2, CLM, Pro Fortran, and MacFortran are trademarks of Absoft Corporation
Apple, the Apple logo, Velocity Engine, OS 9, and OS X are registered trademarks of Apple Computer, Inc.
AMD64 and Opteron are trademarks of AMD Corporation
CF90 is a trademark of Cray Research, Inc.
IBM, MVS, RS/6000, XL Fortran, and XL C/C++ are trademarks of IBM Corp.
Macintosh, NeXT, and NeXTSTEP, are trademarks of Apple Computer, Inc., used under license.
MS-DOS is a trademark of Microsoft Corp.
Pentium, Pentium Pro, and Pentium II are trademarks of Intel Corp.
PowerPC is a trademark of IBM Corp., used under license.
Sun and SPARC are trademarks of Sun Microsystems Computer Corp.
UNIX is a trademark of the Santa Cruz Operation, Inc.
Windows 95/98/NT/ME/2000 and XP are trademarks of Microsoft Corp.
All other brand or product names are trademarks of their respective holders.

Copyright © 1991-2007 Absoft Corporation and its licensor(s).
All Rights Reserved

Printed and manufactured in the United States of America.

10.0030907

Fortran User Guide

Contents

CHAPTER 1 INTRODUCTION.....	1
Introduction to Absoft Pro Fortran.....	1
Absoft Fortran 90/95	1
Absoft FORTRAN 77	1
Conventions Used in this Manual	2
Road Maps	2
Fortran Road Maps	2
Year 2000 Problem.....	3
Fortran 90/95 DATE_AND_TIME Subroutine.....	4
Unix Compatibility Library.....	4
CHAPTER 2 GETTING STARTED	5
Compiling Basics	5
Application Basics	11
CHAPTER 3 USING THE ABSOFT EDITOR.....	13
The Absoft Editor.....	13
Text Selection.....	13
Using Compilers.....	14
Pop-up menus.....	14
Creating New Source Files	15
Manipulating Windows	16
Using the Editor Menus	16
Application Menu.....	17
Preferences.....	17
Format.....	17
Environment.....	19
Tools	21
File Menu	21
New... (⌘N).....	21
Open... (⌘O)	22
Open Recent	22
Open Selected.....	22
Open Complement.....	22
Close (⇧⌘W).....	22
Close All.....	22
Save (⌘S).....	22

Save As...	22
Save All	23
Revert	23
Print Setup... (⌘P)	23
Print... (⌘P)	23
Page Setup	24
Edit Menu	24
Undo (⌘Z)	25
Redo (⇧⌘Z)	25
Cut (⌘X)	25
Copy (⌘C)	25
Paste (⌘V)	25
Clear	25
Select All (⌘A)	25
Find	26
Find (⌘F)	26
Find Next (⌘F)	26
Find Previous (⇧⌘F)	26
Bookmarks	27
Bookmarks Menu	27
Toggle Bookmark (⌘F2)	27
Previous Bookmark (⇧F2)	27
Next Bookmark (F2)	27
Clear File's Bookmarks (⇧⌘F2)	27
Clear All Bookmarks (Control⌘F2)	27
Format Menu	28
Show Info... (⇧⌘I)	28
Insert Continuation (⌘I)	30
Go to Line (⌘L)	30
Match Brackets (⌘{)	30
Shift Left (⌘[)	30
Shift Right (⌘])	30
Comment	30
Uncomment	30
Convert to Upper Case	31
Convert to Lower Case	31
Tools Menu	31
Compile (⌘Y)	31
Check Syntax (⌘K)	31
Stop (⌘.)	31
Errors	31
Previous Error (⌘D)	31
Next Error (⌘E)	32
Window Menu	32
Close Window (⌘W)	32
Zoom Window	32
Minimize Window (⌘M)	32
Hide Toolbar	32
Customize Toolbar	32
Tile Horizontally	33
Tile Vertically	33
Cascade	33
Window list	33
Help Menu	33
Tools Help	33
Hide ToolTips	34

CHAPTER 4 USING THE COMPILERS.....	35
Compiling Programs.....	35
Using the Command Line.....	35
File Name Conventions.....	36
Using the Absoft Developer Tools Interface.....	36
Working with Projects.....	37
Options Dialog.....	38
Target Tab – General Options.....	39
Target Type.....	39
Link Large Data Stubs.....	40
Runtime Stack Trace (-et).....	40
64-bit code (-m64).....	40
Options.....	40
Speed Math (-speed_math=n).....	42
Object File(s) Directory.....	42
Target Directory.....	42
Target Filename.....	42
Target Tab – FPU Options.....	43
FPU Rounding Mode.....	43
FPU Exception Handling.....	44
Don't generate FMA instructions (-Q51).....	44
Other Target Options.....	44
Generate Assembly Language (-S).....	44
Generate Relocatable Object (-c).....	45
Library Specification (-l).....	45
Library Path Specification (-L).....	45
Absoft Developer Tools Interface.....	45
Application Menu.....	46
About Tools.....	46
Preferences.....	46
Hide Tools.....	47
Quit Tools.....	47
File Menu.....	47
New.....	47
Open.....	47
Open Recent.....	47
Close.....	48
Save.....	48
Save As.....	48
Revert To Saved.....	48
Page Setup.....	48
Print.....	48
New File.....	48
Edit Menu.....	48
Undo.....	48
Redo.....	48
View Previous.....	49
Configure Menu.....	49
Add/Remove File(s).....	49
Set Include Paths.....	50
Set Project Options.....	50

File Options.....	50
Remove File Options	51
Remove All File Options	51
MRWE Preferences.....	51
Set Default Options.....	51
Tools Menu.....	52
Search.....	52
Build.....	52
Rebuild All.....	52
Update Dependencies.....	52
Check Syntax	52
Compile.....	52
Edit.....	52
Preprocess	53
Clean	53
Stop	53
Execute.....	53
Debug.....	53
Profile.....	53
Terminal.....	53
Generate Makefile.....	53
Window Menu	54
Hide Toolbar	54
Customize Toolbar.....	54
Tile Horizontally.....	54
Tile Vertically	54
Project.....	54
Output	55
Help Menu	56
Tools Help.....	56
Hide ToolTips.....	56
Absoft Fortran 95 Options.....	57
General - F95 Options	57
Warning level (-znn)	58
Error Handling (-dq and -ea).....	58
Max Internal Handle (-T nn).....	58
Temporary string size (-t nn).....	58
Cache Control (-YDEALLOC= {MINE ALL CACHE}).....	59
Warn of Non-Standard usage (-en)	59
Suppress warnings (-w).....	59
Suppress Warning number(s) (-Znn).....	59
Use System Module Files (-SysModFiles).....	59
Set Module Paths (-p path)	59
Quiet (-q)	60
Verbose (-v).....	60
Procedure Trace (-B80)	60
Output Version number (-V).....	60
Default Recursion (-eR)	60
Compatibility - F95 Options	61
Disable compiler directive (-xdirective)	61
INTEGER and LOGICAL sizes (-in)	62
Character Argument Parameters (-YCFRL={0 1}).....	62
Demote Double Precision to Real (-dp).....	62
Promote REAL and COMPLEX (-N113)	62
One trip DO loops (-ej).....	62
Static storage (-s).....	62

Check Array Boundaries (-Rb)	63
Check Array Conformance (-Rc)	63
Check Substrings (-Rs)	63
Check Pointers (-Rp)	63
Pointers Equivalent to Integers (-YPEI={0 1})	63
Format - F95 Options	64
Free-Form (-ffree)	64
Fixed-Form (-ffixed)	65
Alternate Fixed form (-falt_fixed)	65
Fixed line length (-W nn)	65
YEXT_NAMES={ASIS UCS LCS}	65
External Symbol Prefix (-YEXT_PFX=string)	65
External Symbol Suffix (-YEXT_SFX=string)	65
Treat as Big-Endian (-N26)	65
Treat as Little-Endian (-N27)	66
Escape Sequences in Strings (-YCSLASH=1)	66
No Dot for Percent (-YNDFP=1)	66
MS Fortran 77 Directives (-YMS7D)	66
Common Block - F95 Options	67
COMMON Block Name Prefix (-YCOM_PFX=string)	67
COMMON Block Name Suffix (-YCOM_SFX=string)	67
COMMON Block Name Character Case (-YCOM_NAMES={UCS LCS})	68
Other F95 Options	68
Stack Size (-stack_size:size)	68
Disable Position Independent Code (-no-fpic)	68
Speculative Execution (-B156)	68
Inline CABS (-B157)	68
Address Optimizations (-B158)	68
Safe Floating-Point (-safefp)	69
Disable Default Module File Path (-nodefaultmod)	69
Variable Names Case Sensitivity (-YVAR_NAMES={ASIS UCS LCS})	69
Symbol Names Case Sensitivity (-YALL_NAMES={ASIS UCS LCS})	69
Ignore CDEC\$ directives (-YNO_CDEC)	69
Absoft Fortran 90/95 Compiler Directives	69
NAME Directive	70
FREE[FORM] Directive	70
FIXED Directive	70
NOFREEFROM Directive	71
FIXEDFORMLINESIZE Directive	71
ATTRIBUTES Directive	71
PACK[ON] Directive	71
PACKOFF Directive	72
STACK Directive	72
UNROLL Directive	72
NOUNROLL Directive	72
Absoft FORTRAN 77 Options	73
General - F77 Options	74
Max Internal Handle (-T nn)	74
Temporary string size (-t nn)	74
Suppress Warnings (-w)	75
Warn of non-ANSI Usage (-N32)	75
Quiet (-q)	75
Verbose (-v)	75
Check Array Boundaries (-C)	75
Conditional Compilation (-x)	75
Control - F77 Options	76

Compiler Directives (-Dname[=value])	76
Compatibility - F77 Options	77
Integer Sizes (-i2 and -i8)	77
Vax/Mainframe Compatibility	78
Folding to Lower Case (-f).....	78
Static Storage (-s).....	78
Folding to Upper Case (-N109).....	78
One-Trip DO Loops (F66) (-d).....	78
Append underscore to names (-N15).....	78
Miscellaneous - F77 Options	79
Promote REAL and COMPLEX (-N113).....	79
Escape Sequences in Strings (-K)	80
Format - F77 Options.....	80
ANSI Fortran 77 Fixed	81
Fortran 90 Free-Form (-8)	81
VAX Tab-Format (-V).....	81
Wide Format (-W).....	81
Treat as Big-Endian (-N26).....	81
Treat as Little-Endian (-N27).....	81
COMMON Block - F77 Options	82
Align COMMON Variables (-N34)	82
Set COMMON Block Name (-N22).....	83
Don't Mangle COMMON Block Name (-N110).....	83
C/C++ Options	83
General – C/C++ Options	84
Max Internal Handle (-T n).....	84
Max Errors (-maxerr n)	85
Diagnostic Messages.....	85
Suppress All Warnings (-w31).....	85
Suppress Warnings (-w16).....	85
Suppress Informationals (-w8).....	85
Suppress Anachronisms (-w2).....	85
Suppress Template Warnings (-w1)	86
Treat Anachronisms as Errors (-wp).....	86
Treat All Warnings as Errors (-wabort)	86
No Alias Optimizations (-N19)	86
Quiet (-q)	86
Verbose (-v)	86
Procedure Trace (-N124).....	86
Verbose Templates (-ptv)	86
Include Tree to Stderr (-H)	86
No inlines (-inline none).....	86
Exception Handling (-except on off).....	87
Preprocessor – C/C++ Options	87
Defines (-D name[=value])	87
Undefines (-U name).....	88
Do not search standard system directories	88
Preprocess files only	88
Format – C/C++ Options	89
C++ (-c++).....	89
ANSI C (-A).....	89
K and R C (-K).....	90
Linker Options.....	90
General - Link Options	91
Produce Map File.....	91

Suppress Warnings	91
Verbose.....	91
Report Duplicate Symbols.....	92
Exclude libac.a.....	92
Why Load	92
Report Undefined Symbols.....	92
Add Framework(s).....	92
Passing Options To The Linker.....	92
Undefine A Symbol (-u).....	92
Linker Options (-X and -Xlinker).....	92
Other Link Options.....	93
Plug-ins.....	93
VAST	94
IMSL Library	95
LAPACK Library	95
UNIX Library.....	96
VAX/VMS Library.....	96
BUILD OPTIONS	97
CHAPTER 5 PORTING CODE.....	99
Porting Code from VAX.....	99
Compile Time Options and Issues.....	100
Runtime Issues.....	101
Porting Code from IBM VS FORTRAN.....	101
Compile-time Options and Issues.....	102
Porting Code From Microsoft FORTRAN (PC version).....	102
Compile-time Options and Issues.....	102
Porting Code from Sun Workstations.....	103
Porting Code from the NeXT Workstation.....	104
Porting Code from the IBM RS/6000 Workstation.....	104
Porting Code from Intel 386/486/Pentium Computers	104
Porting Code To/From Other Macintosh Systems	104
Language Systems Fortran	104
Other Absoft Compilers	106
Other Porting Issues	106
Memory Management.....	106
Dynamic Storage	106
Static Storage.....	107
Naming Conventions.....	107
Procedure Names.....	107
COMMON Block Names	108
File and Path Names.....	108
Tab Character Size	108
Runtime Environment	109
Floating Point Math Control.....	111

Rounding Direction	111
Exception Handling	112

CHAPTER 6 THE MACINTOSH RUNTIME WINDOW ENVIRONMENT..... 113

Using MRWE	113
The MRWE Window	114
How Your Program And MRWE Work Together	115
Working With Text in MRWE	115
Using The MRWE Default Menus	116
File Menu	116
Save (⌘S)	116
Save As	116
Page Setup	116
Print Window... (⌘P)	116
Quit (⌘Q)	117
Edit Menu	117
Undo (⌘Z)	117
Cut (⌘X)	117
Copy (⌘C)	117
Paste (⌘V)	117
Clear	117
Font and Size Menus	117
Programming with MRWE	117
Program Organization: Fortran VS. Macintosh	118
MRWE Event Loop Operation	119
Customizing Menus	120
Adding Menus	120
Special Characters	120
Menus and the READ statement	121
Removing a Menu or Menu Item	122
Menu Response Routines and <code>mrwe_DoMenu</code>	122
Adding Checkmarks to Menus	123
Enabling/Disabling Menu Items	123
Launching OTHER APPLICATIONS	124
Apple Events	125
Apple Event Target	125
Apple Event Class and ID	126
Extra Information in an Apple Event	126
Receiving Apple Events	129
Error Codes Returned from Apple Event Routines	132
Other Examples of Apple Events	132
Sending a request to the Finder	132
Using other standard Apple Events	133
Sending information between MRWE applications	133
Scripting	133
Further Information About Apple Events	134
Creating Multiple Windows	134
Showing Alert Messages	135
SetMrwePrefs	136
Effects selected Application bundle, *.r file for *.rsrc file. To affect future applications built via the command line modify /Application/Absoft10/R/includes/mrweprefs.r	136

Termination Options	136
Window Size Options	137
Window Size	137
Text characteristics	137
CHAPTER 7 BUILDING PROGRAMS	137
The Components of an Application	138
Working with Resources	138
Creating Object Files	138
Fsplit - Source Code Splitting Utility	139
Building Programs	140
The Elements of amake	141
Using Macros	142
Advantages of using macros	142
Defining macros	142
Special macros	143
Cautions in using macros	143
Using Description Files	144
Working with dependency blocks	144
Defining a target more than once	145
Using include directives	145
A sample description file	146
Using Dependency Rules	146
The default rules	147
Creating your own rules	148
amake Usage and Syntax	149
Special Targets	151
Dummy Files	151
Environment Variables	152
Example: Rebuilding an Executable File	152
Error Handling and Cautions	153
Syntax Errors	153
Other Common Errors	154
Cautions	155
CHAPTER 8 INTERFACING WITH OTHER LANGUAGES	157
Interfacing with C	157
Fortran Data Types in C	157
Related Compiler Options	158
Rules for Linking	158
Passing Parameters Between C and Fortran	158
Reference parameters	159
Value parameters	160
Array Parameters	161
Function Results	161
Passing Strings to C	162
Calling Fortran math routines	163
Naming Conventions	163
Accessing COMMON blocks from C	164

Declaring C Structures in Absoft Pro Fortran	164
Interfacing with Assembly Language	164
Debugging.....	164
Compiler Options.....	164
Profiling.....	165
Compiler Options.....	165

APPENDIX A ABSOFT COMPILER OPTION GUIDE

..... **167**

Absoft Pro Fortran Compiler Options	167
FPU Control Options	168
Fortran 90/95 Control Options.....	168
Fortran 90/95 Optimization Options	168
Fortran 90/95 Source Format Options	168
Fortran 90/95 Compatibility Options	169
FORTRAN 77 Control Options	169
FORTRAN 77 Optimization Options	170
FORTRAN 77 Source Format Options	170
FORTRAN 77 Compatibility Options	170

APPENDIX B EXCEPTIONS AND IEEE ARITHMETIC

..... **173**

IEEE_FEATURES	173
IEEE_FEATURES_TYPE	173
IEEE_ARITHMETIC	174
IEEE_CLASS_TYPE	174
IEEE_ROUND_TYPE	175
Subroutines and Functions.....	175
IEEE_EXCEPTIONS.....	179
IEEE_FLAG_TYPE	180
IEEE_STATUS_TYPE.....	180
Subroutines and Functions.....	180
Examples	182

APPENDIX C	TERMINAL PROGRAMMING	183
APPENDIX D	ASCII TABLE	185
APPENDIX E	BIBLIOGRAPHY	189
	Fortran 90/95.....	189
	FORTRAN 77.....	189
APPENDIX F	TECHNICAL SUPPORT	191

CHAPTER 1

Introduction

INTRODUCTION TO ABSOFT PRO FORTRAN

Absoft specializes in the development of Fortran compilers and related tools. Full implementations of Fortran 77 and Fortran 90/95 are available for Macintosh OS X, Windows, and Linux platforms. Absoft will continue to focus on Fortran in the future, but the popularity of C/C++ in the Unix environment has required many of today's Fortran programmers who are moving code to their desktop, to link Fortran code with C libraries. Absoft compilers support most popular inter-language calling conventions implemented on Macintosh OS X systems, providing compatibility with existing libraries and object files, simplifying porting efforts.

This User Guide explains the operation of Absoft Fortran 90/95 and Absoft FORTRAN 77, and the Fx™ debugger on the Macintosh OS X operating system for the Intel family of processors. In the event you have licensed only one of these compilers, please refer only to the appropriate section(s) and disregard the others. All compilers operate in a similar manner, share a common tool set, and are link compatible. A brief summary of each compiler appears below.

Absoft Fortran 90/95

A complete, optimizing ANSI Fortran 90/95 implementation with extensions. Absoft Fortran 90/95 is the result of a five-year joint development effort with Cray Research. It utilizes a version of the CF90 front-end and is source compatible with several Cray F90 releases. It provides full support for interfacing with FORTRAN 77 and C Programming Language libraries.

Absoft FORTRAN 77

Refined over 20 years, with emphasis on porting legacy code from workstations. Absoft Fortran 77 is full ANSI 77 incorporating MIL-STD-1753, Cray-style POINTERS, plus most extensions from VAX FORTRAN as well as many from IBM, Sun, HP, and Cray. Absoft Fortran 77 supports legacy extensions that are not part of the Fortran 90/95 standard. See the chapter on **Porting Code** in this manual for further information. Fortran 77 is fully link compatible with Fortran 90/95 and C/C++ so existing, extended FORTRAN 77 routines can be easily compiled and linked with new Fortran 90/95 or C/C++ code.

CONVENTIONS USED IN THIS MANUAL

There are a few typographic and syntactic conventions used throughout this manual for clarity.

- [] square brackets indicate that a syntactic item is optional.
- ... indicates a repetition of a syntactic element.
- Term definitions are underlined.
- **-option** font indicates a compiler option.
- *Italics* are used for emphasis and book titles.
- Unless otherwise indicated, all numbers are in decimal form.
- FORTRAN examples appear in the following form:

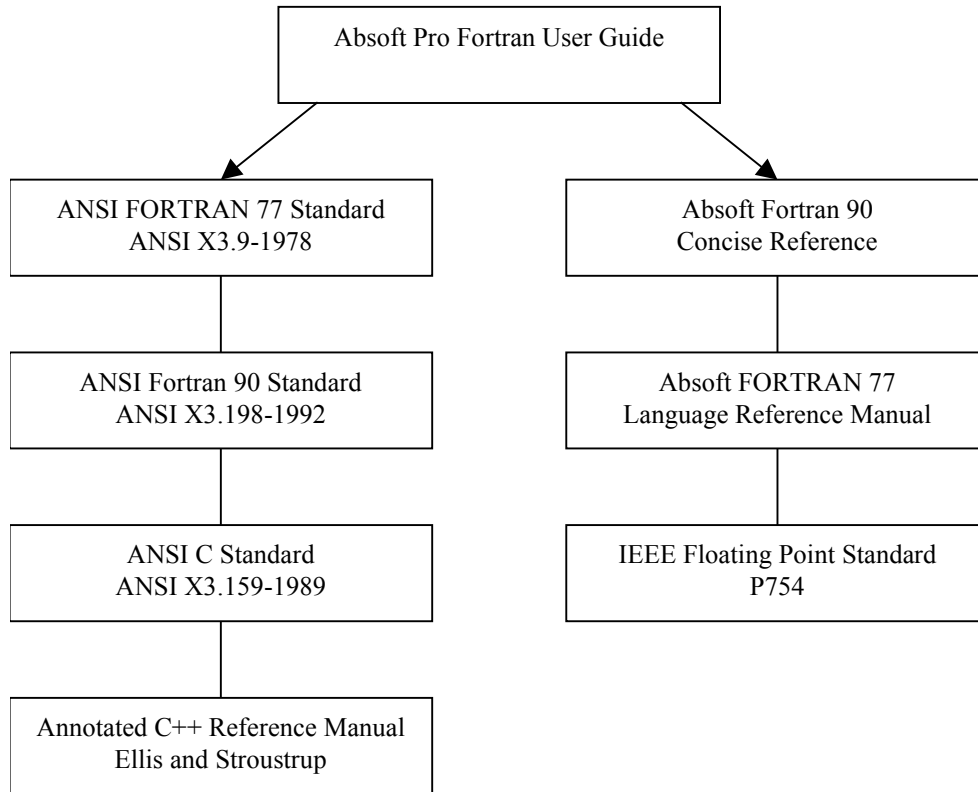
```
PROGRAM SAMPLE
WRITE (9,*) "Hello World!"
END
```

ROAD MAPS

Although this manual contains all the information needed to build programs with Absoft Pro Fortran on Macintosh OS X, there are a number of other manuals that describe Fortran 90/95 and FORTRAN 77 in further detail. The *road map* in this chapter will guide you to these manuals for introductory or advanced reference. The bibliography in appendices lists further information about each manual.

Fortran Road Maps

The Absoft implementation of Fortran 90/95 is detailed in the online manual, *Fortran 90 Concise Reference*, in the Documentation directory of the Pro Fortran CDROM. FORTRAN 77 is detailed in the online manual, *FORTRAN 77 Language Reference Manual*, also in the Documentation directory of the Pro Fortran CDROM. A discussion of floating point precision is at the end of the chapter **Porting Code**. Figure 1-1 shows additional manuals that can be used for referencing the FORTRAN language and internal math operations.



FORTRAN 77 language road map
Figure 1-1

YEAR 2000 PROBLEM

All versions of Absoft Pro Fortran products for Macintosh, Power Macintosh, Windows 95/98, Windows NT, Linux, and UNIX will operate correctly across the date transition to the year 2000. Neither the compilers nor the runtime libraries have ever used 2-digit years in their internal operation. This means the version of Absoft Pro Fortran that you already have will continue to operate correctly. No patches or version updates are required.

The only caveat may be for those porting code from VAX/VMS systems. Since the early 1980s, Absoft Pro Fortran products have included software libraries designed to facilitate porting code from the VAX/VMS environment. Included in these VAX compatibility libraries are two subroutines that emulate the VAX/VMS DATE and IDATE subroutines. These subroutines return the year using a two-digit format. If you use DATE or IDATE in a program that stores or compares dates, you may need to recode portions of your application. Below are listed some of the alternatives supplied with Pro Fortran:

Fortran 90/95 `DATE_AND_TIME` Subroutine

This subroutine is part of the Fortran 90/95 language and returns integer data from the date and real time clock. Refer to the *Fortran 90 Concise Reference* for further information.

Unix Compatibility Library

There are a number of subroutines in the Unix Compatibility Library that return the date and time in both `INTEGER` and `CHARACTER` format. Refer to the manual **Absoft Compatibility Libraries** for information on their format and use.

CHAPTER 2

Getting Started

The tutorial in this chapter introduces the two main functions of the Absoft Pro Fortran Software Development package for Mac OS X: compiling source code and running compiled applications. If you are familiar with the basics of compiling and running programs, please see the table below as a guide to topics you may find useful.

TO DO THIS...	TURN TO THIS SECTION...
Use the editor	Using the Absoft Editor , Chapter 3
Use the compiler	Using the Compilers , Chapter 4
Port from other platforms	Porting Code , Chapter 5
Program Mac OS X	MRWE , Chapter 6
Create applications	Building Programs , Chapter 7
Interface with C	Interfacing With Other Languages , Chapter 8
Debug programs	Using the FX Debugger , Chapter 9

Road map for experienced users

COMPILING BASICS

The Absoft compilers can be run either from a command line or from the Absoft Tools Interface. This chapter describes how to use the Absoft Tools Interface—the command line interface is described in the chapter **Using the Compilers** and in the appendix **Terminal Programming**.

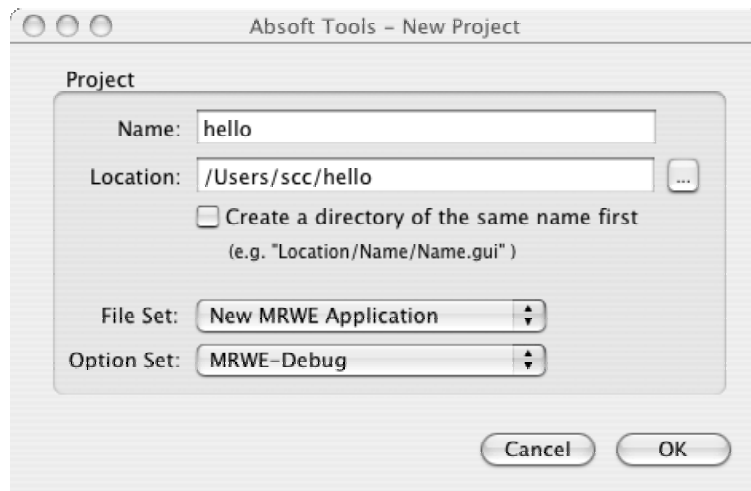
Note: Throughout this chapter and the rest of the manual, it is assumed that the compiler has been installed in the default installation folder **/Applications/Absoft10** on the boot volume. If this is not the case, substitute the correct path in the examples as appropriate.

During the installation process, several example programs were placed in the **/Applications/Absoft10/examples** folder. The example program used in this tutorial is **hello_ex.f**. Follow the tutorial on the next few pages to learn how to use the graphical interface to quickly compile small to medium size programs.

First, start up the interface to the compiler by double clicking the **Absoft Tools** icon in the `/Applications/Absoft10/bin` folder.

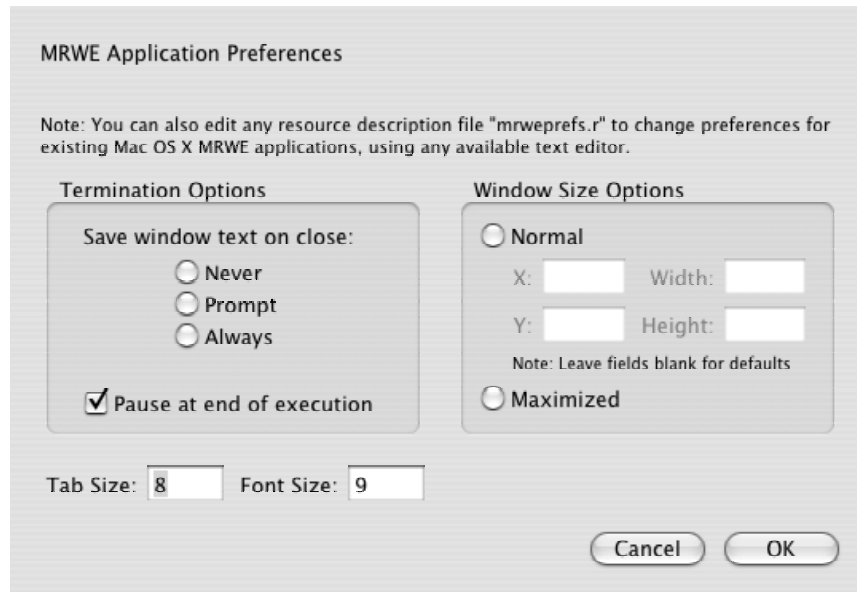
The Absoft Tools Interface is project oriented, so the first thing you need to do is to establish a name and location for your project. Type “**hello**” into the **Name** box. Change the **Location** to a folder in your **home** directory, such as `/Users/user_name/hello` (be sure sure to substitute your actual user name in for *user_name* above).

To create a double clickable windowed application, choose "New **MRWE** Application" (Macintosh Runtime Window Environment) from the **File Set:** menu. This will supply the resource files and other necessary files to have Absoft Tools create a windowed Carbon application. **MRWE** provides an automatic Mac OS X Carbon interface for your program with menus, a scrollable text window for program output, and the ability to print. Choosing **New Fortran 95 Project** or **New Fortran 77 Project** will create a terminal application and supply you only with an empty source file to start with. **Empty Project** creates a Terminal application with no starting source file. For our demonstration here we will create an **MRWE** application.



New Project dialog box

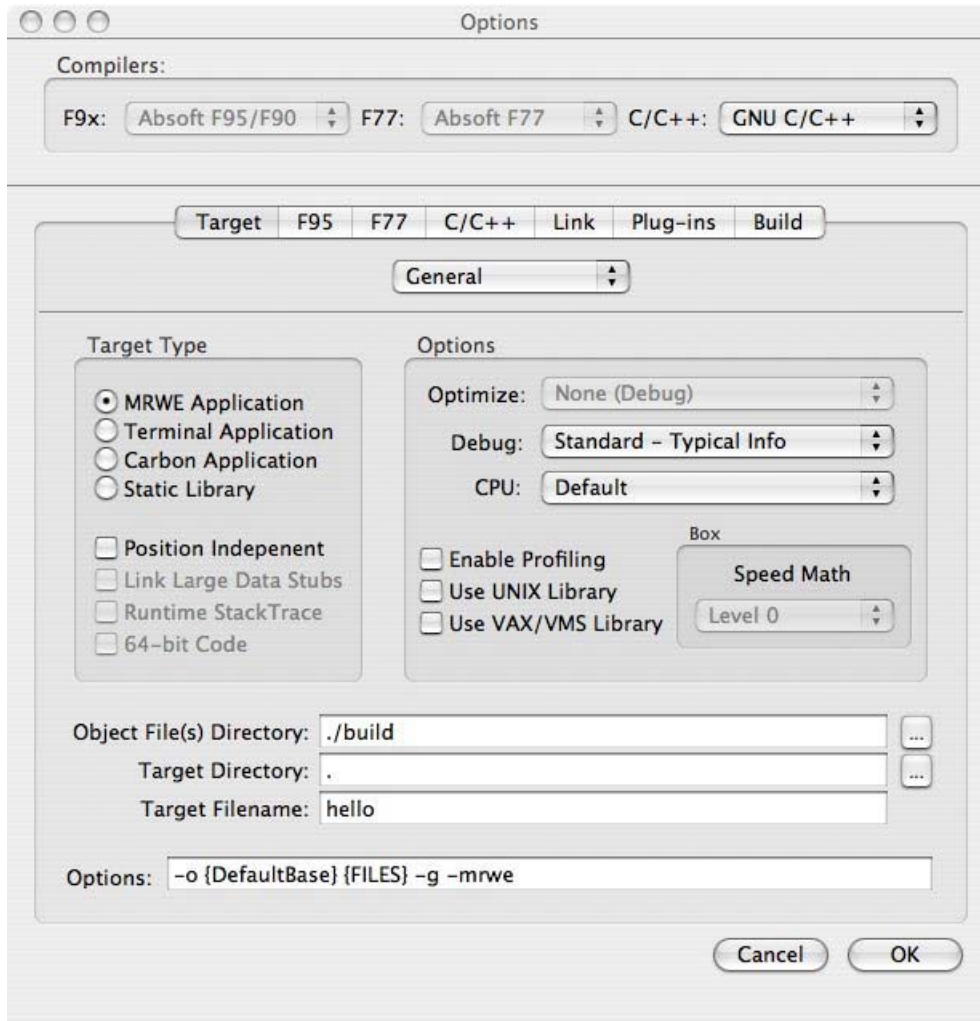
Click on the **OK** button and the **MRWE Preferences** dialog will appear. Most users will want to make sure that the **Pause at end of execution** box is checked. You can alter the initial size of the **MRWE** window in the **Window Size Options** section, or do nothing and accept the default size and position. You can return to this menu later via the **MRWE Options** selection in the **Configure** menu and make any desired adjustments.



MRWE Preferences dialog box

After you set the **MRWE** preferences, click the **OK** button and the project **Options** menu is displayed.

Use this menu page to set project wide options and select which compilers to use. If you know of any options that are necessary for your program you may set them now under the appropriate tab and option subset (see chapter 4, Using the Compilers). You can return to this menu later via the **Set Project Options** selection in the **Configure** menu and make any necessary adjustments.

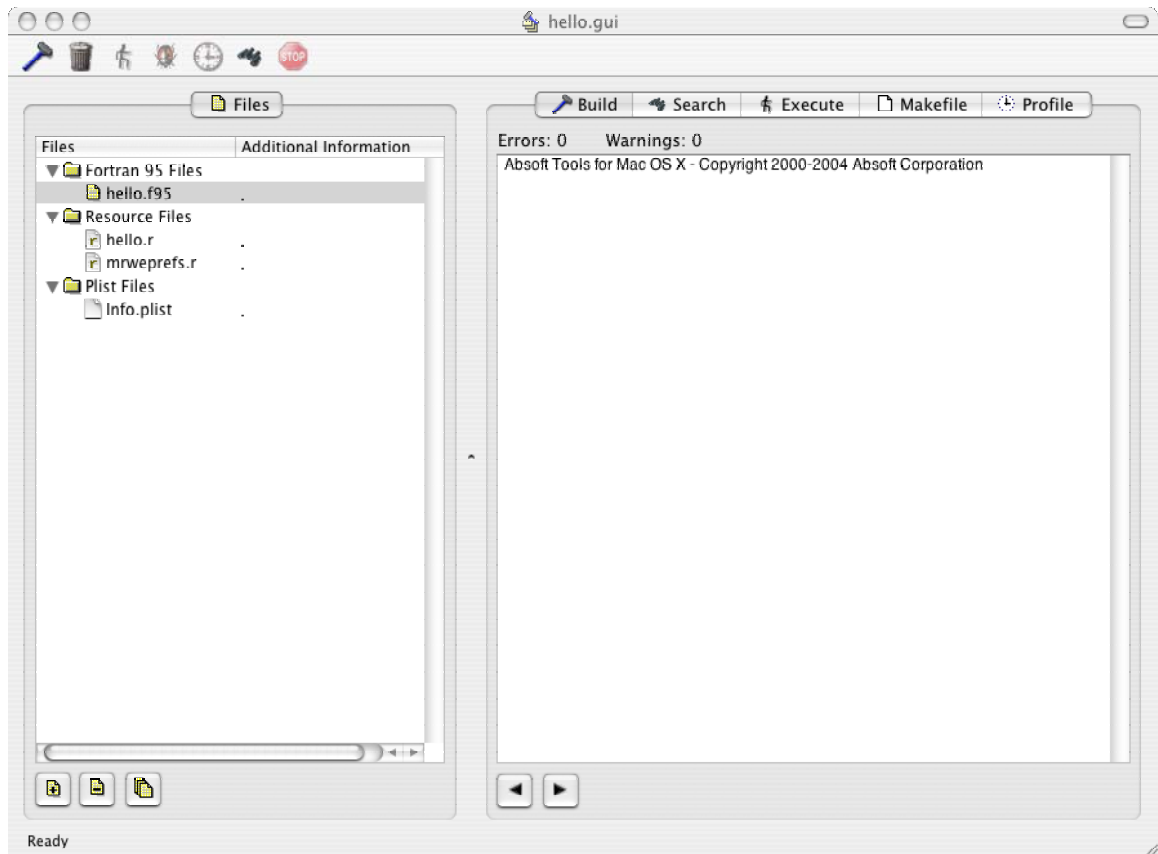


Project options dialog box

Click on the **OK** button to dismiss this dialog and the **Main Project** window will appear. This window maintains all of the files in your project on the left and displays a variety of information about your project on the right.

Each file in the **Files** list on the right side of the **Project Options** menu will be kept in a separate folder based on the file extension. Initially, the list will contain three folders, **Fortran 95 Files**, **Resource Files**, and **Plist Files**.

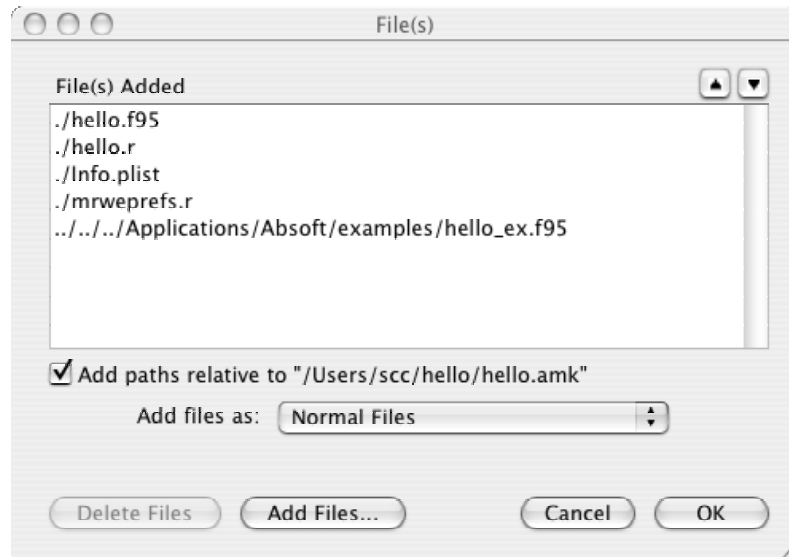
Most users will not need to do anything with the resource or plist files directly, but it is important that they are a part of your **MRWE** project.



Main Project window

Notice that a file named **hello.f95** is already a part of your project. This is a dummy main program file provided by Absoft Tools. You can either edit it to contain your Fortran code (as we will do in this example), or remove it from the project and add your own Fortran source file(s).

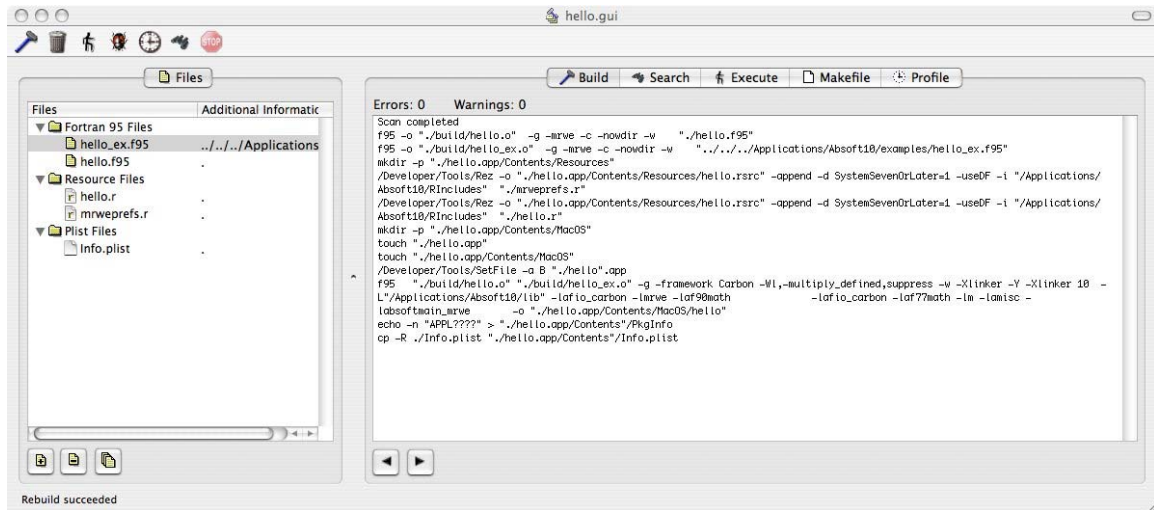
The next step is to specify the file (or files) that the project consists of, in this case **/Applications/Absoft10/examples/hello_ex.f**. This file contains a subroutine to call from your main program. To open the appropriate dialog box, select **Add/Remove File(s)...** from the **Configure** menu. Press the **Add Files** button and navigate to **/Applications/Absoft10/examples**. Highlight **hello_ex.f** and press the **Add** button, then press **OK**.



File selection dialog box

Now you need to create a main program unit and call the **hello_ex** subroutine. Double click on **hello.f95** in the **Fortran 95 Files** folder on the main Project window. This will launch the Absoft Editor, ready to modify **hello.f95**. Add the text **call hello_ex()** on the line just before the **STOP** statement. Go to the **File** menu and select **Save**. You may now quit the Absoft Editor, see chapter 3 for more information about the Absoft Editor.

The last step is to build (compile) your application is to choose the **Build** command from the **Tools** menu. This will compile both **hello.f95** and **hello_ex.f95** and create the executable application **hello.app**, which will be located in the folder specified in the project **Location** field. The “.app” file extension may not be visible in the Finder, but the executable application icon will look like a piece of paper with a stylized “A” on it.



Main Project window after successful build

More detailed information concerning the creation of an application can be found in the chapter **Using the Compilers**.

APPLICATION BASICS

The best way to run your application is to double click the application icon from the Finder. You can also choose the **Execute** command from the **Tools** menu in Absoft tools.

Additional examples that may be helpful in writing Fortran 90/95 or FORTRAN 77 programs can be found in the **/Applications/Absoft10/examples** directory. Most examples have an Absoft Tools project you can launch to build the program. Each example source file starts out with a large comment, referred to as the header. Before compiling an example, look at the header in the source code. It will list all of the compiler options necessary to insure that the example will compile and run correctly. In addition, the header describes the purpose of the example and other useful information.

CHAPTER 3

Using the Absoft Editor

This chapter describes how to use the Absoft Editor to create and edit source files written in FORTRAN. Since word processors embed formatting characters in a document, using a word processor to create source files is not recommended. You can create source files in a word processor or another editor and export them in text format, but the features of the Absoft Editor make this unnecessary. The Absoft Editor incorporates powerful features for editing FORTRAN 77, FORTRAN 90/95, C, and C++ source files. However, this chapter will concentrate specifically on editing FORTRAN programs.

The Absoft Editor is a powerful tool for creating and maintaining program source files. It is source language sensitive and will display keywords and comments in different text colors, making them easier to distinguish in your source code.

With the Absoft Editor, you can edit multiple files at the same time, launch a compiler, and return to the editor to correct syntax errors detected by the compiler. The Absoft Editor is a Macintosh OS X program.

THE ABSOFT EDITOR

Basic editing functions are available as menu commands and there is usually more than one way to initiate any command:

- Select the command from the menu or tool bar.
- Type in the key equivalent (such as typing the Control and the letter O for the Open command).

Text Selection

Text may be selected for copying or deleting in two different manners. For small amounts of text, you can drag the cursor over the text while holding the mouse button down. For larger amounts of text, click the mouse button at the beginning of the selection, hold the **Shift** key down, and then click the mouse button again at the end of the selection.

Using Compilers

The editor can be used to run a compiler to either check the syntax of the source file or compile it into an executable application. Default compiler options are set with the **Preferences** control, described later in this chapter. Specific options for individual files can be set either in the **Source Info** dialog for the file (also described later) or in the **Option** Toolbar.

The **Option** Toolbar appears below the editor's menu bar:



You can choose from a predefined set of options or type in your own custom set for the file.

The **Goto** Toolbar also appears below the editor's menu bar, to the right of the **Option** Toolbar. It contains three controls that are of assistance when navigating source code files.



The first drop-down menu is used to select the type of objects to locate such as functions, modules, derived types, and so on. Once a type is selected, the second drop-down menu displays the names of all those objects in the file. Select an item from this menu to navigate directly to the location of that object in the source file. The third control will refresh the list if you are actively creating new source with new objects.

Pop-up menus

Holding down the **control** key and clicking the mouse button with the cursor positioned over an open file window will display a pop-up menu of context sensitive commands.

Cut

This command removes the selected text from the front-most window and places it on the clipboard. Text on the clipboard may be pasted into other windows.

Copy

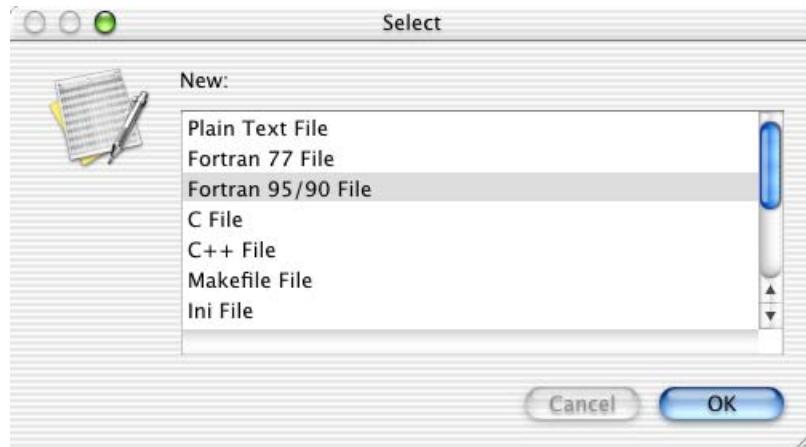
The Copy command copies the selected text from the front-most window and places it on the clipboard. Text on the clipboard may be pasted into other windows.

Paste

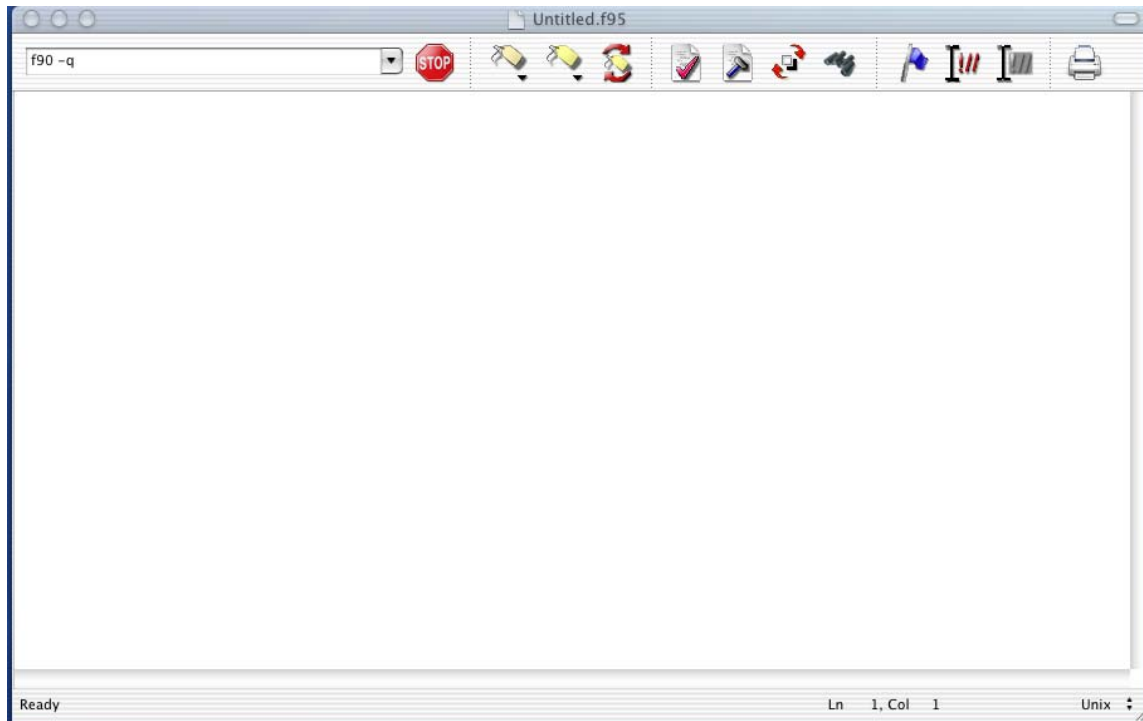
This command replaces the selected text in the front-most window with the text on the clipboard. If no text is selected in the front-most window, the clipboard text is inserted at the insertion point.

