Pro Fortran
Windows™
User Guide
For 32-bit and 64-bit Windows
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 Map ..................................................................... 2
Windows Programming Road Map .................................................. 3

Year 2000 Problem .......................................................................... 3
Fortran 90 DATE_AND_TIME Subroutine ...................................... 3
Unix Compatibility Library .............................................................. 4

CHAPTER 2 GETTING STARTED .................................................. 5

Compiling Basics ............................................................................ 5

Application Basics ......................................................................... 10

CHAPTER 3 USING THE ABSOFT EDITOR ................................. 11

The Absoft Editor ........................................................................... 11
Tab Indenting ................................................................................. 11
Text Selection ................................................................................. 12
Insert/Overstrike ......................................................................... 12
Bookmarks .................................................................................. 12
Using Compilers ......................................................................... 12
Pop-up menus ............................................................................. 13

Creating New Source Files ........................................................... 15
Manipulating Windows .................................................................. 16

Using the Editor Menus ................................................................. 16
File Menu ...................................................................................... 16
New (Ctrl+N) ............................................................................. 16
Open (Ctrl+O) ............................................................................ 16
Close (Ctrl+F4) .......................................................................... 16
Close All .................................................................................. 17
Save (Ctrl+S) ............................................................................ 17
Save As ..................................................................................... 17
Save All .................................................................................. 17
Revert ....................................................................................... 17
Preferences ............................................................................... 17
Format ....................................................................................... 18
Environment ............................................................................. 20
CHAPTER 4 USING THE COMPILERS

Compiling Programs ................................................................. 35

Using the Command Line ........................................................ 35

Using the Absoft Developer Tools Interface ................................ 36
  Working with Projects ......................................................... 36
  Options Property Sheet ...................................................... 37
  Target Property Page – General Options .............................. 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absoft Developer Tools Interface</td>
<td>44</td>
</tr>
<tr>
<td>File Menu</td>
<td>44</td>
</tr>
<tr>
<td>New</td>
<td>44</td>
</tr>
<tr>
<td>Open</td>
<td>44</td>
</tr>
<tr>
<td>Close</td>
<td>44</td>
</tr>
<tr>
<td>Save</td>
<td>44</td>
</tr>
<tr>
<td>Save As</td>
<td>44</td>
</tr>
<tr>
<td>Preferences</td>
<td>45</td>
</tr>
<tr>
<td>Edit Menu</td>
<td>46</td>
</tr>
<tr>
<td>Cut (Ctrl+X)</td>
<td>46</td>
</tr>
<tr>
<td>Copy (Ctrl+C)</td>
<td>46</td>
</tr>
<tr>
<td>Paste (Ctrl+V)</td>
<td>46</td>
</tr>
<tr>
<td>Delete (Del)</td>
<td>46</td>
</tr>
<tr>
<td>View Menu</td>
<td>46</td>
</tr>
<tr>
<td>Toolbars</td>
<td>46</td>
</tr>
<tr>
<td>Status Bar</td>
<td>46</td>
</tr>
<tr>
<td>Previous (Ctrl+F6)</td>
<td>47</td>
</tr>
<tr>
<td>Next (Ctrl+F7)</td>
<td>47</td>
</tr>
<tr>
<td>Full Pathnames</td>
<td>47</td>
</tr>
<tr>
<td>Refresh (F5)</td>
<td>47</td>
</tr>
<tr>
<td>Configure Menu</td>
<td>47</td>
</tr>
<tr>
<td>Add/Remove File(s)</td>
<td>47</td>
</tr>
<tr>
<td>Set Include Path(s)</td>
<td>48</td>
</tr>
<tr>
<td>Set Project Options</td>
<td>48</td>
</tr>
<tr>
<td>File Options</td>
<td>48</td>
</tr>
<tr>
<td>Set Default Options</td>
<td>48</td>
</tr>
<tr>
<td>Atlas</td>
<td>49</td>
</tr>
<tr>
<td>Tools Menu</td>
<td>49</td>
</tr>
<tr>
<td>Search</td>
<td>49</td>
</tr>
<tr>
<td>Build</td>
<td>49</td>
</tr>
<tr>
<td>Rebuild All</td>
<td>49</td>
</tr>
<tr>
<td>Update Dependencies</td>
<td>49</td>
</tr>
<tr>
<td>Check Syntax</td>
<td>49</td>
</tr>
<tr>
<td>Compile</td>
<td>49</td>
</tr>
<tr>
<td>Edit</td>
<td>50</td>
</tr>
<tr>
<td>Preprocess</td>
<td>50</td>
</tr>
<tr>
<td>Touch</td>
<td>50</td>
</tr>
<tr>
<td>Clean</td>
<td>50</td>
</tr>
<tr>
<td>Stop</td>
<td>50</td>
</tr>
<tr>
<td>Execute</td>
<td>50</td>
</tr>
<tr>
<td>Debug</td>
<td>50</td>
</tr>
<tr>
<td>Command Prompt</td>
<td>50</td>
</tr>
<tr>
<td>Generate Makefile</td>
<td>50</td>
</tr>
<tr>
<td>Customize</td>
<td>51</td>
</tr>
<tr>
<td>Window Menu</td>
<td>51</td>
</tr>
<tr>
<td>Cascade</td>
<td>51</td>
</tr>
<tr>
<td>Tile Horizontally</td>
<td>51</td>
</tr>
</tbody>
</table>
Absoft Fortran 95 Options ................................................................. 62
General - F95 Options ................................................................. 63
  Warn of Non-Standard usage (-en) ............................................. 63
  Suppress warnings (-w) ............................................................ 63
  Suppress Warning number(s) (-Znn) ....................................... 63
  Quiet (-q) ............................................................................ 64
  Verbose (-v) .......................................................................... 64
  Warning level (-znn) ............................................................... 64
  Error Handling (-dq and -ea) .................................................. 64
  Output Version number (-V) .................................................... 64
  Default Recursion (-eR) .......................................................... 64
  Max Internal Handle (-T nn) .................................................... 65
  Temporary string size (-t nn) .................................................... 65

Absoft Fortran 77 Options ................................................................ 53
General - F77 Options ................................................................. 53
  Suppress Warnings (-w) .......................................................... 54
  Warn of non-ANSI Usage (-N32) ............................................ 54
  Quiet (-q) ............................................................................ 54
  Show Progress (-v) ............................................................... 54
  Check Array Boundaries (-C) .................................................. 54
  Conditional compilation (-x) ................................................... 54
  Max Internal Handle (-T nn) .................................................... 55
  Temporary string size (-t nn) .................................................... 55

Control - F77 Options ................................................................. 56
  Compiler Directives (-Dname=value) ........................................ 56

Compatibility - F77 Options .......................................................... 57
  Folding to Lower Case (-f) ....................................................... 57
  Static Storage (-s) ................................................................. 58
  Folding to Upper Case (-N109) ............................................... 58
  One-Trip DO Loops (F66) (-d) ............................................... 58
  Append underscore to names (-N15) ....................................... 58
  Promote REAL and COMPLEX (-N113) .................................. 58
  Integer Sizes (-i2 and -i8) ..................................................... 58

Format - F77 Options .................................................................. 59
  ANSI Fortran 77 Fixed ........................................................... 59
  Fortran 90 Free-Form (-8) ...................................................... 60
  VAX Tab-Format (-V) ............................................................ 60
  Wide Format (-W) ................................................................. 60
  Treat as Big-Endian (-N26) ..................................................... 60
  Treat as Little-Endian (-N27) .................................................. 60
  Escape Sequences in Strings (-K) .......................................... 60

COMMON Block - F77 Options ..................................................... 61
  Align COMMON Variables (-N34) .......................................... 61
  Set COMMON Block Name (-N22) ........................................ 61
  Don't Mangle COMMON Block Name (-N110) ......................... 62

Help Menu .................................................................................. 52
  Help Topics .......................................................................... 52

About Interface............................................................................. 52
Absoft Fortran 90/95 Compiler Directives

STACK Directive

FIXED Directive

FREE Directive

Ignore CDEC$ directives (Variable Names Case Sensitivity (-YALL NAMES={ASIS | UCS | LCS})

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

YDLL STDCALL={0|1}

YIL={AC90 ,AC77,LF90,MSVC,MSVB,BC,BD,WINAPI}

Disable Default Module File Path (-nodefaultmod)

Check Array Boundaries (-Rb)

Check Array Conformance (-Rc)

Check Substrings (-Rs)

Check Pointers (-Rp)

Character Argument Parameters (-YCFRL=[0|1])

Pointers Equivalent To Integers (YPEI=[0|1])

DVF/CVF Character Arguments (-YVF_CHAR)

Format - F95 Options

Free-Form (-f free)

Fixed-Form (-f fixed)

Alternate Fixed form (-f alt_fixed)

Fixed line length (-W nn)

YEXT NAMES={ASIS | UCS | LCS}

Treat as Big-Endian (-N26)

Treat as Little-Endian (-N27)

External Symbol Prefix (-YEXT_PFX=string)

External Symbol Suffix (-YEXT_SFX=string)

Escape Sequences in Strings (-YCSLASH=1)

No Dot for Percent (-YNDFP=1)

MS Fortran 77 Directives (-YMS7D)

Miscellaneous - F95 Options

COMMON Block Name Prefix (-YCOM_PFX=string)

COMMON Block Name Suffix (-YCOM_SFX=string)

COMMON Block Name Character Case (-YCOM NAMES={ ASIS | UCS | LCS })

Loop unrolling (-U and -H nn and -h nn)

Safe Floating-Point (-safep)

Add Microsoft GLOBAL prefix (-YMSFT_GLB_PFX)

Other F95 Options

SSE2 instructions (-msse2 and -mno-sse2)

Safe Floating-Point (-safep)

Conditional Compilation (-YX)

Check Argument Interface (-Ra)

Check Argument Count (-Rn)

Disable Default Module File Path (-nodefaultmod)

YIL=\{AC90 ,AC77,LF90,MSVC,MSVB,BC,BD,WINAPI\}

YDLL STDCALL=\{0|1\}

YDLL NAMES={ASIS | UCS | LCS}

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

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

Ignore CDEC$ directives (-YNO_CDEC)
Disables All Warnings (-w) ................................................................. 78
Disables Extensions (-Za) ............................................................... 79
Enable Runtime Debug Checks (-GZ) ............................................ 79
Force Stack Checking (-Ge) ............................................................ 79

Abssoft Linker Options ........................................................................ 79
General - Link Options ................................................................. 80
Stack Size (-stack:reserved[,commit]) ........................................ 80
Heap Size (-heap:reserved[,commit]) .......................................... 80
Map File: (-map:name) ................................................................. 80
Exports Files: (-exports:filename) ................................................ 81
Suppress Warnings: (-warn) ........................................................ 81
Verbose: (-verbose) ...................................................................... 81

Plug-ins ........................................................................................... 81
VAST ............................................................................................. 83
IMSL ............................................................................................. 83
LAPACK ....................................................................................... 84
BLAS ........................................................................................... 84
PLplot Library ............................................................................ 84

Using and Creating DLLs ................................................................... 84
DLL Compatibility ......................................................................... 85
CDECL ......................................................................................... 85
STDCALL ..................................................................................... 85

CHAPTER 5 PORTING CODE .......................................................... 87

Porting Code from VAX .................................................................... 87
Compile Time Options and Issues ................................................ 88

Porting Code from IBM VS FORTRAN ........................................... 90
Compile-time Options and Issues ................................................ 90

Porting Code From Microsoft FORTRAN ..................................... 90
Compile-time Options and Issues ................................................ 91

Porting Code from Sun Workstations .......................................... 92

Porting Code from the NeXT Workstation .................................... 92

Porting Code from the IBM RS/6000 Workstation ....................... 92

Porting Code From Macintosh Systems ........................................... 93
Language Systems Fortran ............................................................ 93
Distribution Issues ....................................................................... 93

Other Porting Issues ....................................................................... 94
Memory Management .................................................................... 94
File and Path Names ..................................................................... 94
Tab Character Size ...................................................................... 95
Runtime Environment ................................................................... 95
Floating Point Math Control ......................................................... 97
Rounding Direction ..................................................................... 97
Exception Handling .................................................................... 98
CHAPTER 6 BUILDING PROGRAMS

An Overview of Program Building ................................................................. 99
The Components of an Application ............................................................ 99
Working with Resources ............................................................................ 100

Creating Object Files .................................................................................. 100
Fsplits - Source Code Splitting Utility ......................................................... 101