CREATING NEW SOURCE FILES

To create a new source file, choose the **New** command from the **File** menu. If **Display new file dialog box** is checked on the **Format** tab of the **Preferences** property sheet (See **Preferences** below) the following dialog will be presented, allowing you to specify the type of new file to create:



Otherwise, the default file type will be determined by the **Use as new file default format** setting on the **Format** tab of the **Preferences** property sheet.



The window will be untitled (it will have the name “Untitled”) until the first time you save it. At that time you will be asked to name the file. Text can be entered and edited using the same basic editing techniques that you use with any Macintosh-based text editor or word processor. You can cut and paste text within the window and move the cursor to enter text at any line.

Manipulating Windows

One or more file windows can be open at any time in the Absoft Editor, allowing you to easily cut and paste text between files.

When working with multiple windows, it is important to note that the Absoft Editor distinguishes between the active window (front-most window) and any inactive windows. Editing commands initiated from the menus will affect or insert text in the active (front-most) window. If you want to know which window is your active window, check the **Window** menu. In the **Window** menu, the active window will have a check mark next to it.

USING THE EDITOR MENUS

The rest of this chapter describes the editor commands in the **Application**, **File**, **Edit**, **Format**, **Tools**, **Window**, and **Help** menus. The name of the command is given, followed by its keyboard equivalent (if any) and a description of its function.

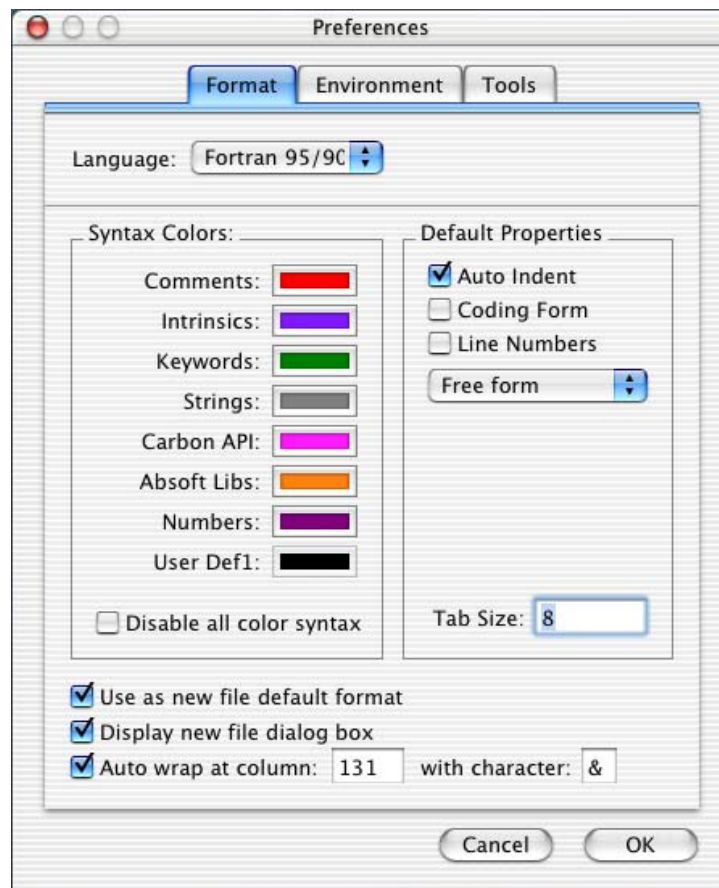
Application Menu

The **Application** menu contains the standard commands for Macintosh OS X applications. It also includes the **Preferences** command which is used to configure the Absoft Editor to your specific editing needs. Preferences consist of three separate property tabs: **Format**, **Environment**, and **Tools**.



Preferences...

Format



The **Format** pane contains the following controls:

Language

This selection box is used to establish the source language for making the settings in the next sections. The default choices are any of **Plain Text**, **Fortran 77**, **Fortran 90/95**, **C**, **C++**, **Makefile**, or **Ini**. Additional, languages may be added by editing the appropriate files in `./Resources/General Resources`. See the **Color** section below for more information.

Syntax Colors

Use this section to change the default use of color and the actual color in the text of **Comments**, **Intrinsics**, **Keywords**, literal **Strings**, and **Numbers** for the selected language. Languages may define up to eight unique colors, and eight or more keyword lists. Some additional color syntax settings for a given language may be viewed, if available, by using the up and down buttons. Additional, user defined keywords may be added by editing the associated resource files located in `./Resources/General Resources/Languages`.

Disable all color syntax

Checking this check box disables syntax coloring for the selected language.

Use as new file default format

This check box controls the default formatting for new files. For example, if you want new (and unnamed) files to default to the Fortran 90/95 format specifications that you have established, choose **Fortran 90/95** from the **Language** selection box and then check this check box.

Display new file dialog box

This checkbox enables the new file type dialog box. Check this check box if you would like to interactively choose the type of file and formatting each time a new file is created. The default file type chosen with the **Use as new file default format** setting will be the initial file type selected in this dialog box.

Auto wrap at column

This option is available when either Fortran 90/95 or Fortran 77 are specified in the **Language** selection. If you place a check in this box, the editor will automatically wrap and continue your source line if you type past the specified column. The character you choose will be placed in column six. The default character is the ‘&’ character.

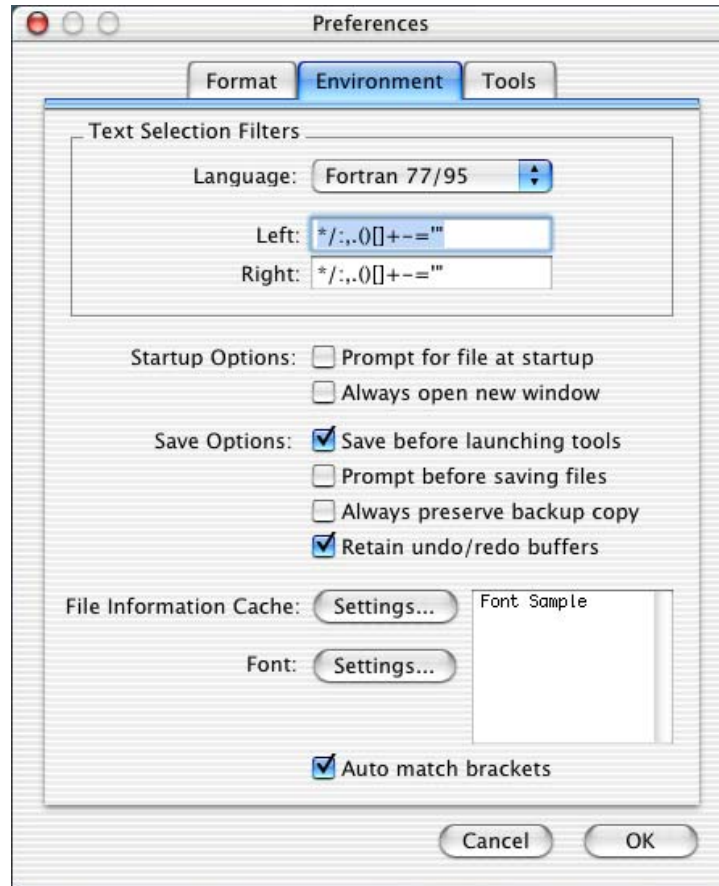
Default Properties

This section controls the default property settings for all files of the currently selected language type (see **Show Info** in the **Edit** menu discussed later in this chapter). You can choose to have **Line Format**, **Coding Form**, and **Line Numbers** shown by default in the text window with your file. (**Coding Form** is available only for Fortran 90/95 and Fortran 77.) You can also set the default **Tab Size** and whether or not you want **Auto Indent** enabled.

The drop down list box is used to confirm or change the interpretation that the Absoft editor applies to the format of the source file.

Use the **Show Info** command in the **Edit** menu to set properties for individual source files.

Environment



The **Environment** pane has the following controls:

Prompt for file at startup

Check this check box if you want the Absoft Editor to prompt you for the name of a file to open rather than starting up with a blank screen or automatically creating a new file.

Always open new window

Check this check box if you want the Absoft Editor to automatically create a new file rather than starting up by prompting you for the name of a file to open or starting up with a blank screen.

Save before launching tools

Check this check box if you want the Absoft Editor to save changes to any open files before starting a compiler.

Prompt before saving files

Check this check box if you want the Absoft Editor to issue a prompt, allowing you to confirm or cancel saving changes to any open files before starting a compiler.

Always preserve backup copy

Check this box if you want the Absoft Editor to maintain a backup copy of the file before every **Save** command. The backup file will have the extension `.aeb`.

Text Selection Filters

The controls in this section establish the logic used for selecting text elements. These controls allow you to select text based on the semantics and syntax rules of the programming language of the source file.

Font Settings

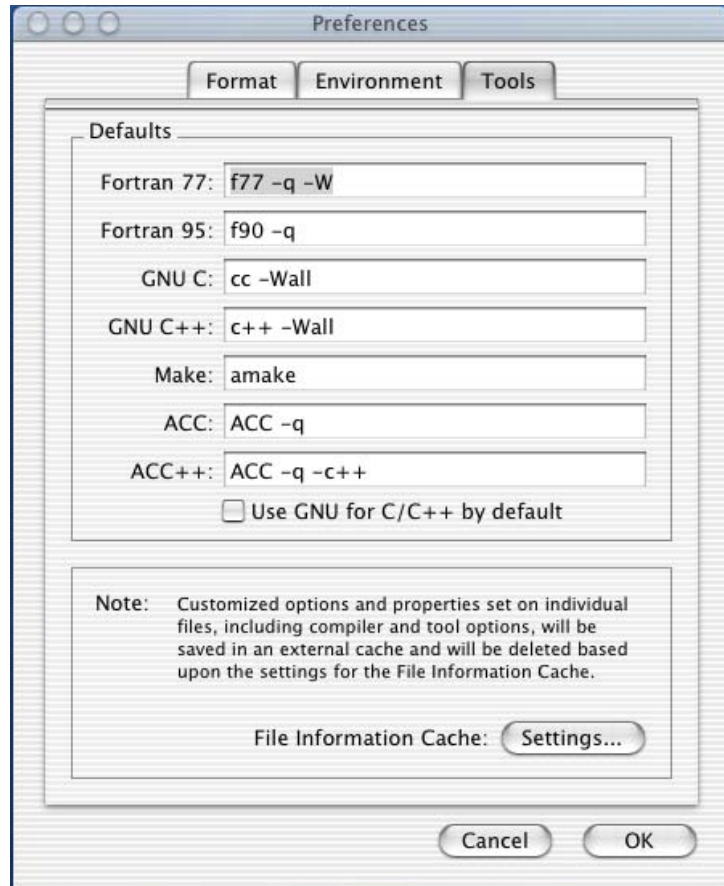
Use this button to select the text font and size you wish to use.

Cache Settings

The editor maintains certain information about each file that you open including the current cursor position, the tab size, etc. Use this **Settings** button to specify how long this information should be maintained since the last time the file was opened.

Tools

The text boxes in this section allow you to set the path to the tool that you prefer to use for each of the source file types. You can also supply default options for individual tools with these text boxes.



File Menu

The **File** menu contains commands for creating, opening, saving, and closing files. There are also commands for printing and for establishing your preferences for the way that the Absoft Editor operates.

New...(⌘N)

This command creates a new window for entering and editing text. The window will be untitled (it will have the name "Untitled") until the first time you save it.

File	
New	⌘N
Open...	⌘O
Open Recent	▶
Open Selected	
Open Complement	⌘⇧↑
Close	⇧⌘W
Close All	⌘⇧W
Save	⌘S
Save As...	⇧⌘S
Save All	⌘⇧S
Revert	
Page Setup...	⇧⌘P
Print...	⌘P

Open...(⌘O)

Use this command to open an existing file. This command displays a standard file selection dialog box to select the file to be opened. If you select a file that is already open, the window that contains that file will be brought to the front of the editor.

Open Recent

Up to 16 files will appear in this list. They represent the file that have been most recently opened in the Absoft Editor. They are listed as a convenience for quickly opening files for editing.

Open Selected

This command opens the file selected in an include statement. It may be a Fortran or a C include statement.

Open Complement

This command opens the complement of the current programming language file. If a C or a Fortran source file is open, the header or include file with the same root name will be opened. If a header or an include file is opened, the source file with the same root name will be opened.

Close (⌘W)

This command closes the file displayed in the front-most window. If any unsaved changes had been made to the text, you will be asked to save it.

Close All

This command closes all files. If any unsaved changes had been made to any files, you will be asked to save them.

Save (⌘S)

Choose this command to save the text in the front-most editor window. The first time you save, you will be asked to provide a name and a path for the file. Thereafter, each time you save, the changes will automatically be written to this file. If no changes have been made, this menu command is dimmed and unavailable.

Save As...

Use the **Save As** command to save the text in the front-most editor window to a different file. A standard file save dialog will appear, allowing you to specify the name of file. The front-most editor window becomes the newly named file.

Save All

Use this command to save the text in all open windows.

Revert

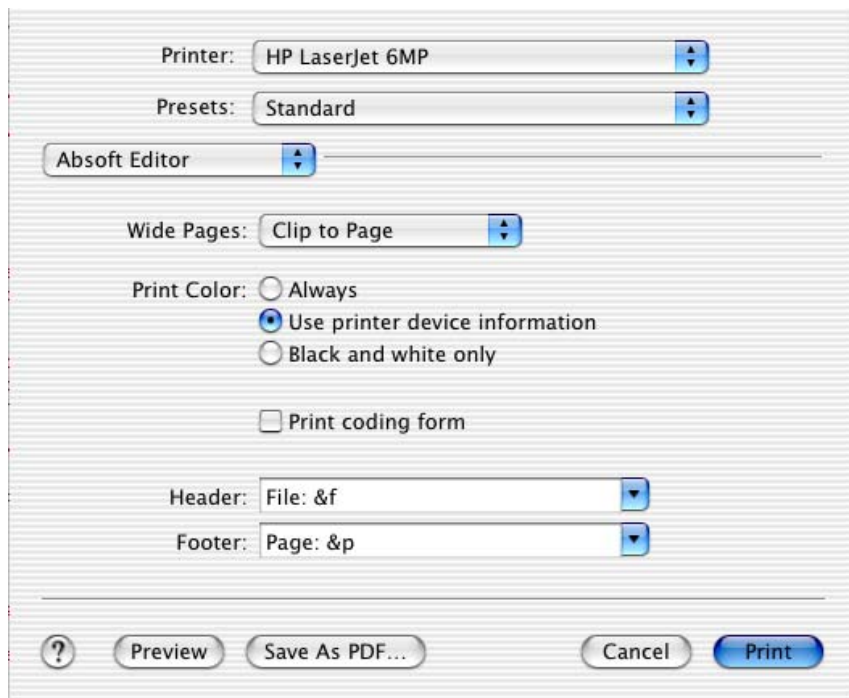
The **Revert** command restores the text in the editor window to its previously saved state.

Print Setup...(⌘P)

Use this command to display the standard **Print Setup** dialog for printing.

Print... (⌘P)

This command prints the front-most window to the currently selected printer.



In addition to the standard printing options, the Absoft Editor supports the following additional print options by selecting the **Absoft Editor** category from the pop-up menu:

Wide Pages

You can select from **Tile Across Page(s)**, **Clip to Page**, or **Fit to Page** to tile document text across multiple pages, crop horizontal printing to the width of a single printed page, or scale the document size so that all text fits horizontally on a printed page. The default is **Clip to Page**.

Print Color

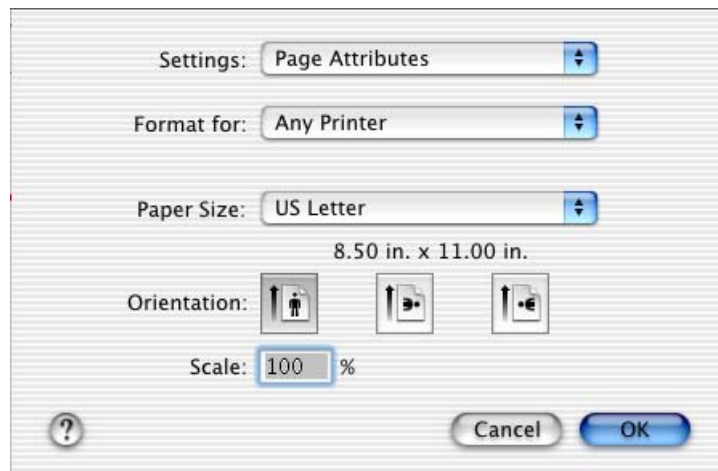
Select the desired color printing support from **Always**, **Use printer device information**, or **Black and white only**. The default is to use the current printers device information to determine level of color printing supported.

Header and Footer

Use these text boxes to configure the default page headers and footers. The default is to display the file name in the header and the page number in the footer. Additional options include printing the date and/or time.

Page Setup...

Use this command to open the **Page Setup** dialog:



This dialog allows you to specify a page header and footer. Click on the right-arrow buttons to insert the text you wish to display in the Header and the Footer: **File Name**, **Page Number**, **Date**, and **Time**. This dialog also allows you to specify the document margins.

Edit Menu

The commands in the **Edit** menu are used for performing standard editor functions, such as Cut, Copy, and Paste.



Undo (⌘Z)

Use this command to undo changes made in the front-most window.

Redo (⇧⌘Z)

Use this command to redo commands that were undone with the **Undo** command.

Cut (⌘X)

This command removes the selected text from the front-most window and places it on the clipboard. Text on the clipboard may be pasted into other windows.

Copy (⌘C)

The Copy command copies the selected text from the front-most window and places it on the clipboard. Text on the clipboard may be pasted into other windows.

Paste (⌘V)

This command replaces the selected text in the front-most window with the text on the clipboard. If no text is selected in the front-most window, the clipboard text is inserted at the insertion point.

Clear

Similar to cut, this command removes the selected text from the front-most window, but does not place a copy on the clipboard.

Select All (⌘A)

Use this command to select all of the text in the file displayed in the front-most window.

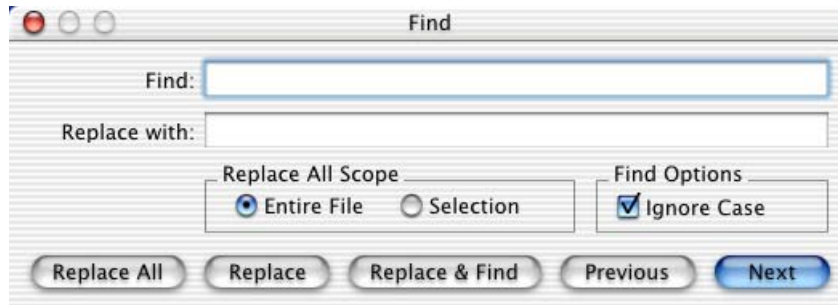
Find

This command displays the Find sub-menu with commands for finding and replacing text within the file.

Find...	⌘F
Find Next	⌘G
Find Previous	⇧⌘G
Use Selection for Find	⌘J
Jump to Selection	⌘J

Find (⌘F)

Use this command to open the **Find** dialog for locating specified text within the front-most window.



The controls in the **Find** dialog are used as follows:

Find

Enter the text string you wish to locate here. A history of previous uses of this command is maintained, allowing you to select strings that you located earlier.

Replace

Enter the text string that will replace found text. This text is used with the **Replace All**, **Replace**, and **Replace & Find** buttons.

Ignore Case

Check this box to find text occurrences in your source file that match your specified text regardless of capitalization and case.

Find Next (⌘F)

This command repeats the last **Find** command in the front-most window by searching forwards in the file.

Find Previous (⇧⌘F)

This command repeats the last **Find** command in the front-most window by searching backwards in the file.

Bookmarks

Bookmarks provide an easy way to “save your place” in a file so that you can later return there quickly. Positioning the insertion caret on the line where you want the bookmark set and then typing **⌘F2** sets (or unsets) a bookmark. In other words, **⌘F2** toggles a bookmark.

A bookmark appears as a small flag at the beginning of the line. Pressing the **F2** key alone moves the insertion caret to the next bookmarked line in the file. Holding the **Shift** key down and pressing the **F2** key moves the insertion caret to the previous bookmarked line in the file. Holding the **Shift** key down and typing **⌘F2** clears all bookmarks in the file.

Bookmarks Menu

The Bookmarks sub-menu provides commands for setting, clearing, and moving between bookmarks.

Toggle Bookmark	⌘F2
Previous Bookmark	⇧F2
Next Bookmark	F2
Clear File's Bookmarks	⇧⌘F2
Clear All Bookmarks	⌘⇧F2

Toggle Bookmark (⌘F2)

Use this command to set or unset a bookmark on the line where the insertion caret is positioned.

Previous Bookmark (⇧F2)

Use this command to move to a previous bookmark location in the file.

Next Bookmark (F2)

Use this command to move to the next bookmark location in the file.

Clear File's Bookmarks (⇧⌘F2)

Use this command to remove all bookmarks in the file.

Clear All Bookmarks (Control⌘F2)

Use this command to remove all bookmarks in all files.

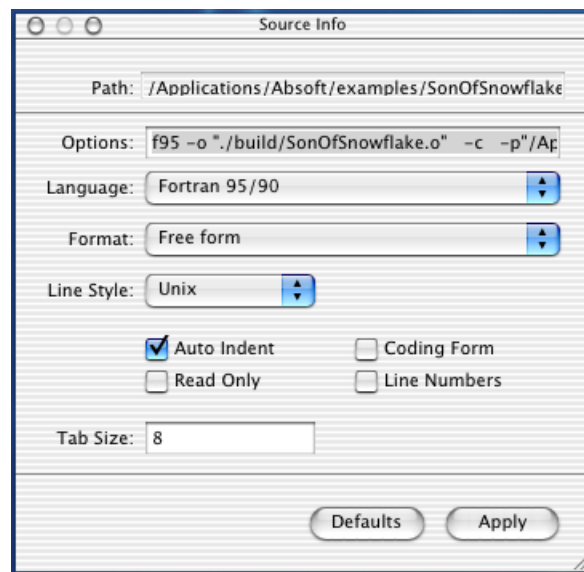
Format Menu

The commands in the **Format** menu provide you with a number of useful commands for managing and formatting your source files.

Show Info... (⇧⌘I)

The **Show Info** dialog is used to control certain characteristics of the display of the file in the front-most window. It also allows you set specific commands and options if you use the Absoft Editor to compile the source file. The characteristics that can be controlled include:

Format	
Show Info	⇧⌘I
Insert Continuation	⌘I
Go to Line...	⌘L
Match Brackets	⌘{
Shift Left	⌘[
Shift Right	⌘]
Comment	⌘/
Uncomment	⇧⌘/
Convert to Upper Case	
Convert to Lower Case	



Options

If the Absoft Editor was *not* started from the Absoft Developer Tools Interface, this field contains the tool option settings for the type of file as established on the **Tools** page of the **Preferences** dialog (see **Preferences** in the **File** menu, discussed earlier in this chapter, for more information).

If the Absoft Editor was started from the Absoft Developer Tools Interface, this field contains the compiler settings for the project and source file.

Language

Use this selection box to establish or change the source language setting for this file.

Format

This drop down list box is used to confirm or change the interpretation that the Absoft editor applies to the format of the source file.

Line Style

This selection box controls how end-of-record (end-of-line) characters are interpreted. Lines in DOS files end with a carriage return/line feed pair. Macintosh lines end with a carriage return only. Files on a Unix system have lines terminated by line feeds only.

The file must be saved before changes to this parameter will take affect.

Auto Indent

This check box enables automatic spacing of new lines to the column where the previous line started.

Coding Form

FORTRAN 77 specifies the purpose of individual columns within a source statement record. A check in this check box will color certain columns to make them easier to identify. The specific columns are 6, 72-80, and 132-140. For more information on source record fields, refer to the section **FORTRAN 77 ANSI Standard** in **The Fortran 77 Program** chapter of the *FORTRAN 77 Language Reference Manual*.

Read Only

A check in this check box will prevent you from making any inadvertent changes to a file that you wish to maintain as read only.

Line Numbers

A check in this check box will display the line numbers of the source lines in the file.

Tab Size

The value in this edit box controls how many spaces a tab is expanded to. Note that for column oriented languages such as FORTRAN 77, the value in this field should agree with how the compiler interprets tab characters.

Insert Continuation (⌘I)

Use this command to insert a Fortran continuation line immediately after the line on which the cursor is positioned. The cursor will be repositioned to the next character position following the continuation character. The continuation character used is defined on the **Format** page of the **Preferences** dialog. See **Preferences** in the **Application** menu, described earlier for more information.

Go to Line (⌘L)

This command opens the **Goto** dialog. Enter the line number of the line you wish to go to and click on the **Select** button.



Match Brackets (⌘t)

When editing any file, this command may be used to find the matching closing character for the opening character next to the cursor on the line where the cursor is positioned. The characters it will match are: (), {}, [], and <> – parentheses, braces, brackets, and less-than greater-than. Double-clicking on an opening or closing character will select the text that the characters enclose.

Shift Left (⌘l)

Use this command to shift selected text to the left by one tab stop.

Shift Right (⌘r)

Use this command to shift selected text to the right by one tab stop.

Comment

This command inserts an exclamation mark (!) in column one of the current line or the selected lines.

Uncomment

This command deletes an exclamation mark (!) from column one of the current line or the selected lines.

Convert to Upper Case

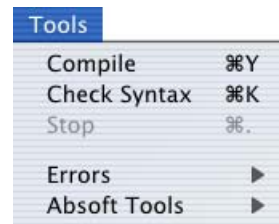
This command converts the selected text to upper case.

Convert to Lower Case

This command converts the selected text to lower case.

Tools Menu

The commands in the **Tools** menu are used to invoke various functions of the language compilers you use. If the compiler issues any diagnostics, they will be reported to you in the **Errors** window. Errors are indicated with a red circle (stop sign) next to the diagnostic message and warnings are indicated with an inverted yellow triangle (caution sign). You can double-click on any line in this window to go directly to the corresponding line in the Absoft editor window. The **Errors** window is automatically opened only if the compiler issues warnings or errors.



Compile (⌘Y)

Use this command to compile the file in the front-most window. The tool that you have selected for the file type will be started with the command line shown in the Option toolbar.

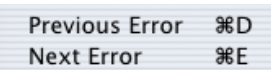
Check Syntax (⌘K)

This command invokes the check syntax phase of the compiler that you have selected for the type of file in the front-most window.

Stop (⌘.)

Use this command to stop the compiler started with either the **Check Syntax** or **Compile** command.

Errors



This sub-menu contains commands for moving to the previous or next error or warning in the source file.

Previous Error (⌘D)

If you compiled your source file with the **Compile** command in the **Tools** menu and error diagnostics were issued by the compiler, you can use this command to go to the previous error in the source file.

Next Error (⌘E)

If you compiled your source file with the **Compile** command in the **Tools** menu and error diagnostics were issued by the compiler, you can use this command to go to the next error in the source file.

Window Menu

The commands in this menu allow you to arrange the open windows in the Absoft Editor and to bring a specific window to the front.

Close Window (⌘W)

This command closes the front-most window in the Absoft Editor. It is the same as the **Close** command in the **File** menu.

Zoom Window

This command maximizes the front-most window.

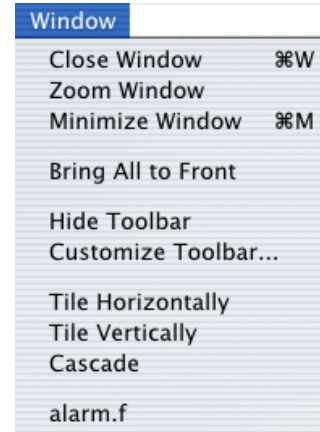
Minimize Window (⌘M)

This command minimizes the front-most window and places it on the docking bar.

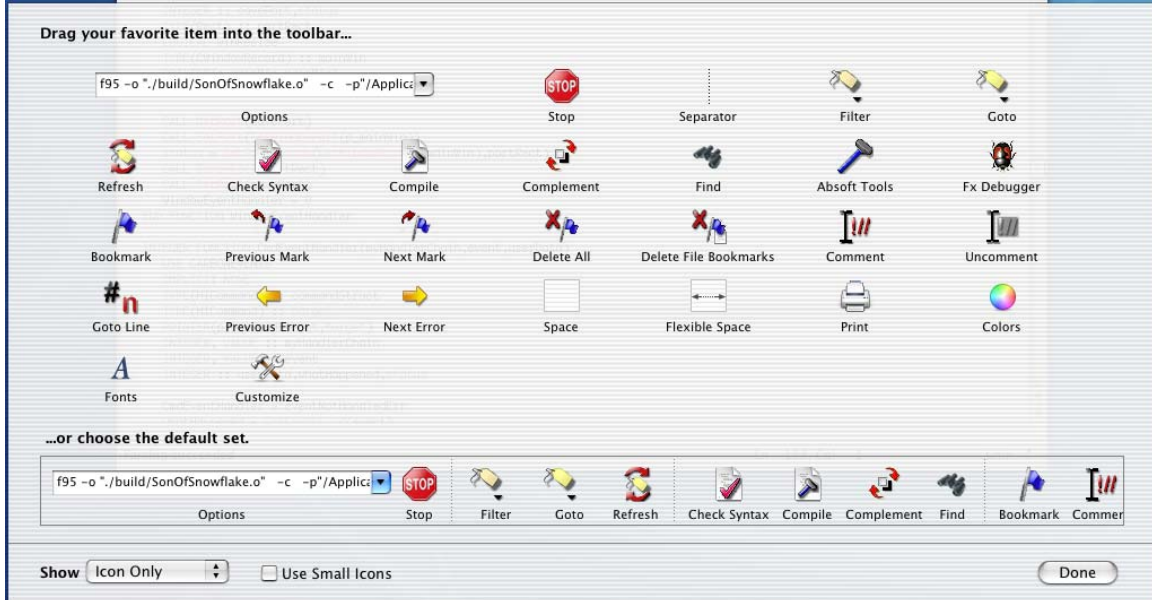
Hide Toolbar

This command removes the toolbar from the front-most window.

Customize Toolbar...



This menu command opens the Toolbar editor that allows you to customize the toolbar.



Tile Horizontally

Use this command to arrange your windows horizontally based on the order in which they were opened.

Tile Vertically

This command is used to arrange your windows vertically based on the order in which they were opened.

Cascade

This command is used to arrange your windows in a cascaded fashion.

Window list

Selecting the name of a window in this list will bring it to the front, restoring it to its previous size if it has been minimized.

Help Menu

The commands in this menu are used to obtain help on the listed topics.



Tools Help

Use this command to view Absoft Developer Tools Interface application online help files.

Hide ToolTips

Use this command to disable pop-up tooltips for all windows and controls in the Absoft Developer Tools Interface for the remainder of the session.

CHAPTER 4

Using the Compilers

This chapter describes how to use the Absoft Fortran 90/95 and FORTRAN 77 compilers to create executable files on the Mac OSX operating system. Beginning with an overview of invoking the compilers, this chapter explains how to compile a small number of Fortran source files into an executable application. Next the Absoft Developer Tools Interface, AbsoftTools, application is described with detailed descriptions of options and compiler settings.

COMPILING PROGRAMS

Three methods of compiling programs are available: a traditional command line, the Absoft Developer Tools Interface, and *makefiles*. The command line and the Absoft Developer Tools Interface application are discussed in this chapter. Makefiles and the Absoft make utility, `amake`, are described in the chapter **Building Programs**.

Source file names and compiler options are selected with the mouse pointer in the Absoft Developer Tools Interface application. Arguments to the command line version are typed in on the command line.

USING THE COMMAND LINE

To use a command line version of any of the Absoft compilers, you must first open a Terminal application shell and set a number of environment variables that assist and control the use of the compilers.

A command line version of an individual compiler can be started with one of the commands: `f95`, or `f77`.

```
f95 [options] files
```

```
f77 [options] files
```

The various options are described in the specific compiler options sections later in this chapter.

FILE NAME CONVENTIONS

Compilation is controlled by the two compiler drivers: `f77` and `f95`. These drivers take a collection of files and, by default, produce an executable output file. Acceptable inputs to `f95` are:

File Type	Default form
Free format Fortran 90/95 source files	<code>file.f90</code> or <code>file.f95</code>
Free format Fortran 90/95 preprocessor files	<code>file.F90</code> or <code>file.F95</code>
Fixed format Fortran 90/95 source files	<code>file.f</code>
Fixed format Fortran 90/95 preprocessor files	<code>file.F</code>
C language source files	<code>file.c</code>
Assembly language source files	<code>file.s</code>
Relocatable object files	<code>file.o</code>

Acceptable inputs to `f77` are:

File Type	Default form
FORTRAN 77 source files	<code>file.f</code> or <code>file.for</code>
FORTRAN 77 preprocessor files	<code>file.F</code> or <code>file.FOR</code>
C language source files	<code>file.c</code>
Assembly language source files	<code>file.s</code>
Relocatable object files	<code>file.o</code>

File names that do not have one of these default forms are passed to the linker. It is assumed that the C compiler (`cc`), assembler (`as`), and linker (`ld`) are installed on the system and use standard command line syntax.

Output file names take the form:

File Type	Default form
Assembly language source files	<code>file.s</code>
Relocatable object files	<code>file.o</code>
Precompiled module file	<code>file.mod</code>
Executable object files	<code>a.out</code>

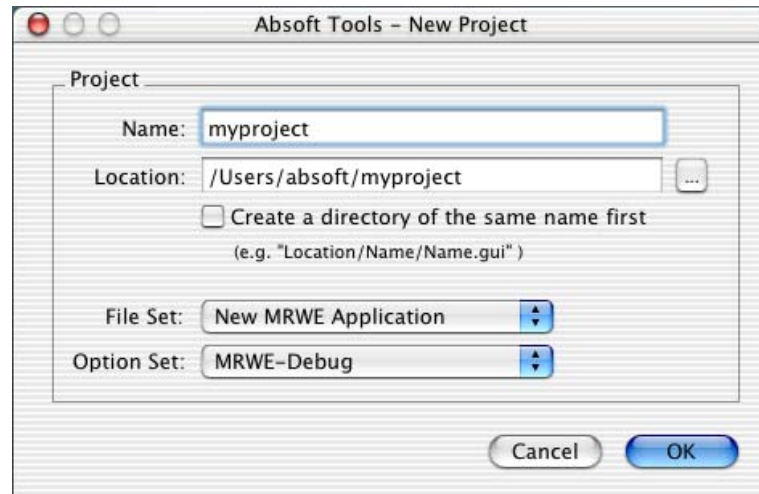
USING THE ABSOFT DEVELOPER TOOLS INTERFACE

The Absoft Developer Tools Interface, `AbsoftTools`, is started by double clicking on the `AbsoftTools` icon, the equivalent alias icon in the Finder, or by clicking on the `AbsoftTools` application icon in the *Dock*.

Working with Projects

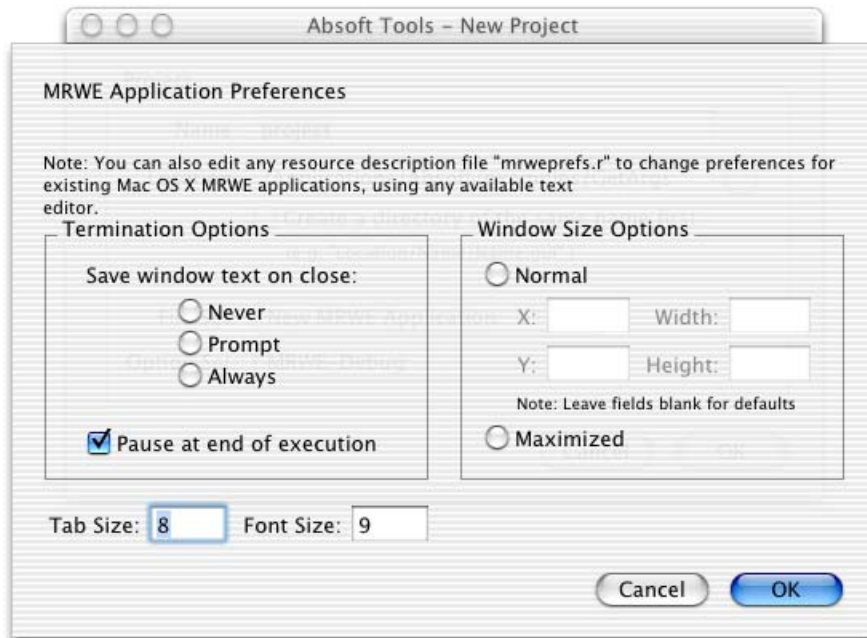
A *project* allows you to organize the entire source, object, include, library, and resource files that constitute an application. It keeps track of which files are associated with the application, which ones are dependent on other files, which ones have been recently modified and need to be rebuilt. Also, it allows you to set specific options to be used with the compilation tool associated with the various files in the project.

The first step in working with a project is to create a new one. Use the **File** menu **New...** command to create a new project. The **New Project** dialog will appear as shown below:



Name is the name that will be applied to the project file, the associated makefile, and the default name of the executable program. **Location** is the directory where the project file and the associated makefile are created. Use the button labeled with an ellipsis to open a directory browser to change the directory. Use the **Create a directory of the same name first** check box to automatically create a subdirectory with the supplied name before creating the actual project files. The **Option Set** drop-down menu is used to choose a specific set of pre-determined options for the project. (See **Set Default Options**, described later in this chapter, for information on configuring default option sets for new projects.) The **File Set** drop-down menu is used to select a default set of source files to create for the project. The **Empty Project** set creates a project with no source files by default. The **New Fortran 95 Project** and the **New Fortran 77 Project** sets create projects containing a new source file with the same name as the project file in the project directory. The **New MRWE Application** menu item creates all the source files necessary for a **Macintosh Runtime Window Environment** application, including a Fortran 95 file, a custom *Info.plist* file and the resource description files.

After you click the **OK** button, if you selected a new MRWE application, the **MRWE Application Preferences** dialog will appear. This dialog contains controls that allow you to customize how MRWE handles text and text saving. It also allows you to specify what your application does at the end of execution: quit immediately or pause automatically. You can also choose the desired size of your MRWE window by providing screen location and sizes or choosing to have your window appear maximized.



After you click the **OK** button to accept these values, the project options panel will be displayed, allowing any of the global compiler options, Plugin options, or Build options to be set or changed. After these are established, begin selection of the files that will constitute the project by choosing the **Add/Remove File(s)** command from the **Configure** menu.

Options Dialog

The Options dialog appears automatically after you confirm the settings for a New Project. The Options dialog can be invoked in several manners: by selecting the **Set Project Options** command to configure options for the current project, using the **Set Default Options** command to configure a set of default options used in the creation of new projects, or using the **Set File Options** command to configure options for the selected source file.

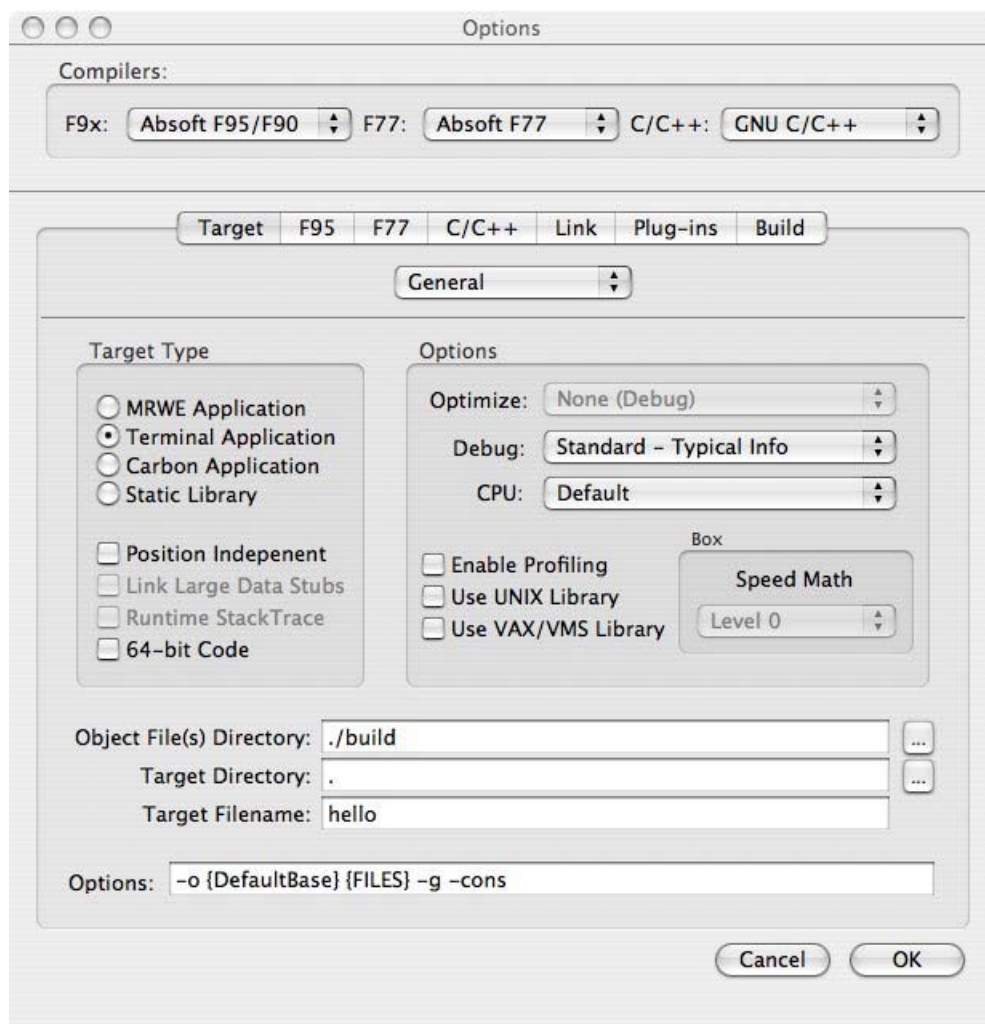
The Options dialog has these features:

- Any button that contains ellipses, such as **Set Module Path(s)...**, will bring up additional dialogs.
- As options are checked in the dialog, they will be added to the **Options** text field in the lower portion of the current options pane.

The Options panel displayed with the **Set Project Options** command also has tabs for the individual compilers and the linker. Within the Options dialog, you can select options to modify the way compiled applications function. When the Build command is invoked, a makefile is constructed and the source code will be compiled and linked into an executable program, ready to run. If any errors occur, they will be listed in the Output pane, under the **Build** tab. See the chapter **Building Programs** for information on makefile commands and using `make`.

Target Tab – General Options

The **Target** tab controls two option subsets that apply to all of the files for a project: **General** and **FPU**. Choose the desired subset from the **Options Subset** drop-down menu. General options are described first:



Target Type

The radio buttons and check boxes in this area of the Target tab control the type of application produced by the compilation process. By default, a stand-alone terminal

application (the **Terminal Application** button) is generated. This type of application runs in the Terminal application command line shell. Use **-cons** to select this option on a command line.

An application linked with MRWE (the **MRWE Application** button) creates an application with a Macintosh style interface (see the chapter, **Macintosh Programming** for more information). Use **-mrwe** to select this option on a command line.

Use the **-plainappl** option (selected with the **Carbon Application** button) when you are creating an application with an interface that you supply. This type of application will have neither an MRWE interface nor a Terminal application shell interface – you are responsible for the interface presented to the user. The option includes all of the standard Carbon API frameworks.

Specify the **Static Library** option if you are creating a static library (see **Linking Programs** and **Creating Libraries** in the chapter titled **Building Programs**).

Link Large Data Stubs

The Carbon and Cocoa development APIs have initialization routines that are called during application startup. For console applications, these are replaced by symbols in the data segment. This imposes a limit on the size of a console application's data segment. The Link Large Data Stubs option removes this limit by supplying stub routines located in the text section.

Runtime Stack Trace (-et)

Use this option to enable a diagnostic stack trace back if a catchable error occurs at runtime. The file name, line number, and list of calling procedures will be displayed at the point of the error. Errors that can be caught include memory access violations, I/O errors, and integer division by zero.

NOTE: This option is not implemented in Pro Fortran V10.0.

64-bit code (-m64)

This option is used to produce 64-bit executables. Programs compiled with the **-m64** option have access to the full addressing capabilities of 64-bit processors.

Options

The **Optimize** and **Debug** drop-down menus are used to control the production of object code. Selecting one of the available optimization choices in the Optimize menu enables compiler optimizations.

The **Optimize** options control compile time optimizations to generate an application with code that executes more quickly. Absoft Pro Fortran is a globally optimizing compiler, so various optimizers can be turned on which affect single statements, groups of statements or entire programs. There are pros and cons when choosing optimizations; the application will execute much faster after compilation but the compilation speed itself will be slow. Some of the optimizations described below will benefit almost any Fortran code, while others should only be applied to specific situations. Any optimization option automatically enables **-cpu:host** (see **CPU** below).

The **Basic (-O1)** option will cause most code to run faster and enables optimizations that do not rearrange your program. The optimizations include common subexpression elimination, constant propagation, and branch straightening.

The **Normal (-O2)** option enables advanced optimizers that can substantially rearrange the code generated for a program. The optimizations include strength reduction, loop invariant removal, code hoisting, and loop closure. This option is not usable with debugging options.

The **Advanced (-O3)** option enables advanced optimizers that can significantly rearrange and modify the code generated for a program. The optimizations include loop permutation (loop reordering), loop tiling (improved cache performance), loop skewing, loop reversal, unimodular transformations, forward substitution, and expression simplification. This option is not usable with debugging options.

The **Fast (-O4)** option enables all of the above optimizations and invokes the IPA (Inter-Procedural Analyzer) linker. IPA can significantly increase compile time. Also, IPA can cause previously working, but incorrect programs to fail. Since IPA may reorganize storage, incorrect pointers or array bounds errors may be exposed. This option is not usable with debugging options.

When **None (Debug)** is selected in the **Optimize** drop-down menu, the **Debug** options menu is enabled allowing you to select the level of symbolic information to produced for debugging with Fx (see the chapter **Using the Fx Debugger**):

Standard produces an object file containing typical debugging information with entry points, line numbers, and program symbols. This is the standard debugging option and is selected on the command line with the **-g** switch.

Full forces the compiler to place information in the debugger symbol tables for all structures whether they have associated storage or not. Normally, the compiler does not place information in the debugger symbol tables for structures that are only declared, but never have storage associated with them. This keeps the symbol tables to a manageable size when include files are used to make structure declarations. This option is selected on the command line with the **-g** and **-N111** switches.

Use the **CPU** options to target object code to a specific type of processor. This option is selected on the command line with **-cpu: *type*** switch. The recognized ***type*** arguments are:

default	automatically establishes <i>type</i> based on the processor in the machine that the program is compiled with
Intel	automatically establishes <i>type</i> based on the processor in the machine that the program is compiled with

The **Enable Profiling** option (**-P** from the command line) causes your program to be instrumented for runtime profiling with the `gprof` tool. For information on using the Mac OS X profiler, see the Mac OS X manual page for *gprof*.

The next two options in this section, **Use UNIX Library** and **Use VAX/VMS Library**, automatically include these compatibility libraries in your project. The compatibility libraries are described in the *Absoft Support Libraries* manual that is included Pro Fortran.

Speed Math (-speed_math=*n*)

The **Speed Math** option enables aggressive floating point optimization. This option should be used with care as high values, while improving performance, may reduce accuracy; possibly to the level inaccurate results. From the command line, enable this option with **-speed_math=*n*** where *n* is an integer from 0-12.

Object File(s) Directory

This is the directory where all object files will be created and maintained.

Target Directory

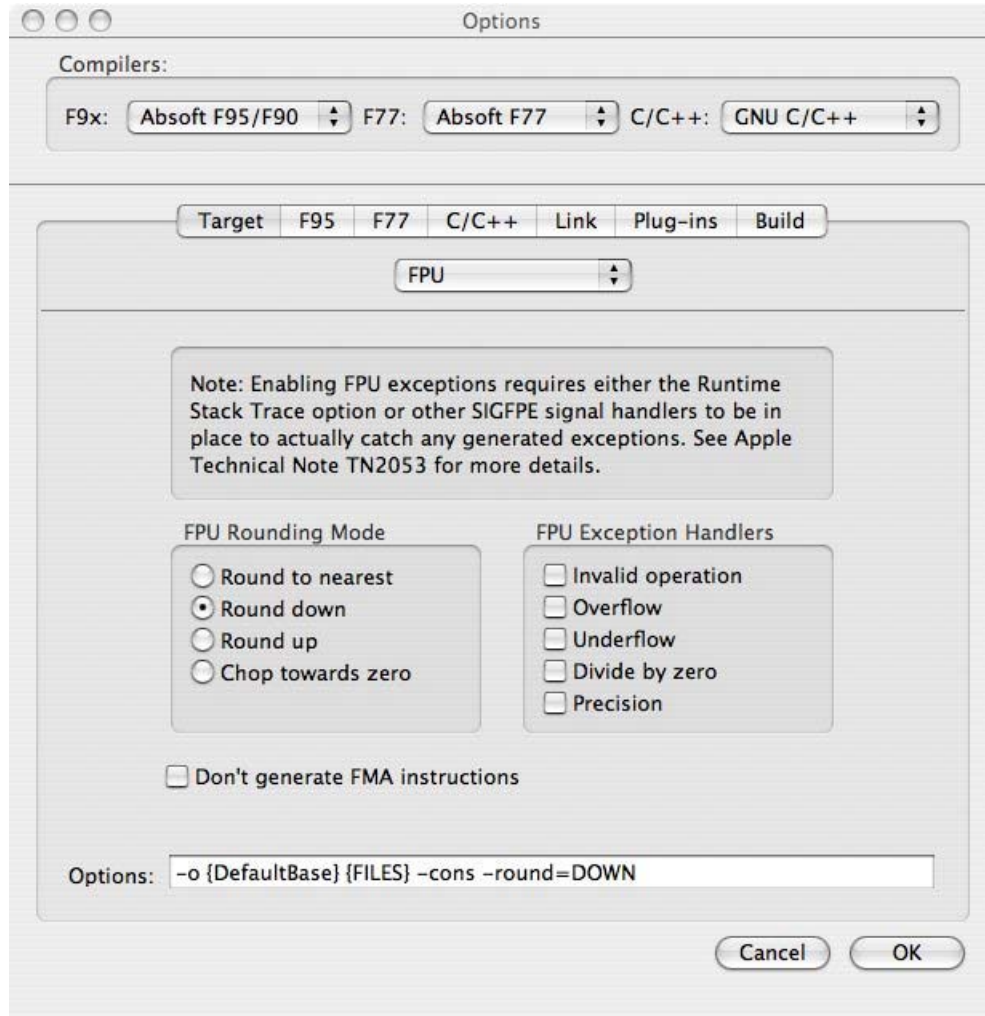
This is the directory where the executable file (or makefile target) will be placed.

Target Filename

The name of the generated executable file can be typed here. If no name is specified, the default is to produce an executable file called `a.out`. The executable file will be placed in the directory specified in the **Target Directory** text box. This option is specified on the command line as **-o *name***, where *name* is the name the executable file will be given.

Target Tab – FPU Options

The **FPU** option subset provides control over the operation of the *Floating-Point Unit* of the processor.



FPU Rounding Mode

These radio buttons control the rounding method used by the floating point unit. The default method is the IEEE P754 default state: **Round to nearest**. The rounding method can be controlled on the command line with the option:

`-round=mode`

where *mode* is one of:

NEAREST
DOWN
UP
TOZERO

FPU Exception Handling

When a floating-point exception is produced, the default action of an application is to supply an IEEE P754 defined value and continue. For undefined or illegal operations (such as divide by zero or square root of a negative number) this value will usually be either Infinity (INF) or Not A Number (NaN) depending on the floating-point operation.

Checking any of the exception boxes will cause the program to stop and produce a core dump, rather than continue, if the exception is encountered. If the program is being debugged, it will stop in the debugger at the statement line that caused the exception. The syntax for using this option on the command line is:

-TENV=exception[,exception,...]

where **exception** is one or more of:

simd_imask	invalid operation exception.
simd_dmask	denormalized operand exception.
simd_zmask	divide by zero exception.
simd_omask	overflow exception.
simd_umask	underflow exception.
simd_pmask	precision exception.

Don't generate FMA instructions (-Q51)

Use of the **-Q51** option will cause the compiler not to use floating-point multiply-add type of instructions. Since there is no rounding performed between the multiplication and addition during the execution of these instructions, numeric results will vary depending on where they are used.

Other Target Options

The following options are not available with the graphical interface to the compiler but may be used with the command line interface or the make facility.

Generate Assembly Language (-S)

Specifying the **-S** option will cause the compilers to generate assembly language output in a form suitable for the system assembler. The file created will have the suffix “.s”. For example, compiling `test.f` with the **-S** option will create `test.s`. If any C source files are given as arguments to `f77` or `f95`, this option will be passed to the C compiler. If no other compiler process control options are specified and there are no relocatable object files specified on the command line, the compilation process will halt after all Fortran 90/95, FORTRAN 77, and any C source code files have been compiled to assembly language source.

Generate Relocatable Object (-c)

Specifying the `-c` option will cause the compilers to generate relocatable object files. In the Macintosh OS X environment, this option indicates that all source files (Fortran 90/95, FORTRAN 77, C, and assembly) should be processed to relocatable object files. If no linker options are present (see below), then the compilation process stops after all object files have been created. If any C source files are given as arguments to `f77` or `f95`, this option will be passed to the C compiler.

Library Specification (-l)

Specifying the `-lname` option will cause the linker to search the library file `libname.a`.

Library Path Specification (-L)

The `-Lpath` option will cause the linker to search the specified directory named in `path` for library files given with succeeding `-l` options.

ABSOFTEVELOPER TOOLS INTERFACE

The following sections describe the menu commands available in the Absoft Developer Tools Interface (AbsoftTools) application.

Application Menu

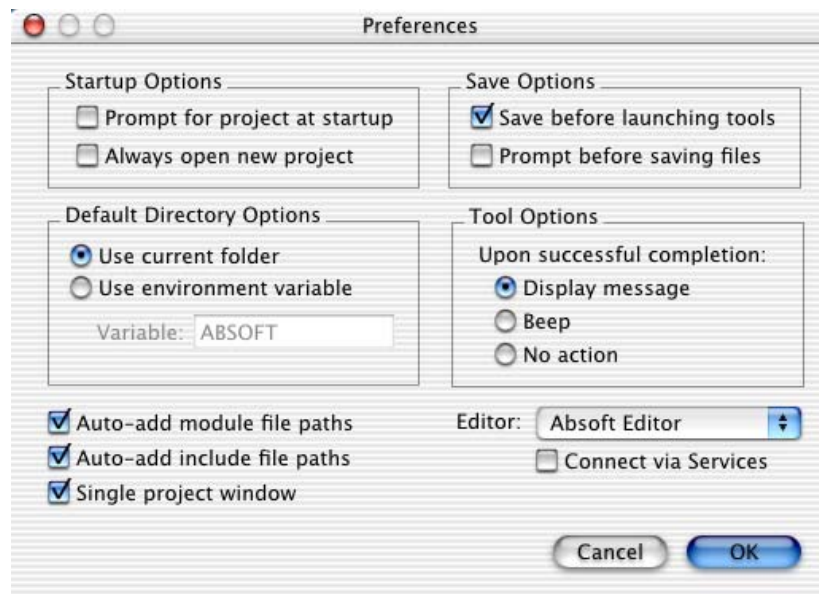
About Tools

This dialog displays various information about the application and its environment, including the version number.

Preferences...



This command opens the **Preferences** dialog for editing and customizing the way the Absoft Developer Tools Interface application operates. The options are divided into several groups:



Preferences

The **Startup Options** check boxes control what action (if any) the Absoft Developer Tools Interface application performs upon startup. If the **Prompt for project at startup** box is checked, the application will open a standard file dialog for selecting an existing project file. If the **Always open new project** box is checked, the application will open a standard file dialog for creating a new project file, each time the application is launched.

The radio buttons in the **Default Directory Options** area control the master directory used for creating new projects. It can be set to either the current working directory, or the directory indicated by an environment variable. The default environment variable is `ABSOFT` and points to the Pro Fortran installation directory.

The radio buttons under **Tool Options** allow you to specify the type of notification you wish to receive from the various developer tools.

Check the **Auto-add module file paths** box to automatically pass the path of any module that has been incorporated into the project to the appropriate tool.

If the **Auto-add include file paths** box is checked, the path of any include or header file that has been incorporated into the project is automatically passed to the appropriate compiler, as if the **Set Include Paths** command in the **Configure** menu had been used to specify these directories.

The **Single project window** control toggles modes between single window projects and dual project and output window style projects in the Absoft Developer Tools Interface application.

Use the **Editor** menu and the **Connect via Services** menu to set the editor application to launch for source file editing, and the inter-process communication style used to talk with the specified editor, respectively. Services are currently only provided by the **TextEdit+** application or the **Absoft Editor**. Selecting **BBEdit** from the pop-up menu will allow complete two-way inter-process communication between the retail version of BBEEdit for Mac OS X and Absoft Developer Tools Interface via a set of custom AppleEvents.

Hide Tools

Hide the Absoft Developer Tools Interface application.

Quit Tools

Exit the Absoft Developer Tools Interface application.

File Menu

New

Use this command to create a new project file. All file selections, include paths, and option settings are reset to default values. See **Working with Projects**, above, for details on creating a new project.

Open

This command is used to open a previously saved Absoft Developer Tools Interface project file with all file selections and option settings restored from that session.

Open Recent

Use the menu items on the submenu to open recently used project files.



Close

This command closes the current project. If you have a previously unsaved project already open, you will be prompted to save it before continuing.

Save

Use this command to save your current file selections and option settings so that they can be restored at a later time.

Save As

The **Save As** command saves your files selections and option settings to a different file than the one currently in use. This command is useful for making a copy of the current settings before making changes.

Revert To Saved

The **Revert To Saved** command restores the previously saved state of the current project file.

Page Setup...

Use this command to display the standard **Page Setup** dialog box for printing.

Print

The **Print** command prints the active output tab. You can print the entire window by pressing the **control** key while issuing this command.

New File

Use this command to create a new source file for the current project. The specified file will be added to the project file and opened using the default source file editor application.

Edit Menu

Undo

Use this command to undo the last editing action, if possible.

Redo

Use this command to reverse the last editing undo action, if possible.

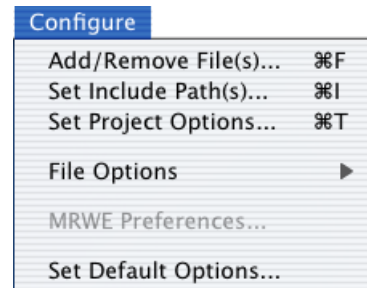


View Previous

The View menu contains two commands: **Previous** and **Next**. If error or warning diagnostics are issued by any the developer tools in the **Build** tab of the output pane, you can use the **Previous** command to go to the previous error in the source file. Use the **Next** command to go to the next error in the source file. These menu items can also be used when the **Search** tab is active to go to the next match in the list.

Configure Menu

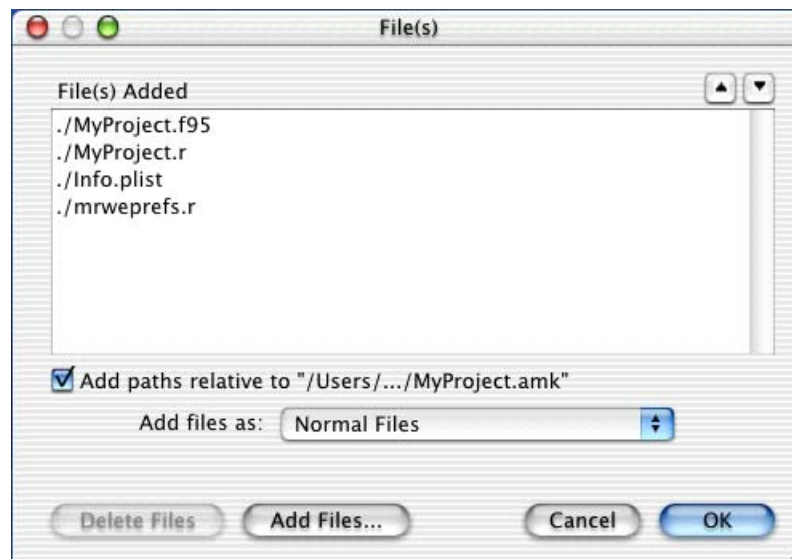
The primary menu for managing projects is the Configure Menu. This menu is used to specify the files that comprise the project, paths to search for include files, and the tool options used to build the application or library.



Add/Remove File(s)

Use this command to open the file selection dialog. Select the project's source files, external object files, and libraries here by clicking either the **Add Files...** button to add additional files or the **Delete Files** button to remove the selected files. Note that the type of file displayed can be limited with the **Add files as** drop-down menu. If the source files you are adding will be processed with the VAST preprocessor (see **Plug-ins** later in this chapter), they must be selected and added to the project with the appropriate **Add files as** setting.

To add multiple files at once you can either shift-click an extended selection range of files or command-click to select a non-contiguous range of files from the standard file dialog. To add files that may otherwise be inaccessible, hold down the Shift or Option key while selecting the **Add/Remove File(s)** menu item.



The up and down pointing arrows above the **File(s) Added** list box are used to change the compilation order of the source files and to set the order in which external object files and library files are presented to the linker.

You can also add include and header files to the project. The paths to these files will automatically be supplied to the appropriate compiler if the **Auto-add include file paths** box is checked in Preferences.

The **Add file paths relative to “...”** check box can be used to toggle relative file paths on or off as desired. The default is to use file paths relative to the current project file. The current state of the check box will be used for all newly added files or paths.

Set Include Paths

Use this command to select additional directory paths to be searched for include and header files, in the same manner that source files are added to the project. This is the **-I** option on the command line and is used to supply a comma separated list of directory paths which are prepended to file names used with the Fortran `INCLUDE` statement or the C/C++ `#include` directive.

```
-Ipath[,path...]
```

The paths are prepended in the order presented with the **-I** option when the include file is not first found in the local directory and when it is not itself an absolute path (a full file specification). Paths supplied with the **-I** option are searched before the path specified with the `ABSOFTH` shell variable.

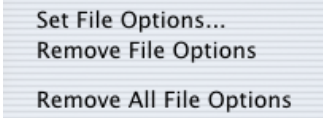
The **Add file paths relative to “...”** check box can be used to toggle relative file paths on or off as desired. The default is to use file paths relative to the current project file. The current state of the check box will be used for all newly added files or paths.

Set Project Options

This command opens the **Options** dialog introduced earlier. The remaining tabs available in this dialog and the options they control will be discussed in detail in the following sections.

File Options

The **File Options** command allows you to set individual options for a specific project source file and tool. The menu command becomes active after you first select a source file in the **Files** tab. Option choices made will affect only the file name displayed in the options dialog title bar. Options set using this command will override any options set by the **Set Project Options** command.



```
Set File Options...
Remove File Options
Remove All File Options
```

Remove File Options

The **Remove File Options** command allows you to remove individual options for a specific file. The menu command becomes active after you first select a source file in the **Files** tab.

Remove All File Options

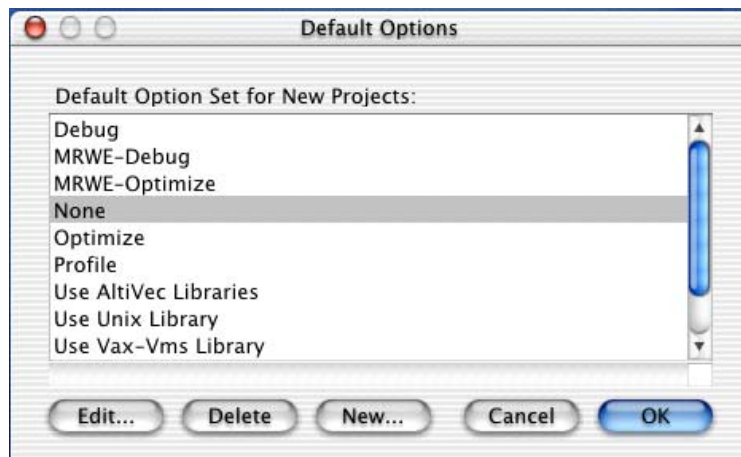
The **Remove All File Options** command allows you to remove all individual source file options for the entire project file.

MRWE Preferences...

If you have created an MRWE application (see **Working with Projects** earlier in this chapter), this menu selection will display the **MRWE Application Preferences** dialog. With this dialog you can modify how your application handles text and text saving. You can also modify the text font and size as well as whether your application pauses or exits when it has completed.

Set Default Options

Use this command to control the default options that are set when a new project is created. Several default settings were established when Absoft Pro Fortran was installed, including: **None**, **Debug**, and **Optimize**. Other common configurations are also supplied.



Set Default Options

Click on the settings you want to use when a new project is created. You can edit the individual settings by clicking on the **Edit** button. The Options dialog (described earlier in this chapter) will appear giving you complete control over all of the general, compiler, linker options. You can also add your own default settings by clicking on the **New** button. Delete an option set with the **Delete** button.

Tools Menu

This menu provides access to the various developer tools you will work with: compilers, editors, debuggers, profilers, preprocessors, and makefiles.

Search

Use this command to search for strings of text in files.

Build

This is the primary command for building and updating your project. When you have finished adding the files to your project with the **Add/Remove File(s)** command in the **Configure** menu and have set all of the options, use this command to compile and link your program or library. This command is also used when you change options or edit files. It will recompile only those files that have been changed.

Rebuild All

Use this command to completely rebuild your project. All of the files will be processed regardless of whether they have been modified since the last build or not.

Update Dependencies

This command is used to force the Absoft Developer Tools Interface to rescan all source and include files for build dependencies.

Check Syntax

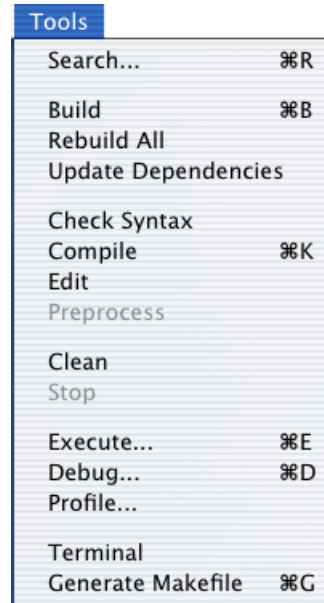
The **Check Syntax** command will check syntax only for the file currently selected in the Files tab of the project window.

Compile

The **Compile** command will compile the file currently selected in the Files tab of the project window.

Edit

Use this command to open for editing the file currently selected in the Files tab of the project window.



Preprocess

The **Preprocess** command is available only if you have the VAST parallel or VAST Altivec preprocessor installed. It executes the installed multi-processor preprocessor and leaves the intermediate file on your disk so that you can examine it.

Clean

Use this command to delete the executable and all of the object files in your project. It will also delete any `.rsrc` files if your project incorporates resource description files (`.r`). For a complete list of what files will be deleted see the appropriate section of the associated makefile in the **Makefile** tab of the Output pane.

Stop

The Stop command terminates the currently executing tool.

Execute

Use this command to execute or run your program after it has finished building. Standard input can be redirected from a file if necessary in the Execute dialog. Standard Output for the executable will be shown in the Execute tab of the output pane by default.

Debug

Use this command to debug your program with the Absoft Fx debugger. See the chapter **Using the Fx Debugger** for more information.

Profile

The **Profile** command will run your program and produce execution time statistics. Note that you must first have produced a special instrumented executable by selecting the **Enable Profiling** option on the **Target** tab of the **Options** dialog.

Terminal

This command opens a Terminal and sets the current working directory to that of your project.

Generate Makefile

Use this command to create a makefile that can be used with the Absoft make utility, `amake`. See the chapter **Building Programs** for information on makefile commands and using `amake`.

Window Menu

The commands in this menu allow you to arrange the open windows in the Absoft Developer Tools Interface, manage the tool bar, and to bring a specific window to the front.



Hide Toolbar

Use this command to show or hide the toolbar.

Customize Toolbar...

This command displays the standard OS X toolbar dialog for customizing toolbars.

Tile Horizontally

Select this command to tile all open Absoft Developer Tools Interface applications windows in a horizontal fashion.

Tile Vertically

Select this command to tile all open Absoft Developer Tools Interface applications windows in a vertical fashion.

Project

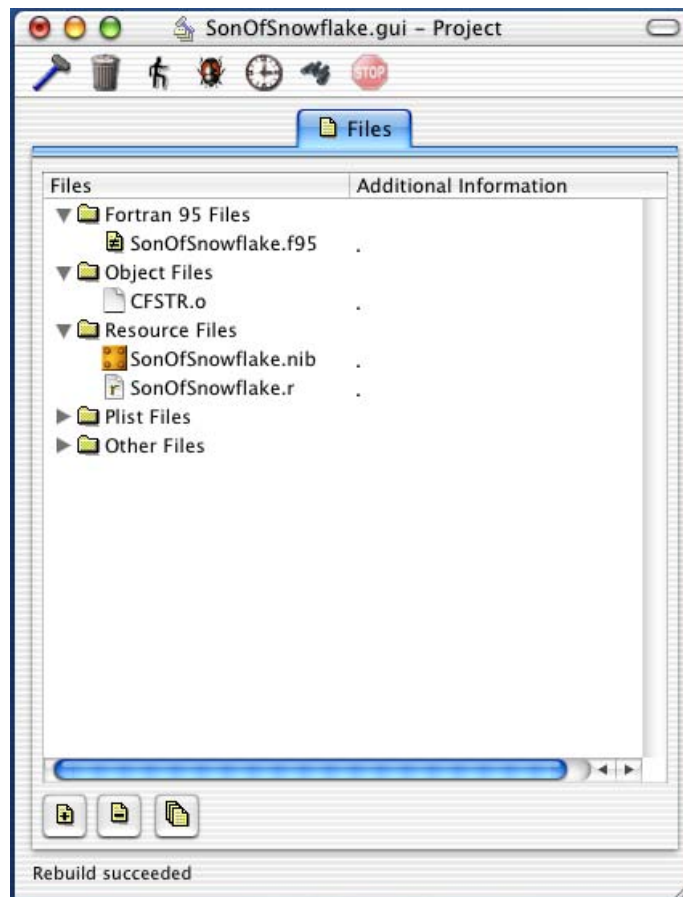
Use this command to activate or re-open the Project window for the current project file. This menu item is only enabled if the **Single Window Mode** option has been turned off in the Preferences dialog.

The toolbar is displayed across the top of the **Project** window, or above the Project pane, if the **Single Window Mode** option is turned on. The toolbar provides quick mouse access to many tools used in the Absoft Developer Tools Interface.



- Build** Build the program using the selected options.
- Clean** Delete all current object files, intermediate files, and the executable file for the active program.
- Run** Execute the program.
- Debug** Debug the program.
- Profile** Profile the program

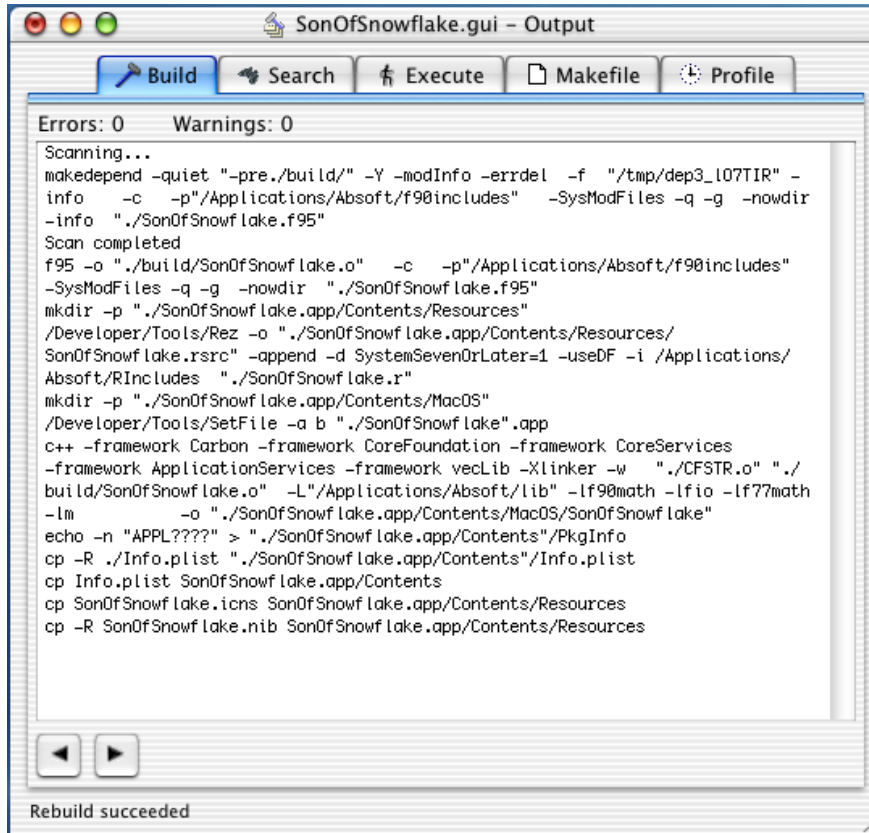
- Search** Search for text or expression.
- Stop** Halt execution of current developer tool.
- Plus** Add new source file to project.
- Minus** Remove selected file from project.
- Files** Add, remove, or re-order file(s) in this project



Output

Use this command to activate or re-open the Output window for the current project file. This menu item is only enabled if the **Single Window Mode** option has been turned off in the Preferences dialog.

The Output window, or pane, contains several tabs. **Build** shows the latest build or compile results for the current project file. **Search** shows the latest search results for the current project. **Execute** displays standard output when the program is launched from the Absoft Developer Tools Interface application. **Makefile** show the associated makefile for the current project document. **Profile** displays data files produced by the profiler.

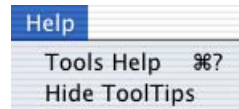


Left Display previous item in source file editor.

Right Display next item in source file editor.

Help Menu

Tools Help



Use this command to view Absoft Developer Tools Interface application online help files.

Hide ToolTips

Use this command to disable pop-up tooltips for all windows and controls in the Absoft Developer Tools Interface for the remainder of the session.

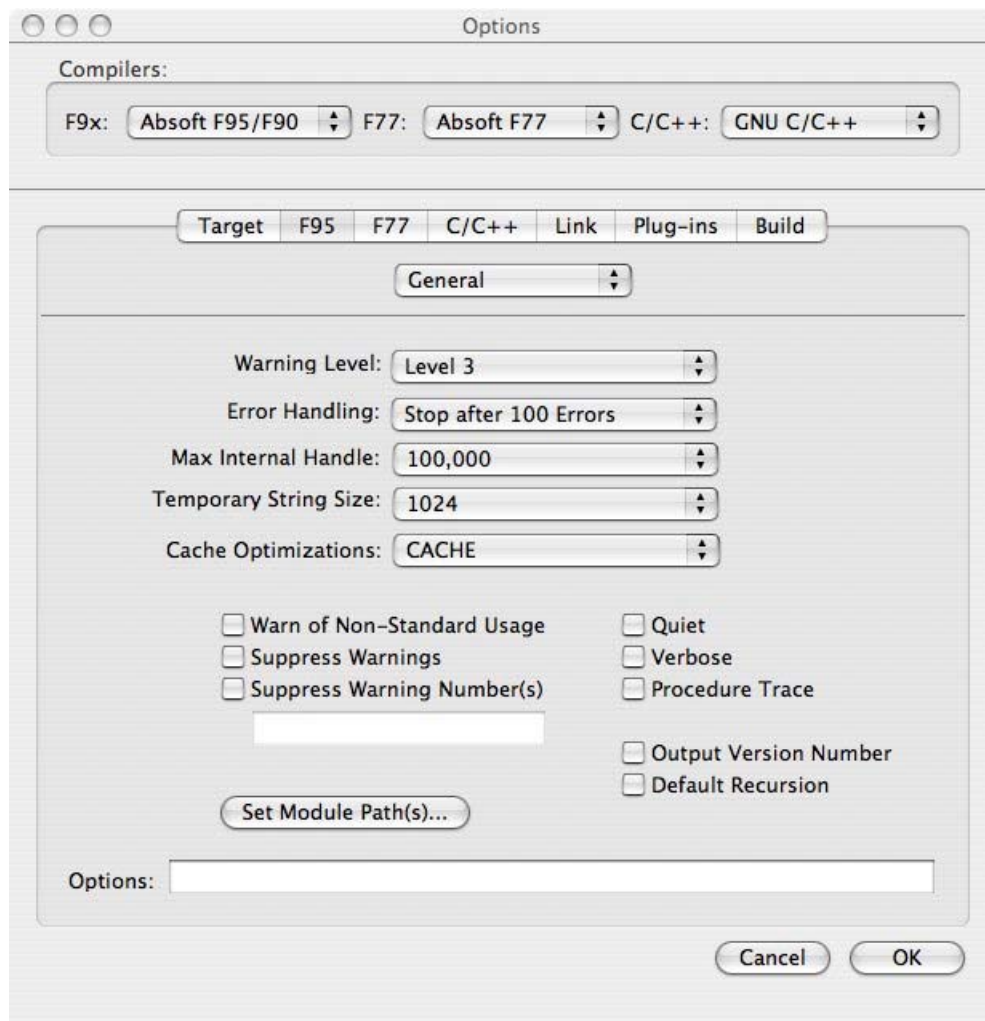
ABSOFORT FORTRAN 95 OPTIONS

The compiler options detailed in this section give you a great deal of control over the compilation and execution of Fortran 90/95 programs. Select the **Set Project Options** command in the **Configure** menu to access the Options dialog. The Fortran 90/95 options fall into four categories: General, Compatibility, Format, and Common Block.

For quick reference, the options listed in the sections that follow are in the order in which they appear in the option tabs. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked in the Absoft Developer Tools Interface application, the same letters will appear in the in the corresponding **Options** box.

General - F95 Options

When this subset of the **F95** options tab is selected, options for controlling various aspects of compiling the Fortran 90/95 programs are available. Click on the box next to the option to add the option for compiling.



Warning level (-znn)

Use the **-znn** option to suppress messages by message level, where **nn** is a message level. Diagnostics issued at the various levels are:

0	errors, warnings, cautions, notes, comments
1	errors, warnings, cautions, notes
2	errors, warnings, cautions
3	errors, warnings
4	errors

The default level is **-z3**; the compiler will issue error and warning diagnostics, but not cautions, notes, and comments. See also the **-znn** option.

Error Handling (-dq and -ea)

Normally, the Absoft Fortran 90/95 compiler will stop if more than 100 errors are encountered. This many errors usually indicate a problem with the source file itself or the inability to locate an `INCLUDE` file. If you want the compiler to continue in this circumstance, select the **Allow > 100** or **-dq** option. The **Stop on Error** or **-ea** option will cause the f95 compiler to abort the compilation process on the first error that it encounters.

Max Internal Handle (-T nn)

This option is used to change the number of handles used internally by the compiler. Under most conditions, the default value of 100000 handles is sufficient to compile even extremely large programs. However, under certain circumstances, this value may be exceeded and the compiler will issue a diagnostic indicating that the value should be increased.

The default value can be increased by powers of ten by specifying the **-T nn**, where **nn** is a positive integer constant. When this option is specified, the number of handles will be 100000×10^{nn} bytes.

Temporary string size (-t nn)

In certain cases the compiler is unable to determine the amount of temporary string space that string operations will require. The compiler will assume that the operation in question will require 1024 bytes of temporary string space. This default value can be increased by powers of ten by specifying the **-t nn**, where **nn** is a positive integer constant. When this option is specified, the default temporary string size will be 1024×10^{nn} bytes.

Cache Control (-YDEALLOC= {MINE | ALL | CACHE})

This option is used to control the underlying runtime memory management associated with the Fortran 95 `ALLOCATE` and `DEALLOCATE` statements. By default the runtime caches memory that has been deallocated (**CACHE**). Specifying **MINE** will cause all user allocated memory to be returned via a call to `free(2)` when a call to `DEALLOCATE` is executed. Specifying **ALL** will cause all user allocated memory to be returned via a call to `free(2)` and return any compiler allocated memory that has been cached. The tradeoff is minimizing memory use (**ALL/MIME**) versus speed of execution (**CACHE**).

Warn of Non-Standard usage (-en)

Use of the `-en` option will cause the compiler to issue a warning whenever the source code contains an extension to the Fortran 90/95 standard. This option is useful for developing code which must be portable to other environments.

Suppress warnings (-w)

Suppresses the listing of warning messages. For example, unreachable code will generate a warning message.

Suppress Warning number(s) (-Znn)

Use the `-Znn` option to suppress messages by message number, where `nn` is a message number. This option is useful if the source code generates a large number of messages with the same message number, but you still want to see other messages. See also the `-znn` option.

Use System Module Files (-SysModFiles)

Checking this box automatically adds the `f90includes` include directory to the search path for modules.

Set Module Paths (-p path)

The Absoft Fortran 90/95 compiler will automatically search the local directory and `$(ABSOFT)/f90includes` for precompiled module files. Use this command to open the file selection dialog to add additional search paths. Paths specified are searched in a position dependent order. If module files are maintained in other directories, use the `-p` option to specify a path or complete file specification. See **Fortran 90/95 Module Files** in the chapter, **Building Programs** for more information.

Note: there must be a space between the option (`-p`) and the path.

Quiet (-q)

The Absoft Fortran 90/95 compiler normally displays information to standard output (the command line window) as it compiles an application. Enabling the **-q** option will suppress any messages printed to standard output. Errors will still be printed, however.

Verbose (-v)

Enabling the **-v** option will cause the **f95** command, described later in the **Building Programs** chapter, to display the commands it is sending to the compiler and linker.

Procedure Trace (-B80)

Specifying the **-B80** option will cause the compiler to generate code to write the name of the currently executing procedure to standard out. This option is useful for tracing program execution and quickly isolating execution problems.

Output Version number (-V)

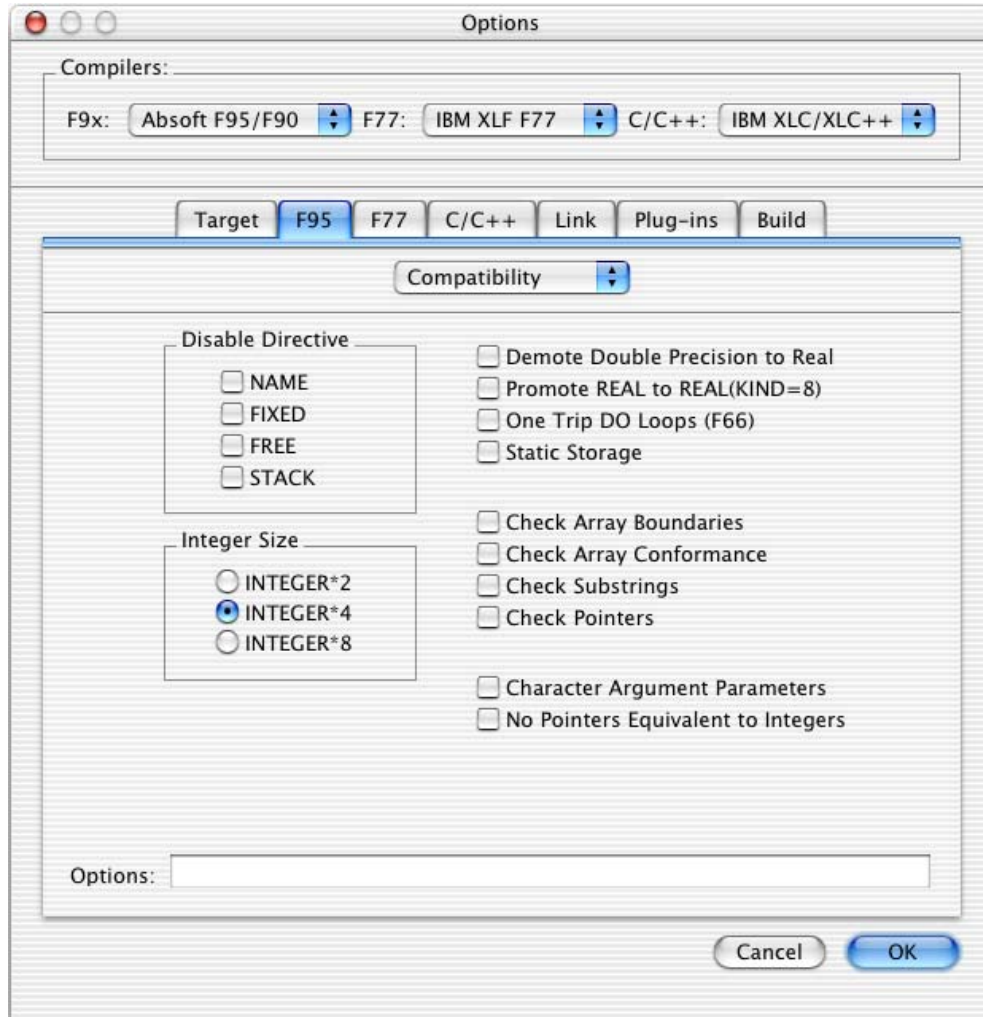
The **-V** option will cause the `f95` compiler to display its version number. This option may be used with or without other arguments.

Default Recursion (-eR)

If you select the **-eR** option, all `FUNCTIONS` and `SUBROUTINES` are given the `RECURSIVE` attribute. Normally, if the compiler detects a recursive invocation of a procedure not explicitly given the `RECURSIVE` attribute, a diagnostic message will be issued. The **-eR** option disables this.

Compatibility - F95 Options

When this subset of the **F95** options tab is selected, options for enhancing the compatibility of Fortran 90/95 programs with other programming languages are available. Click on the box next to the option to add the option for compiling.



Disable compiler directive (*-xdirective*)

The **-x** option is used to disable compiler directive in the source file. **directive** may be any of the following:

NAME
FIXED
FREE
STACK

See the section **Absoft Fortran 90/95 Compiler Directives**, later in this chapter, for more information on using compiler directives in your source code.

INTEGER and LOGICAL sizes (-in)

Without an explicit length declaration, `INTEGER` data types default to thirty-two bits or four bytes (`KIND=4`). The `-i2` option can be used to change this default length to sixteen bits or two bytes (`KIND=2`). The `-i8` option can be used to change the default `INTEGER` size to 64 bits or 8 bytes (`KIND=8`). However, an explicit length specification in a type declaration statement always overrides the default data length.

Character Argument Parameters (-YCFRL={0|1})

Use the `-YCFRL=1` option to force the compiler to pass `CHARACTER` arguments in a manner that is compatible with `g77` and `f2c` protocols. Use the `-YCFRL=0` option (the default) to pass `CHARACTER` arguments in a manner that is compatible with Absoft Compilers on other platforms. **Note:** this option should be used consistently on all files that will be linked together into the final application.

Demote Double Precision to Real (-dp)

The `-dp` option will cause variables declared in a `DOUBLE PRECISION` statement and constants specified with the `D` exponent to be converted to the default real kind. Similarly, variables declared in a `DOUBLE COMPLEX` statement and complex constants specified with `D` exponents will be converted to the complex kind in which each part has the default real kind.

Promote REAL and COMPLEX (-N113)

Without an explicit length declaration, single precision `REAL` and `COMPLEX` data types default to thirty-two bits or four bytes (`KIND=4`) and sixty-four bits or eight bytes (`KIND=4`), respectively. The `-N113` option is used to promote these to their double precision equivalents (`KIND=8`). This option does not affect variables which appear in type statements with explicit sizes (such as `REAL (KIND=4)` or `COMPLEX (KIND=4)`).

One trip DO loops (-ej)

Fortran 90/95 requires that a `DO` loop not be executed if the iteration count, as established from the `DO` parameter list, is zero. The `-ej` option will cause all `DO` loops to be executed at least once, regardless of the initial value of the iteration count.

Static storage (-s)

The `-s` option is used to allocate local variables statically, even if `SAVE` was not specified as an attribute. In this way, they will retain their definition status on repeated references to the procedure that declared them. Two types of variables are not allocated to static storage: variables allocated in an `ALLOCATE` statement and local variables in recursive procedures.

Check Array Boundaries (-Rb)

When the **-Rb** compiler option is turned on, code will be generated to check that array indexes are within the bounds of an array. Assumed size arrays whose last dimension is * cannot be checked. In addition, file names and source code line numbers will be displayed with all run time error messages.

Check Array Conformance (-Rc)

The **-Rc** compiler option is used to check array conformance. When array shapes are not known at compile time and where they must conform, runtime checks are created to insure that two arrays have the same shape.

Check Substrings (-Rs)

When the **-Rs** compiler option is turned on, code will be generated to check that character substring expressions do not specify a character index outside of the scope of the character variable or character array element.

Check Pointers (-Rp)

Use **-Rp** compiler option is used to generate additional program code to insure that Fortran 90 style `POINTER` references are not null.

Pointers Equivalent to Integers (-YPEI={0|1})

This option controls whether or not the compiler will allow or accept a CRI style pointer to be equivalent to an integer argument. By default the Absoft Fortran 90/95 compiler allows this. Even with this relaxed error checking the compiler will correctly choose the right interface for the following example:

```
interface generic
  subroutine specific1(i)
    integer i
  end subroutine specific1
  subroutine specific2(p)
    integer i
    pointer (p,i)
  end subroutine specific2
end interface
call generic(i)
call generic(loc(i))
end
```

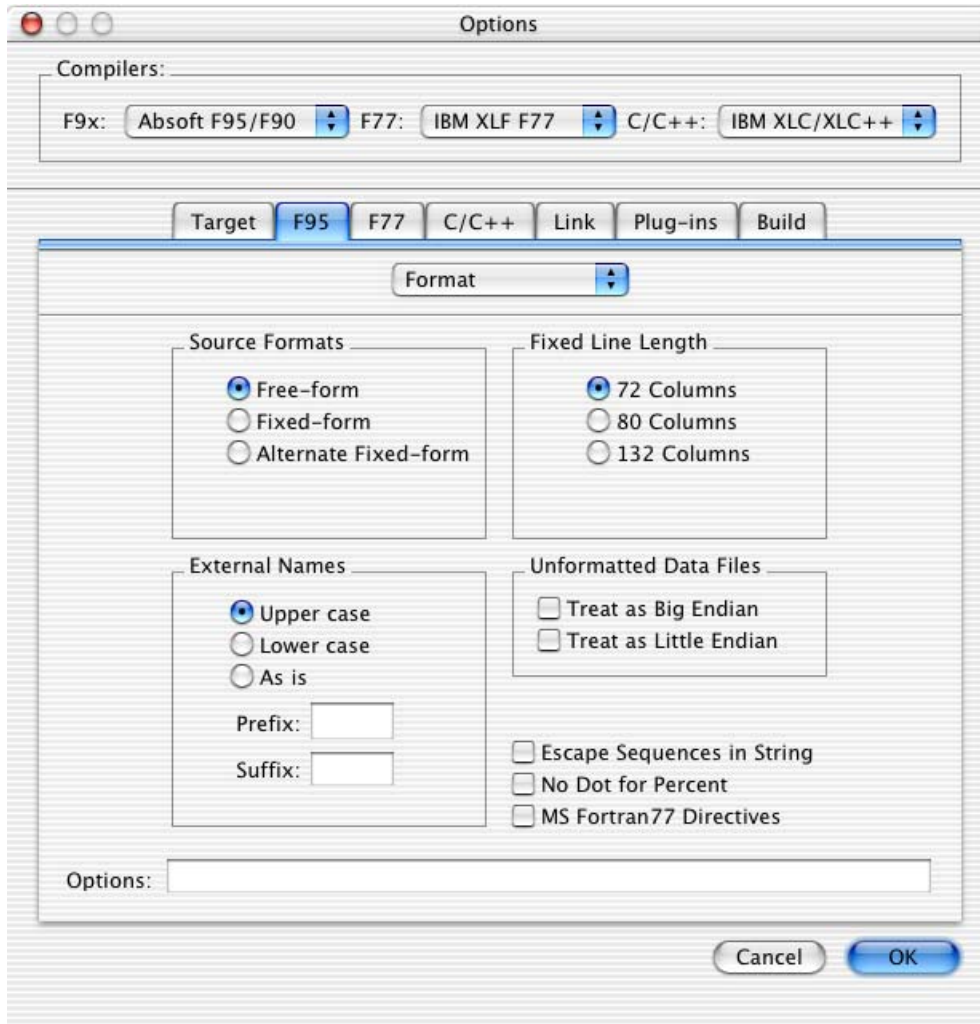
Regardless of the switch setting, this example will compile and the executable generated will be equivalent to:

```
call specific1(i)
call specific2(loc(i))
```

Format - F95 Options

For compatibility with other Fortran environments and to provide more flexibility, the compiler can be directed to accept source code that has been written in a number of different formats. The two basic formats are free-form and fixed-form.

This subset of the **F95** options tab displays options for controlling how Fortran 90/95 interprets the format of source files. These options allow Absoft Fortran 90/95 to accept older or variant extensions of Fortran source code from other computers such as mainframes.



F95 Format Options

Free-Form (-ffree)

The **-f free** option instructs the compiler to accept source code written in the format for the Fortran 90/95 Free Source Form. This is the default for file names with an extension of “.f95”.

Fixed-Form (-ffixed)

The **-ffixed** option instructs the compiler to accept source code written in the format for the Fortran 90/95 Fixed Source Form which is the same as the standard FORTRAN 77 source form.

Alternate Fixed form (-falt_fixed)

The **-falt_fixed** option instructs the compiler to accept source code written in following form:

If a tab appears in columns 1 through 5, then the compiler examines the next character. If the next character is not a letter (a-z, or A-Z) then it is considered a continuation character and normal rules apply. If it is a zero, a blank, another tab, or a letter, the line is not a continuation line.

Fixed line length (-W nn)

Use the **-W** option to set the line length of source statements accepted by the compiler in Fixed-Form source format. The default value of *nn* is 72. The other legal values for *nn* are 80 and 132 — any other value produces an error diagnostic.

YEXT_NAMES={ASIS | UCS | LCS}

The **-YEXT_NAMES** option is used to specify how the external names of globally visible symbols, such as `FUNCTION` and `SUBROUTINE` names, are emitted. By default, names are emitted entirely in upper case. Set this option to **LCS** to emit names entirely in lower case. Set this option to **ASIS** to force external names to emitted exactly as they appear in the source program. This option controls how external names will appear to other object files.

External Symbol Prefix (-YEXT_PFX=string)

The **-YEXT_PFX** option can be used to prepend a user specified *string* to the external representation of external procedure names.

External Symbol Suffix (-YEXT_SFX=string)

The **-YEXT_SFX** option can be used to append a user specified *string* to the external representation of external procedure names.

Treat as Big-Endian (-N26)

Use this option to force the compiler to consider the byte ordering of all unformatted files to be big-endian by default. The `CONVERT` specifier in the `OPEN` statement may be used to override this setting for individual files.

Treat as Little-Endian (-N27)

Use this option to force the compiler to consider the byte ordering of all unformatted files to be little-endian by default. The `CONVERT` specifier in the `OPEN` statement may be used to override this setting for individual files.

Escape Sequences in Strings (-YCSLASH=1)

If the `-YCSLASH=1` option is turned on, the compiler will transform the following escape sequences marked with a `'\'` embedded in character constants:

<code>\a</code>	Audible Alarm (BEL, ASCII 07)
<code>\b</code>	Backspace (BS, ASCII 8)
<code>\f</code>	Form Feed (FF, ASCII 12)
<code>\n</code>	Newline (LF, ASCII 10)
<code>\r</code>	Carriage Return (CR, ASCII 13)
<code>\t</code>	Horizontal Tab (HT, ASCII 09)
<code>\v</code>	Vertical Tab (VT, ASCII 11)
<code>\x[h]</code>	Hexidecimal, up to 2 digits
<code>\o[o[o]]</code>	Octal number, up to 3 digits
<code>\\</code>	Backslash

The default is `-YCSLASH=0`.