Linking Programs ......................................................................................... 102
@ ........................................................................................................... 102
-ALIGN ................................................................................................. 103
-ALLOWBIND ......................................................................................... 103
-ALLOWISOLATION ............................................................................. 103
-ASSEMBLYDEBUG ............................................................................. 103
-ASSEMBLYLINKRESOURCE ............................................................... 103
-ASSEMBLYMODULE ......................................................................... 103
-ASSEMBLYRESOURCE ...................................................................... 103
-BASE .................................................................................................. 103
-CLRIMAGETYPE ............................................................................. 103
-CLRSUPPORTLASTERROR .................................................................. 103
-CLRTHREADATTRIBUTE ................................................................... 103
-CLRUNMANAGEDCODECHECK ......................................................... 103
-DEBUG ............................................................................................... 104
-DEF .................................................................................................... 104
-DEFAULTLIB ..................................................................................... 104
-DELAY ................................................................................................. 104
-DELAYLOAD ....................................................................................... 104
-DELAYSIGN ....................................................................................... 104
-DLL .................................................................................................... 104
-DRIVER ................................................................................................. 104
-ENTRY ................................................................................................. 104
-errorReport .......................................................................................... 104
-EXPORT .............................................................................................. 104
-FIXED .................................................................................................. 105
-FORCE ................................................................................................. 105
-FUNCTIONPADMIN ........................................................................... 105
-HEAP .................................................................................................. 105
-IDLOUT ............................................................................................... 105
-IGNOREIDL ......................................................................................... 105
-IMPLIB ................................................................................................. 105
-INCLUDE ............................................................................................... 105
-INCREMENTAL ................................................................................... 105
-KEYCONTAINER ............................................................................... 105
-KEYFILE ............................................................................................. 105
-LARGEADDRESSAWARE ..................................................................... 105
-LIBPATH ............................................................................................... 106
-LTCG ................................................................................................. 106
-MACHINE ............................................................................................ 106
-MANIFEST ............................................................................................ 106
-MANIFESTDEPENDENCY .................................................................. 106
-MANIFESTFILE (Name Manifest File) ................................................. 106
-MAP .................................................................................................... 106
-MAPINFO ............................................................................................ 106
-MERGE ................................................................................................. 106
-MIDL ................................................................................................... 106
-NOASSEMBLY ................................................................. 106
-NODEFAULTLIB .......................................................... 106
-NOENTRY ................................................................. 107
-NOLOGO ................................................................. 107
-NXCOMPAT .............................................................. 107
-OPT ................................................................. 107
-ORDER ................................................................. 107
-PDB ................................................................. 107
-PDBSTRIPPED ........................................................... 107
-PGD ................................................................. 107
-PROFILE .............................................................. 107
-RELEASE .............................................................. 107
-SAFESEH .............................................................. 107
-SECTION .............................................................. 108
-STACK ................................................................. 108
-STUB ................................................................. 108
-SUBSYSTEM ......................................................... 108
-SWAPRUN .............................................................. 108
-TLBID ................................................................. 108
-TLBOUT ............................................................... 108
-TSAWARE ............................................................. 108
-VERBOSE .............................................................. 108
-VERSION .............................................................. 108
-WX ................................................................. 108

Creating Libraries ........................................................................ 109
 DLL Import Libraries ...................................................................... 110
 Syntax of a Definition File ............................................................... 110
 Name Mangling ........................................................................... 112

Building Programs ....................................................................... 113
 The Elements of amake .................................................................. 114
 Using Macros ............................................................................... 114
   Advantages of using macros ......................................................... 114
   Defining macros ......................................................................... 115
   Special macros ......................................................................... 116
   Cautions in using macros ............................................................ 116
 Using Description Files ................................................................... 117
   Working with dependency blocks .................................................. 117
   Defining a target more than once .................................................. 118
   Using include directives .............................................................. 118
   A sample description file ............................................................ 119
 Using Dependency Rules ................................................................. 119
   The default rules ......................................................................... 120
   Creating your own rules .............................................................. 122
 amake Usage and Syntax ............................................................... 123
   Special Targets ......................................................................... 124
   Dummy Files ............................................................................. 125
   Environment Variables ................................................................ 125
   Example: Rebuilding an Executable File ........................................ 126
 Error Handling and Cautions ............................................................ 127
   Syntax Errors ............................................................................. 127
   Other Common Errors ................................................................ 128
   Cautions ................................................................................... 128

Absoft Fortran 90/95 ...................................................................... 129
 Compiler Directives ......................................................................... 129
CHAPTER 7 WINDOWS PROGRAMMING...........135

Using MRWE............................................................................................................................135
The MRWE Window..................................................................................................................136
How Your Program And MRWE Work Together .......................................................................137
Working With Text in MRWE .......................................................................................................138
  mrwe_clear function ..................................................................................................................139
Working With Graphics in MRWE ...............................................................................................139
Adding Menu Commands to MRWE ............................................................................................141
  mrwe_InsertMenu function .........................................................................................................141
  mrwe_AppendMenu function .......................................................................................................142
  mrwe_CheckMenuItem subroutine ...............................................................................................142
  mrwe_EnableMenuItem subroutine .............................................................................................142
  mrwe_DeleteMenu function .......................................................................................................143
Using Timer Functions in MRWE ................................................................................................143
Using The MRWE Default Menus ...............................................................................................144
File Menu....................................................................................................................................144
  Save.........................................................................................................................................144
  Save As.................................................................................................................................144
  Print.......................................................................................................................................144
  About....................................................................................................................................144
  Exit......................................................................................................................................144
Edit Menu....................................................................................................................................144
  Undo (Ctrl+Z)... .......................................................................................................................144
  Cut (Ctrl+X)..........................................................................................................................145
  Copy (Ctrl+C)........................................................................................................................145
  Paste (Ctrl+V)........................................................................................................................145
  Delete....................................................................................................................................145
Format Menu................................................................................................................................145
  Font.......................................................................................................................................145
  Color ...................................................................................................................................145
Window Menu.............................................................................................................................145
  Cascade (Shift+F5)... ...............................................................................................................146
  Tile (Shift+F6)........................................................................................................................146
  Arrange Icons........................................................................................................................146
Modifying MRWE .......................................................................................................................146

Using The Win32 API .................................................................................................................146
Program Organization: FORTRAN VS. WINDOWS .................................................................147
How Absoft Fortran 90 Interfaces with Windows .......................................................................148
Locating the Interface Files .........................................................................................................148
Passing Arguments ....................................................................................................................148
  CHARACTER Arguments .......................................................................................................150
INTERFACING WITH C

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPLICIT NONE Statement</td>
<td>151</td>
</tr>
<tr>
<td>Using Derived TYPEs, STRUCTUREs and RECORDs with Win32</td>
<td>152</td>
</tr>
<tr>
<td>Passing Procedure Pointers to Win32 Routines</td>
<td>153</td>
</tr>
<tr>
<td>Handles</td>
<td>154</td>
</tr>
<tr>
<td>Fortran Data Types in C</td>
<td></td>
</tr>
<tr>
<td>Passing arguments Between C and FORTRAN</td>
<td>156</td>
</tr>
<tr>
<td>Reference Parameters</td>
<td>157</td>
</tr>
<tr>
<td>Value Parameters</td>
<td>158</td>
</tr>
<tr>
<td>Indirection (the LOC Function)</td>
<td>159</td>
</tr>
<tr>
<td>Function Results</td>
<td>159</td>
</tr>
<tr>
<td>A Call to C from FORTRAN</td>
<td>160</td>
</tr>
<tr>
<td>A Call to FORTRAN from C</td>
<td>160</td>
</tr>
<tr>
<td>Passing Strings to C</td>
<td>160</td>
</tr>
<tr>
<td>Calling FORTRAN Math Routines</td>
<td>161</td>
</tr>
<tr>
<td>Calling FORTRAN I/O Routines</td>
<td>162</td>
</tr>
<tr>
<td>Naming Conventions</td>
<td>162</td>
</tr>
<tr>
<td>Procedure Names</td>
<td>162</td>
</tr>
<tr>
<td>Accessing COMMON Blocks from C</td>
<td>162</td>
</tr>
<tr>
<td>Declaring C Structures In FORTRAN</td>
<td>163</td>
</tr>
</tbody>
</table>

APPENDIX A ABSOFT COMPILER OPTION GUIDE

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absoft Compiler Options</td>
<td>165</td>
</tr>
<tr>
<td>Floating Point Unit Control Options</td>
<td>165</td>
</tr>
<tr>
<td>FORTRAN 90/95 General Options</td>
<td>165</td>
</tr>
<tr>
<td>FORTRAN 90/95 Compatibility Options</td>
<td>166</td>
</tr>
<tr>
<td>FORTRAN 90/95 Format Options</td>
<td>166</td>
</tr>
<tr>
<td>FORTRAN 90/95 Miscellaneous Options</td>
<td>167</td>
</tr>
<tr>
<td>FORTRAN 90/95 Other Options</td>
<td>167</td>
</tr>
<tr>
<td>FORTRAN 77 General Options</td>
<td>167</td>
</tr>
<tr>
<td>FORTRAN 77 Control Options</td>
<td>167</td>
</tr>
<tr>
<td>FORTRAN 77 Compatibility Options</td>
<td>167</td>
</tr>
<tr>
<td>FORTRAN 77 Format Options</td>
<td>168</td>
</tr>
<tr>
<td>FORTRAN 77 COMMON Options</td>
<td>168</td>
</tr>
<tr>
<td>FORTRAN 77 Other Options</td>
<td>168</td>
</tr>
</tbody>
</table>
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 Windows, Macintosh 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. To facilitate this process, certain Absoft Fortran implementations are object code compatible with C/C++ objects, allowing users to create mixed Fortran/C applications from within a single development environment.

Absoft Fortran is a native 32-bit application designed for Windows™ NT/2000/XP/Vista.

Absoft Fortran implementations include all of the tools necessary for you to create stand-alone, double-clickable Windows applications. The purpose of this User Guide is to offer step-by-step instructions on the operation of each compiler, writing, compiling, debugging, linking and running your program. MRWE, Absoft's application framework, can automatically build a standard Windows interface for each compiled application. MRWE is written entirely in Fortran and the complete source code is included as an example for Windows programming in Fortran. Graphics routines, various libraries, and utilities are also provided. Their operation is explained in various appendices.

Absoft Fortran 90/95

A complete ANSI Fortran 90/95 implementation plus 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 the Win32 API directly from Fortran and is capable of building DLLs. Several popular VAX and workstation extensions have also been added.

Absoft FORTRAN 77

Refined over 15 years, with emphasis on porting legacy code from workstations, Absoft Fortran 77 is full ANSI 77 with 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 which 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 so existing, extended FORTRAN 77 routines can be easily compiled and linked with new Fortran 90/95 code. The entire Win32 API is supported and DLLs can be created directly from Fortran.
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 is used for emphasis and book titles.
- On-screen text such as menu titles and button names look the same as in pictures and on the screen (e.g. the File menu).
- The modifier keys on PC keyboards are Shift, Alt, and Control. They are always used with other keys and are referenced as in the following:
  - Shift-G press the Shift and ‘G’ keys together
  - Alt-F4 press the Alt and F4 function keys together
  - Control-C press the Control and ‘C’ keys together
- 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 write programs with Absoft Fortran for Windows, there are a number of other manuals that describe FORTRAN extensions and the Windows programming environment in further detail. The two road maps in this chapter will guide you to these manuals for introductory or advanced reference. The bibliography in the Appendices lists further information about each manual.

FORTRAN Road Map

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

![FORTRAN language road map]

**Windows Programming Road Map**

Absoft Fortran 90/95 and FORTRAN 77 each provide complete access directly to the Windows System Services routines. The Windows Programming chapter of this manual describes the interface to these routines, but does not describe each of the hundreds of routines available. The Win32 SDK Help command in the Absoft Pro Fortran menu describes most of the Win32 API functions. Programmers wishing to make use of these routines to add graphics to their programs or to extend the user interface provided by MRWE may wish to obtain additional documentation on the Win32 programming model.

**YEAR 2000 PROBLEM**

All versions of Absoft Pro Fortran products for Macintosh, Power Macintosh, Windows 95/98, Windows NT/2000/XP, Linux, and UNIX 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.

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 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 *Support Library Guide* 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 Windows: 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.

<table>
<thead>
<tr>
<th>TO DO THIS...</th>
<th>TURN TO THIS SECTION...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the editor</td>
<td>Using the Absoft Editor, Chapter 3</td>
</tr>
<tr>
<td>Use the compiler</td>
<td>Using the Compilers, Chapter 4</td>
</tr>
<tr>
<td>Port from other platforms</td>
<td>Porting Code, Chapter 5</td>
</tr>
<tr>
<td>Create applications</td>
<td>Building Programs, Chapter 6</td>
</tr>
<tr>
<td>Program Windows</td>
<td>Windows Programming, Chapter 7</td>
</tr>
<tr>
<td>Debug programs</td>
<td>FX Debugger Manual</td>
</tr>
</tbody>
</table>

Road map for experienced users

COMPILING BASICS

The Absoft compilers can be run either from a command line or from the Absoft Developer Tools Interface. This chapter describes how to use the Developer Tools Interface —the command line interface is described in the Chapter 4, Using the Compilers.

Note: Throughout this chapter and the rest of the manual, it is assumed that the compiler has been installed on the C: drive. If this is not the case, substitute the correct drive letter in the examples as appropriate.

Selecting Programs from the Start menu and then Developer Tools Interface from the Absoft submenu opens the Absoft Developer Tools Interface. It can also be started by selecting Run... from the Start menu and typing c:\Absoft10\bin\Atools or by typing Atools at a console or command line window. The Windows Command Prompt or should be opened by selecting Programs from the Start menu and then Development Command Prompt from the Absoft submenu. Opening the console window in this way automatically sets the environment variables by executing the c:\Absoft10\bin\absvar.bat batch file.
During the installation process, several example programs were placed in the \texttt{c:\Absoft\examples} directory. The example program used in this tutorial is \texttt{hello.f}. Follow the tutorial on the following 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:

<table>
<thead>
<tr>
<th>What to do</th>
<th>How to do it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke the Absoft Developer Tools Interface.</td>
<td>Select \texttt{Programs} from the \texttt{Start} menu, then select \texttt{Developer Tools Interface} from the \texttt{Absoft Pro Fortran} submenu.</td>
</tr>
</tbody>
</table>

\textbf{Note:} The first time you run the interface, it may ask you if you want certain standard file extensions to be associated with the Absoft Editor. This will allow you to automatically open the editor by double-clicking on files with these extensions. You can choose to have this association established at this time, or defer the decision to later.