No Dot for Percent (-YNDFP=1)

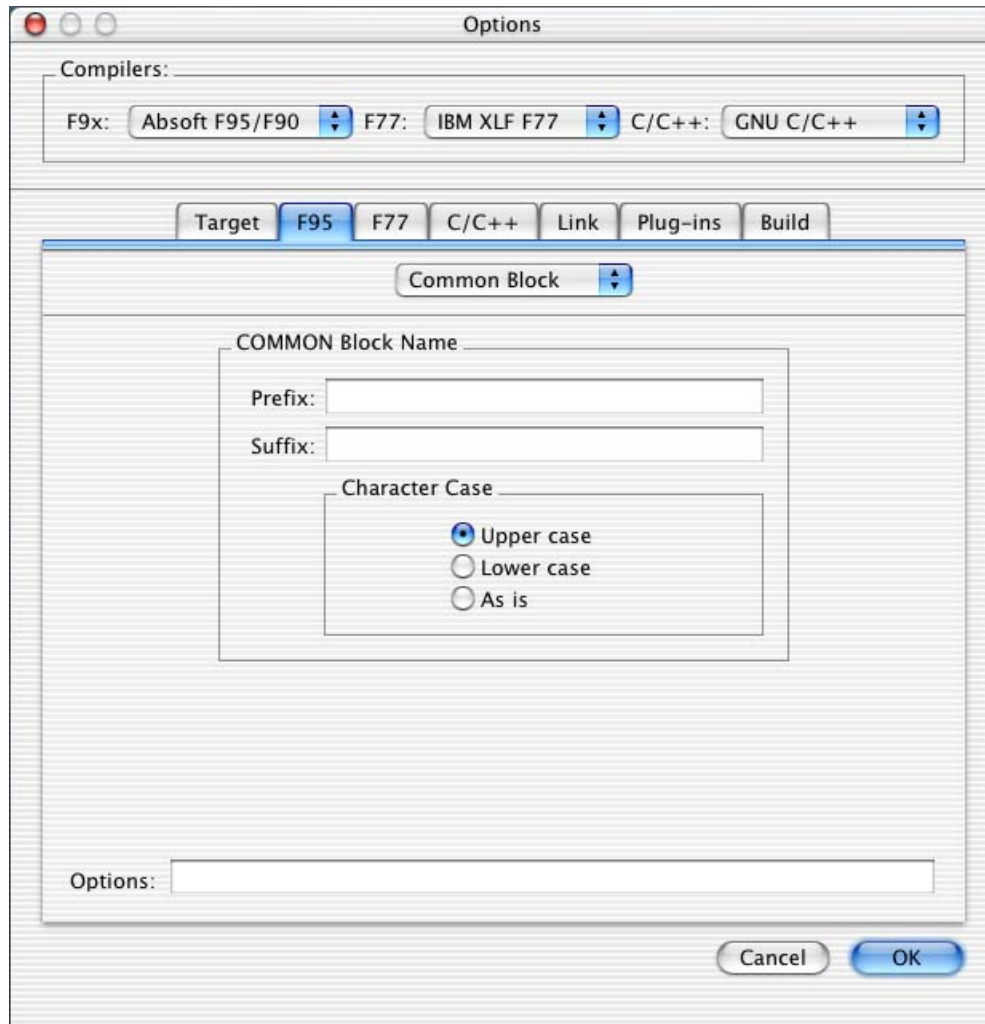
This option instructs the compiler to disallow the use of a `'.'` (period) as a structure field component dereference operator. The default is to allow both `'%'` (percent), which is the Fortran 90/95 standard, and a period which is typically used with DEC style `RECORD` declarations. The use of a period may cause certain Fortran 90/95 conforming programs to be mis-interpreted (a period is used to delineate user defined operators and some intrinsic operators). The default is `-YNDFP=0`. This switch implements Fortran 90/95 standard parsing for structure component referencing.

MS Fortran 77 Directives (-YMS7D)

The `-YMS7D` option causes the compiler to recognize Microsoft Fortran 77 style directives in the form of `$directive` where the dollar-sign character is in column one of the source file. `directive` must be from the set of supported MS directives.

Common Block - F95 Options

This subset of the **F95** options tab displays options for controlling how Fortran 90/95 treats common block names.



F95 Common Block Options

COMMON Block Name Prefix (-YCOM_PFX=*string*)

The **-YEXT_PFX** option can be used to prepend a user specified **string** to the external representation of `COMMON` block names.

COMMON Block Name Suffix (-YCOM_SFX=*string*)

The **-YCOM_SFX** option can be used to append a user specified **string** to the external representation of `COMMON` block names.

COMMON Block Name Character Case (-YCOM_NAMES={UCS | LCS})

The **-YCOM_NAMES** option is used to specify how the external names `COMMON` blocks are emitted. The default (**-YCOM_NAMES=UCS**) is to emit `COMMON` block names entirely in upper case. Set this option to **LCS** to emit names entirely in lower case.

Other F95 Options

The following options are not available with the graphical interface to the compiler but may be entered manually in the option box, used with the command line interface, or with the make facility (See the chapter, **Building Programs**).

Stack Size (-stack_size:size)

Use this option to establish the amount of memory reserved for stack use only. **size** is entered as a hexadecimal value (e.g. 8000000). This option sets the stack size in the application and is not subject to the ulimit maximum of 64 MB.

Disable Position Independent Code (-no-fpic)

Use this option to turn off position independent code generation.

Speculative Execution (-B156)

The **-B156** option enables speculative execution. It removes loop invariant expressions from within conditionals in loops. If the expression would normally be executed, this option may improve performance. If the expression would not normally be executed, this option may impair performance. This option is automatically enabled with the **-O2** optimization option. It may be disabled with the **+B156** option.

Inline CABS (-B157)

The **-B157** option turns on inlining of the `CABS` and `CDABS` functions. This may overflow for extremely large values; the library routine will not overflow. This option is automatically enabled with the **-O2** optimization option. It may be disabled with the **+B157** option.

Address Optimizations (-B158)

The **-B158** option enables optimization of address calculations in loops. This will tend to reduce register pressure within the loop at the cost of more instructions. This option is automatically enabled with the **-O2** optimization option. It may be disabled with the **+B158** option.

Safe Floating-Point (-safefp)

The **-safefp** option is used to disable optimizations that may produce inaccurate or invalid floating point results in numerically sensitive codes. The effect of this option is to preserve the FPU control word, enable NAN checks, disable `CABS` inlining, and disable floating-point register variables.

Disable Default Module File Path (-nodefaultmod)

The Absoft Fortran 90/95 compiler will automatically search the directory `$(ABSOFT)/f90includes` for precompiled module files. Use the **-nodefaultmod** to disable this.

Variable Names Case Sensitivity (-YVAR_NAMES={ASIS | UCS | LCS})

The **-YVAR_NAMES** option is used to specify how the case of variable names is treated. By default, variable names are processed entirely in upper case (**UCS**), regardless of the how they appear in the source code. Set this option to **LCS** to fold variable names to lower case. Set this option to **ASIS** to force variable names to be processed exactly as they appear in the source program.

Symbol Names Case Sensitivity (-YALL_NAMES={ASIS | UCS | LCS})

The **-YALL_NAMES** option is used to specify how the case of all symbolic names is treated. By default, symbolic names are processed entirely in upper case (**UCS**), regardless of the how they appear in the source code. Set this option to **LCS** to fold all symbolic names to lower case. Set this option to **ASIS** to force symbolic names to be processed exactly as they appear in the source program. This option is the same as using the **-YVAR_NAMES**, **-YCOM_NAMES**, and **-YEXT_NAMES** options, which may appear after the **-YALL_NAMES** option to control an individual symbolic name type.

Ignore CDEC\$ directives (-YNO_CDEC)

The compiler recognizes `CDEC$` directives that contain conditional compilation directives. Use this option to disable them.

Absoft Fortran 90/95 Compiler Directives

Compiler directives are lines inserted into source code that specify actions to be performed by the compiler. They are not Fortran 90/95 statements. If you specify a compiler directive while running on a system that does not support that particular directive, the compiler ignores the directive and continues with compilation.

A *compiler directive line* begins with the characters `CDIR$` or `!DIR$`. How you specify compiler directives depends on the source form you are using.

If you are using fixed source form, indicate a compiler directive line by placing the characters `CDIR$` or `!DIR$` in columns 1 through 5. If the compiler encounters a nonblank character in column 6, the line is assumed to be a compiler directive continuation line. Columns 7 and beyond can contain one or more compiler directives. If you are using the default 72 column width, characters beyond column 72 are ignored. If you have specified 80 column lines, characters beyond column 80 are ignored.

If you are using free source form, indicate a compiler directive line by placing the characters `!DIR$` followed by a space, and then one or more compiler directives. If the position following the `!DIR$` contains a character other than a blank, tab, or newline character, the line is assumed to be a compiler directive continuation line.

If you want to specify more than one compiler directive on a line, separate each directive with a comma.

NAME Directive

The `NAME` directive allows you to specify a case-sensitive external name in a Fortran program. You can use this directive, for example, when writing calls to C routines. The case-sensitive external name is specified on the `NAME` directive, in the following format:

```
!DIR$ NAME (fortran="external" [, fortran="external"]...)
```

where: *fortran* is the name used for the object throughout the Fortran program whenever the external name is referenced.

external is the external name.

FREE[FORM] Directive

The `FREE` or `FREEFORM` directive specifies that the source code in the program unit is written in the free source form. The `FREE` directive may appear anywhere within your source code. The format of the `FREE` directive is:

```
!DIR$ FREE
```

You can change source form within an `INCLUDE` file. After the `INCLUDE` file has been processed, the source form reverts back to the source form that was being used prior to processing the `INCLUDE` file.

FIXED Directive

The `FIXED` directive specifies that the source code in the program unit is written in the fixed source form. The `FIXED` directive may appear anywhere within your source code. The format of the `FIXED` directive is:

```
!DIR$ FIXED
```

You can change source form within an `INCLUDE` file. After the `INCLUDE` file has been processed, the source form reverts back to the source form that was being used prior to processing the `INCLUDE` file.

NOFREEFORM Directive

The `NOFREEFORM` directive is the same as the `FIXED` directive (see above) and specifies that the source code in the program unit is written in the fixed source form.

FIXEDFORMLINESIZE Directive

The `FIXEDFORMLINESIZE` directive specifies the line length for fixed-form source code. The format of the `FIXEDFORMLINESIZE` directive is:

```
!DIR$ FIXEDFORMLINESIZE:{72|80|132}
```

ATTRIBUTES Directive

The `ATTRIBUTES` directive can be used to apply special attributes to simplify passing variables between Fortran 90/95 and other languages. The format of the `ATTRIBUTES` directive is:

```
!DIR$ ATTRIBUTES attr-list::sym-list
```

where: *attr-list* is a comma separated list of attributes from the following set.

```
ALIAS  
C  
REFERENCE  
STDCALL  
VALUE
```

sym-list is a comma separated list of symbols.

The `ALIAS` attribute takes the form of

```
ALIAS:external
```

where: *external* is the is the external name of the procedure.

PACK[ON] Directive

The `PACK` or `PACKON` directive specifies that sequenced structure fields be aligned on byte even byte or word (four-byte) boundaries. The default is 1 (byte). The format for this compiler directive is:

```
!DIR$ PACK [= {1|2|4}]
```

The packing directives affect the current program unit being compiled (if there is one), or the next program unit (when there is no current program unit). The packing directive is reset to the default (`PACKOFF`) after the end of each program unit. A packing directive affects only derived-types found below the directive in the source code.

PACKOFF Directive

The `PACKOFF` directive returns structure field alignment to the default for the machine architecture which is alignment on the most efficient boundary for the data type. The format for this compiler directive is:

```
!DIR$ PACKOFF
```

STACK Directive

The `STACK` directive causes the default storage allocation to be the stack in the program unit that contains the directive. This directive overrides the `-s` command line option in specific program units of a compilation unit. The format for this compiler directive is:

```
!DIR$ STACK
```

UNROLL Directive

The `UNROLL` directive is used to control loop unrolling by the compiler. Loop unrolling is automatically enabled with the `-O3` option. Use this directive to control loop unrolling independent of the `-O3` option. The format for this compiler directive is:

```
!DIR$ UNROLL N
```

where *N* is the count of the number of times to unroll the loop. If *N* is 0, the count is automatic. If *N* is 1, loop unrolling is disabled.

NOUNROLL Directive

The `NOUNROLL` directive is used to disable loop unrolling by the compiler. Loop unrolling is automatically enabled with the `-O3` option. Use this directive to disable loop unrolling in all circumstances. The format for this compiler directive is:

```
!DIR$ NOUNROLL
```

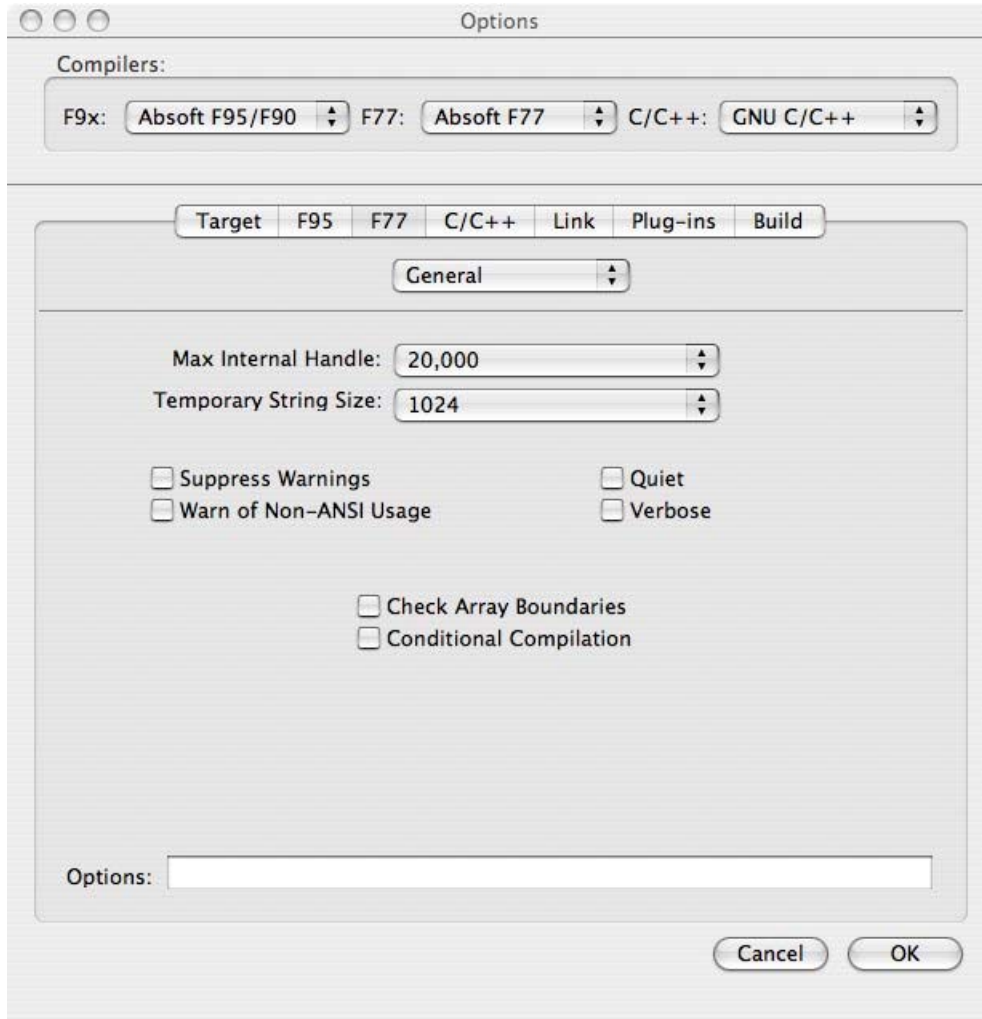

ABSOFORT FORTRAN 77 OPTIONS

The compiler options detailed in this section give you a great deal of control over the compilation and execution of FORTRAN 77 programs. Select the **Set Project Options** command in the **Configure** menu to access the Options dialog. The options for FORTRAN 77 fall into several categories: General, Control, Compatibility, Miscellaneous, Format, and Common Block.

For quick reference, the options listed in the sections that follow are in the order in which they appear on the F77 tab of the options dialog. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked in the Absoft Developer Tools Interface application, the same letters will appear in the in the corresponding **Options** box.

General - F77 Options

These options control the general characteristics of the FORTRAN 77 components of the program being built.



F77 General Options

Max Internal Handle (-T nn)

This option is used to change the number of handles used internally by the compiler. Under most conditions, the default value of 20000 handles is sufficient to compile even extremely large programs. However, under certain circumstances, this value may be exceeded and the compiler will issue a diagnostic indicating that the value should be increased.

Temporary string size (-t nn)

In certain cases the compiler is unable to determine the amount of temporary string space that string operations will require. This undetermined length occurs when the `REPEAT`

function is used or when a `CHARACTER*(*)` variable is declared in a subroutine or function. In these cases, the compiler will assume that the operation in question will require 1024 bytes of temporary string space. This default value can be changed by specifying the `-t nn`, where `nn` is a positive integer constant. When this option is specified, the default temporary string size will be `nn` bytes.

Suppress Warnings (-w)

Suppresses the listing of warning messages. For example, unreachable code or a missing label on a `FORMAT` statement generate warning messages. Compile time diagnostic messages are divided into two categories: errors and warnings. Error messages indicate that the compiler was unable to generate an output file. Warning messages indicate that some syntactic element was not appropriate, but the compiler was able to produce an output file.

Warn of non-ANSI Usage (-N32)

Use of the `-N32` option will cause the compiler to issue a warning whenever the source code contains an extension to the ANSI FORTRAN 77 standard (American National Standard Programming Language FORTRAN, X3.9-1978). This option is useful for developing code which must be portable to other environments.

Quiet (-q)

The Absoft Fortran 77 compiler normally displays information to standard output (the command line window) as it compiles an application. Enabling the `-q` option will suppress any messages printed to standard output. Errors will still be printed, however.

Verbose (-v)

Enabling the `-v` option will display the individual commands that are sent to the command line window, such as the front and back ends of the compiler and the linker.

Check Array Boundaries (-C)

When the `-C` compiler option is turned on, code will be generated to check that array indexes are within the bounds of an array. Exceptions: arrays whose last dimension is `*` and dummy arguments whose last dimension is 1 cannot be checked. In addition, file names and source code line numbers will be displayed with all run time error messages.

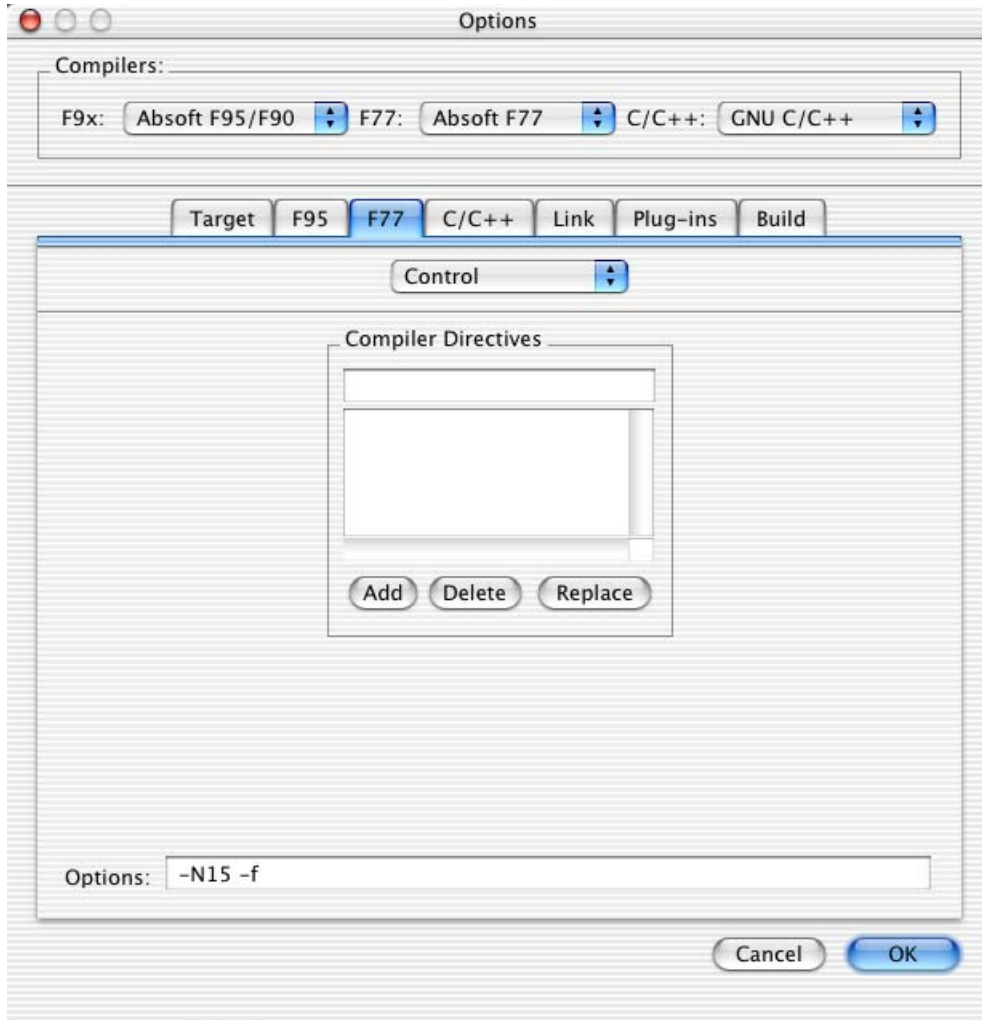
Conditional Compilation (-x)

Statements containing an X or a D in column one are treated as comments by the compiler unless the `-x` compiler option is selected. This option allows a restricted form of conditional compilation designed primarily as a means for easily removing debugging code from the final program. When the `-x` option is selected, any occurrence of an X or a D in column one is replaced by a blank character. The only source formats for which conditional compilation is valid are standard FORTRAN 77, VAX Tab-Format, and wide

format. The compiler also incorporates a complete set of statements for conditional compilation which are described in the **Conditional Compilation Statements** section **The FORTRAN 77 Program** chapter of the *Absoft FORTRAN 77 Language Reference Manual*.

Control - F77 Options

When this subset of the **F77** tab is selected, a dialog is shown for setting compiler directives:



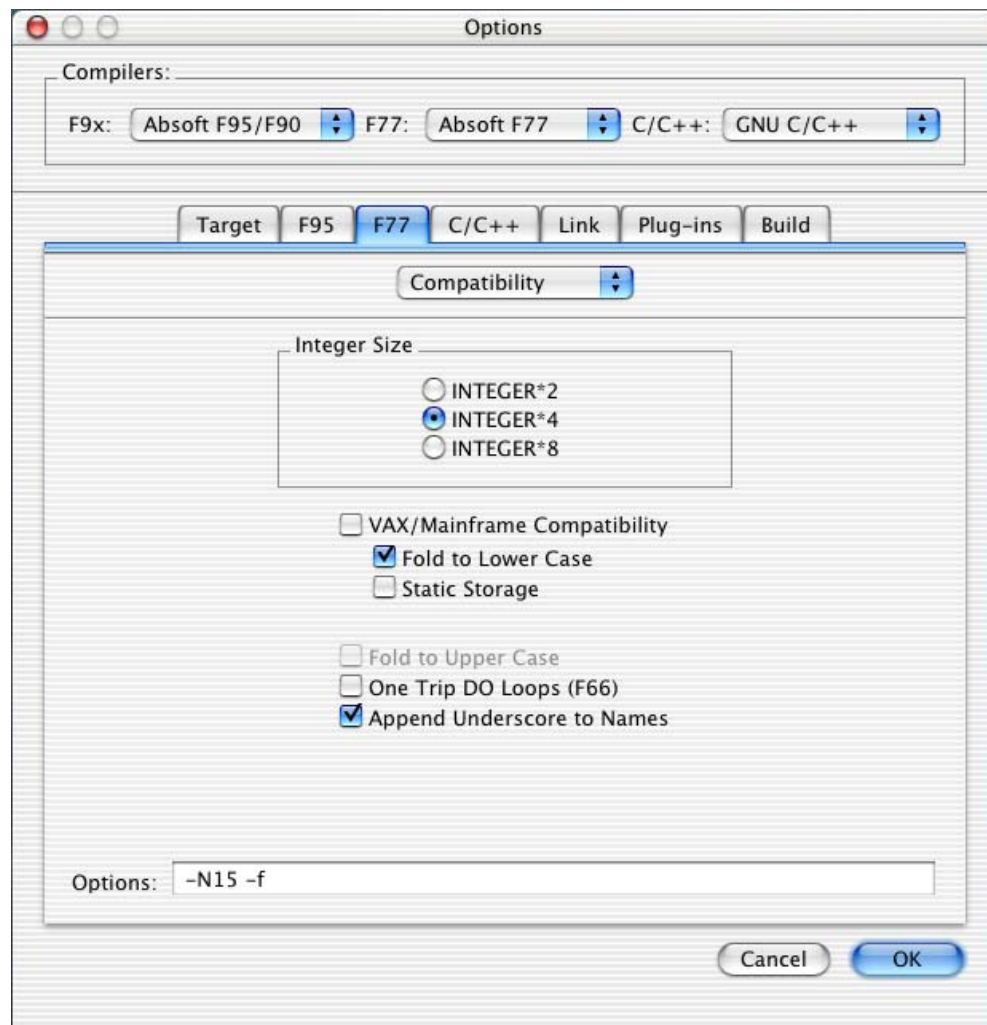
Compiler Directives (-Dname[=value])

Use this text box to enter the names and optional values of conditional compilation variables. The **-D** option is used to define conditional compilation variables from the command line. *value* can only be an integer constant. If *value* is not present, the variable is given the value of 1. Conditional compilation is described in the **Conditional**

Compilation Statements section of the chapter **The FORTRAN 77 Program** of the *Absoft FORTRAN 77 Language Reference Manual*.

Compatibility - F77 Options

This subset of the **F77** tab of the options dialog displays compatibility options for compiling FORTRAN programs. These options allow Absoft Fortran 77 to accept older or variant extensions of FORTRAN source code from other computers such as mainframes. Many of these can be used for increased compatibility with FORTRAN compilers on various mainframe computers.



Integer Sizes (-i2 and -i8)

Without an explicit length declaration, `INTEGER` and `LOGICAL` data types default to thirtytwo bits (four bytes). The `-i2` option can be used to change this default length to sixteen bits (two bytes) for both `INTEGER` and `LOGICAL`. The `-i8` option can be used to change the default `INTEGER` size to 64 bits (8 bytes). However, an explicit length specification in a type declaration statement always overrides the default data length.

Vax/Mainframe Compatibility

The VAX and mainframe compatibility switches may be quickly turned on by clicking in the **VAX/Mainframe Compatibility** check box. Selecting this option is the same as specifying **-f -s** from the command line.

Folding to Lower Case (-f)

The **-f** option will force all symbolic names to be folded to lower case. By default, the compiler considers upper and lowercase characters to be unique, an extension to FORTRAN 77. If you do not require case sensitivity for your compilations or specifically require that the compiler not distinguish between case, as in FORTRAN 77, use this option. This option should be used for compatibility with VAX and other FORTRAN environments.

Static Storage (-s)

In FORTRAN 66, all storage was static. If you called a subroutine, defined local variables, and returned, the variables would retain their values the next time you called the subroutine. FORTRAN 77 establishes both static and dynamic storage. Storage local to an external procedure is dynamic and will become undefined with the execution of a `RETURN` statement. The `SAVE` statement is normally used to prevent this, but the **-s** compiler option will force all program storage to be treated as static and initialized to zero. The **-N1** compiler option causes the definition of variables initialized in `DATA` statements to be maintained after the execution of a `RETURN` or `END` statement. This option should be used for compatibility with VAX and other FORTRAN environments.

Folding to Upper Case (-N109)

By default, the compiler considers upper and lowercase characters to be unique, an extension to FORTRAN 77. If you do not require case sensitivity for your compilations or specifically require that the compiler not distinguish between case, as in FORTRAN 77, including the **-N109** option on the compiler invocation command line will force all symbolic names to be folded to upper case.

One-Trip DO Loops (F66) (-d)

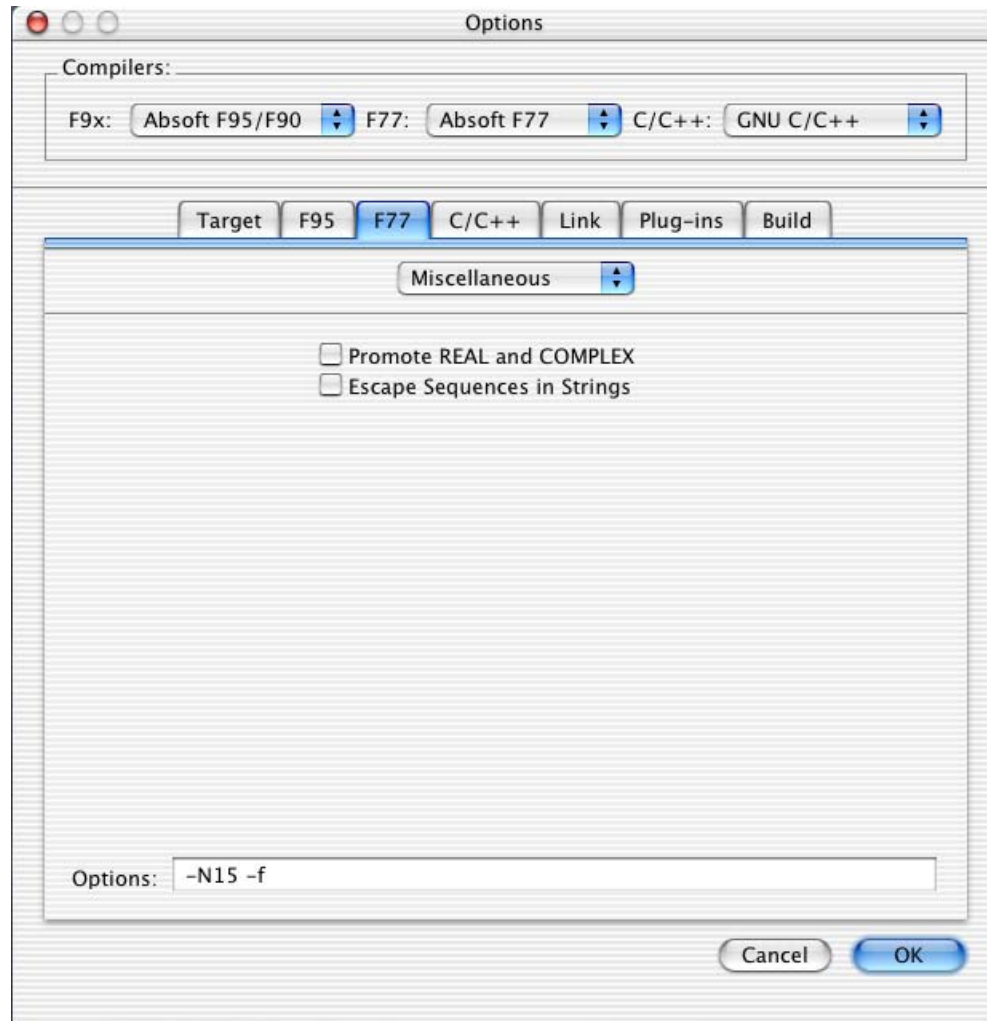
FORTRAN 66 did not specify the execution path if the iteration count of a `DO` loop, as established from the `DO` parameter list, was zero. Many processors would execute this loop once, testing the iteration count at the bottom of the loop. FORTRAN 77 requires that such a `DO` loop not be executed. The **-d** option will cause all `DO` loops to be executed at least once, regardless of the initial value of the iteration count.

Append underscore to names (-N15)

Use of the **-N15** option will cause the compiler to define `SUBROUTINE` and `FUNCTION` names with a trailing underscore. This option can be used to avoid name conflicts with the system libraries or to interface with other FORTRAN environments.

Miscellaneous - F77 Options

This subset displays a set of options for compiling FORTRAN 77 programs. These options allow Absoft Fortran 77 to accept older or variant extensions of FORTRAN 77 source code from other computers such as mainframes. Many of these can be used for increased compatibility with FORTRAN compilers on various mainframe computers.



F77 Miscellaneous Options

Promote REAL and COMPLEX (-N113)

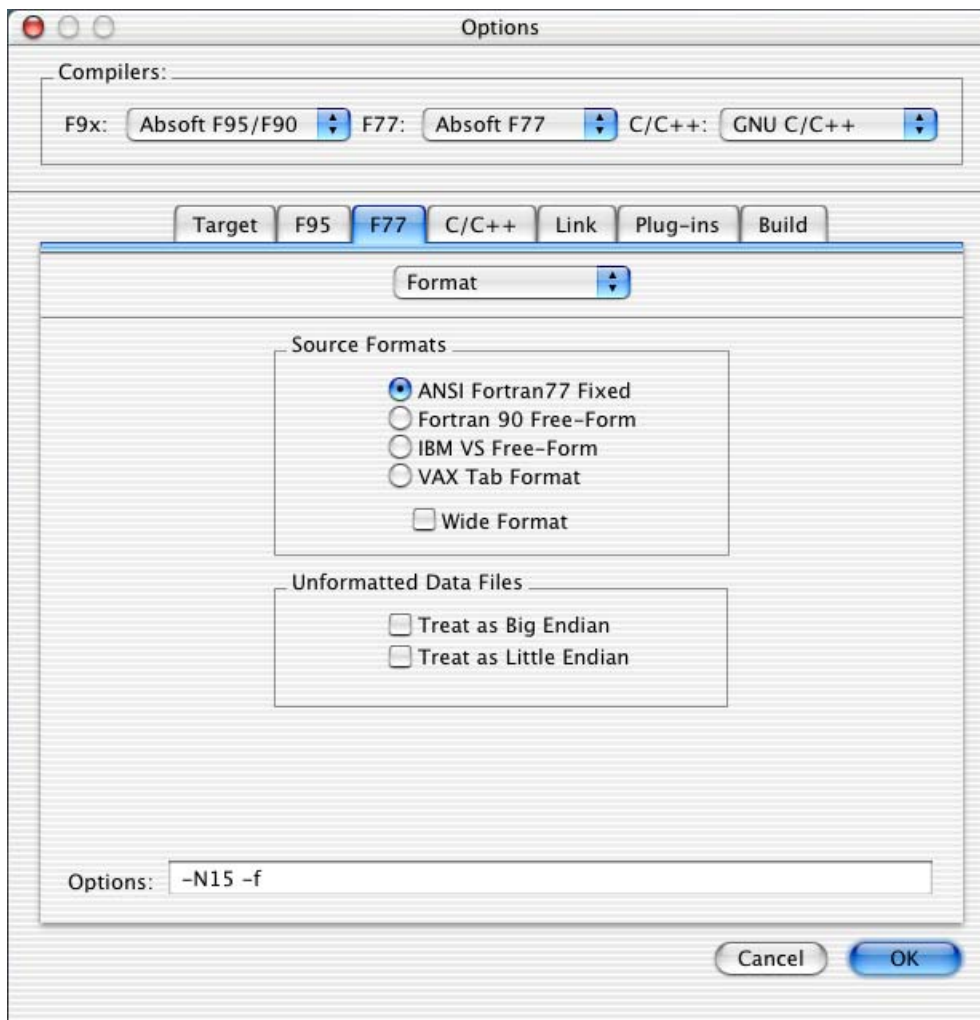
Without an explicit length declaration, single precision `REAL` and `COMPLEX` data types default to thirty-two bits (four bytes) and sixty-four bits (eight bytes), respectively. The **-N113** option is used to promote these to their double precision equivalents: `DOUBLE PRECISION` and `DOUBLE COMPLEX`. This option does not affect variables which appear in type statements with explicit sizes (such as `REAL*4` or `COMPLEX*8`).

Escape Sequences in Strings (-K)

If the **-K** option is turned on, the compiler will transform certain escape sequences marked with a ‘\’ embedded in character constants. For example ‘\n’ will be transformed into a newline character for your system. Refer to chapter **The FORTRAN 77 Program** of the *Absoft FORTRAN 77 Language Reference Manual* for more information on the escape sequences that are supported.

Format - F77 Options

For compatibility with other FORTRAN environments and to provide more flexibility, the compiler can be directed to accept source code which has been written in a variety of different formats. The default setting is to accept only ANSI standard FORTRAN source code format. See the chapter **The FORTRAN 77 Program** of the *Absoft FORTRAN 77 Language Reference Manual* for more information on alternative source code formats.



F77 Format Options

ANSI Fortran 77 Fixed

The default source form is ANSI FORTRAN 77 as described in the chapter **The FORTRAN 77 Program** of the *Absoft FORTRAN 77 Language Reference Manual*. There is no option for this setting.

Fortran 90 Free-Form (-8)

Use of the **-8** option instructs the compiler to accept source code written in the format for the Fortran 90 Free Source Form.

VAX Tab-Format (-V)

Use of the **-V** option causes the compiler to accept source code in the form specified by VAX Tab Format.

Wide Format (-W)

Use of the **-W** option causes the compiler to accept statements which extend beyond column 72 up to column 132.

Treat as Big-Endian (-N26)

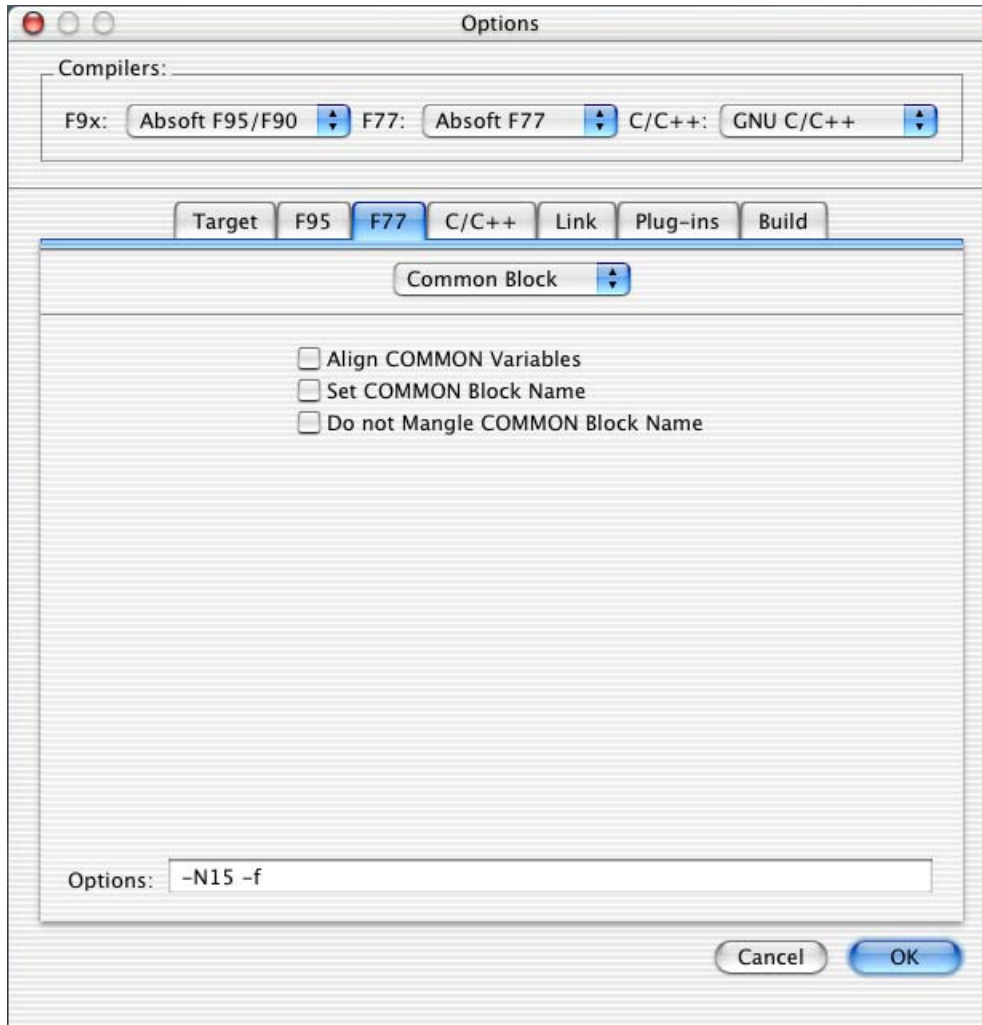
Use this option to force the compiler to consider the byte ordering of all unformatted files to be big-endian by default. The `CONVERT` specifier in the `OPEN` statement may be used to override this setting for individual files.

Treat as Little-Endian (-N27)

Use this option to force the compiler to consider the byte ordering of all unformatted files to be little-endian by default. The `CONVERT` specifier in the `OPEN` statement may be used to override this setting for individual files.

COMMON Block - F77 Options

Several options are available pertaining to the usage of common blocks in FORTRAN 77 source code.



Align COMMON Variables (-N34)

If a `COMMON` block is defined in a manner which causes a misaligned storage location, the `-N34` option can be used to insert space to eliminate the misalignment. This option may invalidate your code if the same `COMMON` block is defined differently in different program units.

Set COMMON Block Name (-N22)

The **-N22** option is used to change the scheme the compiler employs for generating global names for `COMMON` blocks. The default is to prepend the characters “_c” to the `COMMON` block name. This option causes the compiler to append a single underscore (`_`) instead. See also the **-N110** option below.

Don't Mangle COMMON Block Name (-N110)

The **-N110** option prevents the compiler from mangling (changing) the global names for `COMMON` blocks. The default is to prepend the characters “_c” to the `COMMON` block name so that it does not conflict with other global names such as external procedure names. This option causes the compiler to emit the `COMMON` block name exactly as it appears in source.

C/C++ OPTIONS

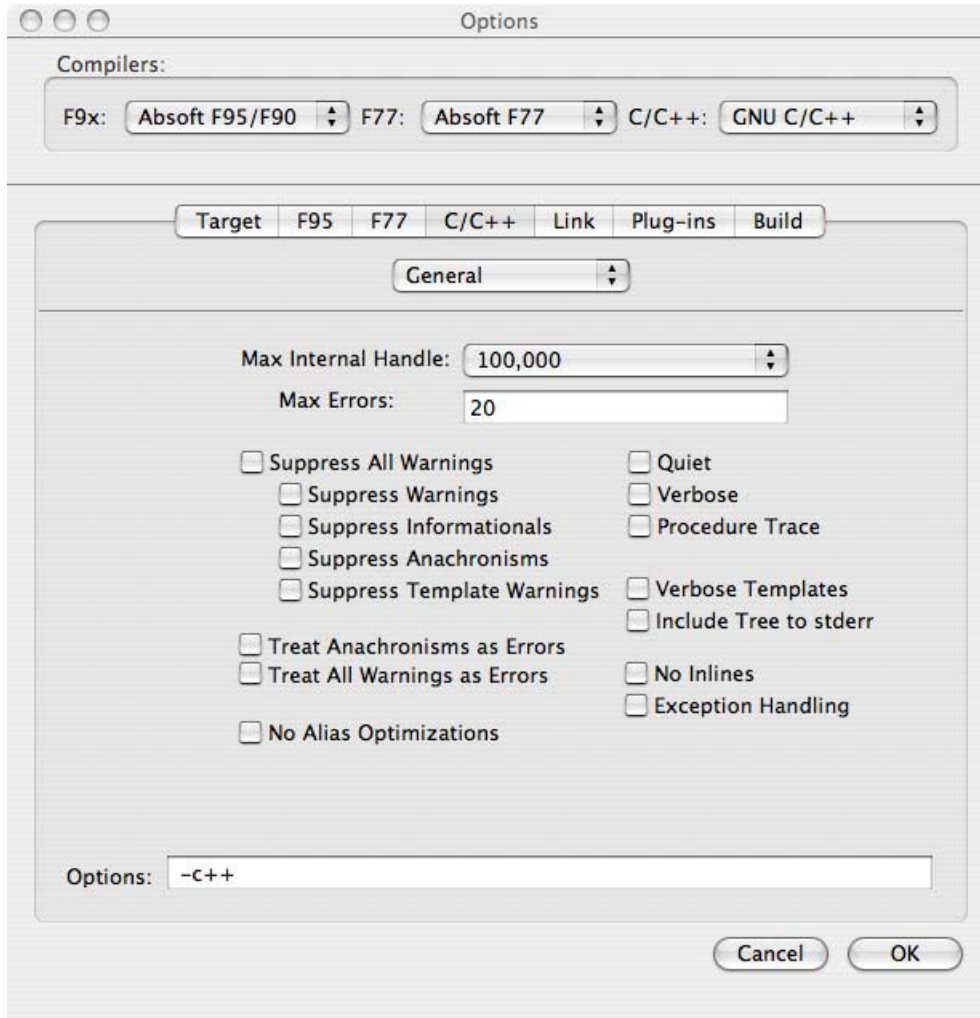
The compiler options detailed in this section give you a great deal of control over the compilation and execution of C/C++ programs with either the Absoft compiler or the GNU compiler. The choice of specific compiler is made on the **Build** tab of the **Options** dialog described at the end of this chapter.

This section of the manual describes the options of the Absoft C/C++ compiler only. The GNU options in the Absoft Developer Tools Interface application are provided as a convenience; for more information on using the Mac OS X C/C++ compiler, see the Macintosh OS X manual page for `cc`.

Select the **Set Project Options** command in the **Configure** menu to access the Options dialog. The C/C++ options fall into three categories: General, Preprocessor, and Format. For quick reference, the options listed in the sections that follow are in the order in which they appear in the dialog. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked in the Absoft Developer Tools Interface application, the same letters will appear in the in the corresponding **Options** box.

General – C/C++ Options

These options control some general characteristics of the C/C++ components of the program being built. For information on using specific options of the Mac OS X C/C++ compiler, see the Mac OS X manual page for *cc*.



C/C++ General Options

Max Internal Handle (-T *n*)

This option is used to change the number of handles used internally by the compiler. Under most conditions, the default value of 10000 handles is sufficient to compile even extremely large programs. However, under certain circumstances, this value may be exceeded and the compiler will issue a diagnostic indicating that the value should be increased.

Max Errors (-maxerr *n*)

This option sets the maximum number of errors that can be emitted before the compiler aborts. Setting a reasonable number, such as 20, prevents the compiler from issuing hundreds of meaningless error messages when a brace or semi-colon is missing.

Diagnostic Messages

The level of diagnostic reporting can be controlled with the next seven options. The Absoft C/C++ compiler categorizes diagnostic messages as follows:

<i>error</i>	non-recoverable syntactic error such as a missing brace or semi-colon
<i>warning</i>	the usage or construction may be an error or non-portable, but is allowed on the assumption that you know what you are doing
<i>informational</i>	the element is legal but may be the cause of a subtle bug
<i>anachronism</i>	currently allowed construction which may not be supported in a future release
<i>template</i>	show the point of instantiation of a template function when that instantiation caused an error

Suppress All Warnings (-w31)

Checking this box suppresses the listing of all diagnostic warning messages. Various levels of warnings may be suppressed by checking the individual warning boxes as discussed below.

Suppress Warnings (-w16)

Checking this box suppresses the listing of most warning messages. The only diagnostic warnings still reported are informational and anachronisms.

Suppress Informationals (-w8)

This check box controls the listing of informational warning messages such as using a member name in a derived class that was defined in the base class, referencing a function which does not have a prototype, or defining a variable which is never used.

Suppress Anachronisms (-w2)

Checking this box suppresses the listing of anachronism warning messages in C++ compilations such as a trailing comma in a parameter list, a temporary used for non-`const` reference, or the use of a count in `delete[]`.

Suppress Template Warnings (-w1)

This option suppresses warnings issued when an error occurs while instantiating a template function.

Treat Anachronisms as Errors (-wp)

This check box causes the usage of C++ anachronisms to be treated as errors. This option should always be used when developing new code to avoid the use of deprecated features.

Treat All Warnings as Errors (-wabort)

This check box causes any warning condition to be treated as an error.

No Alias Optimizations (-N19)

The **-N19** option is selected to prevent the compiler from performing address optimizations when pointer aliases are present.

Quiet (-q)

The Absoft C/C++ compiler normally displays information to standard output as it compiles an application. Enabling the **-q** option will suppress any messages printed to standard output. Errors will still be printed to the standard diagnostic output, however.