The Absoft Developer Tools Interface is project oriented, so the first thing you must do is to establish a name and location for your project.

<table>
<thead>
<tr>
<th>What to do</th>
<th>How to do it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the project name and location.</td>
<td>Type \texttt{hello} in the \texttt{Name} box. Change the \texttt{Location} to the \texttt{Examples} directory by typing the path, or using the browse button to navigate to it.</td>
</tr>
</tbody>
</table>

New Project dialog box
Immediately after naming the project and specifying its location, the project options property page is displayed. Use this property page to set project wide options. The only change you will want to make here is to set the target type to **MRWE Application**. MRWE is the Microsoft Runtime Windows Environment. It provides an automatic Windows interface for your program with menus, a scrollable text window for program output, and the ability to print.

<table>
<thead>
<tr>
<th>What to do</th>
<th>How to do it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the project <strong>Target Type</strong> to <strong>MRWE Application</strong> (a Windows program).</td>
<td>Click on the <strong>MRWE Application</strong> button in the upper left corner.</td>
</tr>
</tbody>
</table>

![Project options dialog box](image)

Project options dialog box
The next step is to specify the file (or files) that the project consists of:

<table>
<thead>
<tr>
<th>What to do</th>
<th>How to do it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add the file <strong>Hello.f</strong> to the project.</td>
<td>Click the <strong>OK</strong> button on the project options dialog. The file section dialog will open automatically</td>
</tr>
</tbody>
</table>

If you are not already in the **Examples** folder, browse to that directory. Click on the file named **Hello.f** and add it to the lower **File(s) Added** box by clicking on the **Add** button. Alternately, double-click on **Hello.f** and it will appear in the lower box.

Click on the **Close** button to dismiss this dialog and the project sources window will appear. This window maintains all of the files in your project. Each file type will be kept in a separate folder. If you wish, you can also manage the files in your project directly from this window; you delete selected files and drag new files into this window.
The last step is to build (compile) your application:

<table>
<thead>
<tr>
<th>What to do</th>
<th>How to do it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile the source file <strong>Hello.f</strong> into the application file <strong>Hello.exe</strong>.</td>
<td>Choose the <strong>Build</strong> command from the <strong>Tools</strong> menu.</td>
</tr>
</tbody>
</table>

The compiler will then create **hello.exe** from **Hello.f**. More detailed information concerning the creation of an application can be found in the chapter **Using the Compilers**.
APPLICATION BASICS

<table>
<thead>
<tr>
<th>What to do</th>
<th>How to do it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute the compiled</td>
<td>Choose the Execute</td>
</tr>
<tr>
<td>application.</td>
<td>command from the Tools</td>
</tr>
<tr>
<td></td>
<td>menu.</td>
</tr>
</tbody>
</table>

You can also select Run... from the Start menu, browse to the c:\Absoft\examples directory, select hello.exe, and run it or double-click on the application icon in an Explorer window.

Additional examples that may be helpful in writing Fortran 90/95 or FORTRAN 77 programs can be found in the c:\Absoft\examples directory. 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 Windows™ 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).

Tab Indenting

When one complete line of text or more is selected, the tab key can be used for indenting the block of text as follows:

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>shift the block right one tab stop</td>
</tr>
<tr>
<td>Ctrl+Tab</td>
<td>shift the block right one space</td>
</tr>
<tr>
<td>Shift+Tab</td>
<td>shift the block left one tab stop</td>
</tr>
<tr>
<td>Shift+Ctrl+Tab</td>
<td>shift the block left one space</td>
</tr>
</tbody>
</table>
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 left mouse button down. For larger amounts of text, click the left mouse button at the beginning of the selection, hold the Shift key down, and then click the left mouse button again at the end of the selection.

Insert/Overstrike
Text may be entered in either insert or overstrike mode. Insert mode is the normal, default mode. Overstrike mode is entered by pressing the Insert key on your keyboard and is shown with the OVR indicator on the status bar.

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 Ctrl+F2 sets (or unsets) a bookmark. In other words, Ctrl+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. Typing Shift+Ctrl+F2 clears all bookmarks in the file.

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 Properties sheet for the file (also described later) or in the Option Toolbar.

The Option Toolbar appears below the editor’s menu bar:

![Compilertools.png](attachment:Compilertools.png)

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

Clicking the right mouse button with the cursor positioned over an open file window will display a pop-up menu of context sensitive commands:

<table>
<thead>
<tr>
<th>Open graph.inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check Syntax</td>
</tr>
<tr>
<td>Match Brackets</td>
</tr>
<tr>
<td>Insert Continuation</td>
</tr>
<tr>
<td>Comment Lines</td>
</tr>
<tr>
<td>Uncomment Lines</td>
</tr>
<tr>
<td>Toggle Bookmark</td>
</tr>
<tr>
<td>New Bookmark</td>
</tr>
<tr>
<td>Cut</td>
</tr>
<tr>
<td>Copy</td>
</tr>
<tr>
<td>Paste</td>
</tr>
<tr>
<td>Properties</td>
</tr>
</tbody>
</table>

**Open**

If the insertion caret is positioned on an include statement, the file named in the include statement can be opened with this command if it exists in the current directory. If an include file with the same root file name as the source file exists in the current directory, you can use this command to open it.

**Check Syntax**

This is a shortcut for the **Tools** menu **Check Syntax** command. It will check the syntax of your file without compiling it. For more information, see **Check Syntax** in the **Tools** menu described later in this chapter.

**Match Brackets**

This is a shortcut for the **Match Brackets** command in the **View** menu. See **Match Brackets** in the **View** menu later in this chapter for details.

**Insert Continuation**

Use this command to insert a Fortran continuation line immediately after the line on which the cursor is positioned. It is a shortcut for the **Insert Continuation** command in the **View** menu. See **Insert Continuation** in the **View** menu later in this chapter for details.

**Comment Lines**

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

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

Toggle Bookmark

Use this command to set or delete a bookmark on the line containing the insertion caret.

Next Bookmark

This command moves the insertion caret to next bookmark in the file.

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.

Properties

This command opens the Source Properties dialog allowing you to control certain display characteristics of your file. See Properties in the Edit menu discussed later in this chapter for more information.
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 Windows-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 cut and paste text between files. File windows have a Title Bar, with a System menu in the upper left corner, and vertical and horizontal Scroll Bars.

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 **File**, **Edit**, **View**, **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.

**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… (Ctrl+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.

**Open… (Ctrl+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.

**Close (Ctrl+F4)**

Fortran User Guide
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 (Ctrl+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.

**Preferences...**

The `Preferences` command opens a property sheet that is used to configure the Absoft Editor to your specific editing needs. Preferences consist of three separate tabbed property pages: `Format`, `Environment`, and `Tools`. 
Format

The Format page 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.

Color

Use this section to change the default use of color, and the actual color in the text, of Comments, Intrinsics, Keywords, and literal Strings 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 colorations

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 new files of the currently selected language type (see Properties in the Edit menu discussed later in this chapter). You can choose to have Line Numbers, Invisibles, and Coding Form 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 Properties command in the Edit menu to set properties for individual source files.
Environment

The Environment page 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 check 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`.

Retain undo/redo buffers

This check box determines if the Absoft Editor will preserve undo/redo buffers across `Save` commands.

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.

Max MRU's

An MRU is a Most Recently Used file - "Used" in the sense that it was opened for editing. A list of the most recently used files appears in the `File` menu and you can immediately open one by selecting it from the menu. Use the `MRU's` edit box to set the number (up to 16) of MRU's the Absoft Editor will track.

Auto match brackets

Select this button to see a visual indication of matching bracket pairs either by inserting new brackets or moving the insertion caret over an existing bracket.
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.

![Preferences dialog box](image)

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

- **Print (Ctrl+P)**
  
  This command prints the front-most window to the currently selected printer.

- **Print Preview**
  
  The **Print Preview** command opens a preview window showing you how the text in the front-most window will appear when printed using the currently selected printer and printer setup.
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. Additionally, this dialog allow you to 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 the level of color printing supported. Select the Print coding form check box if you would also like to print any visible onscreen coding forms when printing your documents.

Most Recently Used Files

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.

Exit

Use this command to quit the Absoft Editor. If any windows have been not been saved, you will be prompted to save them before finishing.
Edit Menu

The commands in the **Edit** menu are used for performing standard editor functions, such as Cut, Copy, and Paste. There is also a Properties command for specifying individual editing preferences for the front-most window.

**Undo (Ctrl+Z)**

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

**Redo (Ctrl+Y)**

Use this command to redo the last change made in the front-most window.

**Cut (Ctrl+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 (Ctrl+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 (Ctrl+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.

**Delete (Del)**

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**

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

Use this submenu to select formatting related commands which will apply to the selected line or text in the file displayed in the front-most window. This submenu gives you control over indenting source lines by shifting lines left or right, the ability to comment or uncomment lines of code, and the ability to convert selected text to either lower case or upper case.

Bookmarks Menu

Use this submenu to select line marking related commands that will apply to the selected in the file displayed in the front-most window. This submenu gives you menu access to the various marking commands such as:

- **Toggle Bookmark (Ctrl+F2)**: toggle bookmark for the current line
- **Next Bookmark (F2)**: display next bookmark in the current file
- **Previous Bookmark (Shift+F2)**: display previous mark in the current file
- **Clear File’s Bookmarks (Shift+Ctrl+F2)**: clear all bookmarks in the current file
- **Clear All Bookmarks (Shift+Alt+F2)**: clear all bookmarks regardless of file
Properties… (Alt+Enter)

The Properties dialog is used to control certain characteristics of the display of the file in the front-most window. The Source Properties dialog can also be opened by clicking the right mouse button with cursor positioned over an open file window and choosing Properties from the pop-up menu which is displayed: The characteristics that can be controlled include:

![Source Properties dialog]

**Line Numbers**

Use this check box to control whether line numbers are displayed or not.

**Invisibles**

This check box determines whether non-printing characters, such as tabs, carriage returns, and line-feeds are displayed graphically along with the printable text.

**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**.
Language

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

Source 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.

File Format

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.

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.

Auto Indent

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

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.

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 property sheet (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.

Path

This field displays the full absolute pathname of the active document.
View Menu

The commands in the View menu allow you to turn the Status Bar and Toolbars on or off and to customize the toolbar. There are also commands for finding and replacing text and text patterns.

Toolbars

Use this command to show or hide Toolbars. The Toolbars are located directly beneath the menu bar and contain various buttons that can be used as shortcuts for some of the more commonly used menu commands. The Toolbars are dockable and may be detached and repositioned wherever convenient within the editor's frame. The Toolbars can be customized by adding, removing, and reorganizing the buttons. See the Customize... command below.

Status Bar

This command is used to show or hide that Status Bar located at the bottom of the Absoft Editor window. The Status Bar indicates the state of certain keys such as “caps lock” and “num lock”, and “read only”. It also indicates the line and column numbers of the insertion caret and the format of the source file (Dos, Unix, or Mac). Clicking on the clock toggles it between displaying the time or the date.

Find… (Ctrl+F)

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

The controls in the Find dialog are used as follows:

Find what
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.

**Match Case**

Check this box to restrict the Find command to text occurrences in your source file that match your specified text exactly in case.

**Direction**

Click on the appropriate radio button in the Direction box to indicate your choice for search direction. **Up** means towards the beginning of the file from the current position of the cursor. **Down** means towards the end of the file from the current position of the cursor.

**Find Next (F3)**

This command repeats the last Find command in the front-most window.

**Replace...**

This command is used to find and replace text. It displays a dialog box similar to that for the Find command with a few additions.

**Go to Line (Ctrl+L)**

This command opens the Go To dialog. Enter the line number of the line you wish to go to and click on the OK button.
Match Brackets (Ctrl+M)

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, and brackets. Double-clicking on an opening or closing character will select the text that the characters enclose.

Insert Continuation (Ctrl+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 property sheet. See Preferences in the Menu, described earlier for more information.

Previous Error (Ctrl+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 (Ctrl+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.

Refresh (F5)

If you have made changes to your source file, you can use this command to re-parse your source file and update the related file navigation information that is displayed in the Goto toolbar.
**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.

**Check Syntax (Ctrl+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.

**Compile**

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.

**Note:** If the file in the front-most window is a makefile (the extension is .amk), this command becomes **Build**.

**Stop**

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

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

You can even create your own custom toolbars and then drag the commands you want on to the new toolbar.
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.

New Window

This command opens an additional window for the current or front-most file. You can use this window to scroll to a different section of the file.

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.

Arrange Icons

Use this command to arrange the icons for any minimized windows.

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.

Contents

Choose this command to access the Absoft Editor on line manual.

Using Help

This command is used to obtain help on using the Windows help facility.