Verbose (-v)

Enabling the **-v** option will cause `ACC` to display the commands it is sending to the compiler and linker.

Procedure Trace (-N124)

Specifying the **-N124** option will cause the compiler to generate code to write the name of the currently executing procedure to standard out. This option is useful for tracing program execution and quickly isolating execution problems.

Verbose Templates (-ptv)

This option causes the compiler to print a message to `stderr` whenever a template is instantiated.

Include Tree to Stderr (-H)

An include file tree will be printed to `stderr` when the **-H** option is enabled.

No inlines (-inline none)

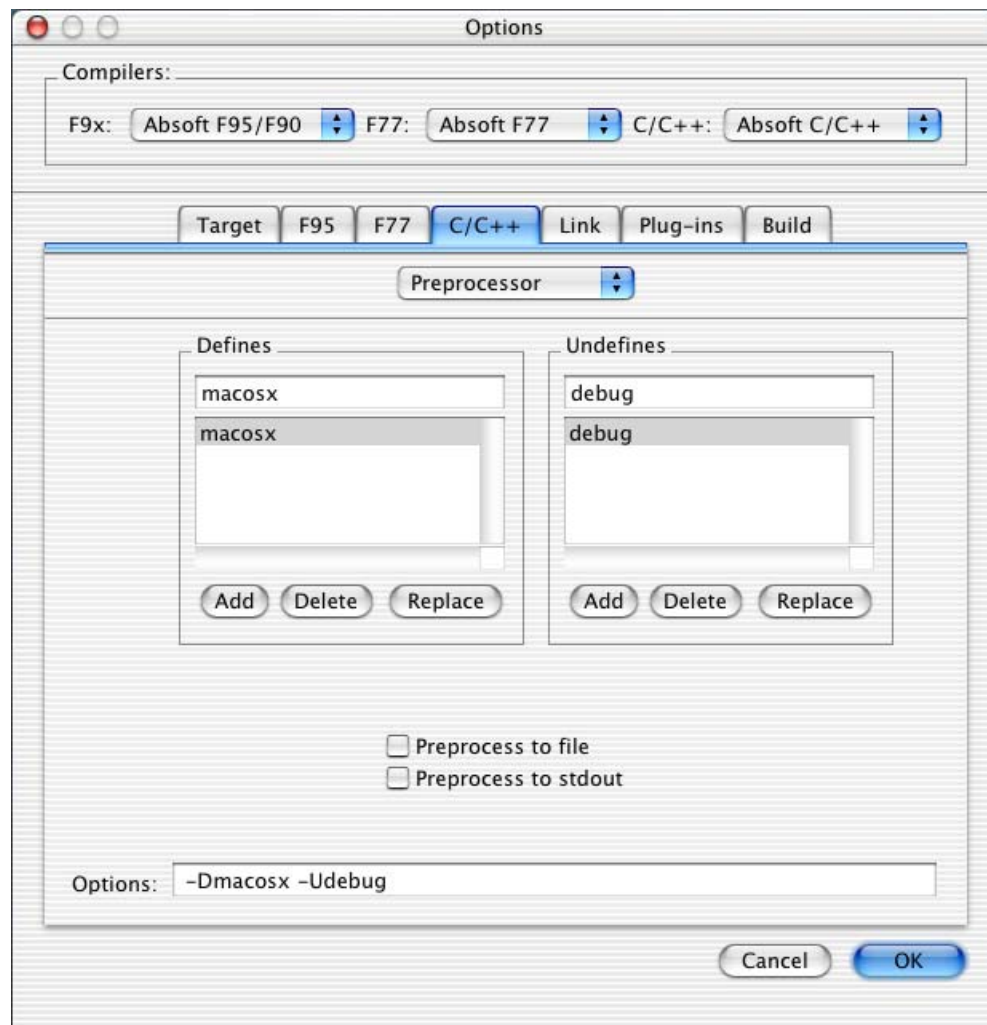
This option prevents the compiler from inlining functions.

Exception Handling (-except on|off)

This option enables C++ exception handling. It allows you to use the `try/catch/throw` constructs in your program. All functions that are called directly or indirectly from a `try` block must be compiled with the **-except on** option. Only specify this option when you are actually using exception handling as there is a considerable overhead to its use.

Preprocessor – C/C++ Options

This subset of the **C/C++** tab of the options dialog displays options for controlling the preprocessor stage of the C and C++ compilation process.



Defines (-D name[=value])

Use these edit and list text boxes to define the names and optional values of preprocessor variables. The **-D** option is used to define preprocessor variables from the command line.

value can only be an integer constant. If *value* is not present, the variable is given the value of 1.

Undefines (-U *name*)

Use these edit and list text boxes to undefine the names preprocessor variables. The **-U** option is used to undefine preprocessor variables from the command line.

Do not search standard system directories

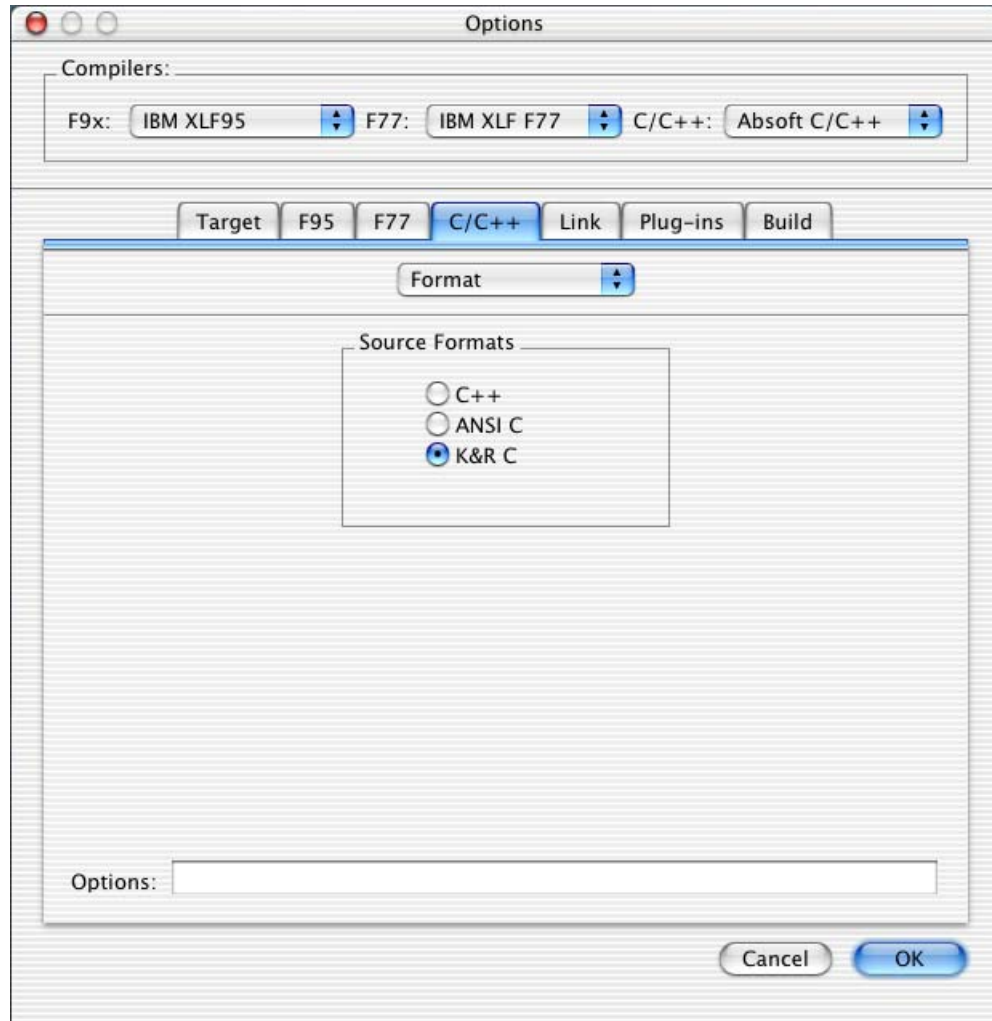
Do not search the standard system directories for header files (**-nostdinc**). Only search directories provided by the **-I** option.

Preprocess files only

The **Preprocess files only** option (**-E**) directs the compiler to place the output of the C preprocessor in a specified file, or to *stdout*.

Format – C/C++ Options

This subset of the **C/C++** tab of the options dialog displays options used to specify the language syntax format of the source files.



C/C++ Format Options

C++ (-c++)

Enabling the **-c++** option allows you to use the namespaces and runtime type information features ANSI C++ in your source code.

ANSI C (-A)

Set the **-A** option for source which conforms to the ANSI X3.159-1989 standard for the C programming language

K and R C (-K)

The **-K** option is set with the radio button titled **K and R C** and should be used with older style Kernighan and Ritchie C. This is the default option.

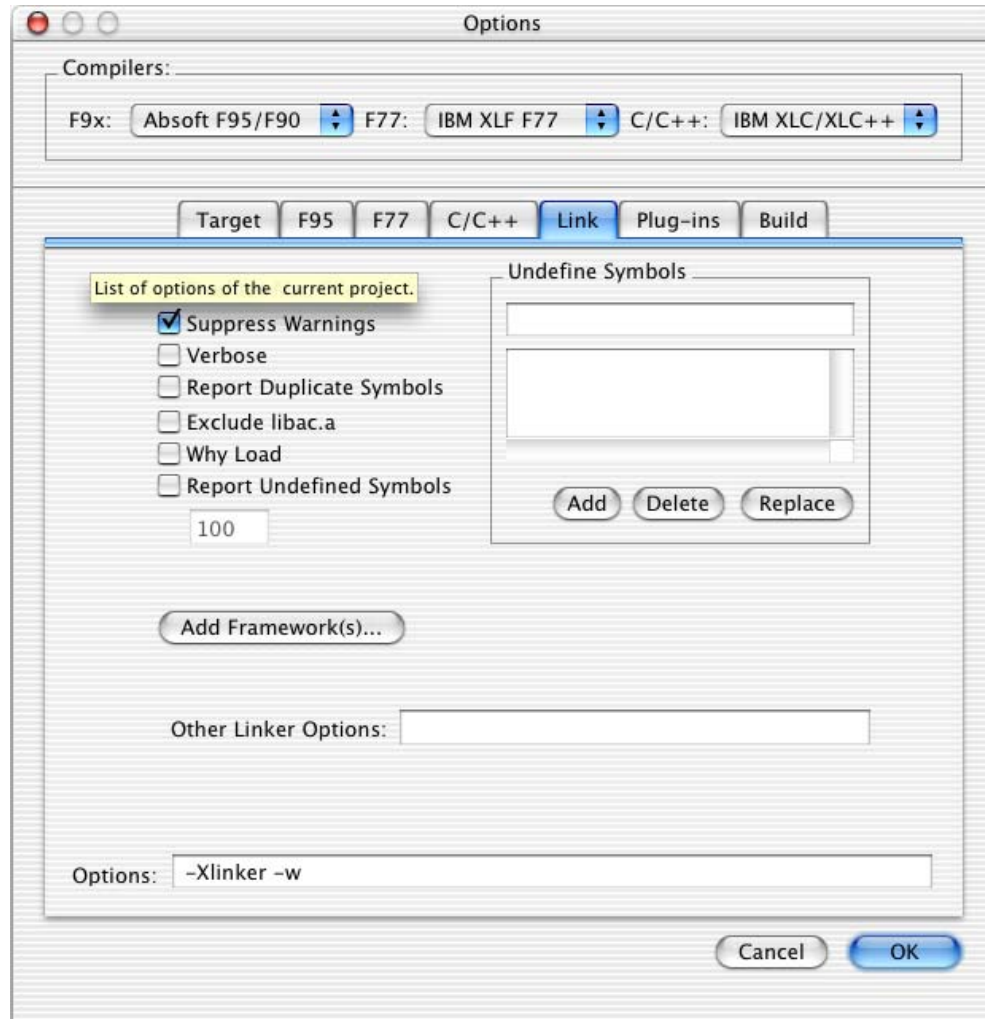
LINKER OPTIONS

The linker options detailed in this section give you control over certain aspects of the program linking process. Select the **Set Program Options** command in the **Configure** menu to access the Options dialog.

For quick reference, the options listed in the sections that follow are in the order in which they appear in the dialog boxes. For information on using specific options of the Mac OS X Linker, see the Mac OS X manual pages for *ld*. When an option is checked, the same letters will appear in the **Options** box.

General - Link Options

There is only one set of options for the linker in the Options dialog:



Produce Map File

Produce a load map, which lists all the segments and sections. The list will contain the address where each input file's section appears in the output file and the section's size.

Suppress Warnings

Use this option to suppress all linker warning messages.

Verbose

This option traces the progress of the linker.

Report Duplicate Symbols

Use this option to treat multiply defined symbols as warnings instead of errors. The first such symbol is used for linking. Other symbols of the same name may be accessed by local references.

Exclude libac.a

Use this option to exclude libac.a from the set of default libraries used for linking. This may be necessary if you are compiling C/C++ code using a compiler other than the Absoft C/C++ compiler.

Why Load

This option indicates why each member of a library is loaded.

Report Undefined Symbols

For the first number undefined symbols this option, displays each file in which the symbol appears, its type and whether the file defines or references it

Add Framework(s)...

This button displays a file browser that allows you add system frameworks to your project. On a command line use **-framework name**.

Passing Options To The Linker

For ease of use within the Macintosh OS X environment, many of the options that are available to the system linker are also available to the f77 and f95 compiler drivers. Specifying any of these options indicates that all files specified on the command line should be processed through the linkage phase. Unless the **-S** or **-c** options are specified, all intermediate files (relocatable objects and/or assembly source) will be deleted. See the system documentation on *ld* for more information regarding these options. In brief, the options are as follows:

Undefine A Symbol (-u)

Specifying the **-u***symbol_name* option will enter *symbol_name* as an undefined symbol to the linker.

Linker Options (-X and -Xlinker)

Use the **-X option** switch to pass an option directly to the linker. The FORTRAN 77 or Fortran 90/95 driver will pass **option** to the linker. If you want to pass an options which takes an argument, use the **-X** option twice.

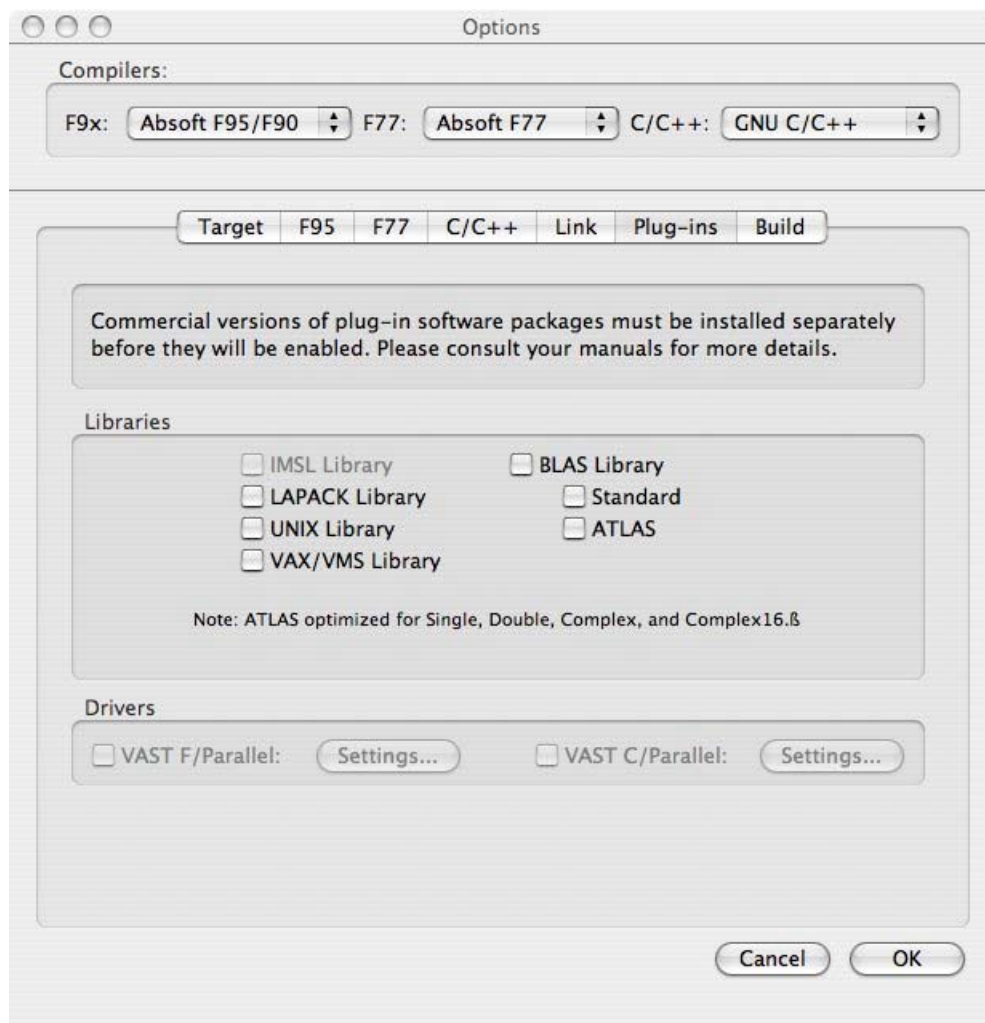
For C and C++, the option is name **-Xlinker**.

Other Link Options

For more information on using specific options of the Mac OS X linker, see the Mac OS X manual page for *ld*.

PLUG-INS

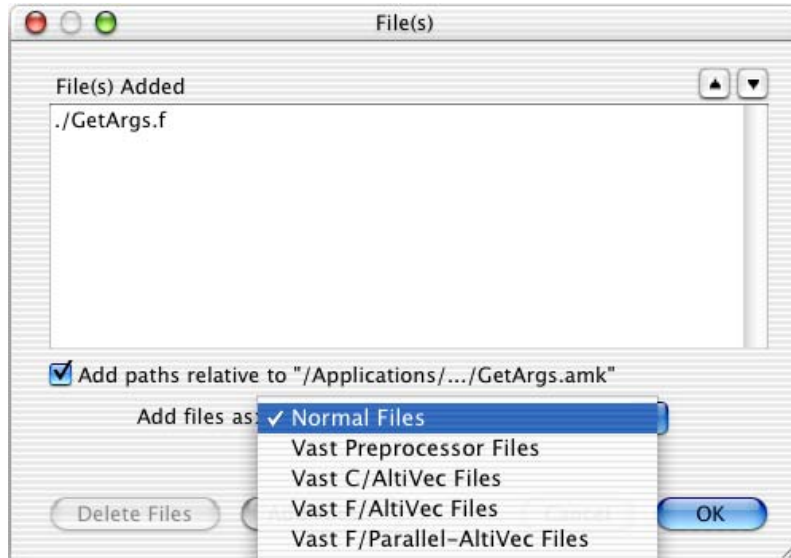
The **Plug-ins** tab of the Options dialog provides a way to integrate any additional tools that you may have purchased into your project. Select the **Set Project Options** command in the **Configure** menu to access the Options dialog.



Two types of plug-ins are available: libraries and preprocessors. The check boxes for these items are enabled only if the product was purchased and installed.

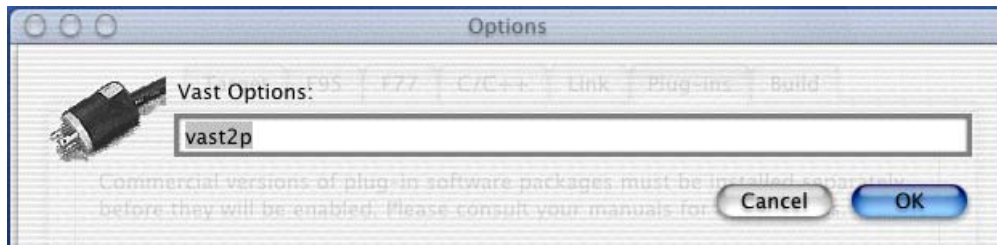
IMPORTANT: Source files that are processed by a VAST preprocessor must be added to the project using the VAST Preprocessor file type. Use the **Add files as** drop-down

menu select the type of preprocessor as required. See the section describing **Add/Remove File(s)** discussed earlier in this chapter.



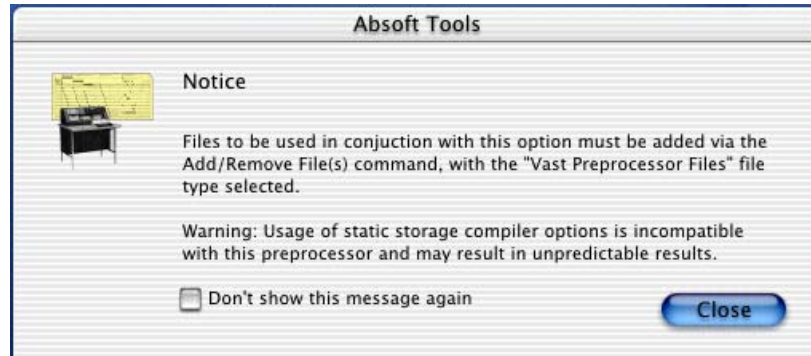
VAST

Crescent Bay Software develops the VAST pre-processors. They are pre-processor and library combinations for multi-processor and Altivec Mac OS X installations and can be used with FORTRAN 77, Fortran 90/95, or C/C++ programs. Click the **Settings** tab to display a dialog for specifying additional preprocessor options.



Documentation for the VAST preprocessors is provided in electronic format on the Absoft Pro Fortran CDROM.

IMPORTANT: The VAST preprocessors do not support the use of automatic static storage (**-s** option) by either Absoft Fortran 90/95 or FORTRAN 77.



The VAST preprocessors can be invoked from a command line or within a custom makefile by using the supplied compiler drivers, pf90 and pf77, for Fortran 90/95 and FORTRAN 77, respectively. The VAST drivers v90, v77, vlf, vlf90, vlf95 are used for VAST-F/AltiVec support for Absoft Fortran 90/95, Absoft FORTRAN 77, IBM XLF FORTRAN 77, IBM XLF Fortran 90, and IBM XLF Fortran 95. The combined multi-processor and AltiVec VAST drivers are pv90, pv77, pvlf, pvlf90, and pvlf95. The VAST-C/AltiVec driver for IBM XLC is vlc.

IMSL Library

The IMSL check box controls the use of the Visual Numerics IMSL Math and Statistics libraries. Complete documentation is provided on the Pro Fortran CDROM. Place a check in this box to automatically link against the library.

LAPACK Library

These libraries contain the basic LAPACK and LAPACK90 libraries obtained from www.netlib.org. LAPACK is used for the most common problems in numerical linear algebra including linear equations, linear least squares, eigenvalue, and singular value problems. LAPACK90 is the Fortran 90/95 interface for LAPACK. Source code for these libraries is supplied with Pro Fortran.

UNIX Library

The Unix library supplied routines compatible with those provided by Sun Microsystems and other Unix based Fortran compilers. Documentation on the routines in this library is available in the *Compatibility Libraries* manual supplied with Pro Fortran. Source code to all library routines is supplied.

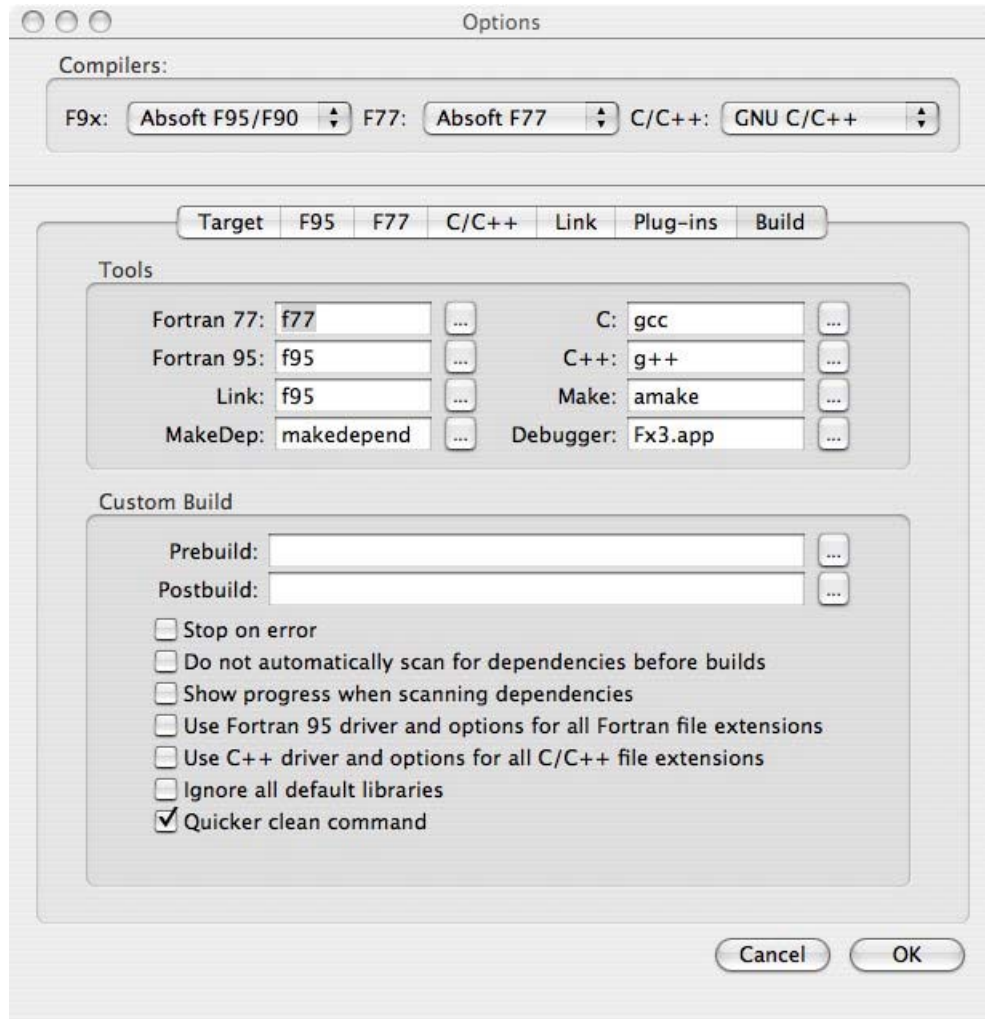
VAX/VMS Library

The VMS library has a few additional routines with calling conventions that match VAX FORTRAN. Documentation on the routines in this library is available in the *Compatibility Libraries* manual supplied with Pro Fortran. Source code to all library routines is supplied.

None of the routines in this library are part of the ANSI FORTRAN 77 or Fortran 90/95 standards and should be used with caution if portability between platforms is a concern.

BUILD OPTIONS

Use this tab of the Options dialog to specify the tools that are used to build an application. The Tools section specifies the compilers, linker, make facilities, and any additional developer tools. The default path to these tools is the BIN directory of the main Absoft directory defined by the environment variable `ABSOFTE` (e.g. Applications\Absoft\Bin). Paths specified by the environment variable `PATH` will also be searched. Use fully qualified paths for tools not residing in these directories.



Build Options

The Prebuild and Postbuild edit boxes in the Custom Build section can be used to specify files containing additional macros, rules, dependencies, and commands to be inserted into the makefile. Examine the Makefile tab in the output window (discussed later in this chapter) for more information.

The check boxes at the bottom of this property page allow you to:

1. Discontinue the build process after the first error is encountered
2. Prevent the build process from updating file dependencies
3. Display file dependency scan progress
4. Specify that only the Fortran 90/95 compiler be used for all FORTRAN language compilations.
5. All normal C and C++ file extensions will be handled by the C++ compiler only
6. Indicate that no default libraries names are to be supplied to the linker
7. Clean up output files using wildcard characters instead of deleting each output file individually.

CHAPTER 5

Porting Code

This chapter describes issues involved in porting FORTRAN 77 code from other platforms. One of the major design goals for Absoft Fortran 77 is to permit easy porting of FORTRAN 77 source code from mainframe computers such as VAX and IBM, and from workstations such as Sun. The result is the rich set of statements and intrinsic functions accepted by Absoft Fortran 77. The last section of this chapter describes Macintosh OS X specific issues about porting code.

As a general rule when porting code, use the following two compiler options:

- f** Fold all symbols to lower case.
- s** Force all program storage to be treated as static and initialized to zero.

Ported programs that have incorrect runs or invalid results are usually caused by the differences between the Macintosh and other environments such as floating point math precision or stack-size issues. See the section **Other Porting Issues** later in this chapter for special considerations when porting code to the Macintosh. In addition, you may want to use this option:

- c** Check array boundaries and generate better runtime errors. Using this option makes programs slightly larger and they will execute slower.

If you want to use the Absoft debugger, Fx, add the **-g** option to generate debugging information.

PORTING CODE FROM VAX

Absoft Fortran 77 automatically supports most of the VAX FORTRAN language extensions. Below is a list of key VAX FORTRAN extensions that are supported and a list of those that are not supported. For a complete list of VAX extensions, refer to Appendix H. Using various options, the compiler can also accept VAX Tab-Format source lines and/or 132-column lines. Otherwise, only ANSI FORTRAN 77 fixed format lines are accepted.

Key Supported VAX FORTRAN Extensions

- NAMELIST—the NAMELIST terminator may be either “\$” or “&”
- STRUCTURE, RECORD, UNION, MAP, %FILL statements
- DO WHILE loops
- INCLUDE statement
- ENCODE, DECODE, ACCEPT, TYPE, and most OPEN I/O specifiers
- Hollerith and hexadecimal constant formats

- “!” comments

Key Unsupported VAX FORTRAN Extensions

- Absoft Fortran 77 uses IEEE floating point representation
- I/O statements DELETE, DEFINE FILE, and REWRITE
- Data dictionaries

Compile Time Options and Issues

Absoft Fortran 77 can be made even more compatible with VAX FORTRAN by using a group of compiler options collectively referred to as the “VAX compatibility options”, listed below. When using the Commando interface for the compiler, they may be invoked by a single check box.

- f** Fold all symbols to lower case.
- s** Force all program storage to be treated as static and initialized to zero.

VAX-compatible time, date, and random number routines are available by linking with the file `vmplib.o` in the `FLibraries` folder. They are:

DATE subroutine	returns current date as CHARACTER*9
IDATE subroutine	returns current date as 3 INTEGER*4
TIME subroutine	returns current time as CHARACTER*8
SECNDS subroutine	returns seconds since midnight
RAN function	returns random number

The following list of VAX FORTRAN “qualifiers” shows the equivalent Absoft Fortran 77 options or procedures:

/ANALYSIS_DATA	no equivalent
/CHECK_BOUNDS	-C to check array boundaries
/CHECK_NONE	do not use the -C option
/CHECK_OVERFLOW	no equivalent
/CHECK_UNDERFLOW	no equivalent
/CONTINUATIONS	no equivalent
/CROSS_REFERENCE	no equivalent
/DEBUG	-g to generate debugging information
/D_LINES	-x to compile lines with a “D” or “X” in column 1
/DIAGNOSTICS	append \geq filename to the <code>f77</code> command line to create a file containing compiler warning and error messages (type Option-> for the \geq character)
/DML	no equivalent
/EXTEND_SOURCE	-W to permit source lines up to column 132 instead of 72
/F77	do not use the -d option

/NOF77	-d for FORTRAN 66 compatible DO loops
/G_FLOATING	see the section Numeric Precision later in this chapter
/I4	do not use the -i option
/NOI4	-i for interpreting INTEGER and LOGICAL as INTEGER*2 and LOGICAL*2
/LIBRARY	no equivalent
/LIST	a symbol table dump may be generated with the -D option
/MACHINE_CODE	-S to generate an assembly source file that <i>can</i> be assembled
/OBJECT	no equivalent
/OPTIMIZE	-O to use basic optimizations
/PARALLEL	no equivalent
/SHOW	no equivalent
/STANDARD	-N32 to generate warnings for non-ANSI FORTRAN 77 usage
/WARNINGS DECLARATIONS	the IMPLICIT NONE statement may be used to generate warnings for untyped data items
/WARNINGS NONE	-w to suppress compiler warnings

The tab size on the Macintosh may be different than the VAX. You can set the tab size for a file by pressing ⌘-Y while editing a file and typing the tab size for the file. For more information about tab size, see the section **Tab Size** later in this chapter.

Runtime Issues

If the program is having problems with I/O, make sure you are using the **-N3** and **-N51** options described in detail in sections **Use record lengths in I/O** and **RECL Defines 32-bit words** in chapter **Using the Compiler**.

PORTING CODE FROM IBM VS FORTRAN

Absoft Fortran 77 automatically supports most of the IBM VS FORTRAN language extensions. Below is a list of key VS FORTRAN extensions that are supported and not supported. Using a compiler option, Absoft Fortran 77 can also accept VS FORTRAN Free-Form source lines which use 80 columns, otherwise, only ANSI FORTRAN 77 fixed format lines are accepted.

Key Supported VS FORTRAN Extensions

- “*” comments in column 1
- Can mix CHARACTER and non-CHARACTER data types in COMMON blocks
- The NAMELIST terminator may be an ampersand “&”
- Hollerith constants

Key Unsupported VS FORTRAN Extensions

- Absoft Fortran 77 uses IEEE floating point representation (more accurate)
- Debug statements
- I/O statements `DELETE`, `REWRITE`, and `WAIT`
- `INCLUDE` statement syntax is different

Compile-time Options and Issues

Absoft Fortran 77 can be made even more compatible with VS FORTRAN by using these compiler options:

- f Fold all symbols to lower case
- s Force all program storage to be treated as static and initialized to zero

PORTING CODE FROM MICROSOFT FORTRAN (PC VERSION)

Absoft Fortran 77 automatically supports many of the Microsoft FORTRAN language extensions. Below is a list of key Microsoft FORTRAN extensions that are supported and not supported. Absoft Fortran 77 does not have the code size restrictions found in the segmented Microsoft FORTRAN models.

Key Supported Microsoft FORTRAN Extensions

- The NAMELIST terminator may be an ampersand “&”
- The Free-Form Source Code is very similar to VS FORTRAN (-V option)
- `AUTOMATIC` statement
- `STRUCTURE`, `RECORD`, `UNION`, `MAP` statements
- `SELECT CASE` statements
- `DO WHILE` loops
- `INCLUDE` statement
- `OPEN` statement displays standard file dialog when using `FILE=""`
- Conditional compilation statements

Key Unsupported Microsoft FORTRAN Extensions

- Metacommands
- MS-DOS specific intrinsic functions
- `INTERFACE TO` statement

Compile-time Options and Issues

Absoft Fortran 77 can be made even more compatible with Microsoft FORTRAN by using these compiler options:

- f** Fold all symbols to lower case
- s** Force all program storage to be treated as static and initialized to zero

The following list of Microsoft FORTRAN metacommands shows the equivalent Absoft Fortran 77 options or procedures:

\$DEBUG	-C to check array boundaries and other run-time checks
\$DECLARE	the <code>IMPLICIT NONE</code> statement may be used to generate warnings for untyped data items
\$DO66	-d for FORTRAN 66 compatible <code>DO</code> loops
\$FLOATCALLS	all floating point is calculated inline or with a threaded math library in Absoft Fortran 77
\$FREEFORM	-v for IBM VS FORTRAN Free-Form source code
\$INCLUDE	use the <code>INCLUDE</code> statement
\$LARGE	not necessary — Absoft Fortran 77 does not have the data size restrictions found in the segmented Microsoft FORTRAN models
\$LINESIZE	not applicable
\$LIST	no equivalent
\$LOOPOPT	-U for loop unrolling optimization; -R for loop invariant removal
\$MESSAGE	no equivalent
\$PACK	use <code>\$PACKON</code> and <code>\$PACKOFF</code>
\$PAGE	not applicable
\$PAGESIZE	not applicable
\$STORAGE:2	-i for interpreting <code>INTEGER</code> and <code>LOGICAL</code> as <code>INTEGER*2</code> and <code>LOGICAL*2</code>
\$STORAGE:4	do not use the -i option
\$STRICT	-N32 to generate warnings for non-ANSI FORTRAN 77 usage
\$SUBTITLE	not applicable
\$TITLE	not applicable
\$TRUNCATE	no equivalent

If code ported from MS-DOS does not compile because of many errors, the end-of-line characters within the file may not match the return character (decimal 13 or Control-M) the Macintosh expects. To *remove* extraneous linefeeds from files transferred from MS-DOS which have both a linefeed and a return character, use the **Replace...** item in the **Find** menu and type control-J for the string to find and nothing for the string to replace.

PORTING CODE FROM SUN WORKSTATIONS

Absoft Fortran 77 automatically supports most of the Sun FORTRAN language extensions. Below is a list of key Sun FORTRAN extensions that are supported and not

supported. The Sun FORTRAN compiler appends an underscore to all external names to prevent collisions with the C library. Absoft Fortran 77, by default, does not append an underscore to maintain compatibility with the Macintosh. The **-N15** option may be used to append underscores to routine names.

Key Supported Sun FORTRAN Extensions

- NAMELIST; the NAMELIST terminator may be either “\$” or “&”
- STRUCTURE, RECORD, POINTER, UNION, MAP, %FILL statements
- DO WHILE loops
- INCLUDE statement
- ENCODE, DECODE, ACCEPT, TYPE, and most OPEN I/O specifiers
- Hollerith and hexadecimal constant formats
- “!” comments in column 1

Absoft has a compiler for Sparc-based SunOS systems. It has the same features and language extensions as Absoft Fortran 77. The compilers are 100% source-compatible.

PORTING CODE FROM THE NEXT WORKSTATION

Absoft FORTRAN 77, formerly available, but now discontinued on the NextStep operating system for either Motorola or Intel microprocessors had the same optimizations and language extensions as Absoft Fortran 77. The object-oriented extensions of the NeXT compiler are specific to the NextStep environment and are not supported with Absoft Fortran 77 for the Macintosh OS X. The compilers are 100% source-compatible.

PORTING CODE FROM THE IBM RS/6000 WORKSTATION

Absoft FORTRAN 77, formerly available, but now discontinued for the IBM RS/6000 computer and had the same optimizations and language extensions as Absoft Fortran 77 for Windows with Intel or PowerPC processors. The compilers are 100% source compatible.

PORTING CODE FROM INTEL 386/486/PENTIUM COMPUTERS

Absoft Pro Fortran is available for the Intel Pentium systems including Windows 95, Windows 98, and Windows/NT. It has the same optimizations and language extensions as Absoft Pro Fortran for the Macintosh. The compilers are 100% source compatible.

PORTING CODE TO/FROM OTHER MACINTOSH SYSTEMS

Language Systems Fortran

Absoft Fortran 77 and Language Systems Fortran share many extensions implemented in other compilers. In addition, Absoft Fortran 77 automatically supports most of the

Language Systems Fortran specific language extensions. Below is a list of key Language Systems extensions that are supported and a list of those that are not supported. For a complete list of Language Systems extensions and their usage, refer to Appendix N.

Key Supported Language Systems Fortran Extensions

- `STRING` declaration statement
- `POINTER` declaration statement
- `LEAVE` control statement
- `GLOBAL`, `CGLOBAL`, and `PBGLOBAL` statements
- `CEXTERNAL` and `PEXTERNAL` statements
- `INT1`, `INT2`, `INT4`, and `JSIZEOF` intrinsic functions

Key Unsupported Language Systems Fortran Extensions

- variables in `FORMAT` statements
- Language Systems Fortran compiler directives

Other Absoft Compilers

Over the past 15 years, Absoft has offered several different compilers for a number of Macintosh environments. This section outlines some of the differences between these products.

MacFortran	This 68000 compiler supported ANSI FORTRAN 77 and compiled programs directly from the Finder without using MPW. Although it lacked optimizations and support for many of the extensions in Absoft Pro Fortran for Macintosh, it compiled very fast and was easy to use.
MacFortran/020	This 68000 compiler was the same as MacFortran but it could also produce faster code for 68020 and 68030 systems that incorporated a floating point unit.
MacFortran II	This 68000 compiler is very similar to Absoft Pro Fortran for Macintosh. It supports many of the same optimizations and extensions, but is designed for 68000 based Macintoshes.

OTHER PORTING ISSUES

Not all porting and compatibility issues can be solved automatically by Absoft Pro Fortran or by using various option combinations. There are six issues that must be addressed on a program-by-program basis for the Macintosh computer:

Memory Management	Tab Character Size
Naming Conventions	Numeric Precision
File and Path Names	Floating Point Math Control

Memory Management

Local variables and temporary values are stored in the stack frame. All other storage is allocated statically in the data and/or bss sections.

Dynamic Storage

Storage for variables local to a function or a subroutine is allocated in the stack frame. As a result, local variables are undefined when execution of a function or subroutine begins and become undefined again when execution terminates. This can cause difficulties in two areas.

First, problems may arise when porting Fortran applications from environments that statically allocate all memory; the application may expect variables to retain their definition status across procedure references. However, it produces applications that

make more effective use of memory and provides the ability to call functions and subroutines recursively. The next section describes how to declare static storage space.

Second, the Macintosh OS X stack is limited to 512 KB and large arrays allocated in the stack frame may overflow the stack. You can increase the stack size with the `ulimit` command (`ulimit` is a bash command - the csh equivalent to `ulimit -s is limit stack`) to raise the stack size limit:

```
# ulimit -s
512
# ulimit -s 32768
# ulimit -s
32768
```

Static Storage

There are three ways to define static storage in Fortran. The first two allow static variables to be defined selectively and are either placing them in `COMMON` blocks or using the `SAVE` statement. The third method, using the `-s` compiler option, forces *all* program storage to be treated as static. Static memory is allocated out of the data and/or bss sections and remains defined for as long as the application runs. In addition, all static storage will be initialized to zero when the application begins execution.

Naming Conventions

Global names in Fortran include all procedure names and `COMMON` block names, both of which are significant to 31 characters. All global names in Absoft FORTRAN 77 are case sensitive unless one of the compiler character case options has been selected. All global names in Absoft Fortran 90/95 are upper case unless one of the compiler character case options has been selected. All other symbols are manipulated as addresses or offsets from local labels and are invisible to the linker.

Procedure Names

Names of functions and subroutines in Fortran programs will appear in the assembly language source output or object file records exactly as they are stated in the Fortran source code with a leading underscore (“_”) prepended. This is identical to how the C Programming Language represents symbolic names on Macintosh OS X.

If a FORTRAN 77 subroutine is defined as:

```
SUBROUTINE SUB(...)
.
.
.
RETURN
END
```

It will be defined in assembly language as:

```
        .text
        .globl _sub
_sub:
        .
        .
        .
        blr
```

COMMON Block Names

The convention in Absoft Pro Fortran is to precede the name given in the COMMON statement with the characters “_c”. BLANK common uses the characters _blank.

For example, the COMMON block declaration:

```
COMMON /the_block/ a, b, c
```

Will produce the following assembler directive:

```
.comm _cthe_block, 0x0000000c
```

File and Path Names

When the compiler encounters the Fortran INCLUDE statement, it takes the CHARACTER constant immediately following as a file name, searches for the file, and, if the file is found, copies its contents into the source file. If an absolute or relative path name is specified, the compiler will search only that path. If only a file name is given, the compiler will first look for the file in the current directory. It will then search any directory defined by the environment variable F77INCLUDES. Additional search paths may be specified with the -I compiler option.

Tab Character Size

The compiler assumes a standard tab size of eight spaces. This is the default for most editors. When the compiler encounters a tab character (ASCII 9) during compilation, it is replaced with the appropriate number of spaces for alignment to the next tab stop. By setting the environment variable TABSIZE, the tab size used by the compiler can be changed. The following command line for the Bourne shell will set the tab size for the compiler to four spaces:

```
TABSIZE=4
export TABSIZE
```

Runtime Environment

A number of the aspects of the runtime environment can be controlled with the `ABSOFRT_FLAGS` environment variable. This variable can be a combination of any of the following switches (the leading minus sign is required for each switch and multiple switches must be separated by one or more spaces):

-defaultcarriage

Causes the units preconnected to standard output to interpret carriage control characters as if they had been connected with `ACTION='PRINT'`.

-fileprompt

Causes the library to prompt the user for a filename when it implicitly opens a file as the result of I/O to an unconnected unit number. By default, the library creates a filename based on the unit number.

-vaxnames

Causes the library to use 'vax style' names (`FORnnn.DAT`) when creating a filename as the result of I/O to an unconnected unit number.

-unixnames

Causes the library to use 'unix style' names (`fort.nnn`) when creating a filename as the result of I/O to an unconnected unit number.

-bigendian

Causes the library to interpret all unformatted files using big endian byte ordering.

-littleendian

Causes the library to interpret all unformatted files using little endian byte ordering.

-noleadzero

Causes the library to suppress the printing of leading zeroes when processing an `Fw.d` edit descriptor. This only affects the limited number of cases where the ANSI standard makes printing of a leading zero implementation defined.

-reclen32

Causes the library to interpret the value specified for `RECL=` in an `OPEN` statement as 32-bit words instead of bytes.

-f90nlexts

Allows `f90 namelist` reads to accept non-standard syntax for array elements. Without this flag, the following input results in a runtime error:

```
$ONE  
A(1)=1,2,3,4  
$END
```

When `-f90nlexts` is set, the values are assigned to the first four elements of `A`.

-nounit9

Causes `UNIT 9` not to be preconnected to standard input and output.

-maceol

Formatted sequential files are in Classic Macintosh format where each record ends with a carriage return,

-doseol

Formatted sequential files are in Windows format where each record ends with a carriage return followed by a line feed.

-unixeol

Formatted sequential files are in Unix format where each record ends with a line feed.

-hex_uppercase

Data written with the `Z` edit descriptor will use upper case characters for `A-F`.

-strict_nan_input

Floating point input values for `NaN` and `Infinity` must be exactly the case insensitive strings `"nan"` and `"infinity"`.

-relaxed_input

Floating point input values must conform exactly to requirements of the Fortran standards.

Floating Point Math Control

This section describes the basic information needed to control the floating-point unit (FPU) built into Intel. The FPU provides a hardware implementation of the IEEE Standard For Binary Floating Point Arithmetic (ANSI/IEEE Std 754-1985). As a result it allows a large degree of program control over operating modes. There are two aspects of FPU operation that can affect the performance of a FORTRAN program:

Rounding direction

Exception handling

A single subroutine is provided with the compiler that is used to retrieve the current state of the floating-point unit or establish new control conditions:

```
CALL fpcontrol(cmd,arg)
```

where: *cmd* is an INTEGER variable that is set to 0 to retrieve the state of the floating point unit and 1 to set it to a new state.

arg is an INTEGER variable that receives the current state of the floating point unit if *cmd* is 0 and contains the new state if *cmd* is 1.

Rounding Direction

The first aspect of FPU operation that may affect a FORTRAN program is rounding direction. This refers to the way floating-point values are rounded after completion of a floating-point operation such as addition or multiplication. The four possibilities as defined in the `fev.inc` include file are:

FE_TONEAREST	round to nearest
FE_TOWARDZERO	round toward zero
FE_UPWARD	round toward +infinity
FE_DOWNWARD	round toward -infinity

Exception Handling

The second aspect of FPU operation that affects FORTRAN programs is the action taken when the FPU detects an error condition. These error conditions are called exceptions, and when one occurs the default action of the FPU is to supply an error value (either Infinity or NaN) and continue program execution. Alternatively, the FPU can be instructed to generate a floating point exception and a run time error when an exception takes place. This is known as *enabling the exception*. The five exceptions that can occur in a FORTRAN program are:

FE_INEXACT	inexact operation
FE_DIVBYZERO	divide-by-zero
FE_UNDERFLOW	underflow
FE_OVERFLOW	overflow
FE_INVALID	invalid argument

NOTE At the time this manual was written Macintosh OS X does not support floating exception handling.