About Editor...

Use this command to see the version number of the Absoft Editor.
CHAPTER 4

Using the Compilers

This chapter describes how to use the Absoft Fortran 95 compiler to create executable files from source files. Beginning with an overview of invoking the compiler, this chapter explains how to compile a small number of files into an executable application. Next, the Absoft Developer Tools Interface is described with detailed descriptions of options and compiler settings.

COMPILING PROGRAMS

Four methods of compiling programs are available: a traditional command line, the Absoft Developer Tools Interface, makefiles, and directly from the editor. The command line and the Absoft Developer Tools Interface are discussed in this chapter. Makefiles and the Absoft make utility, \texttt{amake}, are described in the chapter \textbf{Building Programs}. Using the editor to compile files is outlined in the \textbf{Using the Absoft Editor} chapter. All four methods allow you to compile source files quickly and easily.

Source file names and compiler options are selected with the mouse pointer in the Absoft Developer Tools Interface. Arguments to the command line version are typed in on the command line. In the Absoft Editor, options for the individual compilers are set on the \textbf{Tools} tab of the \textbf{Preferences} command in \textbf{File} menu.

USING THE COMMAND LINE

To use a command line version of any of the Absoft compilers, you must first open a command line window and set a number of environment variables that assist and control the use of the compilers. A shortcut to open a command line window has been provided in the Absoft menu: \textbf{Development Command Prompt}. This command is a shortcut to the normal command line prompt for your system. On startup, it executes the batch file \texttt{absvars.bat} in the \texttt{Absoft\Bin} directory that sets a number of environment variables. Examine the batch file for further details.

A command line version of an individual compiler can be started with one of the commands: \texttt{f95}, \texttt{f77}, or \texttt{cl}.

\begin{verbatim}
f95 options files
f77 options files
cl options files
\end{verbatim}

The various options are described in the specific compiler options sections later in this chapter.
USING THE ABSOFT DEVELOPER TOOLS INTERFACE

The Absoft Developer Tools Interface is started from Windows by selecting Programs from the Start menu and then Developer Tools Interface from the Absoft Pro Fortran submenu. The first time you use the interface, a dialog may be displayed asking if you want certain standard file extensions (such as *.f, *.for, *.f90, *.f95, *.c, *.cpp, etc.) associated with the Absoft Editor. If you have other development systems installed on your system and don’t want to change your existing file associations, you can stop this dialog from being shown by unchecking the Check File Associations box in the Preferences dialog; see below.

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, and 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 box will appear as shown below:

![New Project Dialog Box]

Name is the name that will be applied 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 File Set drop down menu to select the initial file of the project. An Empty Project will have no files associated with it; you add them after creating the new project. The other menu choices will automatically create an empty source file of the specified type as the first (and possibly only) file in the project.
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.

After you click on the OK button to accept these values, the project options property sheet will be displayed, allowing any of the global compiler options, DLL 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 Property Sheet

The Options property sheet appears automatically after you accept the settings for a New Project. You can also open it by selecting the Set Project Options command from the Configure menu. The property sheet displayed with the Set Project Options command also has tabs for the individual compilers and the linker.

Target Property Page – General Options

The Target property page controls two option subsets that apply to all of the files for a project: General and FPU. Choose the subset from the Options Subset drop-down menu. General is described first:
Target Type

The radio buttons and check boxes in this area of the Target property page control the type of application produced by the compilation process. By default, a stand-alone console application (the Console Application button) is generated. This type of application runs in a command line window. Use -cons to select this option on a command line.

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

Use the -plainappl option (selected with the 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 console interface – you are responsible for the interface presented to the user. The option includes all of the standard Windows API import libraries and leaves the linker -entry and -subsystem arguments to their respective defaults.

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

Specify the Dynamic-Link Library option if you are creating a DLL (see Using and Creating DLLs below and Linking Programs and Creating Libraries in the Building Programs chapter).

The Use Dynamic-Link Libraries check box controls whether or not DLLs (as opposed to the static libraries) are used for runtime services such as I/O and intrinsic functions. Using DLLs can reduce the size of the application, but requires that the DLLs reside in a one of the following locations: C:\Windows\System, the directory from which the application was started, or one of directories specified in the PATH environment variable.

If you have a 64-bit edition of Windows, you can use the 64-bit Code Generation check box to create 64-bit applications. This option is necessary if you are producing programs that need to use more memory than a 32-bit system can provide.

Common Options

The settings in this section control commonly used options. All compilers also share these options.

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.
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 box, the Debug box is enabled allowing you to select the level of symbolic information to be produced for debugging with Fx (see the Fx Debugger Manual):

Minimum produces an object file containing debugging information with entry points and line numbers only. No information for program symbols is produced. Use this option when you are only interested in stepping through the program. This option is selected on the command line with the -gmin switch.

Standard produces an object file containing 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.

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:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>486</td>
<td>non-Pentium Intel processor</td>
</tr>
<tr>
<td>p5</td>
<td>Pentium</td>
</tr>
<tr>
<td>p6</td>
<td>Pentium Pro, Pentium II, and Pentium III</td>
</tr>
<tr>
<td>p7</td>
<td>Pentium 4</td>
</tr>
<tr>
<td>athlon</td>
<td>AMD Athlon and Duron</td>
</tr>
<tr>
<td>athlon64</td>
<td>AMD Athlon64</td>
</tr>
<tr>
<td>host</td>
<td>automatically establishes type based on the processor in the machine that the program is compiled with. If the CPU type cannot be determined, p5 is assumed.</td>
</tr>
</tbody>
</table>
The **UNIX Lib** option automatically adds this compatibility library to your project. Depending on your needs and other code that may be incorporated into your project, you can choose to between all upper case (**UCS**) or all lower case (**LCS**) external global symbols. The compatibility libraries are described in the *Support Library* manual with the other Absoft Pro Fortran manuals.

The **Use VAX/VMS Library** automatically includes this compatibility library in your project. The compatibility libraries are described in the *Support Library* manual with the other Absoft Pro Fortran manuals.

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 name will be used with an extension of `.exe`. 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 either `-out:name` or `-o name`, where `name` is the name the executable file will be given.
Target Property Page – FPU Options

The **FPU** subset page provides control over four aspects of the operation of the *Floating-Point Unit* of the processor: rounding mode, exception handling, control word state, and FPU stack integrity.

![Options](image)

### FPU Rounding Mode

These radio buttons control the rounding used in the floating point operations. The default method is to turn off optimizations that may be harmful to floating point calculations. The **Simple optimizations** button, `-OPT:roundoff=1` from the command line, allows simple optimization that may affect floating point accuracy. The **Extensive optimizations** button, `-OPT:roundoff=2` from the command line, allows more extensive optimization that may affect floating point accuracy. The **Allow all optimizations** button, `-OPT:roundoff=3` from the command line, allows all optimizations affecting floating point accuracy.
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.
Build Property Page

Use this page to specify the tools that are used to build an application. The Tools section specifies the compilers, linker, and make facilities. The default path to these tools is the BIN directory of the main Absoft directory (e.g. C:\Absoft\Bin). Use fully qualified paths for tools not residing in this directory.

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

ABSOFT DEVELOPER TOOLS INTERFACE

The following sections describe the menu commands available in the Absoft Developer Tools Interface.

File Menu

New

Use this command to create a new project. All file sections, include paths, and option settings are reset. If you have a previously unsaved project already open, you will be prompted to save it before continuing. 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 with all file selections and option settings restored from that session.

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.
Preferences

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

Preferences

The Startup Options check boxes control what action (if any) the Absoft Developer Tools Interface performs upon startup. If the Check file associations box is checked, the application file type associations of common programming language source file name extensions will be checked to determine if either the Absoft Developer Tools Interface or the Absoft Editor is the registered application to start.

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 which points to the Pro Fortran installation directory.

The radio buttons under Compile Options allow you to specify the type of notification you wish to receive from the compilers.

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.
Edit Menu

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

Cut (Ctrl+X)

This command removes the selected text from the active control and places it on the clipboard. Text on the clipboard may be pasted into other windows, or controls.

Copy (Ctrl+C)

The Copy command copies the selected text from the active control and places it on the clipboard. Text on the clipboard may be pasted into other windows, or controls.

Paste (Ctrl+V)

This command replaces the selected text from the active control with the text on the clipboard. If no text is selected in the active control, the clipboard text is inserted at the insertion point.

Delete (Del)

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

View Menu

The commands in the View menu allow you to turn the Status Bar and Toolbar on or off and to customize the toolbar. There are also commands for configuring and refreshing the project file view, and navigating errors in the build tab of the output view.

Toolbars

Use this command to show or hide Toolbars. The Toolbars are located directly beneath the menu bar and contain various buttons that can be used as shortcuts for some of the more commonly used menu commands. The Toolbars are dockable and may be detached and repositioned wherever convenient within the interface’s frame. The Toolbars can be customized by adding, removing, and reorganizing the buttons. See the Customize... command below.

Status Bar

This command is used to show or hide that Status Bar located at the bottom of the Absoft Developer Tools Interface window. The Status Bar indicates the state of certain keys such as “caps lock” and “num lock”. Clicking on the clock toggles it between displaying the time or the date.
Previous (Ctrl+F6)

Use this command to go to the previous item. This command is only available if the Build tab of the Output window is active and there are errors or warnings, or if the Search tab of the Output window, is active, and there are matches for the current search expression.

Next (Ctrl+F7)

Use this command to go to the next item. This command is only available if the Build tab of the Output window is active and there are errors or warnings, or if the Search tab of the Output window, is active, and there are matches for the current search expression.

Full Pathnames

Use this command to toggle display of full pathnames in the project file view for the current project.

Refresh (F5)

Use this command to refresh the file status of the project file view for the current project. Valid file references will appear in the standard colors. Files that cannot be located on the current hardware will appear with a red file icon.

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. Note that the type of file displayed can be limited with the Files of type drop down menu. If the source files you are adding will be processed with VAST preprocessor (see Plug-ins later in this chapter), they must be selected and added to the project with the appropriate Files of type setting.

The upward and downward 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.
Set Include Path(s)...

Use this command to select additional directory paths to be searched for include and header files. 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 Absoft Windows shell variable.

Set Project Options...

This command opens the Options property sheet introduced earlier. The property pages available on this sheet 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 file. The commands sub menus become active after you first select a source file in the Project window. Any option choices made will affect only the file name displayed in the options dialog title bar.

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.
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.

**Atlas...**

This command opens the Atlas build application. Atlas is a system that you can use to create a BLAS (Basic Linear Algebra System) library customized for your specific hardware environment.

**Tools Menu**

This menu provides access to the various tools you will work with: compilers, editors, debuggers, 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**

Use this command to check the syntax of the file selected in the **Project** window.

**Compile**

The **Compile** command will compile the file currently selected in the **Project** window.
Edit

Use this command to open for editing the file currently selected in the Project window.

Preprocess

The Preprocess command is only available with the Pro FortranMP version. It executes the installed multi-processor preprocessors and leaves the intermediate file on your disk so that you can examine it.

Touch

The Touch command is used to update the modification date of the file selected in the Project window. This will force the file to be recompiled with the next build.

Clean

Use this command to delete the executable and all of the object files in your project. It will also delete any .res files if your project incorporates resource compiler source files (.rc).

Stop

The Stop command terminates a currently executing build.

Execute

Use this command to execute or run your program after it has finished building.

Debug

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

Command Prompt

This command will open a Development Command Prompt. This is a command prompt window with the appropriate environment variables set for using the Absoft Pro Fortran tools.

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.
Customize...

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

You can even create your own custom toolbars and then drag the commands you want on to the new toolbar.

**Window Menu**

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

**Cascade**

**Tile Horizontally**

Use this command to vertically arrange multiple opened windows in a non-overlapped fashion.

**Tile Vertically**

Use this command to arrange multiple opened windows side by side. This command is used to arrange your windows vertically based on the order in which they were opened.

**Arrange Icons**

Use this command to arrange the icons for any minimized windows.
Intelli-tile

Use this command to arrange multiple opened windows side by side, in an intelligent manner, which takes into account the information displayed in the various windows to aid in the sizing process.

Tile with Editor

Use this command to arrange multiple opened windows side by side, in an intelligent manner, which takes into account the information displayed in the various windows to aid in the sizing process, and also involves the MDI frame window of the Editor into this sizing process.

Output

Use this command to open and activate the output window, which contains information on the latest build, latest search, latest profile, and the current state of the associated makefile.

Project

Use this command to open and activate the project window.

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.

Help Topics

Choose this command to access the Absoft Editor on line manual.

About Interface...

Use this command to see the version number and other related information regarding the Absoft Developer Tools Interface.
ABSOFT 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 Property Sheet. The options for FORTRAN 77 fall into several categories: General, Control, Compatibility, Miscellaneous, Format, and Common.

For quick reference, the options listed in the sections that follow are in the order in which they appear on the F77 Property Page. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked, the same letters will appear in the Selected F77 Options box.

General - F77 Options

These options control the general characteristics of the FORTRAN 77 components of the program being built. They are primarily concerned with debugging information.
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.

Show Progress (-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 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 in The FORTRAN 77 Program chapter of the Absoft FORTRAN 77 Language Reference Manual.
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.
Control - F77 Options

When this subset of the F77 options property page 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 options property page 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.