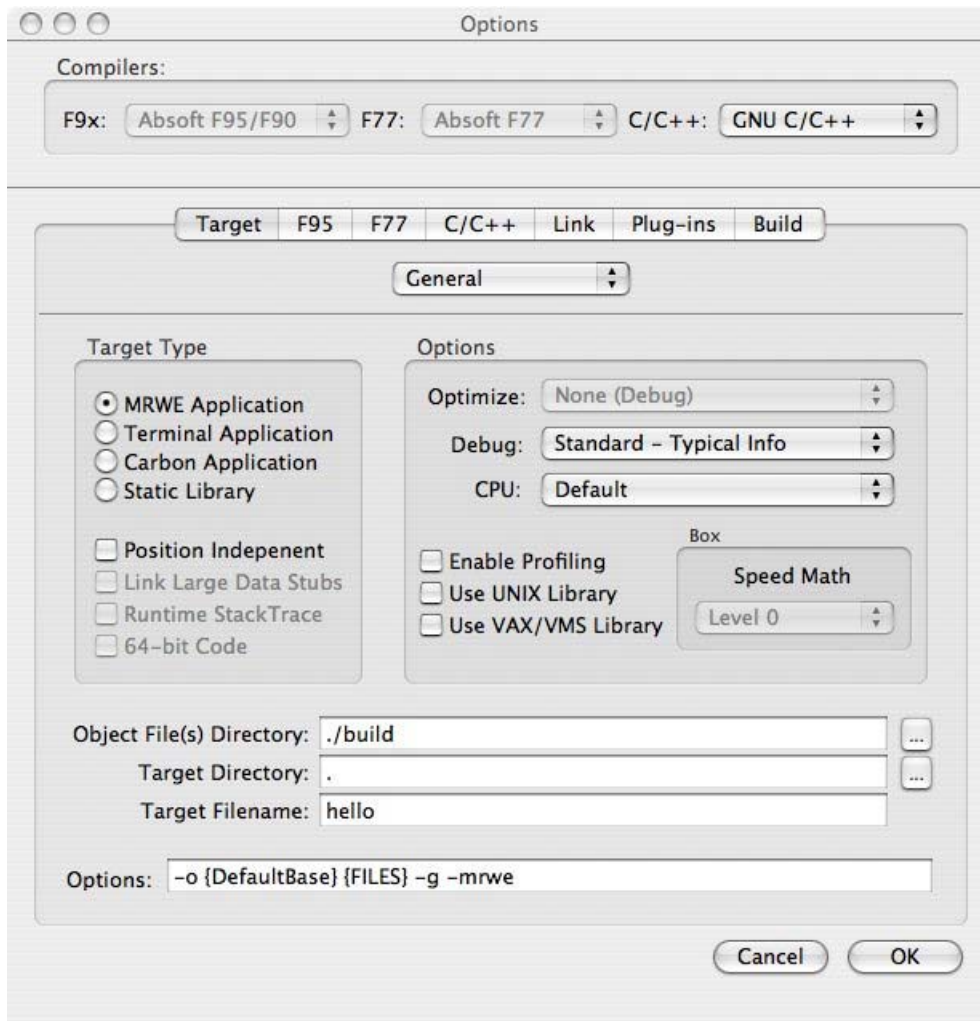
CHAPTER 6

The Macintosh Runtime Window Environment

This chapter discusses the Macintosh Runtime Window Environment (MRWE), a special feature of Absoft Pro Fortran which gives your Fortran 90/95 and FORTRAN programs a Macintosh interface with windows and menus. Usually, when you want to create applications with windows and menus, you need to know how to use the Macintosh Toolbox and the Apple user-interface guidelines. When you create an application using MRWE, your program will automatically exhibit these features without the need for intensive Macintosh programming. MRWE is convenient and flexible to use.

USING MRWE

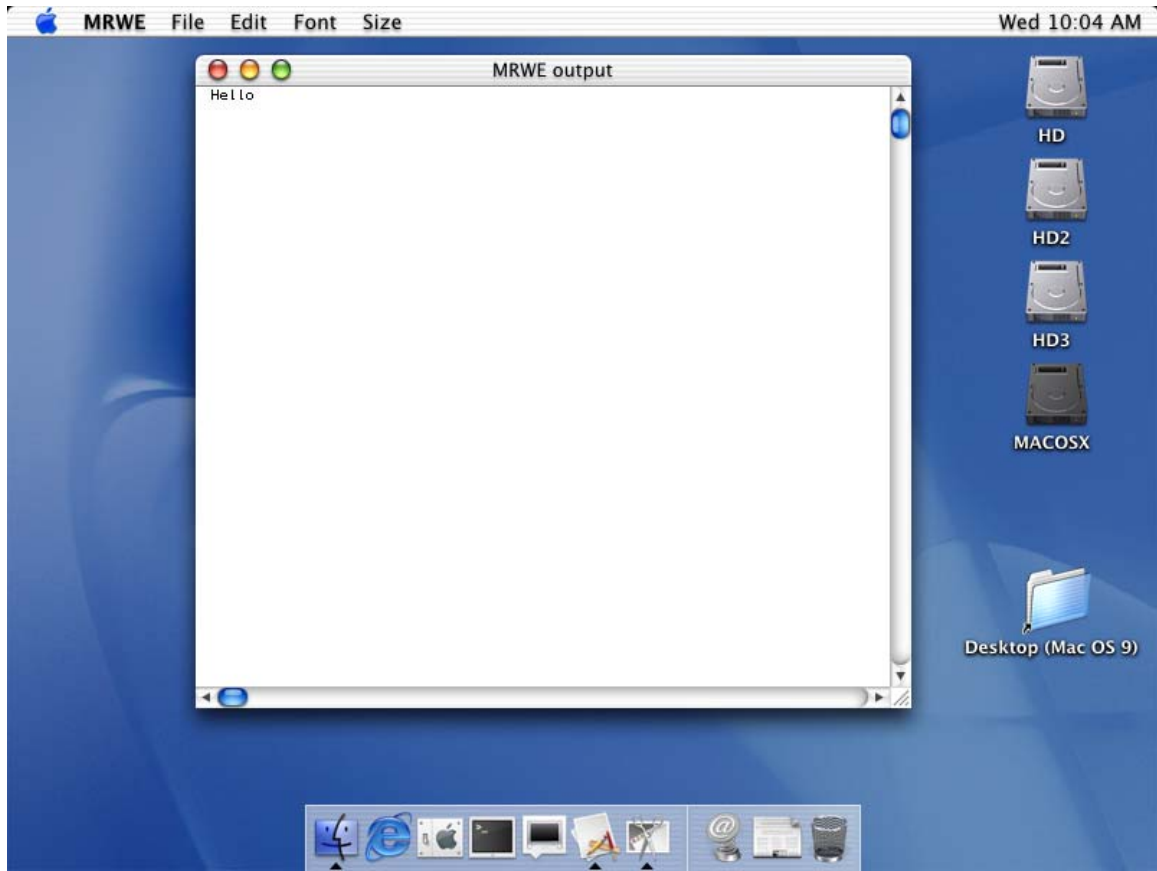
MRWE is a collection of pre-compiled Fortran routines contained in the library file `libmrwe.a` that you can link with your program. Applications generated by Absoft Pro Fortran are usually linked with this library unless explicitly select a different **Target Type**. The Fortran source code for MRWE is in the `FExamples` folder and can be used as an example of how to call Macintosh toolbox routines.



The Absoft Developer Tools Interface allows you to select the format of the executable file. An executable setting of MRWE Application (the default) will cause your program to be built as a stand-alone, double-clickable application with MRWE providing a window and menus interface. If your source code includes its own event loop or relies on the Carbon library even loop and calls Toolbox routines, you should disable MRWE by choosing Carbon Application for the target type.

The MRWE Window

When you launch a program that has been linked with the MRWE library, you will see a blank window on the screen and, above it, a menu bar, as shown below. This window is where MRWE displays standard output from Fortran programs and is where all standard input is typed. It looks like a terminal display, but allows you to see text that has scrolled off the screen.



The Macintosh Runtime Window Environment

Standard input and output are preconnected to the Fortran I/O units 5, 6, 9 and *. Any of these except * may be connected instead to a file by specifying the unit in an OPEN statement. After closing the file on that unit, the unit will be reconnected to standard input and output.

How Your Program And MRWE Work Together

When your program is launched, control is first passed to MRWE, which sets up a Macintosh environment with menus and a window. After the window appears, the application begins executing your Fortran program. By default MRWE is inactive until an I/O statement occurs. You cannot pull down menus or terminate the application at this point. Your Macintosh is completely dedicated to executing your Fortran program at maximum speed.

Working With Text in MRWE

The MRWE that appears is named after the application (truncated to 24 characters) with “ output” appended. Text can be entered from the keyboard.

You can only type into the window when a READ statement is currently active for standard input (i.e. one of the preconnected units), and you can only type on the last line

of the window. Any text that was already in the window when the READ statement began cannot be modified. You can, however, copy any text in the window to the clipboard. Also, when using the extended keyboard, the Home, End, Page Up, and Page Down keys can be used for scrolling. To insert an end-of-file character, type either Command-Return or Command-Enter.

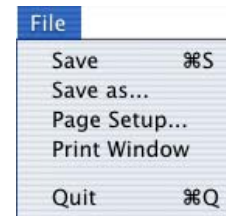
The MRWE window has a text limit of 30K. When this limit is reached, the oldest text in the window is automatically deleted to make room for new output. If you prefer, you can specify that MRWE save all text to a file, indicating your preference when the project is created or by later modifying the `MRWEPrefs.r` resource file.

Using The MRWE Default Menus

An MRWE application automatically has several menus built-in. Following is a description of the commands performed by each item within these default menus.

File Menu

The File menu contains commands for saving and printing the text in the window.



Save (⌘S)

This command saves the MRWE text to a text file. The window title is used for the file name, and if the file already exists, it is overwritten. The window title is the name of the application with the word “output” appended. To automatically save you can indicate your preference when the project is created or by later modifying the `MRWEPrefs.r` resource file..

Save As...

This command saves the window text to a text file. It displays a standard file dialog prompting for a file name in which to save the text. The default file name is the name of the application (truncated to 24 characters) with “output” appended. If the file already exists, you will be prompted to overwrite it.

Page Setup...

Choosing this command displays the standard page setup dialog box for the printer.

Print Window...(⌘P)

To print the contents of the window to the printer, use this menu item. A dialog box is displayed showing the printing status. When a block of text is selected, this menu item changes to **Print Selection...** and prints just the selected text rather than the entire window.

Quit (⌘Q)

This command stops execution of the application by executing a `STOP` statement, thus closing all open Fortran units.

Edit Menu

The Edit menu contains the standard editing commands for cutting, pasting, and copying text.



Undo (⌘Z)

This command is always disabled unless a desk accessory is being used.

Cut (⌘X)

This command removes the selected text in the window and places it on the Clipboard. Text on the Clipboard may be pasted into other applications. Like other editing commands, this command is only available during a `READ` statement, and any text that was already in the window before the `READ` statement began cannot be cut.

Copy (⌘C)

The Copy command places the selected text from the window onto the Clipboard and leaves the text of the window unchanged. Text on the Clipboard may be pasted into other applications.

Paste (⌘V)

This command replaces the selected text of the window with the text on the Clipboard. If no text is selected in the window, the Clipboard text is inserted at the insertion point. Like other editing commands, this command is only available during a `READ` statement, and any text that was already in the window before the `READ` statement began cannot be pasted into.

Clear

This command clears the selected text. Like other editing commands, this command is only available during a `READ` statement.

Font and Size Menus

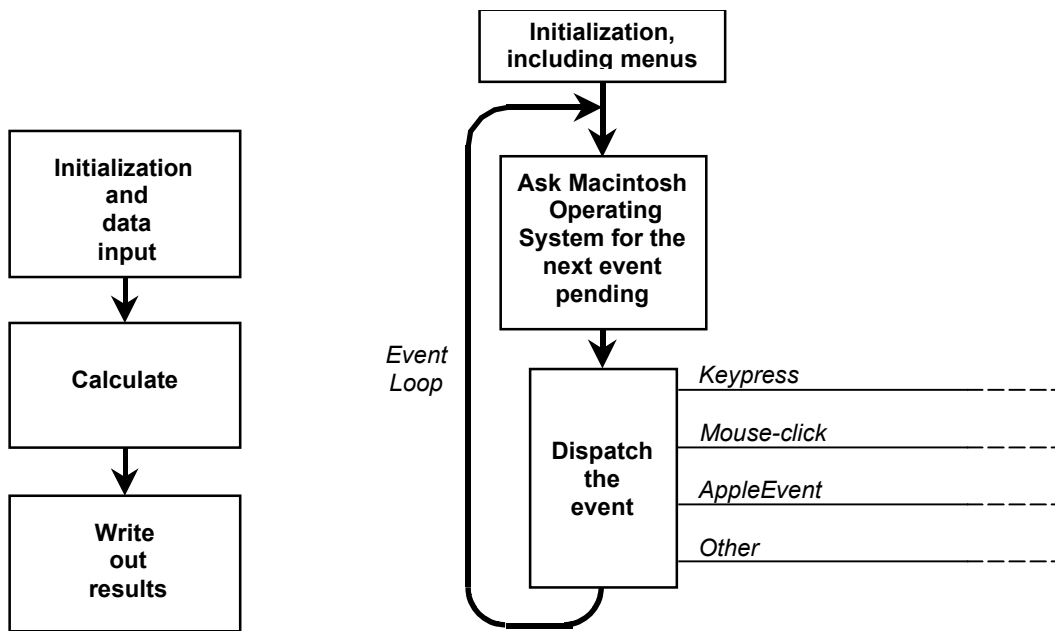
The font and the size menus list available fonts and font sizes. They allow you to change the appearance of text in a window to improve viewing. A checkmark beside an item in these menus shows the current setting for the frontmost window.

PROGRAMMING WITH MRWE

The sections that follow discuss features of MRWE that improve your program's user interface and allow it to communicate with other programs. Menus can be easily customized and messages can be sent to other programs. Before discussing these features, a better understanding of the differences between a typical Macintosh application and a typical Fortran program is necessary.

Program Organization: Fortran VS. Macintosh

Usually a program has the following structure: first, it *initializes* its variables, possibly reading data from files, then it *calculates*, then it *outputs* the results. Some of these steps may be repeated, but the basic order of execution is linear (see the figure below). Once a program begins, there is typically very little interaction with the user.



Typical Fortran Program

Typical Macintosh Program

The Macintosh, on the other hand, presents a very different, highly interactive environment to the user. The interface provided on a Macintosh between the program and the user allows for control and data selections to be made in a graphical manner as well as the more traditional textual methods. The Macintosh communicates user requests to the program through a mechanism known as an *event* that describes an action such as making a menu selection, clicking the mouse button, or typing a key on the keyboard. After the program receives an event it processes it, carrying out whatever action is required.

These actions are usually directed by an *event loop*, in which the program requests from the Macintosh any events pending for the program, carries out the directed action, and then repeats the process (see figure 8-5). When no events are pending, the program will carry out whatever other procedures it was designed to do. Some programs may simply

be idle at this stage if they are designed entirely to react to user input (such as a drawing program).

Unfortunately, there is much more to Macintosh programming than this simple description implies, and it would be necessary to add a great deal of programming to even the simplest Fortran program to add a Macintosh look and feel to it. MRWE can add most of these features to your program and provides facilities that allow you to add additional features without delving into the details of Macintosh programming.

MRWE Event Loop Operation

As the preceding discussion indicates, an event loop is a necessary component of a Macintosh program, and one is built into the MRWE routines. To use it, you must provide subprograms that this event loop can call to respond to user requests. This section will describe how to use this built-in event loop.

Just one call to the subroutine `mrwe_EventLoop` will begin this mechanism, making your program behave automatically in response to user actions.

```
EXTERNAL recur, term
CALL mrwe_EventLoop(recur, term)
```

where: *recur* is the name of a routine that will be called repeatedly while the event loop executes. This routine should execute quickly and return so that the program remains responsive to the user. Specify zero if you do not want to use a recurrent routine.

term is the name of a routine that you want to call when the program terminates. Specify zero if you do not want to use a termination routine.

When using these routines, remember to declare them as external in the subprogram in which `mrwe_EventLoop` is called. After calling `mrwe_EventLoop`, all events will be processed appropriately and, if a menu item is chosen, the subprogram you have associated with it will be called. Before making this call, you will probably want to set up menu items and tell MRWE which routine will respond to particular menu items. This process will be explained in the next section on **Customizing Menus**.

Note that `mrwe_EventLoop` never returns to the procedure from which it was called. It loops endlessly, processing the events sent by the user. Your program will only end when one of the following occurs: the user chooses to quit, one of your menu response routines issues a STOP statement, or there is a runtime error. [**Caution:** `mrwe_EventLoop` must not be called more than once.]

Customizing Menus

This section describes how to customize menus to suit your application. Menus and menu items can be added, deleted, or modified easily using MRWE.

Adding Menus

Menus can be added to your program at any time, but if you call `mrwe_EventLoop` you should probably install some menus and items first. This is because after calling `mrwe_EventLoop`, the user will control most of what the program does next. To install a menu item, call the function `mrwe_AddMenu` specifying the menu name, the item name, and the name of the *menu response routine* that should be called when the item is chosen:

```
INTEGER*4 mrwe_AddMenu
EXTERNAL Routine
itemID = mrwe_AddMenu(menuName, itemName, Routine)
```

where: *menuName* is a CHARACTER expression that specifies the menu that the item will appear in. If the menu does not already appear in the menu bar, it will be created and will appear to the right of the other menus.

itemName is a CHARACTER expression that specifies the name of the item to be added to the bottom of the menu. If the item already exists, then only the *Routine* can be changed.

Routine is the name of the menu response routine to be called when the item is chosen, and should be declared external in the routine from which `mrwe_AddMenu` is called.

itemID, the function result, is an INTEGER*4 value. If an error occurs, *itemID* is -1; otherwise, it is a value that uniquely identifies the menu item. The *itemID* is a composite of the menu number and item number. If an EQUIVALENCE is done of *itemID* to an array of two INTEGER*2 elements, the first element is the menu number and the second element is the item number.

Special Characters

Certain characters have special meanings in menu item names. If you specify any of the characters listed below when creating the item, you must include those characters in exactly the same order when referring to the item by name.

<u>Character</u>	<u>Meaning</u>
<	When followed by the letters B, I, U, O, or S, sets the style of a menu item to bold, italic, underline, outline, or shadow, respectively.
/	When followed by a character, invokes the item from the keyboard by pressing the command key and the character together.
(Disables the menu item. This is not recommended—instead, see “Enabling/Disabling Menu Items” later in this section.
!	When followed by a character, puts that character to the left of the item; also, see “Adding Checkmarks to Menus” later in this section.
-	Makes the item a line separator when it is the first character in the menu item name; typically used as (- to disable the separator.

Special Characters in Menu Item Names

Menus and the READ statement

The menu items that you install in the menu bar by calling `mrwe_AddMenu` will be available to a user of your program after your program calls `mrwe_EventLoop` and as soon as your menu response routines have finished responding to any menu items that have been chosen. The user will also be able to pull down the menus while your program is in the middle of executing a READ statement on standard input (units *, 5, 6, or 9), but the menu items will be disabled. This is because menu response routines must not execute Fortran I/O statements while the program is in the middle of a READ statement on Standard Input.

If you know that your menu response routine and every routine that it calls does not execute any Fortran I/O statements, you can enable the menu item even during a READ on Standard Input. To do this, add a @ character to the beginning of the name of the menu item when you install it with `mrwe_AddMenu`.

Removing a Menu or Menu Item

To remove a menu item or an entire menu, use the function `mrwe_RemoveMenu`:

```
INTEGER mrwe_RemoveMenu
ireturn = mrwe_RemoveMenu(itemID)
```

where: *itemID* is an `INTEGER*4` expression that specifies the item to be removed from the specified menu. It is the same value that was returned by `mrwe_AddMenu` when the item was first installed. If you specify 0 for the item component of *itemID*, the entire menu will be removed.

ireturn is an `INTEGER`. The function will return a value of 0 if successful, or -1 if there was an error.

Important: When a menu item is removed, any items that come after it will be given new item numbers. For example, in a menu with four items: if item #2 is removed, then item #3 becomes item #2, and item #4 becomes item #3. Any variable containing an *itemID* for an item after the one removed will no longer indicate the proper item. Note also that while item numbers are renumbered when an item is removed, menu numbers do not change when a menu is removed.

Menu Response Routines and `mrwe_DoMenu`

When you choose a menu item while running an MRWE application, the routine that performs that function (the “response routine”) is executed. This type of routine should be declared as:

```
INTEGER FUNCTION Routine(itemID)
INTEGER*4 itemID
```

where: *itemID* is the same value that was returned by `mrwe_AddMenu` when the item was first installed. This value can be helpful if more than one item uses the same response routine.

Your response routine is called by the function `mrwe_DoMenu`, which is normally called by `mrwe_DoMouseDown` whenever a mouse-down event occurs with the cursor in the menu bar at the top of the screen. The `mrwe_DoMenu` function can also be called explicitly at any time during the program when you want to execute the routine associated with a menu command. The `mrwe_DoMenu` routine will return one of the following two results: the result of the response routine, or -1 if the item could not be found. Call the `mrwe_DoMenu` function as follows:

```
INTEGER mrwe_DoMenu
ireturn = mrwe_DoMenu(itemID)
```

where: *itemID* is an `INTEGER` expression. It is the same value that was returned by `mrwe_AddMenu` when the item was first installed. It is a composite of the menu and item numbers.

irestult is the `INTEGER` returned by the response routine `mrwe_DoMenu` called. If you declare your response routines as functions, they can return a value through `mrwe_DoMenu`. If you declare them as subroutines, you can call `mrwe_DoMenu` as a subroutine.

Adding Checkmarks to Menus

Checkmarks can be placed beside a menu item after it is created by calling the routine `mrwe_MenuItemCheckmark`:

```
CALL mrwe_MenuItemCheckmark (itemID, state)
```

where: *itemID* is an `INTEGER*4` expression that specifies the menu item where the checkmark will be placed. It is the same value that was returned by `mrwe_AddMenu` when the menu item was installed.
state, is a `LOGICAL` expression. If *state* = `.TRUE.`, a checkmark will be placed next to the item; if *state* = `.FALSE.`, the checkmark will be removed.

Enabling/Disabling Menu Items

The function `mrwe_MenuItemEnable` is used to enable or disable a menu item:

```
INTEGER mrwe_MenuItemEnable  
irestult = mrwe_MenuItemEnable(itemID, state)
```

where: *itemID* is an `INTEGER*4` expression that specifies the menu item which will be enabled or disabled. It is the same value that was returned by `mrwe_AddMenu` when the item was first installed.
state is a `LOGICAL` expression. If the *state* = `.TRUE.`, the menu item will be enabled; if *state* = `.FALSE.`, the item will be disabled.

irestult is an `INTEGER`. The function will return a value of 0 if successful, or -1 if there was an error.

When installing an item by calling `mrwe_AddMenu`, you can also disable it by adding the special character (before the item name. However, this method will not work correctly with the `mrwe_MenuItemEnable` function and is *not recommended*. Instead, install the item as enabled, and then explicitly disable it using `mrwe_MenuItemEnable`.

Further Information About Menus

For further information on menu features, see *Inside Macintosh I*, “The Menu Manager”. To access these features, use the menu number and item number in the itemID returned by `mrwe_AddMenu`. If the routine expects a `MenuHandle`, this `INTEGER*4` value can be derived using the function `GetMHandle`, passing to it the menu number as a `VAL2()`:

```
RECORD /ItemID/ sdItem      ! defined in the file MRWE.inc
EXTERNAL DefaultSettings
INTEGER*4 h_menu

sdItem.both=mrwe_AddMenu('Settings','Use Defaults',DefaultSettings)
h_menu = GetMHandle(VAL2(sdItem.menu))
CALL SetItemMark(VAL4(h_menu), VAL2(sdItem.item),VAL1('•'))
```

Launching OTHER APPLICATIONS

From your Fortran program you can launch other Macintosh applications using the function `mrwe_LaunchApp` as defined below. You can also make an application that is already running become the front-most, active application. [Note: This function can only be used with System 7.]

```
INTEGER mrwe_LaunchApp
ireturn=mrwe_LaunchApp (name, front, useMin)
```

where: *name* is a CHARACTER expression that specifies the application to be launched and can be expressed in one of the following three ways:

- 1) `process=programName` Identifies the application to be launched; specify the path if necessary.
- 2) `creator=XXXX` Launches an application using its *creator type*. However, if there is more than one program with the same creator type, the system will launch the first one it finds.
- 3) `self` Allows an MRWE application to refer to itself. You can use this to bring your program to the front (see example below).

front is a LOGICAL expression that determines whether the specified application should be the frontmost application. If *front*=.TRUE., the application will become the frontmost application. If the application is not currently running, it will be launched in front of all other applications. If the application is already running, it will become the frontmost application. If *front*=.FALSE., the position of the application will remain unchanged. The menu bar at the top of the screen shows the menus of the frontmost application.

useMin is a LOGICAL expression that refers to the amount of memory needed to run the program. If *useMin*=.TRUE., then the program will be launched if the minimum amount of memory specified is available. If *useMin*=.FALSE., then the program is launched only if the preferred amount is available in the system. These values can be viewed and changed in the Finder by selecting the application's icon and choosing Get Info in the File menu. If the program is already running, *useMin* will be ignored.

irestult is an INTEGER error code indicating problems that may prevent launching the application. (See the table later in this chapter for a list of error codes.)

The following statement shows how to launch ResEdit:

```
CALL mrwe_LaunchApp('creator=RSED', .TRUE., .FALSE.)
```

ResEdit is specified by using its creator ID 'RSED'. The value of *front* is .TRUE., so the program is launched in front of all other running applications. Because *useMin* = .FALSE., ResEdit will only be launched if the preferred amount of memory is available.

The next example shows how to make your Fortran program the frontmost application:

```
CALL mrwe_LaunchApp('self', .TRUE., .FALSE.)
```

Apple Events

Apple Events is a System 7 feature that allows Macintosh applications to communicate with each other. By using this feature, you can have your applications *send messages to* and *receive messages from* other applications. MRWE allows you to use both the standard messages defined by Apple Computer and to define your own messages. This section will describe how these events (or messages) are classified, what information you must specify to transmit an event successfully, and examples of how to send and receive various events.

Apple Event Target

The application to which you send an Apple Event is called the *target* and is specified similarly to the way *name* is specified when launching an application (see “Launching Other Applications” above):

<u>Target Specification</u>	<u>Description</u>
<code>process=programName</code>	Sends the message to the application whose name is specified; include the path if necessary.
<code>creator=XXXX</code>	Sends the message to an application whose Creator ID is specified.
<code>sender</code>	Sends the message to the application that sent the last Apple Event received by your application.
<code>browser</code>	Shows the Browser dialog box, allowing the user at runtime to select any <i>currently running</i> application to send the Apple Event to.
<code>browser=prompt</code>	Shows the Browser dialog box with a custom prompt message instead of Choose a program to link to:.
<code>self</code>	Sends a message to itself.
<code>same</code>	Sends an event to the same target to which the most recent event was sent.

Target Specification for Apple Events

Apple Event Class and ID

The *class* and *ID* of an event indicate the action to be carried out—what the sender wants the target application to do. The *class* is a category of events that perform related functions. For example, Finder events make the Finder perform specific tasks such as restarting or shutting down the computer. The *class* and *eventID* arguments are specified with `CHARACTER*4` values. For Finder events, the class is indicated by the abbreviation FNDR. The event ID identifies the specific event being sent, such as “rest” for *restart*. (See the table later in this section for a list of common Apple Events).

Extra Information in an Apple Event

Some Apple Events require additional information in order to complete a message. For example, an Open Document or Print Document event must specify *which* document to act on; this is called extra information. Events can be classified according to the kind of extra information required, if any. The following table shows the three kinds of extra information supported by MRWE and how they are represented as `INTEGER` values.

<u>Kind</u>	<u>Value</u>	<u>Description</u>
signal	0	A signal event needs no extra information. It signals a request to do something or indicates that something has been done. For example, an application can tell another application to quit.
document	1	A document event operates on a document. The name of the document being operated on must be specified. For example, one application can tell another application to print a document.
text	2	A text event includes a character string. Two MRWE applications can pass information back and forth using text events.

Kind of Extra Information for an Apple Event

Typical Apple Events

Apple Computer has defined many Apple Events and has specified the class, event ID, and kind of extra information associated with each event. An application that supports the Apple Events feature can send or receive at least four basic events called Required Apple Events: *Open Application*, *Open Document*, *Print Document*, and *Quit Application*. MRWE supports some of the standard Apple Events and also allows you to define your own events. It allows you to send these events to other programs and to specify routines that will respond if one of these events is received by your application (see **Receiving Apple Events** later in this section). Required Apple Events is a subset of the Core class of Apple Events—the latter is identified by the abbreviation “aevt”.

Following is a table of common Apple Events, their class, event ID, and extra information specifications, and a brief description of what they do.

<u>Event name</u>	<u>Class</u>	<u>Event ID</u>	<u>Kind of extra info</u>	<u>What the event does</u>
Open Document	aevt	odoc	document	Tells the target application to open the specified document.†
Print Document	aevt	pdoc	document	Tells the target application to print the specified document.†
Open Application	aevt	oapp	signal	Tells the receiving application it has just been opened; usually causes an untitled document to appear.†
Quit Application	aevt	quit	signal	Tells the receiving application to begin the process of quitting.
Do Script	misc	dosc	text	Tells an application to perform actions specified in a scripting language.
Restart	FNDR	rest	signal	Tells the Macintosh to restart.*
Shutdown	FNDR	shut	signal	Tells the Macintosh to shut down.*
Sleep	FNDR	slep	signal	On a portable Macintosh, puts the computer in low-power mode.*
Empty Trash	FNDR	empt	signal	Tells the Finder to empty the trash.*
About	aevt	abou	signal	Tells the Finder to show its “About This Macintosh” window.*
Show Clipboard	FNDR	shcl	signal	Tells the Finder to show its clipboard window.*

† When an application is first launched, it will immediately receive one of these three events. Open Application is only sent when neither Open Document nor Print Document will be sent upon launch. When MRWE launches an application, it automatically sends an Open Application event.

* These Apple Events only work if sent to the Finder on the Macintosh running the application. Also, they all work the same as the corresponding Finder menu items.

Common Apple Event Specifications

Sending Apple Events

An Apple Event can be sent from one application to another as follows:

```
INTEGER mrwe_SendAE
ireturn=mrwe_SendAE(class,eventID,extraKind,target,extraInfo)
```

where: *class*, a CHARACTER*4 expression, is a category of related events, such as FNDR for Finder events.

eventID, a CHARACTER*4 expression, identifies which event to send, such as 'odoc' for an Open Document event or 'quit' for a Quit Application event.

extraKind, an INTEGER, specifies the kind of extra information, if any, that needs to be sent along with the event. Its value can be 0 for a signal event, 1 for a document event, or 2 for a text event.

target, a CHARACTER expression, identifies the application that the event is sent to.

extraInfo, a CHARACTER expression, is the extra information needed for certain kinds of Apple Events.

iresult is an INTEGER error code indicating problems that may occur while sending the event, such as an inability to send the event or the target application's inability to receive the event properly. (See the table for a list of error codes.)

For example, you could send an Apple Event from an MRWE application to any application which understands Apple Events, and tell it to print a certain text file. To have your program tell Microsoft Word™ to print the document called *results*, you would use the following statement:

```
ires = mrwe_SendAE('aevt','pdoc',1,'creator=MSWD','results')
```

It only takes one function call to send an event. The first argument identifies the target application (Microsoft Word), the second and third arguments define the class and event ID, and the fourth and fifth arguments indicate that the event is a document event and specify the name of the file.

Receiving Apple Events

MRWE provides a mechanism for receiving Apple Events, but it is up to you to determine how your program will react when it receives a event. When using Apple Events, you must specify routines that respond to particular events, just as you would specify a response routine when adding menu items (see “Customizing Menus” earlier in this chapter). For example, you may write one routine to respond to Open Document events and another to respond to Print Document events. You need to tell MRWE which events you want to respond to and the name of the routine that will handle the event. You do this by installing an Apple Event response routine using the function `mrwe_AddAppleEvent`.

```
LOGICAL mrwe_AddAppleEvent  
EXTERNAL Routine  
iresult=mrwe_AddAppleEvent(class,eventID,extraKind,Routine)
```

where: *class*, a CHARACTER*4 expression, identifies the category of the event to be received.

eventID, a CHARACTER*4 expression, identifies which event within the class will be received, such as 'odoc' for an Open Document event or 'quit' for a Quit Application event.

extraKind, an INTEGER, specifies the kind of extra information, if any, that will be received along with the event. Its value can be 0 for a signal event, 1 for a document event, or 2 for a text event.

Routine is the name of the response routine MRWE will call when the event is received, and should be declared external in the routine from which `mrwe_AddAppleEvent` is called.

lresult is a LOGICAL value indicating whether the response routine was installed. If *lresult*=.TRUE., the routine was installed properly. If *lresult*=.FALSE., the response was not installed because Apple Events are not supported by System 6.

For example, a routine that handles Print Document events could be installed with the following call, where `PrintDoc` is the name of a function that will be called every time the application receives a Print Document event.

```
lres = mrwe_AddAppleEvent('aevt', 'pdoc', 1, PrintDoc)
```

Response routines should be declared like this for *signal* events:

```
INTEGER FUNCTION Routine(class, eventID)
CHARACTER class*4, eventID*4
```

or like this for a response routine that responds to *document* or *text* Apple Events:

```
INTEGER FUNCTION Routine(class, eventID, extraInfo)
CHARACTER class*4, eventID*4, extraInfo*(*)
```

For a document event, *extraInfo* is the path and filename of the document to act on. For a text event, it is a text string. The INTEGER value that your function should return is an error code. It should return 0 if everything went well, or one of the error codes shown in error code table to indicate to the sender that a problem occurred while responding to the event. There is one response routine that is built into MRWE to handle Quit Application events, called `mrwe_QuitAE`. Normally, there should be no need to change this routine.

When your application is launched by the Finder under System 7, the Finder will send either an Open Application event, an Open Document event, or a Print Document event. The following table shows several methods you can use to launch an application and the corresponding Apple Event your application will receive:

<u>Method of launch</u>	<u>Apple Events received</u>
Double-click the application icon or single-click it and choose Open from the File menu.	An Open Application event.
Double-click a document icon.	An Open Document event, if successful. The creator of the document must match that of the application (Mrwe by default). Also, if there is more than one application with the same creator, the application that is launched may not be the one you expect.
Select the application icon and the icons of one or more documents, then double click on any selected icon or choose Open in the File menu.	An Open Document event for each document whose creator matches the application's creator.
Select the icons of one or more documents, then drag them to the application icon.	An Open Document event for each document.
Select the application icon and the icons of one or more documents, then choose Print in the File menu.	A Print Document event for each document whose creator matches the application's creator. Immediately after the Print Document events, a Quit Application event will be received.
Launch the application from another application.	Probably an Open Application event, although it depends on the application doing the launch.

Apple Events Received Upon Launch

Error Codes Returned from Apple Event Routines

Apple Event routines return an `INTEGER` error code. If the value returned is 0, no error occurred; if it is not 0, an error occurred either in sending the Apple Event, or in the target application receiving the event. Following is a list of error codes and their meanings:

<u>Error code</u>	<u>Meaning</u>
0	No error occurred in either sending or receiving the event.
-108	Not enough memory available to complete the action. If sending an Apple Event to an application that is not currently running, the preferred amount of memory is not available in which to launch it.
-128	The user of the application pressed Cancel in the Browser dialog.
-1701	The wrong kind was specified for an Apple Event, or the extra information in the event could not be extracted properly.
-1712	No response from the target application within the limit of seven seconds.
-1717	No handler installed to handle Apple Events of that class and eventID. Handlers should be installed <i>before</i> the call to <code>mrwe_EventLoop</code> is made.
-5553	The operating system does not support the requested feature. For example, System 6 does not support either Apple Events or using <code>mrwe_LaunchApp</code> to launch other applications.

Error Codes for Apple Events

Other Examples of Apple Events

Following are some additional examples of common events you might wish to transmit between applications:

Sending a request to the Finder

If you are running lengthy processes overnight or over the weekend, you may want to turn off your machine after all processes are completed. To do this, simply send the Finder a *Shut Down* Apple Event. This will turn off the computer if the Shut Down item in the Finder's Special menu can turn off the system. When sending an Apple Event to the Finder, you must properly specify the path of the System Folder that contains the Finder. For example, if your hard disk is named "HD", you could send a Shut Down event with the following statement:

```
ires = mrwe_SendAE('FNDR','shut',0,'process=HD:System Folder:Finder','')
```

Using other standard Apple Events

To cause another application (which supports Apple Events) to open a particular file, you would send that application an *Open Document* event specifying the file:

```
ires = mrwe_SendAE('aevt','odoc',1,'creator=MSWD','test')
```

To allow your application to respond to Open Document events, install a response routine to handle Open Document events:

```
ires = mrwe_AddAppleEvent('aevt','odoc',1,OpenDocProc)
```

Sending information between MRWE applications

One application could send a document event to another application telling it to process the file in a certain way. When the second application finishes, it can send a signal event back to the sender to indicate completion.

Another way to send messages is through text events. The *class* and *eventID* are not limited to the examples shown in the table earlier in this section. They can be defined as any four characters you choose to distinguish different events. Following is an example of how to send information through text events. The sending application would include:

```
CHARACTER*30 internalFile  
WRITE (internalFile,*) a,b,c  
ires=mrwe_SendAE('mrwe','guas',2,'process=SecondApp',internalFile)
```

and the receiving application would include:

```
EXTERNAL GuasProc  
.  
ires=mrwe_AddAppleEvent('mrwe','guas',2,GausProc)  
.  
END  
.  
INTEGER FUNCTION GausProc(class,type,internalFile)  
CHARACTER*4 class,type  
CHARACTER*(*) internalFile  
.  
READ (internalFile,*) a,b,c
```

Scripting

Scripting refers to text commands that an application can interpret and execute. An Apple Event called Do Script sends text commands from one application to another that has a scripting language, allowing the scriptable application to be controlled. It is not enough for an application to have a scripting language; it must also accept the Do Script Apple Event.

```
ires = mrwe_SendAE('misc','doscl',2,'creator=WILD','Answer "Press '
& 'OK to flash the screen." with "Flash" '//CHAR(13)//'Flash')
```

Further Information About Apple Events

MRWE supports sending and receiving only certain kinds of Apple Events. This is because the information carried in an Apple Event can be very diverse and complex. Some programs may use Apple Events to carry information in a form MRWE cannot recognize. To learn more about Apple Events, consult *Inside Macintosh VI*, “Apple Events Manager” (Chapter 6), and the *Apple Events Registry*, published by Apple Computer, Inc.

Creating Multiple Windows

You can easily open additional windows in MRWE. While the standard window described at the beginning of this chapter shows characters in the standard input and output units, additional windows can show characters written to other units. By specifying `ACCESS="window"` in an `OPEN` statement, you open a window connected to the unit specified in the `OPEN` statement. Any read or write functions associated with that unit will appear in a window. The `ACCESS` specifier can be expressed in the following three ways:

```
ACCESS="window"
```

```
ACCESS="window, height, width"
```

```
ACCESS="window, height, width, top_edge, left_edge"
```

where: *height* and *width* define the dimensions of the window in pixels (not including the title bar of the window). Both arguments should be expressed as positive integers.

top_edge and *left_edge* define the distance in pixels from the top of the screen to the top of the window and from the left edge of the screen to the left edge of the window, respectively. Both arguments are expressed as signed integers (either positive or negative).

The following is a simple example of how to use the `OPEN` statement to create an additional window in an MRWE application:

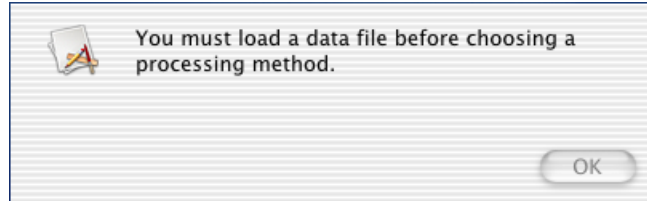
```
OPEN (7,FILE='Second Window',ACCESS='window, 200, 360')
WRITE (7,*) 'This text will appear in the second window'
READ (7,*)      ! This is like a PAUSE statement, but in the new window
CLOSE (7)
```

The first line creates a window titled 'Second Window' and connects it to unit 7. The size will be 200 pixels tall and 360 pixels wide. Since the location of the window is not specified, the window will open using the MRWE default for window location. The

second line writes the statement to the window. The `READ` statement in the third line will function like a `PAUSE` statement and wait for the user to press the Return or Enter key. When either key is pressed, the program will continue to the fourth line, which will close the window.

Showing Alert Messages

Macintosh applications often announce messages when a problem occurs by showing an Alert dialog box:



Show Alert Box Example

To show an Alert dialog box, use:

```
CALL mrwe_ShowAlert(button,message)
```

where: *button* is an `INTEGER` whose value indicates the contents of the button that dismisses the Alert.

message is a `CHARACTER` string containing the text of the message to be displayed.

Following are the possible values of the *button* argument and the corresponding buttons that will appear in the dialog box:

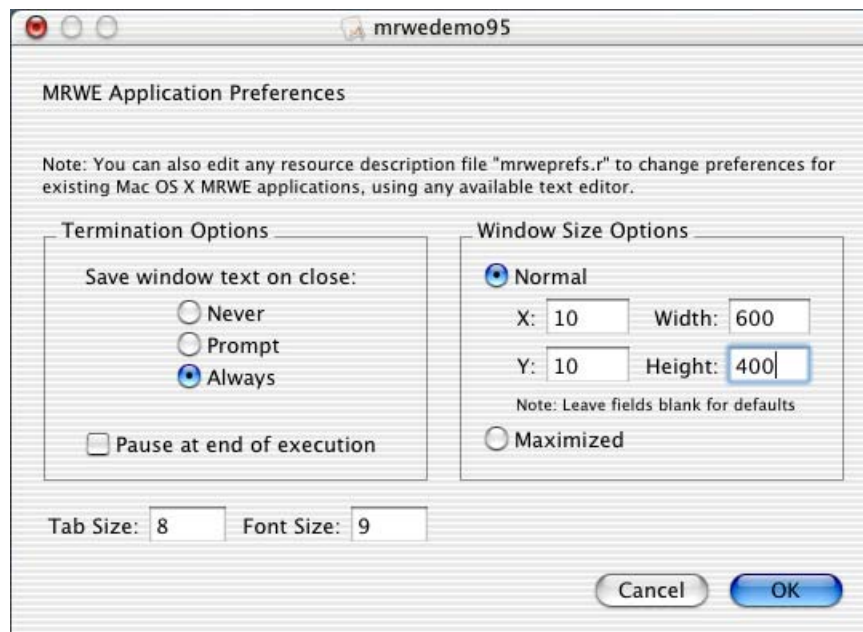
<u>Value</u>	<u>Button</u>
0	Ok
1	Cancel
2	Continue
3	Quit (causes the program to stop)
4	Ok (widens the dialog box by 15 percent)

“Show Alert” Buttons

SetMrwePrefs

The MRWE interface to your Fortran program can be easily customized to meet your needs by specifying the text characteristics and how the application behaves when it quits. The initial attributes of the MRWE interface are established when you create or modify the options of a project that includes MRWE. Additionally, the SetMrwePrefs program allows you to modify these characteristics after the application has been built.

SetMrwePrefs is located in the /Applications/Absoft10/bin. Use the finder to navigate to this folder and double click on the SetMrwePrefs application. Use the Open command in the file menu to select the program that you want to customize. The following dialog window will open:



SetPrefs Dialog Box
Figure 8-7

Below is a brief description of the options that are available to you within the SetPrefs Tool. Options are grouped according to the way they appear in the SetPrefs dialog box and are followed by the appropriate command line arguments.

Effects selected Application bundle, *.r file for *.rsrc file. To affect future applications built via the command line modify /Application/Absoft10/Rincludes/mrweprefs.r.

Termination Options

These options control the actions of the MRWE window before the application stops executing.

Pause at end of execution

Click the checkbox if you want MRWE to pause after the program ends, allowing you to read output in the window. By default, MRWE immediately quits.

Save window text on close

When MRWE quits, text in the window can be saved. If this option is never, text will not be saved. If this option is prompt, a dialog will ask whether text should be saved. To automatically save text without a prompt, use always.

Window Size Options

These options affect only the standard MRWE window, not windows opened with the ACCESS="window" specifier in an OPEN statement.

Window Size

When MRWE starts, this specifies the size of the window. If the `normal` is selected, the window is the default size. If `maximize` is selected, the window is maximized. You can specify explicit location and size if `normal` is selected.

Text characteristics

These options control the appearance of text characters in the window. You may prefer a text style different than the default for easier reading. Also, the text style of the window controls the text style when you print from MRWE.

Tab Size

This is the tab modulo size MRWE. If the value is greater than 20 or less than 0, the value is the default, 8. If the value is 0, tabs are passed as is to the application.

Font Size (-fsize *fontsize*)

Type in a font size in points or select one from the pop-up menu; the default is 9.

CHAPTER 7

Building Programs

This chapter covers the specifics of building Fortran 90/95 and FORTRAN 77 programs, including a discussion of the make facility. This chapter details the Absoft tools available for advanced programming and linking using the command line. The `Fsplit` utility is

also described. You use each tool on the command line – the syntax and a description of each command is given below.

The Components of an Application

Program code, system calls, library routines, and features of the Macintosh OS X operating system and interface are all important components of an application. Output from tools such as `amake` and `ld` are combined with your object code to create a Macintosh application.

Working with Resources

A resource is one of the most important concepts in Macintosh programming. A resource is a collection of information used by the Macintosh OS X operating system, such as menus, dialog definitions, or icons. These and other types of special information are stored in the executable image of a program file. The application itself may use some of the resources and other applications may use the resources for getting information about the application.

Resources are added to your program by the linker and are created using special tools and programs. Various dialog editors provide an interactive method of modifying existing resources or copying resources between files. The Macintosh program, `rez`, included with the Apple developer tools, is a resource compiler that creates new resources based on a textual resource description file.

CREATING OBJECT FILES

After you create and edit source files, or port files from other environments (see the chapter, **Porting Code**), these files are compiled using one or more of the Absoft compilers (described in the chapter, **Using the Compilers**).

The compiler is invoked by using one of the commands: `f95` or `f77` – these commands control both components of the compiler (front and back ends), the system assembler (`as`) and the linker (`ld`) (see the section below on **Linking Programs**). The features of the `f95` and `f77` commands simplify the process of creating finished applications, especially if you are working with a limited number of source files.

To initiate one of the Absoft compilers from the command line, follow these command syntax guidelines:

```
f95 [option...] [file...]  
f77 [option...] [file...]
```

where `option...` represents one or more of the compiler options described in the chapter, **Using the Compilers**. These options must begin with a dash (-); if more than one option is used, separate each option with a space. Also, some arguments appended to an individual option, such as a filename, may need to be separated from the option letter with a space — see the chapter, **Using the Compilers** for specific option rules.

When these commands are invoked on the command line, each *file* will be compiled to generate an executable application. By default, the resulting application will be given a name the `a.out`. To compile `hello.f` with the static local storage option, and generate an application named `welcome`, enter:

```
f77 -s -o welcome.exe hello.f
```

The option, `-o name`, specifies the name of the executable file overriding the default name of `a.out`. The name of the file must appear after the `-o` option as shown above. This option is passed directly to the linker; therefore, it has no effect when used in conjunction with the `-c` option. In this case, a space is required between the `-o` and `name`.

Remember that the `f77` and `f95` commands are used to control the compilation process. The actual compilers consist of the front-end (parsers and syntax analyzers) and the back-end (code generator).

If you need to create object files that are to be combined in a library, use the compiler commands with the `-c` option. This will suppress any linking functions and an executable file will not be created, as in the following example:

```
f95 -c Hello.f95 Goodbye.f95
```

The files are compiled into the object files `Hello.o` and `Goodbye.o`. After a source file has been compiled into an object file, it contains object code as well as any symbolic external references not known at compile time.

Since the linker is directly accessed in the `f77` and `f95` commands, any set of options may be passed directly to the linker. To do this, append the following option to the compiler command:

```
-X opts
```

The argument *opts* is a string enclosed in quotes to be passed to the linker. For example, `-X -v` will pass the `-v` option (display additional information) to the linker.

Fsplit - Source Code Splitting Utility

When you need to manage large files, work on small portions of Fortran code, or port code from other environments, you may want to split large, cumbersome source files into one procedure per file. This can be done using the `Fsplit` tool. The command syntax for the tool is shown below.

```
Fsplit [option...] [file...]
```

`Fsplit` splits FORTRAN source files into separate files with one procedure per file. The following command line will generate individual files for each procedure:

```
Fsplit largefile.f
```

A procedure includes block data, function, main, program, and subroutine program declarations. The procedure, `proc`, is put into file `proc.f` with the following exceptions:

- An unnamed main program is placed in `MAIN.f`.
- An unnamed block data subprogram is placed in a file named `blockdataNNN.f`, where `NNN` is a unique integer value for that file. An existing block data file with the same name will not be overwritten.
- Newly created procedures (non-block data) will replace files of the same name.
- File names are truncated to 14 characters.

Output files are placed into the directory in which the `fsplit` command was executed. The tab size is pulled from the environment variable `TABSIZE` if it exists, otherwise, a tab size of 8 is used. Options for the command are:

- v Verbose progress of `fsplit` is displayed on standard diagnostic.
- V Source files are in VAX FORTRAN Tab-Format.
- I Source files are in IBM VS FORTRAN Free-Form.
- 8 Source files are in Fortran 90 Free Source Form.
- W Source files are in wide format.

BUILDING PROGRAMS

It is often necessary in software development to maintain large numbers of files, many of which are dependent on other files in some way. It can become very difficult and time-consuming to manage these complex file relationships manually and to ensure that the appropriate files are updated when modifications are made to other related files. For example, when a source file is altered, it is necessary to recompile it in order to build or rebuild an updated object file and to link the object file with the appropriate auxiliary files (such as libraries) to form a complete and up-to-date executable file. It may also be necessary to use multiple languages and other programming resources during this process.

The Absoft `amake` utility allows you to automate much of this process of file maintenance by keeping a record of file dependencies according to rules that are either *built-in* to `amake` or *specified by the user*. (The `amake` utility is also referred to as "amake", the "make program", or the "make command" throughout this section.) Following these rules, the program determines whether any files need to be updated, and if so, rebuilds them automatically. If a file needs to be updated and does not exist, `amake` will create it based on the dependency rules for that file.

While `amake` is used primarily in software development, it can also be employed in other types of routine project management activities that involve file dependency relationships such as deleting temporary files, updating documents, or performing backups. In this

section, we will focus on the use of `amake` to maintain an up-to-date executable file during the course of a software project.

The major advantages of using `amake` in this type of environment are that it:

- saves considerable time and computing resources since only the files that need to be updated at a particular time are rebuilt;
- simplifies project management by performing many routine functions automatically and helping to coordinate the activities of projects involving multiple programmers; and
- frees programmers from the need to perform routine file maintenance activities manually.

This section discusses the operation of the Absoft `amake` program and explains how you can define your own rules to adapt the program to your particular environment. It also covers the topics of creating description files and macros, command usage and options, using environment variables, and handling errors. The level of presentation assumes a familiarity with programming and the process of developing software, but does not require any previous knowledge of the `amake` utility itself.

The Elements of `amake`

A key concept in understanding the operation of the `amake` program is that of *file dependency*. Files that are required to build (or rebuild) other files are referred to here as *prerequisite files* (or *prerequisites*). A file that is dependent on these prerequisites is called a *target file* (or *target*). For example, an object file (the target) is dependent on one or more source files (the prerequisites). The `amake` program searches through a *dependency tree* to establish the relationships between targets and prerequisites. If a prerequisite file has been updated more recently than its target file (or at exactly the same time), `amake` will (re)build the target file. [Note: The term *(re)build* is used in this section to indicate that a file will be *built* (created) if it does not exist, or *rebuilt* (updated) if it does exist.]

As mentioned above, the Absoft `amake` program operates based on rules that are: built-in to the program, specified by the user, or a combination of both. The program uses information from the following sources to determine whether a particular file needs to be (re)built and, if so, how this will be done:

- A description file supplied by the user that specifies:
 - (a) dependency relationships between targets and prerequisites,
 - and
 - (b) the commands needed to (re)build the target file.
- File names and the date/time each file was last modified.

- A set of default rules that define how files are (re)built based on the relationships between their suffixes.

Using Macros

Before discussing how a description file is created and used, it is necessary to have some understanding of how macros are used with `amake`. The term *macro*, as used here, refers to a symbol or character string that substitutes for something else, such as a set of commands. Macros are very useful in defining dependency relationships.

Advantages of using macros

The `amake` tool allows you to define macros easily, either within the description file itself, or as arguments on the `amake` command line. By using macros, you can:

- Represent recurring strings, such as file names or commands, in simplified form, reducing redundancy and thus, file size.
- Improve the consistency, readability, and maintainability of your description files.
- Allow for variation in the value of a macro from one (re)build to the next, and for values to be changed globally simply by redefining the corresponding macro.

Defining macros

A *macro definition* is made up of three basic elements: a name, followed by an equal sign, followed by a symbol or string that defines what the macro represents (in description files, usually a command string). You invoke a macro by placing a `$` symbol immediately before the name and enclosing the name in either parentheses `()` or braces `{ }`. [Exception: A name of only one character can be invoked without being enclosed in parentheses or braces.] By convention, macro names are written in uppercase characters, but any combination of upper or lower case letters or other non-reserved characters is acceptable. The following are examples of valid macro definitions and their corresponding invocations:

Macro Definition

```
DEBUGOPT = -g
SRCFILES = one.f two.f
OBJFILES = one.o two.o
ALLFILES = $(SRCFILES) $(OBJFILES)
```

Macro Invocation

```
$(DEBUGOPT)
$(SRCFILES)
$(OBJFILES)
$(ALLFILES)
```

The last example invokes the two previous macros within the definition, producing a list of the two FORTRAN source files and two object files as follows:

```
one.f two.f one.o two.o
```

The order of precedence for macro definitions is (from highest to lowest): the `make` command line, the description file, and the default definitions.

Special macros

The `make` utility includes a set of special-purpose macros that you may find useful in building your description files and rules. The most commonly-used are:

<u>Macro</u>	<u>Function</u>
<code>\$\$@</code>	Represents the <i>full name of the current target</i> —for use only on a (re)build command line. (When building a library it represents the name of the library.)
<code>\$\$*</code>	Represents the <i>base name of the current target</i> —for use only on a (re)build command line.
<code>\$\$<</code>	Represents a <i>current prerequisite</i> —for use only on a (re)build command line.
<code>\$\$\$@</code>	Represents the <i>base name of the current target</i> —for use only on a dependency line.
<code>\$\$?</code>	Represents a <i>list of prerequisites</i> that have been changed more recently than the current target—for use only on a (re)build command line.

Other special macros that are provided with Absoft `make` include:

<u>Macro</u>	<u>Function</u>
<code>MAKE</code>	Used for recursive makes—that is, when a <code>make</code> command is included as part of a description file.
<code>MAKEFLAGS</code>	Sets the command-line options available to <code>make</code> —usually defined as an environment variable (see Environment Variables later in this section).
<code>SUFFIXES</code>	Contains the default list of suffixes for the <code>.SUFFIXES</code> special target (see Special Targets later in this section).

Cautions in using macros

In addition to being aware of the order of precedence for macro definitions (see above) you should use caution in defining and using macros for the following reasons:

- A description file macro should be defined *before* the first time it is used in a dependency block.
- A macro should be defined only *once* within a description file.
- Macros may not be recursive—a macro may not directly or indirectly reference itself.

- If you reference an undefined macro, `amake` will assign it a null string and *no error message will be given*.
- While other characters are acceptable, it is advisable to use upper-case characters for macro names and to avoid characters that have special meanings in the operating system environment.

Using Description Files

The relationships between target files and their prerequisite files are specified in a *description file* which is called either `makefile` or `Makefile` by default (in that order). This file contains one or more *dependency blocks*, each consisting of the following elements:

- The target file name followed by a colon.
- The prerequisite file names (if any) following the colon.
- White space (a tab or spaces) followed by the commands needed to rebuild the target file.

[Note: Description files are also commonly referred to as *makefiles*. The term *description file* is used in this section for the sake of consistency.]

Working with dependency blocks

The general form of a dependency block is:

```
target: prerequisite1 prerequisite2...
      command(s) to (re)build target
```

For readability and ease of maintenance, we recommend that you:

- Place the target file name, colon, and prerequisite file name(s) on the first line and command(s) on the second line whenever possible; and
- Use a tab rather than spaces to precede commands.

For example, the first line of the following block:

```
test: a.f95 b.f95 libstat.a
      f95 -o test a.f95 b.f95 libstat.a
```

specifies that the target `test` is dependent on the prerequisites `a.f95`, `b.f95`, and `libstat.a`. If any of the three prerequisites have been updated at the same time or after the target, `test` will be rebuilt automatically using the command specified on the second line. The first line is referred to here as the *dependency line* and the second as the *(re)build command line*, or simply, the *command(s)*. [Note: The term *(re)build command line* in this context applies only to dependency blocks and should not be confused with the *make command line* discussed later in this section.]

If desired, the entire dependency block can be placed on one line by including a semicolon after the last prerequisite file name. The example above would look like:
test: a.f95 b.f95 libstat.a; f95 -o test a.f95 b.f95 libstat.a

If a line exceeds the maximum length allowed on your system, or you wish to shorten it and continue it onto the next line, you can use the continuation character for your environment. Using the example above, the backslash character (\) must be the last character on the first line as follows:

```
test: a.f95 b.f95 libstat.a; f95 -o test a.f95 \  
b.f95 libstat.a
```

Defining a target more than once

There may also be times when you will need to define the same target more than once within the same description file. This can be done using the *double-colon* feature of Absoft *amake*. This allows you to define two different sets of prerequisites (and the associated (re)build commands) for the same target. This feature is particularly useful in updating archive libraries. For example:

```
libgraph.a:: vertex.f95  
$(F95) -c -g -DDEBUG vertex.f95  
ar -r libgraph.a vertex.o  
rm vertex.o  
  
libgraph.a:: edge.f95  
$(F95) -c -O edge.f95  
ar -r libgraph.a edge.o  
rm edge.o
```

In this example, two different sets of commands are passed to the Fortran 90/95 compiler during the process of building the library `libgraph.a`.

Using include directives

An `include` directive can be used to include a text file within a description file. Such a text file could consist of macro definitions, dependency blocks, or any other components you would include as part of a description file. An `include` directive consists of the word `include`, left-justified, followed by one or more spaces or tabs, followed by the name of the file that is to be included at that point in the description file. For example:

```
include mymacros.txt
```

Included files are processed before the next line in the current description file. They can also be nested.

A sample description file

The following is an example of a simple description file:

```
# program name
NAME = util

# set FLAGS for command line
F95FLAGS = -g
LDFLAGS =

SRCS = util.f95 build.f95 parse.f95 tstring.f95
OBJS = util.o build.o parse.o tstring.o
PROG = $(NAME)

$(PROG): $(OBJS)
    $(F95) $(F95FLAGS) $(OBJS) -o $(PROG) $(LDFLAGS)

util.o: util.f95 util.inc tstring.inc decl.inc

build.o: build.f95 util.inc tstring.inc decl.inc

parse.o: parse.f95 util.inc tstring.inc decl.inc

tstring.o: tstring.f95 tstring.inc
```

Explanation:

- Lines beginning with a pound sign (#) are interpreted as comments.
- Lines containing an equal sign (=) are macro definitions; macros should be defined before they are used in a dependency block. (See **Defining macros** and **Cautions in using macros** earlier in this section).
- The lines containing a colon are dependency lines.
- Lines indented under dependency lines are (re)build commands.
- A dependency line and a set of (re)build commands together constitute a dependency block.

Although the order of these components may not affect the operation of `amake`, we suggest that you follow the format shown above in creating and maintaining your description files, that is: *macro definitions*, followed by *user-defined suffix rules*, followed by *dependency blocks*—with each definition, rule, or block separated by a blank line.

Using Dependency Rules

The `amake` utility uses a set of internal rules, commonly referred to as *dependency rules* or *suffix rules* to determine how to (re)build a particular target file. These rules determine file relationships based on filename suffixes. Absoft `amake` looks for dependency rules in two locations:

1. a default file that is automatically read by `amake`, and

2. your description file.

Rules specified in a description file *always* override the corresponding default rules.

The default rules

The default dependency rules (or suffix rules) automatically handle the common file transformations that `amake` performs, such as compiling source files to produce object files. Without these default rules, you would have to specify all file relationships in a description file; this would tend to become very complex and redundant in a large software development project. The default rules are located in:

```
/usr/absoft10/bin/default.mk.
```

The following is a list of the default dependency rules included with Absoft `amake` for Fortran 90/95 and FORTRAN 77 files. The macros shown within these rules are pre-defined in the `default.mk` file. [Note: The numbers on the left are not part of the rules and are included for reference only.]

Default Rules for Fortran 90/95 files

- (1) `.f90:`
 `$(F95) $(F95FLAGS) $(LD_FLAGS) -o $@ $<`
- (2) `.f90.o:`
 `$(F95) $(F95FLAGS) -c $*.f90`
- (3) `.f95:`
 `$(F95) $(F95FLAGS) $(LD_FLAGS) -o $@ $<`
- (4) `.f95.o:`
 `$(F95) $(F95FLAGS) -c $*.f95`

Explanation:

- (1) Compiles a Fortran 90 source file into an executable target.
- (2) Creates an object file from a Fortran 90 source file.
- (3) Compiles a Fortran 95 source file into an executable target.
- (4) Creates an object file from a Fortran 95 source file.

Default Rules for FORTRAN 77 files

- ```
(1) .f:
 $(F77) $(FFLAGS) $(LDFLAGS) -o $@ $<

(2) .f.o:
 $(F77) $(FFLAGS) -c $*.f

(1) .for:
 $(F77) $(FFLAGS) $(LDFLAGS) -o $@ $<

(2) .for.o:
 $(F77) $(FFLAGS) -c $*.for
```

Explanation:

- (1) Compiles a FORTRAN 77 source file into an executable target.
- (2) Creates an object file from a FORTRAN 77 source file.
- (3) Compiles a FORTRAN 77 source file into an executable target.
- (4) Creates an object file from a FORTRAN 77 source file.

### Creating your own rules

In general, it is best to rely on the default dependency rules as much as possible. There will be times, however, when you may need to modify the behavior of `make` by creating your own dependency rules. There are two possible ways to do this:

- Include dependency rules in your description file, or
- Modify the file of default rules by adding your own rule(s), or deleting/changing existing rule(s).

We recommend that you use the first alternative if possible, and avoid modifying the default rules unless absolutely necessary. Since rules in a description file *always* override any corresponding default rules, the first alternative should be sufficient for virtually any circumstance. [Caution: Unless you are replacing an existing default rule, it is advisable to avoid using suffixes that are pre-defined in `make` to avoid conflicts with the default rules.]

The following is an example of a user-specified dependency rule included in the description file discussed earlier in this section:

```
program name
NAME = util

set FLAGS for command line
F95FLAGS = -g
LDFLAGS =

SRCS = util.f95 build.f95 parse.f95 tstring.f95
OBJS = util.o build.o parse.o tstring.o
PROG = $(NAME)

.f95.o:
 $(F95) $(F95FLAGS) /c $<
 cp $< /home/usr/workdir

$(PROG): $(OBJS)
 $(F95) $(F95FLAGS) $(OBJS) /o $(PROG) $(LDFLAGS)

util.o: util.f95 util.inc tstring.inc decl.inc
build.o: build.f95 util.inc tstring.inc decl.inc
parse.o: parse.f95 util.inc tstring.inc decl.inc
tstring.o: tstring.f95 tstring.inc
```

The user-supplied rule:

```
.f95.o:
 $(F95) $(F95FLAGS) -c $<
 cp $< /home/usr/workdir
```

will override the corresponding default rule in the `default.mk` file:

```
.f95.o:
 $(F95) $(F95FLAGS) -c $<
```

Rather than following the default rule for creating an object file from a Fortran 90/95 source file, the new suffix rule will override the default to invoke the Fortran compiler and copy the resulting object file to the working directory. (The default rule *only* invokes the Fortran 90/95 compiler.)

### amake Usage and Syntax

The `amake` command accepts options, description file names, macro definitions, and target file names as arguments in the form:

```
make [options] [description file] [macros] [target(s)]
```

Arguments specified on the `amake` command line override any corresponding definitions found in a description file or in the default dependency rules.

`make` command-line options are specified with a dash (-):

- d Lists the prerequisites for each dependency block that caused `make` to rebuild a target. All prerequisites that are newer than the target are displayed. Useful for determining why certain (re)build commands are executed.
- D Displays the contents of a description file as it is read by the `make` program.
- e Causes environment variables to override macros defined in a description file. By default, user-defined macros override environment variables (see **Environment Variables** below).
- f Takes an argument in the form *filename* which specifies the name of a description file to be used in place of the default name `makefile`. A file name consisting of a dash (-) uses the standard input rather than *filename* as input. If there are no `-f` arguments, the program will search (by default) for a file named `makefile` or `Makefile` in the current directory.
- i Ignores error codes returned by commands. This is equivalent to using the `.IGNORE` special target in a description file (see **Special Targets** below). Useful in situations when it is not necessary that certain commands execute successfully.
- k This option stops processing on the current entry when an error occurs, but continues processing on other branches of the dependency tree that do not depend on the current entry.
- n Displays all commands, but does not execute them. (Command lines beginning with an `@` character are also displayed.) Useful in debugging/testing description files.
- p Prints a complete list of macro definitions, dependency blocks, and suffix rules.
- q Returns a zero or nonzero status code depending on whether the target is or is not up-to-date, respectively. Useful when `make` is called from a script or tool that requires the current target.
- r Does not use the default rules (i.e., does not read in the `default.mk` file). Useful for situations where you want to completely isolate the environment in which `make` operates.
- s Does not print command lines before executing. This is equivalent to using the `.SILENT` special target in a description file.

- t *Touches* the target files (assigning them the current date/time) without executing the commands to (re)build them. Used to bypass the (re)build process for particular targets—should be used with caution.

Any command-line arguments other than options, description file names, or macros are assumed to be the names of targets to be (re)built; these are evaluated in left-to-right order. If there are no such arguments, the first target in the description file whose name does not begin with a period is rebuilt (see below).

### Special Targets

In addition to the options listed above, the following *special targets* can be used in a dependency block (rule) to further customize the behavior of `amake`:

|           |                                                                                                                                                                                                                                                 |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .DEFAULT  | Used when there is no target name specified or default rule for building a target file. A set of pre-defined commands are invoked by the .DEFAULT target.                                                                                       |
| .DONE     | This target and its prerequisites are processed after all other targets have been (re)built.                                                                                                                                                    |
| .IGNORE   | Ignores all error codes; equivalent to the <code>-i</code> option on the make command line.                                                                                                                                                     |
| .INIT     | This target and its prerequisites are processed before any other targets are (re)built.                                                                                                                                                         |
| .SILENT   | Executes commands, but does not send them to the standard output; equivalent to the <code>-s</code> option on the make command line.                                                                                                            |
| .SUFFIXES | Used to add dependency rules to the default rules (specify .SUFFIXES as the target followed by the suffixes to be added as the prerequisites), or to delete the default rules entirely (specify .SUFFIXES as the target without prerequisites). |

### Dummy Files

There may be times when you will want to run `amake` without actually (re)building a target or when you need to force a target to be (re)built regardless of when the last modification was made to a prerequisite. You can do this by using *dummy files*—i.e., specifying one or more filenames in your description file that do not represent an actual file, but that cause the behavior of `amake` to change. Often, this can be used to bypass the established dependency tree and force `amake` to behave in a desired manner.

The most common type of dummy filename is a *dummy target*. For example:

```
clobber :
 rm *.o
```

will execute the commands on the second line without (re)building any files.

### Environment Variables

Each time you run `amake`, the environment variables that exist at that time are read and added to the existing macro definitions. Essentially, environment variables are handled in the same manner as macros by `amake`. As briefly described earlier in this section, the `MAKEFLAGS` variable (also sometimes referred to as the `MAKEFLAGS` macro) defines the command-line options available to `amake` and is usually defined as an environment variable; the `MAKEFLAGS` environment variable is read and processed prior to any options specified on the `amake` command line.

When you run `amake`, the following order of precedence is followed (from highest to lowest priority):

1. command-line arguments
2. description file entries (definitions)
3. environment variables
4. default dependency rules

If you invoke the `/e` command-line option, priority levels 2 and 3 are reversed so that the order of precedence becomes:

1. command-line arguments
2. environment variables
3. description file entries (definitions)
4. default dependency rules

### Example: Rebuilding an Executable File

Generally, in a software development environment, you would run the `amake` utility whenever there is a need to update an executable file, such as after changes have been made to source files or libraries. To summarize the operation of `amake`, the program:

1. Searches for a description file called `makefile` (or, if that name does not exist, `Makefile`) by default, or another name assigned through the `-f` option.



2. Checks dependencies in a bottom-up manner, establishing relationships between targets and their prerequisites and building a dependency tree in the process.
3. (Re)builds target files when they are out-of-date with respect to their prerequisites according to commands specified in the description file, the default rules, or both.

Using our sample description file, `amake` will: read in the macro definitions, check the syntax of all entries, and (re)build the executable file `util` based on the `.f95.o` suffix rule and the dependency blocks (lines) following it:

```
program name
NAME = util

set FLAGS for command line
F95FLAGS = -g
LDLFLAGS =

SRCS = util.f95 build.f95 parse.f95 tstring.f95
OBJS = util.o build.o parse.o tstring.o
PROG = $(NAME)

.f95.o:
 $(F95) $(F95FLAGS) /c $<
 cp $< /home/usr/workdir

$(PROG): $(OBJS)
 $(F95) $(F95FLAGS) $(OBJS) /o $(PROG) $(LDLFLAGS)

util.o: util.f95 util.inc tstring.inc decl.inc
build.o: build.f95 util.inc tstring.inc decl.inc
parse.o: parse.f95 util.inc tstring.inc decl.inc
tstring.o: tstring.f95 tstring.inc
```

### **Error Handling and Cautions**

The following is a list of common errors you may encounter while using `amake` and possible reasons for their occurrence.

#### **Syntax Errors**

##### Error Message

##### Explanation

Badly formed macro

Incorrect syntax for a macro definition—often, the macro name is missing.

Improper macro

An error occurred during macro expansion. Often, the problem is a missing parenthesis or bracket.

|                            |                                                                                                                       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Macro too long ...         | A macro name is too long; cannot be longer than 100 characters.                                                       |
| Rules must be after target | Occurs when a line beginning with a space or tab has been encountered before a dependency line in a description file. |

### Other Common Errors

#### Error Message

#### Explanation

|                                   |                                                                                                                                                                              |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Cannot open file                  | The description file specified in an <code>include</code> directive could not be found or was not accessible. (See <b>Using include directives</b> earlier in this section.) |
| Don't know how to make target     | There is no target entry in a description file, none of the default rules apply, and there is no <code>.DEFAULT</code> rule.                                                 |
| Too many options                  | The <code>amake</code> program has exceeded the allocated space while processing command-line options or a target list.                                                      |
| Too many rules defined for target | Multiple sets of rules have been defined for a target; targets may only have one set of rules.                                                                               |
| Unexpected end of line seen       | The colon in a dependency line is missing.                                                                                                                                   |

### Cautions

In addition to handling the errors described above, particular caution should be exercised as follows when running `amake`:

- Use of the `-t` (touch) or `-i` (ignore errors) options can be destructive in the way that they override the normal behavior of `amake` (see **amake Usage and Syntax** earlier in this section). These options should be used with great care and, if possible, tested first before being used with actual files. The `-t` option, in particular, can save considerable time by "updating" files without (re)building them, but it erases the file relationships that would normally be established.
- Unforeseen problems can arise by changing default rules or variables, such as the `MAKEFLAGS` environment variable. It is best not to change these default values but, if this must be done, caution is advisable.
- Caution should be used when defining and using macros, especially when macros are to be invoked recursively and when using any of the special pre-defined macros described earlier in this section.



## CHAPTER 8

### Interfacing With Other Languages

---

This chapter discusses interfacing Absoft Pro Fortran with the C Programming Language and assembly language, debugging programs, and profiling executables. Although Fortran programs can call C functions easily with just a `CALL` statement, the sections below should be read carefully to understand the differences between argument and data types.

#### INTERFACING WITH C

Absoft Pro Fortran is designed to be fully compatible with the implementation of the standard C Programming Language provided on Macintosh OS X. The linker can be used to freely link C modules with Fortran main programs and vice versa. However, some precautions must be taken to ensure proper interfacing. Data types in arguments and results must be equivalent. The case of global symbols C is significant. The symbolic names of external procedure must match in case and decoration.

#### Fortran Data Types in C

| Fortran                                             | C                                                                 |
|-----------------------------------------------------|-------------------------------------------------------------------|
| LOGICAL*1 l<br>LOGICAL*2 m<br>LOGICAL*4 n           | unsigned char l;<br>unsigned short m;<br>unsigned long n;         |
| CHARACTER*n c                                       | char c[n];                                                        |
| INTEGER*1 i or BYTE i<br>INTEGER*2 j<br>INTEGER*4 k | char i;<br>short j;<br>int k;<br>long k <sup>1</sup> ;            |
| INTEGER*8 l                                         | long long l;                                                      |
| REAL*4 a<br>REAL*8 d                                | float a;<br>double d;                                             |
| COMPLEX*8 c                                         | struct complx {<br>float x;<br>float y;<br>};<br>struct complx c; |
| COMPLEX*16 d                                        | struct dcomp {<br>double x;<br>double y;<br>};<br>struct dcomp d; |

1. On 64-bit systems, `long` is equivalent to `INTEGER*8`.

The storage allocated by the C language declarations will be identical to the storage allocated by the corresponding Fortran declaration.

There are additional precautions when passing Fortran strings to C routines. See the section **Passing Strings to C** later in this chapter for more information.

### Related Compiler Options

Symbols in C programs are case sensitive and are often written entirely in lower case. Occasionally, C functions designed to be used with Fortran will also have a trailing underscore added to the function names. Certain compiler options are helpful when combining procedures written in both Fortran and C.

FORTTRAN 77 code should be compiled with the following options:

|             |                                             |
|-------------|---------------------------------------------|
| <b>-f</b>   | fold symbols to lower case                  |
| <b>-s</b>   | use static storage                          |
| <b>-N15</b> | append trailing underscores to global names |

Fortran 90 code should be compiled with the following options:

|                        |                                             |
|------------------------|---------------------------------------------|
| <b>-YEXT_NAMES=LCS</b> | fold symbols to lower case                  |
| <b>-s</b>              | use static storage                          |
| <b>-YEXT_SFX=_</b>     | append trailing underscores to global names |

C code does not have to be compiled with any special options for the C compiler.

### Rules for Linking

When linking Fortran and C programs, the `f77` or `f90` compiler driver should be used so that the appropriate Fortran and C libraries are included in the final application. The following command will compile the file `f1.f` with the FORTRAN 77 compiler and the file `c1.c` with the C compiler. It will then link the two resulting object files along with `o1.o` and the appropriate libraries to generate an executable application named `exec`:

```
f77 -o exec f1.f c1.c o1.o
```

### Passing Parameters Between C and Fortran

The Absoft Pro Fortran compilers use the same calling conventions as the C programming language. Therefore, a Fortran routine may be called from C without being declared in the C program and vice versa, if the routine returns all results in parameters. Otherwise, the function must be typed compatibly in both program units. In addition, care

must be taken to pass compatible parameter types between the languages. Refer to the table earlier in this chapter.

### Reference parameters

By default, all Fortran arguments to routines are passed by reference, which means pointers to the data are passed, not the actual data. Therefore, when calling a Fortran procedure from C, pointers to arguments must be passed rather than values. Both integer and floating point values may be passed by reference. Consider the following example:

```
SUBROUTINE SUB(a_dummy,i_dummy)
REAL*4 a_dummy
INTEGER*4 i_dummy

WRITE (*,*) 'The arguments are ',a_dummy, ' and ', i_dummy
RETURN
END
```

The above subroutine is called from Fortran using the `CALL` statement:

```
a_actual = 3.3
i_actual = 9
CALL SUB(a_actual, i_actual)
END
```

However, to call the subroutine from C, the function reference must explicitly pass pointers to the actual parameters as follows:

```
int main()
{
 float a_actual;
 int i_actual;
 void SUB();

 a_actual = 3.3;
 i_actual = 9;
 SUB(&a_actual,&i_actual);
 return 0;
}
```

Note that the values of the actual parameters may then be changed in the Fortran subroutine with an assignment statement or an I/O statement.

When calling a C function from Fortran with a reference parameter, the C parameters are declared as pointers to the data type and the Fortran parameters are passed normally:

```
PROGRAM convert_to_radians
WRITE (*,*) 'Enter degrees:'
READ (*,*) c
CALL C_RAD(c)
WRITE (*,*) 'Equal to ',c,' radians'
END
```

---

```
void C_RAD(c)
float *c;
{
 float deg_to_rad = 3.14159/180.0;
 *c = *c * deg_to_rad;
}
```

### Value parameters

Absoft Pro Fortran provides the intrinsic function `%VAL()` for passing value parameters. Function interfaces may also be used to specify which arguments to pass by value. Although it is generally pointless to pass a value directly to a Fortran procedure, these functions may be used to pass a value to a C function. The following is an example of passing a 4-byte integer:

```
WRITE (*,*) 'Enter an integer:'
READ (*,*) i
CALL C_FUN(VAL(i))
END
```

---

```
void C_FUN(i)
int i;
{
 printf ("%d is ",i);
 if (i % 2 == 0)
 printf ("even.\n");
 else
 printf ("odd.\n");
}
```

The value of `i` will be passed directly to `C_FUN`, and will be left unaltered upon return. Value parameters can be passed from C to Fortran with use of the `VALUE` statement. The arguments that are passed by value are simply declared as `VALUE`.

```
void C_FUN()
{
 void FORTRAN_SUB();
 int i;

 FORTRAN_SUB(i);
}
```

---

```
SUBROUTINE FORTRAN_SUB(i)
VALUE i
...
END
```

Note that C will pass all floating-point data as double precision by default, and that the only Fortran data type that cannot be passed by value is `CHARACTER`.



## Array Parameters

One-dimensional arrays can be passed freely back and forth as both language implementations pass arrays by reference. However, since C and Fortran use different row/column ordering, multi-dimensional arrays cannot be easily passed and indexed between the languages.

```
INTEGER ia(10)

CALL C_FUN(ia)
WRITE (*,*) ia

END
```

---

```
void C_FUN(i)
int i[];
{
int j;
 for(i=0; j<10; j++)
 i[j]=j;
}
```

## Function Results

In order to obtain function results in Fortran from C language functions and vice versa, the functions must be typed equivalently in both languages: either `INTEGER`, `REAL`, or `DOUBLE PRECISION`. All other data types must be returned in reference parameters. The following are examples of the passing of function results between Fortran and C. The names are case-sensitive, so trying to call `cmax`, for example, will result in an error at link time.

### A call to C from Fortran

```
PROGRAM callc
INTEGER*4 CMAX, A, B

WRITE (*,*) 'Enter two numbers:'
READ (*,*) A, B
WRITE (*,*) 'The largest of', A, ' and', B, ' is ', CMAX(A,B)
END
```

---

```
int CMAX (x,y)
int *x,*y;
{
 return((*x >= *y) ? *x : *y);
}
```

### A call to Fortran from C

```
main()
{
float QT_TO_LITERS(), qt;

 printf ("Enter number of quarts:\n");
 scanf ("%f",&qt);
 printf("%f quarts = %f liters.\n", qt, QT_TO_LITERS(&qt));
}
```

---

```
REAL*4 FUNCTION QT_TO_LITERS(q)
REAL*4 q;

QT_TO_LITERS = q * 0.9461;
END
```

### Passing Strings to C

Fortran strings are a sequence of characters padded with blanks out to their full fixed length, while strings in C are a sequence of characters terminated by a null character. Therefore, when passing Fortran strings to C routines, you should terminate them with a null character. The following Fortran expression will properly pass the Fortran string to the C routine CPRINT:

```
PROGRAM cstringcall
character*255 string
string = 'Moscow on the Hudson'
CALL CPRINT(TRIM(string)//CHAR(0))
END
```

---

```
void CPRINT (anystring)
char *anystring;
{
 printf ("%s\n",anystring);
}
```

This example will neatly output “Moscow on the Hudson”. If the TRIM function were not used, the same string would be printed, but followed by 235 blanks. If the CHAR(0) function was omitted, C would print characters until a null character was encountered, whenever that might be.

You can also take advantage of the string length arguments that Fortran passes. After the end of the formal argument list, Fortran passes (and expects) the length of each CHARACTER argument as a 32-bit integer value parameter. For example:

```
SUBROUTINE FPRINT(string)
character*(*) string
print *, string
END
```

---

```
#include <string.h>

int main()
{
char string[] = {"Moscow on the Hudson"};
void FPRINT(char *, int);

 FPRINT(string, strlen(string));
 return 0;
}
```

### Calling Fortran math routines

All of the Fortran intrinsic math functions which return values recognized by the C Programming Language can be called directly from C as long as the Fortran run time library, `libf77math.a`, is linked to the application.

Taking the intrinsic function names in lower case and adding two underscores to the beginning forms the names of the functions that can be called.

The following example calls the Fortran intrinsic function `SIN` directly from C:

```
main()
{
float sin_of_a, a, __sin();

 a = 3.1415926/6;
 sin_of_a = __sin(a);
}
```

### Naming Conventions

Global names in FORTRAN include procedure names and COMMON block names, both of which are significant to 31 characters. All global names can be case sensitive, meaning the compiler recognizes the difference between upper and lower case characters. In the FORTRAN 77 compiler, use of the `-f` option will fold global names to lower case, while the `-N109` option will fold global names to upper case. . In the Fortran 95 compiler, use of the `-YEXT_NAMES=LCS` option will fold global names to lower case, while the `-YEXT_NAMES=UCS` option will fold global names to upper case. All other symbols in FORTRAN are manipulated as addresses or offsets from local labels and are invisible to the linker.

### Accessing COMMON blocks from C

COMMON block names are global symbols formed in Absoft Pro Fortran by prepending the characters “\_c” to the name of the COMMON block. The elements of the COMMON block can be accessed from C by declaring an external structure using this name. For example,

```
COMMON /comm/ a,b,c
```

can be accessed with the C declaration:

```
extern struct {
 float a;
 float b;
 float c;
} _CCOMM;
```

### Declaring C Structures in Absoft Pro Fortran

If there are equivalent data types in FORTRAN for all elements of a C structure, a RECORD can be declared in FORTRAN to match the structure in C:

| <u>C</u>                                                                                                    | <u>FORTRAN</u>                                                                                                                    |
|-------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <pre>struct str {<br/>char c;<br/>long l;<br/>float f;<br/>double d;<br/>};<br/>struct str my_struct;</pre> | <pre>STRUCTURE /str/<br/>CHARACTER c<br/>INTEGER*4 l<br/>REAL*4 f<br/>REAL*8 d<br/>END STRUCTURE<br/>RECORD /str/ my_struct</pre> |

By default, the alignment of the C structure should be identical to the FORTRAN RECORD. Refer to the **Specification and DATA Statements** chapter of the *FORTRAN 77 Language Reference Manual* for more information on the FORTRAN RECORD type.

## INTERFACING WITH ASSEMBLY LANGUAGE

If you are interested in interfacing to Fortran through assembly language, refer to the Apple document supplied with the Macintosh OS X developer tools.

## DEBUGGING

Debugging a Fortran program is accomplished with the Absoft source-level debugger, Fx™. This is a multi-language, windowed debugger designed especially for the Intel based Linux computers. The operation of the debugger is detailed in the chapter, **Using the Fx Debugger**. The following paragraphs describe the compiler options and resources necessary to prepare a program for debugging.

### Compiler Options

The **-g** compiler option directs the compiler to add symbol and line number information to the object file. This option should be enabled for each source file that you will want to have source code displayed while debugging. It is not required for files that you are not interested in.

It is recommended that all optimization options be disabled while debugging. This is because the optimizers can greatly distort the appearance and order of execution of the individual statements in your program. Code can be removed or added (for loop unrolling), variables may be removed or allocated to registers (making it impossible to examine or modify them), and statements may be executed out of order.

## **PROFILING**

The Macintosh OS X operating system includes the libraries and tools necessary to obtain procedure level profiles of your application. You simply create an instrumented version of your application (see **Compiler Options** below) and then execute it. The file `gmon.out` will automatically be created. Use *gprof* to display and analyze the results.

### **Compiler Options**

The **-P** compiler option directs the compiler to add the symbol information to the object file necessary to profile an application. Enabling this option will allow the application to report the number of times a particular subroutine is called or a function is referenced.

All other options that you would normally use should be enabled, including optimization.



## Appendix A

### Absoft Compiler Option Guide

---

This appendix summarizes general options for Absoft Pro Fortran compilers and specific options for the Absoft Fortran 90/95 and FORTRAN 77 compilers. Refer to the chapter, **Using the Compilers** for detailed descriptions of the options

#### ABSOF PRO FORTRAN COMPILER OPTIONS

| <i>Option</i>   | <i>Effect</i> □                                                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-c</b>       | suppresses creation of an executable file — leaves compiled files in object code format. □                                                 |
| <b>-g</b>       | generates symbol information for Fx™. □                                                                                                    |
| <b>-Lpath</b>   | library file search path specification.                                                                                                    |
| <b>-lname</b>   | library file specification.                                                                                                                |
| <b>-O</b>       | enables a group of basic optimizations which will cause most code to run faster without the expense of application size or memory usage. □ |
| <b>-o name</b>  | directs the compiler to produce an executable file called <i>name</i> where <i>name</i> is a Macintosh OS X file name.                     |
| <b>-P</b>       | instrument executable for profiling. □                                                                                                     |
| <b>-S</b>       | generates an assembly language output file. □                                                                                              |
| <b>-s</b>       | allocate local variables statically.                                                                                                       |
| <b>-u</b>       | undefine a symbol to the linker.                                                                                                           |
| <b>-v</b>       | directs the compiler to print status information as the compilation process proceeds.                                                      |
| <b>-w</b>       | suppresses listing of all compile-time warning messages. □                                                                                 |
| <b>-Xoption</b> | linker option. □                                                                                                                           |

### FPU CONTROL OPTIONS

**-round=*mode*** set the FPU rounding method.

**-trap=*exception*** enable FPU exceptions.

### FORTRAN 90/95 CONTROL OPTIONS

**-YEXT\_SFX** append trailing characters to procedure names.

**-B112** disable stack alignment.

**-dq** Allow more than 100 error diagnostics.

**-ea** Causes the  $\text{f95}$  compiler to abort the compilation process on the first error that it encounters.

**-en** Causes the compiler to issue a warning whenever the source code contains an extension to the Fortran 90/95 standard.

**-eR** Default recursion

**-g** Generates symbol information for Fx<sup>TM</sup>.

**-Mnn** Suppresses messages by message number.

**-mnn** Suppresses messages by message level.

**-P** Instrument executable for profiling.

**-V** Causes the  $\text{f95}$  compiler to display its version number.

**-v** Directs the compiler to print status information as the compilation process proceeds

**-w** Suppresses listing of all compile-time warning messages.

### FORTRAN 90/95 OPTIMIZATION OPTIONS

**-O[n]** enables a group of basic optimizations which will cause most code to run faster without the expense of application size or memory usage.

### FORTRAN 90/95 SOURCE FORMAT OPTIONS

**-fform** sets the form of the source file to **free**, **fixed**, or **alt\_fixed**.

**-Wn** sets the line length of source statements accepted by the compiler in Fixed-Form source format.



**FORTRAN 90/95 COMPATIBILITY OPTIONS**

- dp** causes variables declared in a `DOUBLE PRECISION` statement and constants specified with the `D` exponent to be converted to the default real kind. □
- ej** causes all `DO` loops to be executed at least once, regardless of the initial value of the iteration count. □
- in** set default integer size to *n* (2 or 8) bytes. □
- N113** set default real size to 8 bytes (`KIND=8`). □
- p path** specify module search path □
- s** allocate local variables statically
  
- Rb** generate code to check array boundaries. □
- Rc** generate code to validate substring indexes. □
- Rp** generate code to check for null pointers. □
- Rs** generate code check array conformance. □
- tn** this option increases the default temporary string size to  $1024 \times 10^n$  bytes. □
- xdirective** disable compiler directive in the source file.
  
- YCFRL** forces the compiler to pass `g77/f2c` compatible `CHARACTER` arguments. □
- YCOM\_NAMES** specify `COMMON` block names externally in upper or lower case. □
- YCOM\_PFX** specify `COMMON` block external name prefix. □
- YCOM\_SFX** specify `COMMON` block external name suffix. □
- YCSLASH** directs the compiler to transform certain escape sequences marked with a `'\'` embedded in character constants. □
- YEXT\_NAMES** Specify procedure names externally in upper, lower, or mixed case. □
- YEXT\_PFX** Specify procedure external name prefix. □
- YEXT\_SFX** Specify procedure external name suffix. □
- YMS7D** Recognize Microsoft style compiler directives beginning with a `'$'` in column 1. □
- YNDFP** disallow the use of a `'.'` as a structure field separator. □
- YPEI** pointers are Equivalent to Integers allows a Cray-style pointer to be manipulated as an integer. □

**FORTRAN 77 CONTROL OPTIONS**

- N15** append trailing underscores to procedure names.
- C** generates code to check that array indexes are within array bounds - file names and source code line numbers will be displayed with all run time error messages □
- D** used to define conditional compilation variables from the command line (**-D name[=value]**) — if *value* is not present, the variable is assigned the value of 1 □

- g**                    generates symbol information for Fx™.□
- lpath**                specify path to search for INCLUDE files.□
- P**                    instrument executable for profiling.□
- q**                    suppress non-diagnostic output.□
- Tnn**                 used to change the number of handles used internally by the compiler.
  
- tnn**                 modifies the default temporary string size to *nn* bytes from the default of 1024 bytes
  
- v**                    directs the compiler to print status information as the compilation process proceeds□
- w**                    suppresses listing of all compile-time warning messages□
- x**                    replaces any occurrence of X or D in column one with a blank character: allows a restricted form of conditional compilation □

### **FORTRAN 77 OPTIMIZATION OPTIONS**

- O[n]**                enables a group of basic optimizations which will cause most code to run faster without the expense of application size or memory usage.

### **FORTRAN 77 SOURCE FORMAT OPTIONS**

- 8**                    directs the compiler to accept source code written in Fortran 90/95 Free Source Form
  
- V**                    directs the compiler to accept VAX Tab-Format source code□
- W**                    directs the compiler to accept statements which extend beyond column 72 up to column 132

### **FORTRAN 77 COMPATIBILITY OPTIONS**

- N15**                causes the compiler to define SUBROUTINE and FUNCTION names with a trailing underscore□
- d**                    causes all DO loops to be executed at least once, regardless of the initial value of the iteration count (FORTRAN 66 convention)□
- f**                    folds all symbolic names to lower case□
- in**                 changes the default storage length of INTEGER from 4 bytes to *n* (2 or 8).□
- K**                    directs the compiler to transform certain escape sequences marked with a ‘\’ embedded in character constants□
- N22**                don’t mangle COMMON block names with leading “\_c”

- N26**            force the compiler to consider the byte ordering of all unformatted files to be big-endian by default
  
- N27**            force the compiler to consider the byte ordering of all unformatted files to be little-endian by default
  
- N109**            folds all symbolic names to UPPER CASE□
- N113**            changes REAL and COMPLEX data types without explicit length declaration to DOUBLE PRECISION and DOUBLE COMPLEX□
  
- s**                forces all program storage to be treated as static: see **-N1** also□



# Appendix B

## Exceptions and IEEE Arithmetic

Three modules are provided to support floating-point exceptions and IEEE arithmetic: `IEEE_FEATURES`, `IEEE_ARITHMETIC`, and `IEEE_EXCEPTIONS`. Use of these modules and the procedures in them ensure portability of programs exploiting features of IEEE arithmetic across platforms. The module `IEEE_ARITHMETIC` contains a `USE` statement for `IEEE_EXCEPTIONS`. Any procedure that uses `IEEE_ARITHMETIC` will have access to the public features of `IEEE_EXCEPTIONS`.

### `IEEE_FEATURES`

This module defines the derived type `IEEE_FEATURES_TYPE` whose components are all private. Its purpose is to express the need for particular IEEE features.

### `IEEE_FEATURES_TYPE`

The only possible values are the following constants:

|                                  |                                                       |
|----------------------------------|-------------------------------------------------------|
| <code>IEEE_DATATYPE</code>       | IEEE data types are available                         |
| <code>IEEE_DENORMAL</code>       | IEEE denormalized values are supported                |
| <code>IEEE_DIVIDE</code>         | IEEE division to the required precision is supported  |
| <code>IEEE_HALTING</code>        | control of halting is supported                       |
| <code>IEEE_INEXACT_FLAG</code>   | inexact exceptions are supported.                     |
| <code>IEEE_INF</code>            | IEEE infinities (positive and negative) are supported |
| <code>IEEE_INVALID_FLAG</code>   | invalid exceptions are supported                      |
| <code>IEEE_NAN</code>            | IEEE NaN (Not a Number) values are supported          |
| <code>IEEE_ROUNDING</code>       | all IEEE rounding modes are supported                 |
| <code>IEEE_SQRT</code>           | SQRT is supported to the IEEE standard                |
| <code>IEEE_UNDERFLOW_FLAG</code> | underflow exceptions supported                        |

**IEEE\_ARITHMETIC**

This module defines the two derived types `IEEE_CLASS_TYPE` and `IEEE_ROUND_TYPE` whose components are all private. The purpose of `IEEE_CLASS_TYPE` is to identify the class of a value. The purpose of `IEEE_ROUND_TYPE` is to specify or inquire the rounding mode. This module also defines two elemental operators for each of these types: `==` and `/=`. The `==` operator returns true if two values of these types are equal and false if they are not. The `/=` operator returns true if two values of these types are not equal and false if they are equal.

The `IEEE_ARITHMETIC` module further provides a number of subroutines and functions for inquiry, performing operations, and setting the IEEE rounding environment.

**IEEE\_CLASS\_TYPE**

The only possible values are the following constants:

|                                     |                                                        |
|-------------------------------------|--------------------------------------------------------|
| <code>IEEE_SIGNALING_NAN</code>     | NaN (Not a Number)                                     |
| <code>IEEE_QUIET_NAN</code>         | NaN (Not a Number)                                     |
| <code>IEEE_NEGATIVE_INF</code>      | negative infinity                                      |
| <code>IEEE_NEGATIVE_NORMAL</code>   | negative number                                        |
| <code>IEEE_NEGATIVE_DENORMAL</code> | negative number smaller than the normal representation |
| <code>IEEE_NEGATIVE_ZERO</code>     | negative zero                                          |
| <code>IEEE_POSITIVE_ZERO</code>     | zero                                                   |
| <code>IEEE_POSITIVE_DENORMAL</code> | positive number smaller than the normal representation |
| <code>IEEE_POSITIVE_NORMAL</code>   | positive number                                        |
| <code>IEEE_POSITIVE_INF</code>      | positive infinity                                      |

## IEEE\_ROUND\_TYPE

The only possible values are the following constants:

**IEEE\_NEAREST**

**IEEE\_TO\_ZERO**

**IEEE\_UP**

**IEEE\_DOWN**

## Subroutines and Functions

The `IEEE_ARITHMETIC` module provides the following *inquiry* functions:

logical function **IEEE\_SUPPORT\_DATATYPE**(x)  
 real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if the data type of the argument is supported in conformance with section 14.8 of the Fortran 2003 Draft Standard.

logical function **IEEE\_SUPPORT\_DENORMAL**(x)  
 real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if denormalized numbers are supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_DIVIDE**(x)  
 real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if division to the accuracy specified by the IEEE standard is supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_INF**(x)  
 real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if infinities numbers are supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_IO**(x)  
 real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if IEEE rounding is supported during formatted input and output conversions for the data type of the argument.

logical function **IEEE\_SUPPORT\_NAN**(x)  
real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if NaN (Not a Number) values are supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_ROUNDING**(round\_value, x)  
type(IEEE\_ROUND\_TYPE), intent(in) :: round\_value  
real(kind=SP), intent(in), optional :: x

returns the value `.TRUE.` if the specified rounding type is supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_SQRT**(x)  
real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if `SQRT` to the accuracy specified by the IEEE standard is supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_STANDARD**(x)  
real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if all capabilities and facilities specified by the IEEE standard are supported for the data type of the argument.

logical function **IEEE\_SUPPORT\_UNDERFLOW**(x)  
real(kind=any), intent(in), optional :: x

returns the value `.TRUE.` if control of underflow mode is supported for the data type of the argument. Control of underflow mode allows specifying gradual or abrupt underflow. On some processors, chopped underflow can produce code that executes faster.