F77 Compatibility Options

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 lower case 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 \texttt{RETURN} statement. The \texttt{SAVE} statement is normally used to prevent this, but the \texttt{-s} compiler option will force all program storage to be treated as static and initialized to zero.

Folding to Upper Case (-N109)

By default, the compiler considers upper and lower case 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 \texttt{-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 \texttt{DO} loop, as established from the \texttt{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 \texttt{DO} loop not be executed. The \texttt{-d} option will cause all \texttt{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 \texttt{-N15} option will cause the compiler to define \texttt{SUBROUTINE} and \texttt{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.

Promote REAL and COMPLEX (-N113)

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

Integer Sizes (-i2 and -i8)

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

![Options](image)

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. In the absence of specification, handling of byte ordering depends on the system.

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. See discussion under N26.

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 which are supported.
COMMON Block - F77 Options

Several options are available to control COMMON blocks in FORTRAN 77.

**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. See also the -N25 option above.

ABSOFT 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 Property Sheet. The Fortran 90/95 options fall into four categories: General, Compatibility, Miscellaneous, and Format.

For quick reference, the options listed in the sections that follow are in the order in which they appear in the dialog boxes. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked, the same letters will appear in the Selected F95 Options box.
General - F95 Options

When this subset of the F95 options property page 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.

F95 General Options

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.
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.

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.

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.
Max Internal Handle (-T \textit{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 \textit{nn}, where \textit{nn} is a positive integer constant. When this option is specified, the number of handles will be $100000 \times 10^{nn}$ bytes.

Temporary string size (-t \textit{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 \textit{nn}, where \textit{nn} is a positive integer constant. When this option is specified, the default temporary string size will be $1024 \times 10^{nn}$ bytes.

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

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

Set Module Paths (-p \textit{path})

The Absoft Fortran 90/95 compiler will automatically search the local directory for precompiled module files. 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.
Compatibility - F95 Options

This subset of the F95 options property page displays options for compiling Fortran 90/95 programs. These options allow Absoft Fortran 90/95 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.

Disable compiler directive (-xdirective)

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

NAME
FIXED
FREE
STACK

See the section Absoft Fortran 90/95 Compiler Directives for more information on using compiler directives in your source code.
Integer Sizes (-i2 and -i8)

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.

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 to REAL(KIND=8) (-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=8), 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.

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.

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

This option controls whether or not the compiler will allow 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:

```fortran
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:

```fortran
call specific1(i)
call specific2(loc(i))
```
DVF/CVF Character Arguments (-YVF_CHAR)

The -YVF_CHAR option causes the compiler to pass and expect CHARACTER arguments in a manner compatible with Digital/Compaq Visual Fortran. The length of the argument (as a value) immediately follows the argument itself as opposed to the more common method of passing the length(s) at the end of the argument list.

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 property page 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 (-f free)

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 (-f fixed)

The -f fixed 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 (-f alt_fixed)

The -f alt_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.

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. In the absence of specification, handling of byte ordering depends on the system.

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. See discussion under N26
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.

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:

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\a</td>
<td>Audible Alarm (BEL, ASCII 07)</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace (BS, ASCII 8)</td>
</tr>
<tr>
<td>\f</td>
<td>Form Feed (FF, ASCII 12)</td>
</tr>
<tr>
<td>\n</td>
<td>Newline (LF, ASCII 10)</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage Return (CR, ASCII 13)</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal Tab (HT, ASCII 09)</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical Tab (VT, ASCII 11)</td>
</tr>
<tr>
<td>\xh[h]</td>
<td>Hexadecimal, up to 2 digits</td>
</tr>
<tr>
<td>\o[o[o]]</td>
<td>Octal number, up to 3 digits</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
</tbody>
</table>

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.
Miscellaneous - F95 Options

These options are used to control the global names of \texttt{COMMON} blocks. Their primary use is for managing the character case and decoration applied to \texttt{COMMON} block names when interfacing with external procedures written in FORTRAN 77 or the C Programming Language.

**COMMON Block Name Prefix (-YCOM\_PFX=\texttt{string})**

The \texttt{-YCOM\_PFX} option can be used to prepend a user specified \texttt{string} to the external representation of \texttt{COMMON} block names.

**COMMON Block Name Suffix (-YCOM\_SFX=\texttt{string})**

The \texttt{-YCOM\_SFX} option can be used to append a user-specified \texttt{string} to the external representation of \texttt{COMMON} block names.

**COMMON Block Name Character Case (-YCOM\_NAMES=\{ ASIS | UCS | LCS \})**

The \texttt{-YCOM\_NAMES} option is used to specify how the external names of \texttt{COMMON} blocks are emitted. The default (\texttt{-YEXT\_NAMES=UCS}) is to emit \texttt{COMMON} block names entirely in upper case. Set this option to \texttt{LCS} to emit names entirely in lower case.
Loop unrolling (-U and -h $nn$ and -H $nn$)

The Absoft Fortran 95 compiler has the ability to automatically unroll some of the loops in your source code. Loops may be unrolled by any power of two. Generally it is beneficial to unroll loops that execute a large number of iterations, while the benefit is small for loops that iterate only a few times. Due to this, only innermost loops are considered for unrolling. The -h $nn$ option will cause the compiler to unroll your innermost loops $nn$ times, where $nn$ is any power of two. The -H $nn$ option will cause the compiler to consider loops containing $nn$ or fewer operations for unrolling. When the -O3 option is used, the default is to only consider loops of a forty operations and unroll them four times. Using the -U option is equivalent to using -h 4 -H 40, causing innermost loops of forty or fewer operations to be unrolled four times. Loop unrolling will provide a speed increase in most cases, but will make your application larger and it will require more memory to compile. Consider the following example:

<table>
<thead>
<tr>
<th>Original code:</th>
<th>Becomes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBROUTINE SUB(A,N,X) INTEGER A(100) DO i=1,N A(i) = X*A(i) END DO RETURN END</td>
<td>SUBROUTINE SUB(A,N,X) INTEGER A(100) DO i=1,MOD(N,4) A(i) = X<em>A(i) END DO DO i=4,N-(MOD(N,4)),4 A(i) = X</em>A(i) A(i+1) = X<em>A(i+1) A(i+2) = X</em>A(i+2) A(i+3) = X*A(i+3) END DO RETURN END</td>
</tr>
</tbody>
</table>

This is similar to the effect of loop unrolling. At least three comparisons and three branch instructions are saved each time the second loop is executed. Note that if your code contains extended range DO loops, unrolling loops will invalidate your program.

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.

Add Microsoft GLOBAL prefix (-YMSFT_GLB_PFX)

This option causes common block names declared with GLOBAL to be prefixed with "__imp_" for the Microsoft linker. It is useful when creating DLLs that will share data through Fortran common blocks.
Other F95 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 (see the chapter, Building Programs).

SSE2 instructions (-msse2 and -mno-sse2)

The -msse2 and -mno-sse2 options enable and disable respectively the use of SSE2 instructions for floating-point operations. This -msse2 option is automatically enabled on processors which support SSE2. It may be disabled with the -mno-sse2 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.

Conditional Compilation (-YX)

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

Check Argument Interface (-Ra)

When the -Ra compiler option is specified, code will be generated to check 1) actual and dummy argument count mismatches, 2) non-writable arguments passed to dummy arguments declared as INTENT OUT or INTENT INOUT, and 3) type/kind mismatches. Note that this option requires that all components of an executable be compiled with the -Ra option, including module procedures.

Check Argument Count (-Rn)

The -Rn compiler option is used to check actual and dummy argument count mismatches. Note that this option requires that all components of an executable be compiled with the -Rn option, including module procedures.

Disable Default Module File Path (-nodefaultmod)

The Absoft Fortran 90/95 compiler will automatically search the directory %ABSOFT%\F90INC for precompiled module files. Use the -nodefaultmod to disable this.
YIL={AC90, AC77, LF90, MSVC, MSVB, BC, BD, WINAPI}

For DLL_IMPORT and DLL_EXPORT, this option controls the calling mechanism and name mangling produced in the object code:

<table>
<thead>
<tr>
<th>Target</th>
<th>Call Mechanism</th>
<th>Name Mangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC90</td>
<td>Default</td>
<td>Uppercase</td>
</tr>
<tr>
<td>AC77</td>
<td>Default</td>
<td>As is</td>
</tr>
<tr>
<td>MSVC</td>
<td>stdcall</td>
<td>As is &amp; @argsize</td>
</tr>
<tr>
<td>MSVB</td>
<td>stdcall</td>
<td>As is</td>
</tr>
<tr>
<td>Borland C</td>
<td>stdcall</td>
<td>As is</td>
</tr>
<tr>
<td>Borland Delphi</td>
<td>stdcall</td>
<td>As is</td>
</tr>
<tr>
<td>WINAPI</td>
<td>stdcall</td>
<td>As is &amp; @argsize</td>
</tr>
</tbody>
</table>

AC90 is the default setting of this option. The arguments to this option are case sensitive.

NOTE: The $DIR NAME directive takes precedence over any name mangling implied by this option.

YDLL_STDCALL={0|1}

This option can be used to override or confirm the default treatment of DLL_IMPORT and DLL_EXPORT symbol names. The default is set according to the language target (see the YIL option described above). If 0, external references will be generated with a normal call/return sequence in which the callee does not pop the argument frame. If 1, then the call/return convention is Microsoft STDCALL and the callee pops the frame.

YDLL_NAMES={ASIS | UCS | LCS}

This option can be used to override or confirm the default treatment of DLL_IMPORT and DLL_EXPORT symbol names. The default is set according to the language target (see the YIL option described above). The !DIR$ NAME directive takes precedence over any name mangling implied by this option (except the frame size mangling done by the back-end). The argument to this option is case sensitive. This option does not affect names declared with the STDCALL attribute.

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.
Variable 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 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, -YEXT_NAMES, and -YDLL_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 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 separated by commas. 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 separated by commas. 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.
NAME Directive

The NAME directive allows you to specify case-sensitivity for externally visible names. You can use this directive, for example, when writing calls to C routines or declaring functions to be called outside of Fortran 90/95. 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 Directive

The FREE 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.

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
```
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. Select the Set Project Options command in the Configure menu to access the Options Property Sheet. 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 boxes. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked, the same letters will appear in the options box.

General – C/C++ Options

The developer tools interface is designed to allow using any C/C++ you want. Although the default options are set for Microsoft Cl, any compiler can be used by entering options in the option box.

Options

C/C++ Options:
- Disable All Warnings (-w)
- Enable Run-time Debug Checks
- Force Stack Checking

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.
Disable Extensions (-Za)

This option is used to assure that only standard C/C++ syntax is used in source programs.

Enable Runtime Debug Checks (-GZ)

This check box enables runtime checking of the stack.

Force Stack Checking (-Ge)

This check box forces stack checking of all functions.

ABSOFT LINKER OPTIONS

The linker options detailed in this section give you control over default memory allocations, linker output file names, and certain other aspects of the program linking process. Select the Set Project Options command in the Configure menu to access the Options Property Sheet. A complete description of the linker can be found in the Building Programs chapter of this manual.

For quick reference, the options listed in the sections that follow are in the order in which they appear in the dialog boxes. Each option is listed with the corresponding option letter(s) and a short description. When an option is checked, the same letters will appear in the Selected Linker Options box.
General - Link Options

This is the only property page for the linker:

**Stack Size (-stack:reserved[,commit])**

Use this option to establish the amount of memory reserved for stack use only. The default `reserved` amount is 0x800000 (8 megabytes). The optional `commit` argument can be used to change the increment value from the default of 4096 bytes.

**Heap Size (-heap:reserved[,commit])**

Use this option to establish the amount of memory reserved for heap use only. The optional `commit` argument can be used to change the increment value from the default of 4096 bytes.

**Map File: (-map:name)**

Entering a file name in this text box directs the linker to create a file containing the address of every symbol (both code and data) in the program. This option is given on the command line as `-link -map:name`, where `name` is the name of the map file.
Exports Files: (-exports:filename)

This option, used in conjunction with creating DLLs, specifies the filename of a file containing the module names to be exported in the DLL.

When used in conjunction with the /lib option, this option specifies the filename of a file that contains all of the exportable names in the library.

Suppress Warnings: (-warn)

Use this option to suppress all linker warnings.

Verbose: (-verbose)

This option writes the specified version number to header of the output file.

PLUG-INS

The Plug-ins property page on the Options Property Sheet 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 Property Sheet.

Two types of plug-ins are available: preprocessors and libraries. The check boxes for these items are enabled only if the product was purchased and installed.
**IMPORTANT:** Source files that are intended to be processed by the VAST preprocessor must be added to the project using the VAST file type. Use the **Files of Type** drop-down list box to set the type to **Vast Files** files. See the section describing **Add/Remove File(s)** discussed earlier in this chapter.

![File Selection Dialog](image)

**File(s)** dialog box with file types and file selection options.
VAST

The VAST pre-processor is developed by Pacific-Sierra Research Corporation. It is a pre-processor and library combination for multi-processor Windows NT/2000/XP installations and can be used with FORTRAN 77 or Fortran 90/95 programs. You must have Pro FortranMP installed. The VAST pre-processor cannot be used with Windows 95/98 as neither operating system supports multi-processor installations. Click the Settings tab to display a dialog for specifying additional preprocessor options.