The `IEEE_ARITHMETIC` module provides the following *elemental* functions for data values that `IEEE_SUPPORT_DATATYPE(x)` returns `.TRUE.`:

elemental type(IEEE\_CLASS\_TYPE) function **IEEE\_CLASS**(x)  
real(kind=any), intent(in) :: x

returns the class of the input argument `x`.

elemental real function **IEEE\_COPY\_SIGN**(x, y)  
real(kind=any), intent(in) :: x, y

returns a result that has the value of `x` and the sign of `y`, even when `x` is a special value such as NaN or infinity.



elemental logical function **IEEE\_IS\_FINITE**(*x*)  
 real(kind=any), intent(in) :: *x*

returns the value `.TRUE.` if the value of *x* is finite. That is its class is one of:

|                        |                        |
|------------------------|------------------------|
| IEEE_NEGATIVE_NORMAL   | IEEE_POSITIVE_DENORMAL |
| IEEE_NEGATIVE_DENORMAL | IEEE_POSITIVE_NORMAL   |
| IEEE_NEGATIVE_ZERO     | IEEE_POSITIVE_ZERO     |

elemental logical function **IEEE\_IS\_NAN**(*x*)  
 real(kind=any), intent(in) :: *x*

returns the value `.TRUE.` if the value of *x* is a NaN.

elemental logical function **IEEE\_IS\_NORMAL**(*x*)  
 real(kind=any), intent(in) :: *x*

returns the value `.TRUE.` if the value of *x* is normal. That is its class is one of:

|                      |                      |
|----------------------|----------------------|
| IEEE_NEGATIVE_NORMAL | IEEE_POSITIVE_NORMAL |
| IEEE_NEGATIVE_ZERO   | IEEE_POSITIVE_ZERO   |

elemental logical function **IEEE\_IS\_NEGATIVE**(*x*)  
 real(kind=any), intent(in) :: *x*

returns the value `.TRUE.` if the value of *x* is negative. That is its class is one of:

|                        |                        |
|------------------------|------------------------|
| IEEE_NEGATIVE_NORMAL   | IEEE_NEGATIVE_ZERO     |
| IEEE_NEGATIVE_DENORMAL | IEEE_NEGATIVE_INFINITY |

elemental real function **IEEE\_LOGB**(*x*)  
 real(kind=any), intent(in) :: *x*

returns the unbiased exponent of *x* as a floating-point value. If *x* is 0.0, -infinity is returned and `IEEE_DIVIDE_BY_ZERO` is signaled.

elemental real function **IEEE\_NEXT\_AFTER**(*x*, *y*)  
 real(kind=any), intent(in) :: *x*, *y*

returns the neighbor of *x* in the direction of *y*. The *kind* of the result is the same as *x*. If *x* is 0.0, the result is the smallest denormalized number.

elemental real function **IEEE\_REM**(*x*, *y*)  
 real(kind=any), intent(in) :: *x*, *y*

returns the exact remainder of *x*/*y*. The function is defined as  $x - y * n$  where *n* is the nearest integer to  $x/y$ . If  $|n - x/y| = 1/2$ , *n* is even. The *kind* of the result is the same as *x*.

elemental real function **IEEE\_RINT**(x)  
real(kind=any), intent(in) :: x

returns the rounded to integer value of x. x is rounded according to the current rounding mode.

elemental real function **IEEE\_SCALB**(x, i)  
real(kind=any), intent(in) :: x  
integer (kind=4), intent(in) :: i

returns the floating-point value of  $x \cdot 2^{i}$ . The *kind* of the result is the same as x.

elemental logical function **IEEE\_UNORDERED**(x, y)  
real(kind=any), intent(in) :: x, y

returns if either x or y is a NaN.

elemental real function **IEEE\_VALUE**(x, class)  
real(kind=any), intent(in) :: x  
type(IEEE\_CLASS\_TYPE), intent(in) :: class

returns a value of the specified IEEE\_CLASS\_TYPE.

The IEEE\_ARITHMETIC module provides the following non-*elemental* subroutines:

subroutine **IEEE\_GET\_ROUNDING\_MODE**(round\_value)  
type(IEEE\_ROUND\_TYPE), intent(out) :: round\_value

the current rounding mode is returned in the round\_value variable.

subroutine **IEEE\_GET\_UNDERFLOW\_MODE**(gradual)  
logical(kind=4). Intent(out) :: gradual

the current underflow mode (gradual/abrupt) is returned in the gradual variable. If the mode is gradual, the value of gradual will be set to .TRUE.. If IEEE\_SUPPORT\_UNDERFLOW\_CONTROL(x) returns .FALSE. this subroutine will produce a runtime error and should not be called.

subroutine **IEEE\_SET\_ROUNDING\_MODE**(round\_value)  
type(IEEE\_ROUND\_TYPE), intent(in) :: round\_value

the rounding mode is set to the mode specified in the round\_value variable.

```
subroutine IEEE_SET_UNDERFLOW_MODE(gradual)
logical(kind=4), intent(in) :: gradual
```

the underflow mode is set to gradual if the value of the `gradual` variable is `.TRUE.`. If the value of the `gradual` variable is `.FALSE.`, the underflow mode is set to abrupt. If `IEEE_SUPPORT_UNDERFLOW_CONTROL(x)` returns `.FALSE.` this subroutine will produce a runtime error and should not be called.

The `IEEE_ARITHMETIC` module provides the following *kind* function:

```
integer(kind=4) function IEEE_SELECTED_REAL_KIND(p, r)
integer(kind=4), intent(in), optional :: p
integer(kind=4), intent(in), optional :: r
```

returns the *kind* of an IEEE floating-point value with the requested precision and exponent range. This function is similar to `SELECTED_REAL_KIND`, but only returns IEEE reals.

## IEEE\_EXCEPTIONS

This module defines the two derived types `IEEE_FLAG_TYPE` and `IEEE_STATUS_TYPE` whose components are all private. The purpose of `IEEE_FLAG_TYPE` is to identify the exception flags. The purpose of `IEEE_STATUS_TYPE` is to save and restore the current floating-point environment.

The `IEEE_EXCEPTIONS` module also provides a number of subroutines and functions for inquiry and getting and setting exception flags.

An important feature of exception flags is that they may or may not halt execution of the program depending on the state of the halting modes. A floating-point operation (such as divide-by-zero) will cause an exception flag to signal, but unless the halting mode for that flag is set to true, execution will continue with an IEEE default value. The `IEEE_GET_FLAG` subroutine can be used to retrieve the current state (signaling or quiet) of a specific exception flag.

**IEEE\_FLAG\_TYPE**

The only possible values are the following constants:

**IEEE\_INVALID**

**IEEE\_OVERFLOW**

**IEEE\_DIVIDE\_BY\_ZERO**

**IEEE\_UNDERFLOW**

**IEEE\_INEXACT**

and the array constants:

**IEEE\_USUAL**

```
type(IEEE_FLAG_TYPE), parameter, dimension(3) :: IEEE_USUAL = &
 (/IEEE_OVERFLOW, IEEE_DIVIDE_BY_ZERO, IEEE_INVALID/)
```

**IEEE\_ALL**

```
type(IEEE_FLAG_TYPE), parameter, dimension(5) :: IEEE_ALL = &
 (/IEEE_USUAL, IEEE_UNDERFLOW, IEEE_INEXACT/)
```

**IEEE\_STATUS\_TYPE**

This type is used to save and restore the floating-point environment.

**Subroutines and Functions**

The `IEEE_EXCEPTIONS` module provides the following *elemental* subroutines:

```
elemental subroutine IEEE_GET_FLAG(flag, flag_value)
type(IEEE_FLAG_TYPE), intent(in) :: flag
logical(kind=4), intent(out) :: flag_value
```

retrieves the state of the specified flag. If the flag is signaling, `flag_value` is set to `.TRUE..` If the flag is quiet, `flag_value` is set to `.FALSE..`

```
elemental subroutine IEEE_GET_HALTING_MODE(flag, halting)
type(IEEE_FLAG_TYPE), intent(in) :: flag
logical(kind=4), intent(out) :: halting
```

retrieves the halting mode of the specified flag. If the `flag` mode is halting, halting is set to `.TRUE..` If the `flag` mode is continue, halting is set to `.FALSE..`

The `IEEE_EXCEPTIONS` module provides the following non-*elemental* subroutines:

```
subroutine IEEE_GET_STATUS(status_value)
type(IEEE_STATUS_TYPE) :: status_value
```

retrieves the state of the floating-point environment.

```
subroutine IEEE_SET_FLAG(flag, flag_value)
type(IEEE_FLAG_TYPE), intent(in) :: flag
logical(kind=4), intent(in) :: flag_value
```

sets the state of the specified flag. If `flag_value` is `.TRUE..`, the `flag` is set to signaling. If `flag_value` is `.FALSE..`, the `flag` is set to quiet.

```
elemental subroutine IEEE_SET_HALTING_MODE(flag, halting)
type(IEEE_FLAG_TYPE), intent(in) :: flag
logical(kind=4), intent(in) :: halting
```

sets the halting mode of the specified flag. If `halting` is `.TRUE..`, the `flag` mode is set to halting. If `halting` is `.FALSE..`, the `flag` mode is set to continue.

```
subroutine IEEE_SET_STATUS(status_value)
type(IEEE_STATUS_TYPE) :: status_value
```

sets the state of the floating-point environment.

**EXAMPLES**

The following example demonstrates the sequence necessary to exercise program control over detection of an exception.

```
subroutine safe_divide(a, b, c, fail)
 use IEEE_EXCEPTIONS
 use IEEE_ARITHMETIC
 real a, b, c
 logical fail
 type(IEEE_STATUS_TYPE) status

 ! save the current floating-point environment, turn halting for
 ! divide-by-zero off, and clear any previous divide-by-zero flag
 call IEEE_GET_STATUS(status)
 call IEEE_SET_HALTING_MODE(IEEE_DIVIDE_BY_ZERO, .false.)
 call IEEE_SET_FLAG(IEEE_DIVIDE_BY_ZERO, .false.)

 ! perform the operation
 c = a/b

 ! determine if a failure occurred and restore the floating-point
 ! environment
 fail = IEEE_IS_NAN(c) .or. .not. IEEE_IS_FINITE(c)
 call IEEE_SET_STATUS(status)

end subroutine safe_divide
```

# Appendix C

## Terminal Programming

---

All of the Absoft tools are also designed to be used in a command line environment. This appendix outlines the use of the Macintosh Terminal application which provides a command line interface to the OS X operating system.

Open a terminal through the Finder by navigating to the `/Applications/Utilities` folder and double clicking on the `Terminal` application. A window will open and a command prompt will appear. The prompt will look similar to:

```
[computer_name:~] username%
```

The `~` symbol is an abbreviation for your home directory (`/Users/username/`). If you enter a change directory command such as “`cd fortran`” (assuming you have a folder named `fortran` in your home directory) the prompt will change to:

```
[computer_name:~/fortran] username%
```

and `/Users/username/fortran` will be the current working directory. To return to your home directory you can enter “`cd ..`”. This will take you up one directory. You can also enter “`cd ~`” from any directory to return to your home directory.

The environment variables must be modified so that the system can find the Absoft compilers. Change the current working directory to your home directory (`cd ~`) and enter the following command to edit a file with a simple text editor:

```
pico .tcshrc
```

This launches the text editor `pico` and specifies that we want to edit the file `.tcshrc`. The file `.tcshrc` is a special file which is read every time you open a terminal. Enter the following text on the first line in the editor:

```
setenv ABSOFT /Applications/Absoft10
```

This creates an environment variable named `ABSOFT` and sets its value to `/Applications/Absoft10` (the location of the Absoft compilers). The existing environment variable `path` must be modified to allow the system to find the Absoft compilers. Enter the following text on the second line in the editor:

```
set path = ($ABSOFT/bin $path)
```

A `$` before an environment variable indicates that we are referencing the value of that variable. This line modifies the `path` environment variable, instructing the system to look in the `/Applications/Absoft10/bin` directory for executable programs.

To exit `pico`, press `Ctrl-X` and answer “y” to the “Save modified buffer” question. Because this file is only read when a terminal session is started, it is necessary to close this terminal and then open a new one. The compilers can now be used from the terminal.

Change the current working directory (with the `cd` command) to one containing a Fortran source file. Use the `ls` command to list the files in the current directory.

Compile and link the Fortran file with the command:

```
f77 source.f -o myprogram
```

or

```
f95 source.f95 -o myprogram
```

These commands will automatically compile and link your source file to produce an executable file named `myprogram`. Enter “`./myprogram`” to run your program. The “`./`” indicates to look in the current working directory for the file. (**Note:** omitting “`-o name`” produces an executable named `a.out`.)

If you just wish to compile your source file into an object file to be linked later, add the `-c` option:

```
f77 source.f -c
```

or

```
f95 source.f95 -c
```

These commands both produce `source.o`. You can link object files together to create executables with the `f77` and `f95` commands as well:

```
f77 source1.o source2.o source3.o -o myprogram
```

or

```
f95 source1.o source2.o source3.o -o myprogram
```

Refer to the chapter **Using the Compilers** for a description of compiler options.



## Appendix D

### ASCII Table

ASCII codes 0 through 31 are control codes that may or may not have meaning on Macintosh. They are listed for completeness and historical reasons and may aid when porting code from other systems. Codes 128 through 255 are extensions to the 7-bit ASCII standard and the symbol displayed depends on the font being used; the symbols shown below are from the Times New Roman font. The Dec, Oct, and Hex columns refer to the decimal, octal, and hexadecimal numerical representations.

| Character | Dec | Oct | Hex | Description      | Character | Dec | Oct | Hex | Description    |
|-----------|-----|-----|-----|------------------|-----------|-----|-----|-----|----------------|
| NULL      | 0   | 000 | 00  | null             |           | 32  | 040 | 20  | space          |
| SOH       | 1   | 001 | 01  | start of heading | !         | 33  | 041 | 21  | exclamation    |
| STX       | 2   | 002 | 02  | start of text    | "         | 34  | 042 | 22  | quotation mark |
| ETX       | 3   | 003 | 03  | end of text      | #         | 35  | 043 | 23  | number sign    |
| ECT       | 4   | 004 | 04  | end of trans     | \$        | 36  | 044 | 24  | dollar sign    |
| ENQ       | 5   | 005 | 05  | enquiry          | %         | 37  | 045 | 25  | percent sign   |
| ACK       | 6   | 006 | 06  | acknowledge      | &         | 38  | 046 | 26  | ampersand      |
| BEL       | 7   | 007 | 07  | bell code        | '         | 39  | 047 | 27  | apostrophe     |
| BS        | 8   | 010 | 08  | back space       | (         | 40  | 050 | 28  | opening paren  |
| HT        | 9   | 011 | 09  | horizontal tab   | )         | 41  | 051 | 29  | closing paren  |
| LF        | 10  | 012 | 0A  | line feed        | *         | 42  | 052 | 2A  | asterisk       |
| VT        | 11  | 013 | 0B  | vertical tab     | +         | 43  | 053 | 2B  | plus           |
| FF        | 12  | 014 | 0C  | form feed        | ,         | 44  | 054 | 2C  | comma          |
| CR        | 13  | 015 | 0D  | carriage return  | -         | 45  | 055 | 2D  | minus          |
| SO        | 14  | 016 | 0E  | shift out        | .         | 46  | 056 | 2E  | period         |
| SI        | 15  | 017 | 0F  | shift in         | /         | 47  | 057 | 2F  | slash          |
| DLE       | 16  | 020 | 10  | data link escape | 0         | 48  | 060 | 30  | zero           |
| DC1       | 17  | 021 | 11  | device control 1 | 1         | 49  | 061 | 31  | one            |
| DC2       | 18  | 022 | 12  | device control 2 | 2         | 50  | 062 | 32  | two            |
| DC3       | 19  | 023 | 13  | device control 3 | 3         | 51  | 063 | 33  | three          |
| DC4       | 20  | 024 | 14  | device control 4 | 4         | 52  | 064 | 34  | four           |
| NAK       | 21  | 025 | 15  | negative ack     | 5         | 53  | 065 | 35  | five           |
| SYN       | 22  | 026 | 16  | synch idle       | 6         | 54  | 066 | 36  | six            |
| ETB       | 23  | 027 | 17  | end of trans blk | 7         | 55  | 067 | 37  | seven          |
| CAN       | 24  | 030 | 18  | cancel           | 8         | 56  | 070 | 38  | eight          |
| EM        | 25  | 031 | 19  | end of medium    | 9         | 57  | 071 | 39  | nine           |
| SS        | 26  | 032 | 1A  | special sequence | :         | 58  | 072 | 3A  | colon          |
| ESC       | 27  | 033 | 1B  | escape           | ;         | 59  | 073 | 3B  | semicolon      |
| FS        | 28  | 034 | 1C  | file separator   | <         | 60  | 074 | 3C  | less than      |
| GS        | 29  | 035 | 1D  | group separator  | =         | 61  | 075 | 3D  | equal          |
| RS        | 30  | 036 | 1E  | record separator | >         | 62  | 076 | 3E  | greater than   |
| US        | 31  | 037 | 1F  | unit separator   | ?         | 63  | 077 | 3F  | question mark  |

| Character | Dec | Oct | Hex | Description       | Character | Dec | Oct | Hex |        |
|-----------|-----|-----|-----|-------------------|-----------|-----|-----|-----|--------|
| @         | 64  | 100 | 40  | commercial at     | ~         | 126 | 176 | 7E  | tilde  |
| A         | 65  | 101 | 41  | upper case letter |           | 127 | 177 | 7F  | delete |
| B         | 66  | 102 | 42  | upper case letter | □         | 128 | 200 | 80  |        |
| C         | 67  | 103 | 43  | upper case letter | □         | 129 | 201 | 81  |        |
| D         | 68  | 104 | 44  | upper case letter | ,         | 130 | 202 | 82  |        |
| E         | 69  | 105 | 45  | upper case letter | f         | 131 | 203 | 83  |        |
| F         | 70  | 106 | 46  | upper case letter | ”         | 132 | 204 | 84  |        |
| G         | 71  | 107 | 47  | upper case letter | ...       | 133 | 205 | 85  |        |
| H         | 72  | 110 | 48  | upper case letter | †         | 134 | 206 | 86  |        |
| I         | 73  | 111 | 49  | upper case letter | ‡         | 135 | 207 | 87  |        |
| J         | 74  | 112 | 4A  | upper case letter | ^         | 136 | 210 | 88  |        |
| K         | 75  | 113 | 4B  | upper case letter | %         | 137 | 211 | 89  |        |
| L         | 76  | 114 | 4C  | upper case letter | Š         | 138 | 212 | 8A  |        |
| M         | 77  | 115 | 4D  | upper case letter | <         | 139 | 213 | 8B  |        |
| N         | 78  | 116 | 4E  | upper case letter | Œ         | 140 | 214 | 8C  |        |
| O         | 79  | 117 | 4F  | upper case letter | □         | 141 | 215 | 8D  |        |
| P         | 80  | 120 | 50  | upper case letter | □         | 142 | 216 | 8E  |        |
| Q         | 81  | 121 | 51  | upper case letter | □         | 143 | 217 | 8F  |        |
| R         | 82  | 122 | 52  | upper case letter | □         | 144 | 220 | 90  |        |
| S         | 83  | 123 | 53  | upper case letter | ‘         | 145 | 221 | 91  |        |
| T         | 84  | 124 | 54  | upper case letter | ’         | 146 | 222 | 92  |        |
| U         | 85  | 125 | 55  | upper case letter | “         | 147 | 223 | 93  |        |
| V         | 86  | 126 | 56  | upper case letter | ”         | 148 | 224 | 94  |        |
| W         | 87  | 127 | 57  | upper case letter | •         | 149 | 225 | 95  |        |
| X         | 88  | 130 | 58  | upper case letter | —         | 150 | 226 | 96  |        |
| Y         | 89  | 131 | 59  | upper case letter | —         | 151 | 227 | 97  |        |
| Z         | 90  | 132 | 5A  | upper case letter | ~         | 152 | 230 | 98  |        |
| [         | 91  | 133 | 5B  | opening bracket   | ™         | 153 | 231 | 99  |        |
| \         | 92  | 134 | 5C  | back slash        | š         | 154 | 232 | 9A  |        |
| ]         | 93  | 135 | 5D  | closing bracket   | >         | 155 | 233 | 9B  |        |
| ^         | 94  | 136 | 5E  | circumflex        | œ         | 156 | 234 | 9C  |        |
| ˘         | 95  | 137 | 5F  | underscore        | □         | 157 | 235 | 9D  |        |
| ˘         | 96  | 140 | 60  | grave accent      | □         | 158 | 236 | 9E  |        |
| a         | 97  | 141 | 61  | lower case letter | ÿ         | 159 | 237 | 9F  |        |
| b         | 98  | 142 | 62  | lower case letter |           | 160 | 240 | A0  |        |
| c         | 99  | 143 | 63  | lower case letter | ı         | 161 | 241 | A1  |        |
| d         | 100 | 144 | 64  | lower case letter | ç         | 162 | 242 | A2  |        |
| e         | 101 | 145 | 65  | lower case letter | £         | 163 | 243 | A3  |        |
| f         | 102 | 146 | 66  | lower case letter | ¤         | 164 | 244 | A4  |        |
| g         | 103 | 147 | 67  | lower case letter | ¥         | 165 | 245 | A5  |        |
| h         | 104 | 140 | 68  | lower case letter |           | 166 | 246 | A6  |        |
| i         | 105 | 151 | 69  | lower case letter | §         | 167 | 247 | A7  |        |
| j         | 106 | 152 | 6A  | lower case letter | ”         | 168 | 250 | A8  |        |
| k         | 107 | 153 | 6B  | lower case letter | ©         | 169 | 251 | A9  |        |
| l         | 108 | 154 | 6C  | lower case letter | ª         | 170 | 252 | AA  |        |
| m         | 109 | 155 | 6D  | lower case letter | «         | 171 | 253 | AB  |        |
| n         | 110 | 156 | 6E  | lower case letter | ¬         | 172 | 254 | AC  |        |
| o         | 111 | 157 | 6F  | lower case letter | -         | 173 | 255 | AD  |        |
| p         | 112 | 160 | 70  | lower case letter | ®         | 174 | 256 | AE  |        |
| q         | 113 | 161 | 71  | lower case letter | —         | 175 | 257 | AF  |        |
| r         | 114 | 162 | 72  | lower case letter | °         | 176 | 260 | B0  |        |
| s         | 115 | 163 | 73  | lower case letter | ±         | 177 | 261 | B1  |        |
| t         | 116 | 164 | 74  | lower case letter | ²         | 178 | 262 | B2  |        |
| u         | 117 | 165 | 75  | lower case letter | ³         | 179 | 263 | B3  |        |
| v         | 118 | 166 | 76  | lower case letter | ´         | 180 | 264 | B4  |        |
| w         | 119 | 167 | 77  | lower case letter | µ         | 181 | 265 | B5  |        |
| x         | 120 | 170 | 78  | lower case letter | ¶         | 182 | 266 | B6  |        |
| y         | 121 | 171 | 79  | lower case letter | ·         | 183 | 267 | B7  |        |
| z         | 122 | 172 | 7A  | lower case letter | ,         | 184 | 270 | B8  |        |
| {         | 123 | 173 | 7B  | opening brace     | ı         | 185 | 271 | B9  |        |
|           | 124 | 174 | 7C  | vertical bar      | °         | 186 | 272 | BA  |        |
| }         | 125 | 175 | 7D  | closing brace     | »         | 187 | 273 | BB  |        |

| Character | Dec | Oct | Hex | Character | Dec | Oct | Hex |
|-----------|-----|-----|-----|-----------|-----|-----|-----|
| ¼         | 188 | 274 | BC  | Ɔ         | 222 | 336 | DE  |
| ½         | 189 | 275 | BD  | Ɓ         | 223 | 337 | DF  |
| ¾         | 190 | 276 | BE  | à         | 224 | 340 | E0  |
| ¿         | 191 | 277 | BF  | á         | 225 | 341 | E1  |
| À         | 192 | 300 | C0  | â         | 226 | 342 | E2  |
| Á         | 193 | 301 | C1  | ã         | 227 | 343 | E3  |
| Â         | 194 | 302 | C2  | ä         | 228 | 344 | E4  |
| Ã         | 195 | 303 | C3  | å         | 229 | 345 | E5  |
| Ä         | 196 | 304 | C4  | æ         | 230 | 346 | E6  |
| Å         | 197 | 305 | C5  | ç         | 231 | 347 | E7  |
| Æ         | 198 | 306 | C6  | è         | 232 | 350 | E8  |
| Ç         | 199 | 307 | C7  | é         | 233 | 351 | E9  |
| È         | 200 | 310 | C8  | ê         | 234 | 352 | EA  |
| É         | 201 | 311 | C9  | ë         | 235 | 353 | EB  |
| Ê         | 202 | 312 | CA  | ì         | 236 | 354 | EC  |
| Ë         | 203 | 313 | CB  | í         | 237 | 355 | ED  |
| Ì         | 204 | 314 | CC  | î         | 238 | 356 | EE  |
| Í         | 205 | 315 | CD  | ï         | 239 | 357 | EF  |
| Î         | 206 | 316 | CE  | ð         | 240 | 360 | F0  |
| Ï         | 207 | 317 | CF  | ñ         | 241 | 361 | F1  |
| Ð         | 208 | 320 | D0  | ò         | 242 | 362 | F2  |
| Ñ         | 209 | 321 | D1  | ó         | 243 | 363 | F3  |
| Ò         | 210 | 322 | D2  | ô         | 244 | 364 | F4  |
| Ó         | 211 | 323 | D3  | õ         | 245 | 365 | F5  |
| Ô         | 212 | 324 | D4  | ö         | 246 | 366 | F6  |
| Õ         | 213 | 325 | D5  | ÷         | 247 | 367 | F7  |
| Ö         | 214 | 326 | D6  | ø         | 248 | 370 | F8  |
| ×         | 215 | 327 | D7  | ù         | 249 | 371 | F9  |
| Ø         | 216 | 330 | D8  | ú         | 250 | 372 | FA  |
| Ù         | 217 | 331 | D9  | û         | 251 | 373 | FB  |
| Ú         | 218 | 332 | DA  | ü         | 252 | 374 | FC  |
| Û         | 219 | 333 | DB  | ý         | 253 | 375 | FD  |
| Ü         | 220 | 334 | DC  | þ         | 254 | 376 | FE  |
| Ý         | 221 | 335 | DD  | ÿ         | 255 | 377 | FF  |



## Appendix E

### Bibliography

---

#### **FORTRAN 90/95**

These books and manuals are useful references for the Fortran 90/95 programming language and the floating point math format used by Absoft Pro Fortran on Linux.

Michael Metcalf and John Reid, *FORTRAN 90/95 explained*, Oxford University Press (1996)

Walter S. Brainerd, Charles H. Goldberg, and Jeanne C. Adams, *Programmer's Guide to Fortran90*, Unicom, Inc (1994)

Jeanne C. Adams, Walter S. Brainerd, Jeanne T. Martin, and Brian T. Smith, *Fortran Top 90*, Unicom, Inc (1994)

James F. Kerrigan, *Fortran 90*, O'Reilly & Associates, Inc (1993)

American National Standard Programming Language Fortran 90, X3.198-1991, ANSI, 1430 Broadway, New York, N.Y. 10018

COMPUTER, *A Proposed Standard for Binary Floating-Point Arithmetic*, Draft 8.0 of IEEE Task P754, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 (1981)

#### **FORTRAN 77**

These books and manuals are useful references for the FORTRAN language and the floating point math format used by Absoft Pro Fortran on Linux.

Page, Didday, and Alpert, *FORTRAN 77 for Humans*, West Publishing Company (1983)

Kruger, Anton, *Efficient FORTRAN Programming*, John Wiley & Sons, Inc. (1990)

Loren P. Meissner and Elliot I. Organick, *FORTRAN 77*, Addison-Wesley Publishing Company (1980)

Harry Katzan, Jr., *FORTRAN 77*, Van Nostrand Reinhold Company (1978)

J.N.P. Hume and R.C. Holt, *Programming FORTRAN 77*, Reston Publishing Company, Inc. (1979)

Harice L. Seeds, *FORTRAN IV*, John Wiley & Sons (1975)

Jehosua Friedmann, Philip Greenberg, and Alan M. Hoffberg, *FORTRAN IV, A Self-Teaching Guide*, John Wiley & Sons, Inc. (1975)

James S. Coan, *Basic FORTRAN*, Hayden Book Company (1980)

American National Standard Programming Language FORTRAN, X3.9-1978, ANSI, 1430 Broadway, New York, N.Y. 10018

COMPUTER, *A Proposed Standard for Binary Floating-Point Arithmetic*, Draft 8.0 of IEEE Task P754, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 (1981)

M. Abramowitz and I.E. Stegun, *Handbook of Mathematical Functions*, U.S. Department of Commerce, National Bureau of Standards (1972)

## Appendix F

# Technical Support

---

The Absoft Technical Support Group will provide technical assistance to all registered users of current products. They will *not* answer general questions about operating systems, operating system interfaces, graphical user interfaces, or teach programming. For further help on these subjects, please consult this manual and any of the books and manuals listed in the bibliography.

Before contacting Technical Support, please study this manual and the language reference manuals to be sure your problem is not covered here. Specifically, refer to the chapter **Using the Compilers** in this manual. To help Technical Support provide a quick and accurate solution to your problem, please include the following information in any correspondence or have it available when calling.

### Product Information:

- Name of product .
- Version number.
- Serial number.
- Version number of the operating system.

### System Configuration:

- Hardware configuration (hard drive, etc.).
- System software release (i.e. 4.0, 3.5, etc).
- Any software or hardware modifications to your system.

### Problem Description:

- What happens?
- When does it occur?
- Provide a small (20 line) step-by-step example if possible.

### Contacting Technical Support:

Address:    Absoft Corporation  
              Attn: Technical Support  
              2781 Bond Street  
              Rochester Hills, MI 48309

## 192 Technical Support

---

|                |                                                           |               |
|----------------|-----------------------------------------------------------|---------------|
| telephone:     | (248) 853-0095                                            | 9am - 3pm EST |
| FAX            | (248) 853-0108                                            | 24 Hours      |
| email          | support@absoft.com                                        | 24 Hours      |
| World Wide Web | <a href="http://www.absoft.com">http://www.absoft.com</a> |               |



# Index

---

- 132 column source code, 65, 82
- 386, porting from, 106
- 64-bit code, 40
- Absoft address, 193
- Absoft Editor commands
  - Edit menu, 24
  - File menu, 21
  - Formatmenu, 28
  - Help menu, 33
  - Tools menu, 31
  - Window menu, 32
- ABSOFRT\_RT\_FLAGS, 111
- Adding Menus, 122
- advanced optimizations, 41
- Alert dialog box, 137
- amake tool, 142
- ANSI C source format, 90
- Apple Events, 127
  - class, 128
  - ID, 128
  - target, 128
- array
  - boundary checking, 63, 75
- ASCII table, 187
- assembly language, 44
  - interfacing with FORTRAN, 166
- ATTRIBUTES directive, 71
- basic optimizations, 41
- bookmarks, 27
  - clear bookmarks, 27
  - next bookmark, 27
  - previous bookmark, 27
  - toggle bookmark, 27
- Bookmarks menu, 27
- C
  - function results, 163
  - interfacing with FORTRAN. (see interfacing FORTRAN and C)
- C/C++
  - options, 84
- carriage returns
  - adding, 105
- Check Syntax menu command, 31
- Clear bookmarks, 27
- Clear menu command, 119
- Clipboard, 119
- Close All menu command, 22
- Close box**, 139
- Close menu command, 22
- command line, 185
- Command-Enter, 118
- Command-Return, 118
- COMMON blocks from C, 166
- COMMON, aligning data, 83
- Compile menu command, 31
- compiler directives, 69
  - ATTRIBUTES directive, 71
  - FIXED directive, 70
  - FIXEDFORMLINESIZE directive, 71
  - FREE[FORM] directive, 70
  - NAME directive, 70
  - NOFREEFORM directive, 71
  - NOUNROLL directive, 72
  - PACK[ON] directive, 71
  - PACKOFF directive, 72
  - STACK directive, 72
  - UNROLL directive, 72
- compiler options, 169
  - TENV, FPU exception handling, 44
  - !N61, use temporary files, 87
  - 8, Fortran 90, 82
  - A, ANSI C, 90
  - A, ANSI C source format, 90
  - B156, speculative execution, 68
  - B157, inline CABS, 68
  - B158, address optimizations, 68
  - B80, procedure trace, 60
  - C, check boundaries, 75
  - c, relocatable object, 45
  - c++, ANSI C++, 90
  - D, define compiler variable, 77
  - d, define preprocessor variable, 88
  - d, one trip DO loops, 79
  - E, preprocess to file, 89
  - ea, stop on error, 58
  - ej, one trip DO loops, 62
  - en, non-standard usage, 59
  - ep, demote Double Precision, 62
  - eq, allow greater than 100 errors, 58
  - eR, default recursion, 60
  - et, runtime stack trace, 40
  - except, exception handling, 88
  - f, case fold, 78, 101
  - f, case folding, 166
  - f, fixed source form, 65
  - f, freed source form, 64
  - g, Fx2 debugging, 101
  - H, include tree to stderr, 87
  - i, integer sizes, 62, 78
  - inline none, no inlines, 87
  - K, escape sequences, 66, 81
  - K, K & R C source format, 90
  - K, K&R C, 91
  - L, library path specification, 45
  - l, library specification, 45
  - m64, 64-bit code, 40
  - maxerr, max errors, 86
  - MS7D, Microsoft directives, 66
  - N1, static storage, 78
  - N109, case fold, 78
  - N110, no Common mangling, 84
  - N113, floating point sizes, 62, 80
  - N124, procedure trace, 87
  - N15, append underscore, 79
  - N22, set Common, 84
  - N26, set big-endian, 65, 82
  - N26, set little-endian, 66, 82
  - N32, non-ANSI, 75
  - N34, align COMMON, 83
  - nodefaultmod, MODULE path, 69
  - no-pic, no position independent code, 68
  - nostdinc, no standard includes, 89

# Index

---

- O1, basic optimizations, 41
- O2, normal optimizations, 41
- O3, advanced optimizations, 41
- O4, IPA optimizations, 41
- P, enable profiling, 42, 167
- p, module files, 59
- ptv, verbose templates, 87
- q, quiet, 87
- q, quiet, 60, 75
- Q51, no fma instructions, 42, 44
- Rb, check array conformance, 63
- Rb, check boundaries, 63
- round=, FPU rounding mode, 43
- Rp, check pointers, 63
- Rs, check substrings, 63
- S, assembly language, 44
- s, static storage, 62, 78, 101
- safefp, safe floating-point, 69
- speed\_math, fast math, 42
- stack\_size, stack size, 68
- SysModFiles, F95 include directory, 59
- T, max internal handle, 58, 74, 85
- t, temporary strings, 58, 75
- u, undefine preprocessor variable, 89
- u, undefine symbol, 93
- v, show progress, 60, 75
- V, show version, 60
- V, VAX Tab-Format, 82
- v, verbose, 87
- VAX compatibility, 78
- W, line length, 65
- w, suppress compiler warnings, 59, 75
- W, wide format, 82
- w1, suppress template warnings, 87
- w16, suppress warnings, 86
- w31, suppress all warnings, 86
- w8, suppress anachronism warnings, 86
- w8, suppress informational warnings, 86
- wp, treat anachronisms as errors, 87
- x, disable compiler directive, 61
- x, conditional compilation, 76
- Xlinker, linker options, 93
- YALL\_NAMES, symbolic names, 69
- YCFRL=1, CHARACTER argument parameters, 62
- YCOM\_NAMES, COMMON block case, 68
- YCOM\_PFX, COMMON block prefix, 67
- YCOM\_SFX, COMMON block suffix, 67
- YDEALLOC, cache control, 59
- YEXT\_NAMES, external names, 65
- YEXT\_PFX, external symbol prefix, 65
- YEXT\_SFX, external symbol suffix, 65
- YNDFP, type elements, 66
- YNO\_CDEC, ignore CDEC\$ directives, 69
- YPEI, pointers equivalent to integers, 63
- YVAR\_NAMES, variable names, 69
- z, suppress messages, 58
- Z, suppress warning number, 59
- compiler version, 60
- compiling FORTRAN source code, 5
- complex data types
  - equivalent declarations in C, 159
- conditional compilation, 76
  - conditional compilation variables, 77
  - continuation lines, 102
  - conventions used in the manual, 2
  - Convert to Lower Case menu command, 31
  - Convert to Upper Case menu command, 31
  - Copy menu command, 25, 119
  - Customize Toolbar menu command, 33
  - Cut menu command, 25, 119
  - DATE subroutine, 102
  - debugging, 166
  - default MODULE path, 69
  - Delete menu command, 25
  - divide by zero exceptions, 114
  - DO loops
    - one trip, 79
  - DO Loops, 62
  - documentation conventions, 2
  - Edit
    - menu, 24
  - elemental functions, 178
  - elemental subroutines, 182
  - enabling the exception*, 114
  - Enabling/Disabling Menu Items, 125
  - end-of-file character, 118
  - Environment Preferences, 19
  - errors, 58
    - at runtime, 103
  - event loop, 121
  - exceptions
    - divide by zero, 114
    - operand error, 114
    - overflow, 114
  - extensions
    - key Microsoft FORTRAN, 104
    - key Sun FORTRAN ones, 106
    - key VAX FORTRAN ones, 101
    - key VS FORTRAN ones, 103
  - extensions to FORTRAN 77, 2
  - external procedure name, 110
  - f77.exe, 140
  - File
    - menu, 21
  - Find menu command, 26
  - Find Next menu command, 26
  - FIXED directive, 70
  - FIXEDFORMLINESIZE directive, 71
  - floating point
    - unit, 114
  - floating point unit
    - exception handling, 114
    - rounding direction, 113
  - FORM='BINARY' specifier, 105
  - Format menu, 28
  - Format Preferences, 17
    - Color, 18
    - Language, 17
  - Fortran 77
    - introduction, 1
    - options, 73
  - FORTRAN 77 extensions, 2
  - Fortran 77 Fixed Source Form, 82
  - Fortran 90 Fixed Source Form, 65
  - Fortran 90 Free Source Form, 64, 82

- Fortran 95
  - options, 57
- FORTRAN math routines
  - calling from C, 165
- FPU exception handling, 44
- FPU rounding mode, 43
- fr.exe, 140
- frameworks, 93
- FREE[FORM] directive, 70
- Fsplit utility tool, 141
- function
  - call to C from FORTRAN, 163
  - call to FORTRAN from C, 164
- g77, 160
- gcc, 160
- Go to Line menu command, 30
- graying of text, 2
- Help menu, 33
- ibe.exe, 140
- IBM RS/6000, porting from, 106
- IDATE subroutine, 102
- IEEE floating point math, 114
- IEEE\_ALL, 182
- IEEE\_CLASS, 178
- IEEE\_CLASS\_TYPE, 180
- IEEE\_COPY\_SIGN, 178
- IEEE\_DATATYPE, 175
- IEEE\_DENORMAL, 175
- IEEE\_DIVIDE, 175
- IEEE\_DIVIDE\_BY\_ZERO, 179, 182
- IEEE\_DOWN, 177
- IEEE\_FEATURES\_TYPE, 175, 176
- IEEE\_FLAG\_TYPE, 181
- IEEE\_GET\_FLAG, 181, 182
- IEEE\_GET\_HALTING\_MODE, 182
- IEEE\_GET\_ROUNDING\_MODE, 180
- IEEE\_GET\_STATUS, 183
- IEEE\_GET\_UNDERFLOW\_MODE, 180
- IEEE\_HALTING, 175
- IEEE\_INEXACT, 182
- IEEE\_INEXACT\_FLAG, 175
- IEEE\_INF, 175
- IEEE\_INVALID, 182
- IEEE\_INVALID\_FLAG, 175
- IEEE\_IS\_FINITE, 179
- IEEE\_IS\_NAN, 179
- IEEE\_IS\_NEGATIVE, 179
- IEEE\_IS\_NORMAL, 179
- IEEE\_LOGB, 179
- IEEE\_NAN, 175
- IEEE\_NEAREST, 177
- IEEE\_NEGATIVE\_DENORMAL, 176
- IEEE\_NEGATIVE\_INF, 176
- IEEE\_NEGATIVE\_NORMAL, 176
- IEEE\_NEGATIVE\_ZERO, 176
- IEEE\_NEXT\_AFTER, 179
- IEEE\_OVERFLOW, 182
- IEEE\_POSITIVE\_DENORMAL, 176
- IEEE\_POSITIVE\_INF, 176
- IEEE\_POSITIVE\_NORMAL, 176
- IEEE\_POSITIVE\_ZERO, 176
- IEEE\_QUIET\_NAN, 176
- IEEE\_REM, 179
- IEEE\_RINT, 180
- IEEE\_ROUND\_TYPE, 176
- IEEE\_ROUNDING, 175
- IEEE\_SCALB, 180
- IEEE\_SELECTED\_REAL\_KIND, 181
- IEEE\_SET\_FLAG, 183
- IEEE\_SET\_HALTING\_MODE, 183
- IEEE\_SET\_ROUNDING\_MODE, 180
- IEEE\_SET\_STATUS, 183
- IEEE\_SET\_UNDERFLOW\_MODE, 181
- IEEE\_SIGNALING\_NAN, 176
- IEEE\_SQRT, 175
- IEEE\_STATUS\_TYPE, 181
- IEEE\_SUPPORT\_DATATYPE, 177, 178
- IEEE\_SUPPORT\_DENORMAL, 177
- IEEE\_SUPPORT\_DIVIDE, 177
- IEEE\_SUPPORT\_INF, 177
- IEEE\_SUPPORT\_IO, 177
- IEEE\_SUPPORT\_NAN, 178
- IEEE\_SUPPORT\_ROUNDING, 178
- IEEE\_SUPPORT\_SQRT, 178
- IEEE\_SUPPORT\_STANDARD, 178
- IEEE\_SUPPORT\_UNDERFLOW, 178
- IEEE\_SUPPORT\_UNDERFLOW\_CONTROL, 180, 181
- IEEE\_TO\_ZERO, 177
- IEEE\_UNDERFLOW, 182
- IEEE\_UNDERFLOW\_FLAG, 175
- IEEE\_UNORDERED, 180
- IEEE\_UP, 177
- IEEE\_USUAL, 182
- IEEE\_VALUE**, 180
- IMSL libraries, 96
- include tree to stderr, 87
- input from keyboard, 117
- inquiry functions, 177
- Insert Continuation menu command, 30
- Intel 386, porting from, 106
- interfacing FORTRAN and C
  - calling FORTRAN math routines, 165
  - compatible type declarations, 159
  - function call to C from FORTRAN, 163
  - function call to FORTRAN from C, 164
  - function results, 163
  - LOC function, 163
  - passing an array, 163
  - passing pointers, 161
  - passing strings, 164
  - passing values, 161
  - reference parameters, 161
    - passing to C, 162
    - passing to FORTRAN, 161
  - VAL function, 162
  - value parameters, 162
- intrinsic functions
  - LOC, 163
  - math, 165
  - VAL, 162
- italicized text, defined, 2

# Index

---

- K & R C source format, 90
- key equivalent, 13
- keyboard input, 117
- kind function, 181
- Language Systems Fortran, 107
- LAPACK library, 96
- Launching OTHER APPLICATIONS, 126
- libmrwe.a file, 115
- library path specification, 45
- library specification, 45
- linefeed character, removing, 105
- Link Large Data Stubs, 40
- Linker options, 91
- LOC, intrinsic function, 163
- MacFortran, 108
- MacFortran II, 108
- MacFortran/020, 108
- Match Brackets menu command, 30
- math routines
  - FORTTRAN, 165
- max errors, 86
- memory management, 109
- menu
  - adding, 122
  - checkmarks, 125
  - deleting, 124
  - deleting an item, 124
  - item, enabling and disabling, 125
- menus
  - Edit, 24
  - File, 21
  - Format, 28
  - Help, 33
  - Tools, 31
  - Window, 32
- metacommands, Microsoft FORTRAN, 105
- Microsoft FORTRAN
  - metacommands, 105
  - porting from, 104
- module files, 59
- MRWE
  - customization, 120
  - Event Loop, 121
  - programming, 120
- MRWE preferences, 138
- mrwe\_AddMenu, 122
- mrwe\_DoMenu, 124
- MS-DOS files, translating, 105
- MS-DOS, porting from, 104
- Multiple Windows, 136
- NAME directive, 70
- naming conventions, 165
- New menu command, 21
- New Window menu command, 32, 33
- Next bookmark, 27
- Next Error menu command, 32
- NeXT, porting from, 106
- no inlines, 87
- NOFREEFORM directive, 71
- non ANSI warnings, 75
- non-elemental subroutines, 180, 183
- normal optimizations, 41
- NOUNROLL directive, 72
- one trip DO, 79
- open additional windows, 136
- Open menu command, 22
- operand error exceptions, 114
- options, 169
- options, manual convention, 2
- other porting issues, 108
- overflow exceptions, 114
- PACK[ON] directive, 71
- PACKOFF directive, 72
- page setup dialog, 118
- Page Setup menu command, 23, 24
- Paste menu command, 25, 119
- pause after the program ends, 139
- Plug-Ins, 94
- porting code, 101
  - doesn't run correctly,, 103
  - from IBM RS/6000, 106
  - from Intel 386, 106
  - from Microsoft FORTRAN, 104
  - from MS-DOS, 104
  - from NeXT, 106
  - from SCO Unix, 106
  - from Sparc, 105
  - from Sun FORTRAN, 105
  - from VAX FORTRAN, 101
  - from VS FORTRAN, 103
- PowerPC options, 45
- ppcbe.exe, 140
- preconnected units, 117
- Preferences menu command, 17
- Previous bookmark, 27
- Previous Error menu command, 31, 32
- Print menu command, 23
- procedure naming conventions, 166
- profiling, 167
- Properties menu command, 28
- qualifiers, VAX FORTRAN, 102
- quiet, 87
- Quit menu command, 119
- RAN function, 102
- rc, resource compiler, 140
- Receiving Apple Events, 131
- relocatable object, 45
- removing linefeed character, 105
- Required Apple Events, 129
- Rever menu command, 23
- road maps*, 2
- RS/6000, porting from, 106
- running compiled applications, 5
- runtime error messages, 103
- Save All menu command, 23
- Save As menu command, 22
- Save menu command, 22
- SCO Unix, porting from, 106
- scriptable application, 135
- SECNDS subroutine, 102
- Select All menu command, 25
- Sending Apple Events**, 130
- SetMrwePrefs, 138
- Shift Left menu command, 30
- show compiler progress, 60
- source line length, 65

- Sparc, Absoft compiler for, 106
- Sparc, porting from, 105
- square brackets, defined, 2
- STACK directive, 72
- stack trace, 40
- static storage, 62
- Stop menu command, 31
- string length, 58
- strings
  - passing FORTRAN to C, 164
- Sun FORTRAN, porting from, 105
- support, 193
- suppress all warning messages, 86
- suppress anachronism warning messages, 86
- suppress informational warning messages, 86
- suppress list of compiler warning messages, 59
- suppress template warning messages, 87
- suppress warning messages, 86
- TABSIZE variable, 142
- target file, updating, 143
- technical support, 193
- Terminal, 185
- Text characteristics, 139
- text selection, 13
- Tile Horizontally menu command, 33
- Tile Vertically menu command, 33
- TIME subroutine, 102
- Toggle bookmark, 27
- Tools menu, 31
- Tools Preferences, 21
- tracing, 60, 87
- treat anachronisms as errors, 87
- undefine symbol, 93
- underlined text, defined, 2
- Undo menu command, 25, 119
- UNIX libraries, 97
- UNROLL directive, 72
- updating target file, 143
- use temporary files, 87
- using compilers, 14
- VALUE statement, 162
- VAST pre-processor, 95
- VAX compatibility, 78
- VAX FORTRAN
  - porting from, 101
  - qualifiers, 102
- VAX Tab-Format source, 82
- VAX/VMS libraries, 97
- verbose, 87
- verbose templates, 87
- VS FORTRAN, porting from, 103
- wide source format, 82
- Window characteristics, 139
- Window menu, 32
- Window menu command, 33
- Y2K bug, 3