![VAST Settings dialog box](image)

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

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

![Notice dialog box](image)

The VAST preprocessor 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.

IMSL

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

The LAPACK check box controls the use of the LAPACK libraries. Complete documentation, source code, and makefiles are provided on the Pro Fortran CDROM. Place a check in this box to automatically link against the library.

BLAS

A BLAS library must be used with IMSL and LAPACK. You can choose either the standard library (source code and makefiles are provided on the Pro Fortran CDROM) or an ATLAS library. If you choose an ATLAS library you can use one of the prebuilt ones supplied on the CDROM or build your own, customized for your specific hardware.

PLplot Library

PLplot is a multi-paltform add-on graphics library. Documentation is provided in the Documentation directory of the Absoft Pro Fortran installation directory and example codes are available in the Examples directory. Two versions are supplied: single and double precision. Place a check in the appropriate box to automatically link against the library.

Note: PLplot uses its own fonts which are supplied in the files plstnd5.fnt and pltxtnd5.fnt. These files must be placed in the same directory as the executable file that uses PLplot.

USING AND CREATING DLLS

Creating a Dynamic-Link Library is easy to do with any of the Absoft compilers. select the Dynamic-Link Library radio button in the Target Type box of the Options Property sheet and the Absoft linker will perform the necessary operations. (Select the Set Project Options command in the Configure menu to access the Options Property Sheet.)

In order for the procedures and data in a DLL to be referenced externally, they must be exported – which means making their names known outside of the DLL. A DLL may contain private procedures that are unknown outside of the DLL by not exporting them.

Procedures in a DLL may be referenced outside of the DLL by importing them which means informing the compiler or linker that they are located in a DLL.

If you are creating an application, the Use Dynamic-Link Libraries box on the Target property page of the Options property sheet can be checked to have the compiler link to the Absoft runtime DLL rather than the static runtime library.
Using the Compilers

**DLL Compatibility**

The actual form of an external entry point name in a DLL (an exported name) is dependent on the system that created the DLL. Various forms of *name mangling* are employed by programming language suppliers. Name mangling involves decorating external names in such a way that they do not conflict with other global names or so that they can supply argument list and stack size information to the linker. In addition, there are two call/return sequences defined in the Win32 API (Application Programming Interface):

**CDECL**

This is the default call/return sequence generated by Absoft compilers. The caller pushes arguments from right to left onto the stack. The callee accesses the parameters in the stack and returns. The caller cleans up (removes the arguments from) the stack.

**STDCALL**

This is the call/return sequence used by most of the Windows operating system functions. The caller pushes arguments from right to left onto the stack. The callee accesses the parameters in the stack, but is also responsible for removing them from the stack.

Obviously, the two mechanisms cannot be intermixed and passing too many or too few arguments with the STDCALL protocol is disastrous (the wrong number of arguments will be removed from the stack by the callee). As a protection against this, STDCALL function names in the object code are often *mangled* by appending a commercial at sign ('@') and the size of the stack (in bytes) to the function name. In this way, the caller and the callee must agree on the number of arguments, or the program will not link.

**Note:** All Absoft languages are self-consistent. No special commands are necessary for inter-language procedure references with Absoft languages. Use the **Default** command in this instance.
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 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 Windows-specific issues about porting code.

The Absoft Fortran 77 compiler is recommended for porting most legacy codes because of the number extensions and features it supports. Consequently, FORTRAN 77 options and language features will be described in this chapter. However, in most cases, the Fortran 90/95 compiler has equivalent options and can also be used. Refer to the Using the Compilers chapter for information on Fortran 90/95 compile time options.

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 Windows 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 Windows. 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 the appendix VAX Extensions in the Absoft FORTRAN 77 Language Reference Manual. 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 Pro Fortran 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 graphical interface for the compiler, they can all be selected with 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 library file vms.lib in the lib directory. The routine names may be referenced as all upper case (-N109), all upper case with an underscore appended (-N15), or all lower case with an underscore appended. The routine names 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
- SECONDS 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 2>filename to the f77 command line to create a file containing compiler warning and error messages
- /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 on “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: no equivalent
- /MACHINE_CODE: no equivalent
- /OBJECT: no equivalent—you can use the COPY command to copy an object file to another name
- /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
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 that 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

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)
- STRUCTURE, RECORD, UNION, MAP statements
- SELECT CASE statements
- DO WHILE loops
- INCLUDE statement
- Conditional compilation statements
Key Unsupported Microsoft FORTRAN Extensions

- Metacommands
- MS-DOS specific intrinsic functions
- INTERFACE TO statement
- OPEN statement displays standard file dialog when using FILE=

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 IMPLICIT NONE statement may be used to generate errors or warnings for untyped data items
- `$DO66` -d for FORTRAN 66 compatible DO loops
- `$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 INCLUDE 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` -O for optimization
- `$MESSAGE` no equivalent
- `$PACK` use $PACKON and $PACKOFF
- `$PAGE` not applicable
- `$PAGESIZE` not applicable
- `$STORAGE:2` -i for interpreting INTEGER and LOGICAL as INTEGER*2 and LOGICAL*2
- `$STORAGE:4` do not use the -i option
- `$STRICT` -N32 to generate warnings for non-ANSI FORTRAN 77 usage
- `$SUBTITLE` not applicable
- `$TITLE` not applicable
- `$TRUNCATE` no equivalent
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 Windows functions and other development languages. The -N15 option may be used to append underscores to routine names.

Key Supported Sun FORTRAN Extensions

- 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

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 Windows with Intel or PowerPC processors. 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 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 the appendix Language Systems Fortran Extensions in the Absoft FORTRAN 77 Language Reference Manual.

Key Supported Language Systems Fortran Extensions

- STRING declaration statement
- POINTER declaration statement
- LEAVE control statement
- GLOBAL, CGLOBAL, and PBLOCAL 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

Distribution Issues

If you plan to distribute executable programs generated with Absoft Fortran 77, you must obtain a copy of the Absoft “Redistribution License Agreement”, complete it, and return it to Absoft. There is no charge for this license or the redistribution of programs created with Absoft Fortran 77. To obtain the Absoft “Redistribution License Agreement”, write to:

Absoft Corporation
2781 Bond Street
Rochester Hills, MI 48309
OTHER PORTING ISSUES

Not all porting and compatibility issues can be solved automatically by Absoft Fortran 77 or by using various option combinations. There are six issues that must be addressed on a program-by-program basis for Windows based computers:

- Memory Management
- Stack Issues
- Tab Character Size
- File and Path Names
- Numeric Precision
- Floating Point Math Control

Memory Management

A Win32 API application’s address space differs slightly between Windows NT/XP and Windows 95/98. A Windows NT/XP address space ranges from 0x00010000 to 0x7ffeffff, while Windows 95/98 makes the space between 0x0040000 and 0x7fffffff available to the application. Since Win32 is a virtual, memory-managed environment, there is no real need for the programmer to be concerned with address spaces or virtual to physical memory mapping. A Win32 application will generally run without any memory management intervention by the programmer. There are however, two tunable memory parameters available that may be used to improve performance with memory usage intensive programs.

The Windows memory manager will automatically allocate memory beyond the initial 0x100000 byte (1 megabyte) heap and stack allocations to your program as it requires it. If you know that your program will use significantly more memory than this, it can be more efficient to reserve it initially rather than allocate it incrementally. Initial stack and heap allocations are established with the -stack and -heap compiler options (refer to the chapter Using the Compilers for details on the stack size and heap size options). The optional commit argument is used to specify the size of actual memory pages.

File and Path Names

Almost every operating system has a unique set of rules for valid file and path names and Windows is no exception. File names may be up to 255 characters in length, but may not contain any of the following characters: ? " / < > * | :. Case is preserved in file names, but file names themselves are not case sensitive. To reference a file in the current directory of a running application, the file name can be used without having to specify a path as in this example with a file called file.dat (8 characters including the period).

```
OPEN(UNIT=5,FILE="file.dat")
```

Path names are the concatenation of drive names, directory names, and file names and are used to specify files in other directories. Each component of a path name is separated by a back slash (\). A full path name always begins with the name of a drive and includes each of the directory names in the path to the file. A full path name is a complete and unambiguous identification for a file. Another type of path name is a partial path name.
that describes the path to a file starting from the current directory. Parent directories can be specified by beginning the path name with two periods and a backslash.

Programmers should be aware that Windows XP “personalizes” directories so the complete path name for a file may appear different from different sign in accounts. The same file will be seen in “Jane’s Documents” by Fred, but in “My Documents” by Jane.

**Tab Character Size**

Absoft Fortran 77 expands each tab character in a FORTRAN source file into the equivalent number of spaces during compilation. The size of a tab character is determined from the following list in order.

- From the environment variable `TABSIZE`, which can be established with the `SET` command as follows:

  ```bash
  SET TABSIZE=6       set tabs to six characters
  SET TABSIZE         "unset" the TABSIZE variable
  ```

- If the environment variable `TABSIZE` is not set, the value 8 is used.

Tabs are also expanded at runtime when reading formatted files. They are expanded modulo `TABSIZE` where `TABSIZE` is an environment variable. If `TABSIZE` is not set, tabs are expanded modulo 8. If `TABSIZE` is set to 0, the tab is passed unmolested to the application.

**Runtime Environment**

A number of the aspects of the runtime environment can be controlled with the `ABSOFT_RT_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.

Floating Point Math Control

This section describes the basic information needed to control the floating point unit (FPU) built into Intel processors. 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:

\[
\text{CALL fpcontrol(cmd, arg)}
\]

where: \( cmd \) is an INTEGER expression 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 or 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 \text{fenv.inc} include file are:

\[
\text{FE_TONEAREST} \quad \text{round to nearest}
\]
**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

For example, to retrieve the state of the FPU, and then enable divide-by-zero exceptions, the following sequence would be used:

```fortran
INCLUDE "fenv.inc"
INTEGER state

CALL fpcontrol(0, state)
CALL fpcontrol(1, FE_DIVBYZERO)
```
CHAPTER 6

Building Programs

This chapter covers the specifics of building Fortran 90/95 and FORTRAN 77 programs, including a discussion of the linker, library manager, and 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.

This chapter also describes the features, capabilities, and extensions in the Absoft implementations of Fortran 90/95 discussed in language reference manuals. The Absoft implementation of Fortran 90/95 is described the Absoft Fortran 90 Concise Reference. The Absoft implementation of FORTRAN 77 is described the Absoft FORTRAN 77 Language Reference Manual.

AN OVERVIEW OF PROGRAM BUILDING

There are several different ways of building an application with the Absoft software development tools. The general overview of building a completed application is as follows:

Create Fortran 90/95 or FORTRAN 77 and compile them into object files with the proper interface and include files. See the section on Creating Object Files later in this chapter.

Create non-code resources with rc.exe, the resource compiler. See the section on Working with Resources in this chapter.

Create the executable program by using the link tool to link object files with the necessary resources and library routines from the Windows system. For more information on the link tool, see the section on Linking Programs, also in this chapter.

The Components of an Application

Program code, system calls, library routines, and features of the Windows operating system and interface are all important components of an application. Output from tools such as amake and lnk are combined with your object code to create a Windows application.
Working with Resources

A resource is one of the most important concepts in Windows programming. A resource is a collection of information used by the Windows 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 Microsoft program, rc.exe, included with the Absoft software development tools, is a resource compiler that creates new resources based on a textual resource description file. rc.exe is documented in a help file supplied with the compiler.

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.exe, f77.exe, or f90.exe – these command control both components of the compiler (front and back ends) and the link tool (see the section below on Linking Programs). The features of the f95, f77, and f90 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...]
- cl [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 same as the base file name of the first file on the command line with an extension of .exe. For example, if you enter f77 hello.f on the command line, the source code from the hello.f file is compiled and an application will be generated in the file hello.exe. To compile hello.f with the static local storage option, and generate an application named welcome.exe, enter:
f77 -s -o welcome.exe hello.f

The option, `-o name`, specifies the name of the executable file overriding the default name of `hello.exe`. 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.obj` and `Goodbye.obj`. 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`, any set of options may be passed directly to the linker. To do this, append the following option to the compiler command:

```
-link opts
```

The argument `opts` is a string enclosed in quotes to be passed to the linker. For example, `-link -verbose` will pass the `-verbose` 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`.  

---

**Fortran User Guide**
• 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.

LINKING PROGRAMS

Linking programs is the process of combining a group of object files into an application. The result is a new executable file. The Absoft tool that provides this feature is the link tool, or otherwise called the Linker. This tool is also used to create a Dynamic-Link Library, or a DLL (see the section Creating Libraries below for details).

The link tool links these object files into an application or tool called the output file. The Linker creates (or replaces) program code and places the object files as a linked application in the output file. The default output name is the root name of the first object file with the extension .exe appended.

If you use the f77 command with the /c option to create an ordinary object file only, it contains object code and symbolic references to global variables and identifiers unknown at compile time. If you execute the f77 command without the /c option, the link command is automatically invoked, creating the executable file.

The format for the link command is:

```bash
link [option...] [file...]
```

The options for the link command are:

@

Specifies a response file
-ALIGN
Specifies the alignment of each section

-ALLOWBIND
Specifies that a DLL cannot be bound

-ALLOWISOLATION
Specifies behavior for manifest lookup.

-ASSEMBLYDEBUG
Adds the DebuggableAttribute to a managed image.

-ASSEMBLYLINKRESOURCE
Create a link to a managed resource.

-ASSEMBLYMODULE
Specifies that a Microsoft Intermediate Language (MSIL) module should be imported into the assembly

-ASSEMBLYRESOURCE
Embeds a managed resource file to an assembly

-BASE
Sets a base address for the program

-CLRIIMAGETYPE
Sets the type (IJW, pure, or safe) of a CLR image.

-CLRSUPPORTLASTERROR
Preserves the last error code of functions called through the P/Invoke mechanism.

-CLRTHREADATTRIBUTE
Specify which threading attribute you want applied to the entry point of your CLR program.

-CLRUNMANAGEDCODECHECK
/CLRUNMANAGEdC0DECHECK specifies whether the linker will apply the SuppressUnmanagedCodeSecurity attribute to linker-generated PInvoke stubs that call from managed code into native DLLs.

-DEBUG
Creates debugging information

-DEF
Passes a module-definition (.def) file to the linker

-DEFAULTLIB
Searches the specified library when resolving external references

-DELAY
Controls the delayed loading of DLLs

-DelayLoad
Causes the delayed loading of the specified DLL

-DelaySign
Partially sign an assembly.

-DLL
Builds a DLL

-DRIVER
Creates a Windows NT kernel mode driver

-ENTRY
Sets the starting address

>ErrorReport
Report internal linker errors to Microsoft.

-EXPORT
Exports a function
-FIXED
Creates a program that can be loaded only at its preferred base address

-FORCE
Forces a link to complete in spite of unresolved or symbols defined more than once

-FUNCTIONPADMIN
Creates a hotpatchable image.

-HEAP
Sets the size of the heap in bytes

-IDLOUT
Specifies the name of the .idl file and other MIDL output files

-IGNOREIDL
Prevents processing attribute information into an .idl file

-IMPLIB
Overrides the default import library name

-INCLUDE
Forces symbol references

-INCREMENTAL
Controls incremental linking

-KEYCONTAINER
Specify a key container to sign an assembly.

-KEYFILE
Specify key or key pair to sign an assembly.

-LARGEADDRESSAWARE
Tells the compiler that the application supports addresses larger than two gigabytes
-LIBPATH
Allows the user to override the environmental library path

-LTCG
Specifies link-time code generation

-MACHINE
Specifies the target platform

-MANIFEST
Create a side-by-side manifest file.

-MANIFESTDEPENDENCY
Specify a <dependentAssembly> section in your manifest file.

-MANIFESTFILE (Name Manifest File)
Change the default name of the manifest file.

-MAP
Creates a mapfile

-MAPINFO
Includes the specified information in the mapfile

-MERGE
Combines sections

-MIDL
Specifies MIDL command line options

-NOASSEMBLY
Suppresses the creation of a .NET Framework assembly

-NODEFAULTLIB
Ignores all (or specified) default libraries when resolving external references
-**NOENTRY**
Creates a resource-only DLL

-**NOLOGO**
Suppresses startup banner

-**NXCOMPAT**
Marks an executable as having been tested to be compatible with Windows Data Execution Prevention feature.

-**OPT**
Controls LINK optimizations

-**ORDER**
Places COMDATs into the image in a predetermined order

-**OUT**
Specifies the output file name

-**PDB**
Creates a program database (PDB) file

-**PDBSTRIPPED**
Creates a program database (PDB) file with no private symbols

-**PGD**
Specify .pgd file for profile guided optionizations.

-**PROFILE**
Produces an output file that can be used with the Performance Tools profiler.

-**RELEASE**
Sets the Checksum in the .exe header

-**SAFESEH**
Specify that the image will contain a table of safe exception handlers.

- **SECTION**
  Overrides the attributes of a section

- **STACK**
  Sets the size of the stack in bytes

- **STUB**
  Attaches an MS-DOS stub program to a Win32 program

- **SUBSYSTEM**
  Tells the operating system how to run the .exe file

- **SWAPRUN**
  Tells the operating system to copy the linker output to a swap file before running it

- **TLBID**
  Allows you to specify the resource ID of the linker-generated type library

- **TLBOUT**
  Specifies the name of the .tlb file and other MIDL output files

- **TSAWARE**
  Creates an application that is specifically designed to run under Terminal Server

- **VERBOSE**
  Prints linker progress messages

- **VERSION**
  Assigns a version number

- **WX**
  Treat linker warnings as errors.
CREATING LIBRARIES

Windows based computers support two types of libraries: static and dynamic. A static library is a collection of object files (modules), each containing one or more routines, which are maintained in a single file — a library. When a library file is presented to the linker, modules that are required to satisfy unresolved external references are selected for inclusion into the application file. The advantage of a library is that only those modules that are required to satisfy unresolved external references are linked into the application. The FORTRAN runtime library fmath.lib is an example of a static library. Not every FORTRAN program requires a hyperbolic tangent function, so it is only linked into those programs that require it.

A dynamic library is similar to a static library in that it contains a collection of routines in object modules. The difference is that the elements of the library are not linked into the final application file, but rather are available for linking when the application is executed. The advantage to this type of library is that the individual applications can be smaller and several applications can share the same library. The disadvantage is that the dynamic or shared library must be available on every computer where the application is to be run.

The link command, introduced earlier, is used to create dynamic libraries or DLL’s by combining multiple object files into a single DLL and the loader glue code. The linker can also be used to create static library files, but the library manager, lib, supplied with the Absoft Fortran compiler, is the primary tool for manipulating static libraries. Use lib to create libraries from multiple object files and to add, delete, or replace object modules in existing libraries.

The syntax of the lib command is given below. Options can be preceded with either a dash (-) or a slash (/).

```
lib [option...] [files...]
```

A new library is created by using the -out option to specify the new library name and the files argument to indicate the object files. To add an object file to an existing library, specify both the existing library name and the object file name in the files argument and use the -out option to specify the resulting library name. The input and output library name may be the same.

```
/out:objname
```

This option is used to copy the objname from the library to a file. The output filename is the same as objname unless the /out option is given. /remove and /extract cannot be used at the same time.

```
/list
```

Use this option to display to standard output a list of the object modules in the library.
/out:filename

This option sets the name of the output file.

/remove:objname

Use this option to delete the specified objname from the library. /remove and
/extract cannot be used at the same time.

DLL Import Libraries

The Win32 API supports two types of libraries: static and dynamic, which were
described earlier in this chapter. One requirement to link against a dynamic link library or
DLL is an import library. An import library is a type of library that describes to the linker
all of the available function entry points in the DLL. \texttt{lnk} automatically creates an import
library when it creates a DLL. If you are given a DLL and no import library, you can still
use the DLL by creating the import library yourself, provided you have a definition file
(described below). The utility \texttt{imptool} creates import libraries from definition files.

The syntax of the \texttt{imptool} command is given below. Options can be preceded with either
a dash (-) or a slash (/).

\texttt{imptool \ [option...] \ [files...]}\n
/def:filename

This option is required. It specifies the definition file (described below) which is
used as input to \texttt{imptool}. The file must have an extension of .def.

/out:filename

This option specifies the name of the output file. If it is not used, the name of the
output will be the root of the definition file name with an extension of .lib
added.

/nounderscores

This option specifies that the symbol names not have an underscore prepended to
the entry names.

/w

This option suppresses warnings. \texttt{imptool} will not output any warning messages.

Syntax of a Definition File

The input file for \texttt{imptool} is a definition file. This file describes the entry points into a
dynamic-link library, the DLL. A definition file has a list of the entry point names, and
some keywords to describe them. The syntax of a definition follows.
There are many statements available that can be used in a definition file. Most of them are not supported by imptool, but are used by other applications. imptool will produce a warning, and ignore them. The three statements that imptool supports are.

**NAME [application][BASE=address]**

This statement specifies the application to which the definition file is associated with. If NAME is not specified, the default will be the output file name with a `.dll` extension. The `BASE` keyword has no effect as far as an import library is concerned; it is used by other applications, but is still legal syntax. This statement can only appear on the first line of the definition file; otherwise it will produce a warning. If you use a NAME statement, you cannot use a LIBRARY statement (described next).

**LIBRARY [library][BASE=address]**

This statement specifies the DLL that the definition is associated with. If this is not specified, the default will be the output file name with a `.dll` extension. The `BASE` keyword has no effect as far as an import library is concerned; it is used by other applications, but is still legal syntax. This statement can only appear on the first line of the definition file; otherwise it will produce a warning. If you use a LIBRARY statement, you cannot use a NAME statement.

**EXPORTS definitions**

This statement makes one or more definitions available as exports to other programs. The syntax of an export definition is:

```
entryname[=internalname] [@ordinal[NONAME]] [DATA] [PRIVATE]
```

- **entryname** the name of the entry point into the DLL
- **internalname** is used by other applications and is ignored by imptool
- **ordinal** the ordinal associated with the entry point
- **NONAME** a keyword that means import by ordinal only
- **DATA** a keyword that means you are importing data
- **PRIVATE** a keyword that means to leave the entry name out of the import library

Other statements ignored by imptool are:

- **DESCRIPTION**
- **STACKSIZE**
- **IMPORTS**
- **SECTIONS**
- **VERSION**
- **NAME** (after the first line)
- **LIBRARY** (after the first line)
Comments in a definition file are signified by a semi-colon (;) at the first position of a line.

**Name Mangling**

Entry point names are the most important parameter when creating a definition file. `imptool` will automatically prepend an underscore (_) to all entry point names unless the `/nounderscores` is specified on the command line. If `/nounderscores` is specified, `imptool` will place the entry point in the import library exactly as it appears in the definition file. The best way to insure that your import library will correctly describe the entry points of a DLL is to specify the entry point name exactly as it appears in the DLL and use the `/nounderscores` option. Name mangling, prepending an underscore to a symbol name is standard in the Win32 API, so it is best to add an underscore yourself and use the `/nounderscores` option.

When creating a definition file, keep in mind that using `stdcall` will require you to append some characters to the entry point names that must be present in the definition file, since that is how they appear in the object code and the DLL. The extra characters are a commercial at sign (@) followed by the size, in bytes, of the parameter block for the function or subroutine. An example follows:

```fortran
stdcall function my_func(integer parameter1, integer parameter2)

This should appear as:

_my_func@8

in your definition file.

Because `imptool` will not have any of this information available regarding the type or calling convention used with the function, it will not add any of these characters, so it is important that you append them yourself.

This form of name mangling, appending @ and a number is different from using an ordinal value. If you are specifying an ordinal value, the value is separated by whitespace on the same line as the entry point name. If the value following the @ is the size of the parameter block required when using `stdcall`, it is appended to the entry point name, with no whitespace between. An example to illustrate this follows:

If using `stdcall` as above:

_my_func@8

should be the export line name in your definition file.

If specifying an ordinal value:

_my_func @32

---

*Fortran User Guide*
should be the export line in your definition file.

When you are creating the definition file it is important to keep in mind how the entry point names are being represented. Again, the most direct way is to add the characters that are needed and use the /nounderscores command line option. This will insure that names are handled exactly as they appear, and are not being mangled by imptool in any way.

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 \texttt{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 \texttt{amake} utility itself.

**The Elements of amake**

A key concept in understanding the operation of the \texttt{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 \texttt{amake} program searches through a file 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), \texttt{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 \texttt{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 \texttt{amake}. The term \textit{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 \texttt{amake} tool allows you to define macros, either within the description file itself, or as arguments on the \texttt{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 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:

<table>
<thead>
<tr>
<th>Macro Definition</th>
<th>Macro Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUGOPT = -g</td>
<td>$(DEBUGOPT)</td>
</tr>
<tr>
<td>SRCFILES = one.f two.f</td>
<td>$(SRCFILES)</td>
</tr>
<tr>
<td>OBJFILES = one.obj two.obj</td>
<td>$(OBJFILES)</td>
</tr>
<tr>
<td>ALLFILES = $(SRCFILES) $(OBJFILES)</td>
<td>$(ALLFILES)</td>
</tr>
<tr>
<td>RESFILES = $(RCFILES:.rc=.res)</td>
<td>$(RESFILES)</td>
</tr>
</tbody>
</table>

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.obj two.obj
```

The order of precedence for macro definitions is (from highest to lowest): the *amake* command line, the description file, and the default definitions.
Special macros

The `amake` 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:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$@</td>
<td>Represents the full name of the current target—for use only on a (re)build command line. (When building a library it represents the name of the library.)</td>
</tr>
<tr>
<td>$*</td>
<td>Represents the base name of the current target—for use only on a (re)build command line.</td>
</tr>
<tr>
<td>$&lt;</td>
<td>Represents a current prerequisite—for use only on a (re)build command line.</td>
</tr>
<tr>
<td>$$@</td>
<td>Represents the base name of the current target—for use only on a dependency line.</td>
</tr>
<tr>
<td>$?</td>
<td>Represents a list of prerequisites that have been changed more recently than the current target—for use only on a (re)build command line.</td>
</tr>
</tbody>
</table>

Other special macros that are provided with Absoft `amake` include:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAKE</td>
<td>Used for recursive makes—that is, when a make command is included as part of a description file.</td>
</tr>
<tr>
<td>MAKEFLAGS</td>
<td>Sets the command-line options available to make—usually defined as an environment variable (see <code>Environment Variables</code> later in this section).</td>
</tr>
<tr>
<td>SUFFIXES</td>
<td>Contains the default list of suffixes for the <code>.SUFFIXES</code> special target (see <code>Special Targets</code> later in this section).</td>
</tr>
</tbody>
</table>

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:

\[
\text{target: prerequisite1 prerequisite2...} \\
\text{command(s) to (re)build target}
\]

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:

\[
\text{test: a.f95 b.f95 lib.lib} \\
\text{f95 -o test.exe a.f95 b.f95 lib.lib}
\]

specifies that the target samp.exe is dependent on the prerequisites a.f95, b.f95, and lib.lib. If any of the three prerequisites have been updated at the same time or after the target, test.exe 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 lib.lib; f95 -o test.exe a.f95 b.f95 lib.lib

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 lib.lib; f95 -o test.exe a.f95 \ b.f95 lib.lib
```

### 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:

```
graph.lib:: vertex.f95
 $(F95) /c /O -DDEBUG vertex.f95
 lb /out:$@ $@ vertex.obj
 del vertex.obj

 graph.lib:: edge.f95
 $(F95) /c /g edge.f95
 lb /out:$@ $@ edge.obj
 del edge.obj
```

In this example, two different sets of commands are passed to the Fortran 90/95 compiler during the process of building the library `graph.lib`.

### 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:

```fortran
# program name
NAME = util

# set FLAGS for command line
F95FLAGS = /g
LDFLAGS =

SRCs = util.f95 build.f95 parse.f95 tstring.f95
OBJs = util.obj build.obj parse.obj tstring.obj
PROG = $(NAME)

$(PROG): $(OBJs)
   $(F95) $(F95FLAGS) $(OBJs) /o $(PROG) $(LDFLAGS)

util.obj: util.f95 util.inc tstring.inc decl.inc
build.obj: build.f95 util.inc tstring.inc decl.inc
parse.obj: parse.f95 util.inc tstring.inc decl.inc
tstring.obj: tstring.f95 tstring.inc
dezine.s: /dev/null

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:

```
c:\absoft\bin\default.mk.
```

The following is a list of the default dependency rules included with Absoft *amake* for Fortran 90/95, FORTRAN 77, and C 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. `.f95`:
   ```
   $(F95) $(F95FLAGS) $(LDFLAGS) -o $@ $<
   ```

2. `.f95.obj`:
   ```
   $(F95) $(F95FLAGS) -c $*.f95
   ```

3. `.f95.lib`:
   ```
   $(F95) $(F95FLAGS) -c $*.f95
   $(LIB) $(LIBFLAGS) $@ $*.obj
   $(RM) $*.obj
   ```

4. `.rc.res`:
   ```
   $(RC) $(RCFLAGS) $<
   ```

**Explanation:**

1. Compiles a Fortran 90/95 source file into an executable target.
2. Creates an object file from a Fortran 90/95 source file.
3. Compiles a Fortran 90/95 source file into an object file and then adds it to a library (.lib) file.
4. Creates a resource file from a resource script.
Default Rules for FORTRAN 77 files

(1) .f:
    $(F77) $(FFLAGS) $(LDFLAGS) -o $@ $<

(2) .f.obj:
    $(F77) $(FFLAGS) -c $*.f

(3) .f.lib:
    $(F77) $(FFLAGS) -c $*.f
    $(LIB) $(LIBFLAGS) $@ $*.obj
    $(RM) $*.obj

(4) .rc.res
    $(RC) $(RCFLAGS) $<

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 object file and then adds it to a library (.lib) file.
(4) Creates a resource file from a resource script.

Default Rules for C files

(1) .c:
    $(CC) $(CFLAGS) $(LDFLAGS) $< -o $@

(2) .c.obj:
    $(CC) $(CFLAGS) -c $<

(3) .c.i:
    $(CC) $(CFLAGS) -P $<

(4) .c.lib:
    $(CC) -c $(CFLAGS) $<
    $(LIB) $(LIBFLAGS) $@ $*.obj
    $(RM) $*.obj

Explanation:
(1) Compiles a C source file into an executable target.
(2) Creates an object file from a C source file.
(3) Creates an intermediate (.i) file by running a C source file through the C preprocessor.
(4) Compiles a C source file into an object file and then adds it to a library (.lib) 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 `amake` 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 `amake` 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:

```plaintext
# program name
NAME = util

# set FLAGS for command line
F95FLAGS = /g
LDFLAGS =

SRCS = util.f95 build.f95 parse.f95 tstring.f95
OBJ = util.obj build.obj parse.obj tstring.obj
PROG = $(NAME)

.f95.obj:
  $(F95) $(F95FLAGS) /c $<
  copy $< c:\usr\workdir

$(PROG): $(OBJ)
  $(F95) $(F95FLAGS) $(OBJ) /o $(PROG) $(LDFLAGS)

util.obj: util.f95 util.inc tstring.inc decl.inc
build.obj: build.f95 util.inc tstring.inc decl.inc
parse.obj: parse.f95 util.inc tstring.inc decl.inc
tstring.obj: tstring.f95 tstring.inc

The user-supplied rule:

.f95.obj:
  $(F95) $(F95FLAGS) -c $<
  copy $< c:\usr\workdir

will override the corresponding default rule in the `default.mk` file:
```
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.

`amake` command-line options may specified with either a dash (`-`) or a slash (`/`):

- `/d` Lists the prerequisites for each dependency block that caused `amake` 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 `amake` 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.
Displays all commands, but does not execute them. (Command lines beginning with an @ character are also displayed.) Useful in debugging/testing description files.

Prints a complete list of macro definitions, dependency blocks, and suffix rules.

Returns a zero or nonzero status code depending on whether the target is or is not up-to-date, respectively. Useful when amake is called from a script or tool that requires the current target.

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 amake operates.

Does not print command lines before executing. This is equivalent to using the .SILENT special target in a description file.

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 predefined 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 /i 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 /s 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 :
   del *.obj

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.obj suffix rule and the dependency blocks (lines) following it:
# program name
NAME = util

# set FLAGS for command line
F95FLAGS = /g
LDFLAGS =

SRCS = util.f95 build.f95 parse.f95 tstring.f95
OBJJS = util.obj build.obj parse.obj tstring.obj
PROG = $(NAME)

.f95.obj:
   $(F95) $(F95FLAGS) /c $<
   copy $< c:\usr\workdir

$(PROG): $(OBJJS)
   $(F95) $(F95FLAGS) $(OBJJS) /o $(PROG) $(LDFLAGS)

util.obj: util.f95 util.inc tstring.inc decl.inc
build.obj: build.f95 util.inc tstring.inc decl.inc
parse.obj: parse.f95 util.inc tstring.inc decl.inc
tstring.obj: 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**

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Badly formed macro</td>
<td>Incorrect syntax for a macro definition—often, the macro name is missing.</td>
</tr>
<tr>
<td>Improper macro</td>
<td>An error occurred during macro expansion. Often, the problem is a missing parenthesis or bracket.</td>
</tr>
<tr>
<td>Macro too long ...</td>
<td>A macro name is too long; cannot be longer than 100 characters.</td>
</tr>
<tr>
<td>Rules must be after target</td>
<td>Occurs when a line beginning with a space or tab has been encountered before a dependency line in a description file.</td>
</tr>
</tbody>
</table>
Other Common Errors

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cannot open file</td>
<td>The description file specified in an include directive could not be found or was not accessible. (See Using include directives earlier in this section.)</td>
</tr>
<tr>
<td>Don't know how to make target</td>
<td>There is no target entry in a description file, none of the default rules apply, and there is no .DEFAULT rule.</td>
</tr>
<tr>
<td>Too many options</td>
<td>The amake program has exceeded the allocated space while processing command-line options or a target list.</td>
</tr>
<tr>
<td>Too many rules defined for target</td>
<td>Multiple sets of rules have been defined for a target; targets may only have one set of rules.</td>
</tr>
<tr>
<td>Unexpected end of line seen</td>
<td>The colon in a dependency line is missing.</td>
</tr>
</tbody>
</table>

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.
ABSOFT FORTRAN 90/95

This section describes the features, capabilities, and extensions in the implementation of Absoft Fortran 90/95 not discussed in language reference manuals.

Compiler Directives

This section describes additional compiler directives available with the Fortran 90/95 compiler. 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 non-blank 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.

PACKON Directive

The PACKON directive allows you to control the memory alignment of the components of a derived type with a SEQUENCE attribute:

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

where: 1, 2, and 4 specify one-, two-, and four-byte boundaries respectively. The default alignment in the absence of an explicit boundary argument is 1-byte boundaries.

A PACKON directive affects the memory alignment of data elements in the current program unit being compiled. It is reset to the default after the end of each program unit.
PACKOFF Directive

Use the PACKOFF directive to restore the memory alignment of derived type components with a SEQUENCE attribute to the default.

!DIR$ PACKOFF

Keywords

This section describes the syntax and use of language extensions available in the form of additional keywords accepted by the Absoft Fortran 90/95 compiler. The primary purpose of these extensions is to assist in converting from other implementations of Fortran 90/95 and for accessing the Win32 API. Using any of the language extensions detailed below impacts the portability of your program.

BYTE

This type specifier can be used anywhere the INTEGER specifier can be used. It cannot be adorned with a KIND attribute specifier or a FORTRAN 77-style *len length specifier:

BYTE [[,attr_spec]...::] entity_decl_list

where: attr_spec is an attribute specification

entity_decl_list is: object_name[array_spec]

DLL_IMPORT

This keyword informs the compiler that the function is to be imported from a DLL. The name mangling (decorations) and call mechanism generated will be appropriate for the compiler options specified (see the chapter, Using the Compilers).

DLL_EXPORT

This keyword informs the compiler that the function is to be an entry exported from a DLL. The name mangling (decorations) and call mechanism generated will be appropriate for the compiler options specified (see the chapter, Using the Compilers).

INQUIRE

The following additional specifiers are available for the INQUIRE statement:

ACCESS = a CHARACTER variable or array element which is defined with the value SEQUENTIAL, DIRECT, or TRANSPARENT depending on the access mode.

BLOCKSIZE = an INTEGER variable or array element which is defined with the size of the physical input/output buffer.
CARRIAGECONTROL = a CHARACTER variable or array element which is defined with the value FORTRAN or LIST. The value FORTRAN indicates that the first character in each record will be interpreted for carriage control.

FLEN = an INTEGER variable or array element which is defined with the length of the file in bytes.

OPEN

The following additional specifiers are available for the OPEN statement:

ACCESS = a CHARACTER expression which must be SEQUENTIAL, DIRECT, or TRANSPARENT. ACCESS=TRANSPARENT is equivalent to the specifier: FORM=BINARY.

BLOCKSIZE = an INTEGER expression which specifies the size of the physical input/output buffer.

CARRIAGECONTROL = a CHARACTER expression which must be FORTRAN or LIST. Setting the value to FORTRAN causes the first character in each record to be interpreted for carriage control. Setting the value to LIST has no effect and is only supported for compatibility.
UNION and MAP

A UNION declaration defines a data area which is shared by one or more fields or groups of fields. Storage allocation associated in this manner must be a component of a derived type, which itself must contain a SEQUENCE statement. The declaration begins with a UNION statement and ends with an END UNION statement. In between are MAP and END MAP statements which define the field or groups of fields which will share the storage area. A UNION declaration is as follows:

```
UNION
  map-declaration
  map-declaration
  [map-declaration]
  ...
  [map-declaration]
END UNION
```

where: map-declaration takes the following form:

```
MAP
  field-declaration
  [field-declaration]
  ...
  [field-declaration]
END MAP
```

A field-declaration contains one or more of the following, a UNION declaration or a component definition statement. Field-declarations cannot have been previously declared or be dummy arguments.

The size of the shared data area for the union is the size of the largest map area contained within the union. The fields of only one of the map areas are defined at any given time during program execution.

Example:

```
UNION
  MAP
    INTEGER*4 long
  END MAP
  MAP
    INTEGER*2 short1, short2
  END MAP
END UNION
```

VALUE

The keyword VALUE can be used both as a declaration statement and as a type attribute. The VALUE keyword can only be associated with a dummy argument and indicates that
the argument is passed by value rather than by reference (the normal Fortran 90/95 method of argument passing).

If an interface definition of a program unit contains dummy arguments with a VALUE attribute, then the actual arguments at the point of reference will automatically be passed by value.

**Data initialization**

Extended, FORTRAN 77-style initialization in a type statement is accepted provided that the Fortran 90/95 “::” separator form of type declaration is not used:

```
type [*len] v[/value/] [,v[/value/]]...
```

where: type can be any of the following specifiers: LOGICAL, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, or DOUBLE COMPLEX.

len is an unsigned integer constant that specifies the length, in bytes, of a variable, an array element, a symbolic constant, or a function.

v is the symbolic name of a variable, an array, a constant, a function, a dummy procedure, or an array declaration.

value is an optional initial value for the preceding variable or array. When initializing an array, value must contain constants separated by commas for each element of the array.

**Fortran 90/95 Module Files**

MODULE definitions can be used in multiple files in one of two ways. Either place them in INCLUDE files and include them in the files that USE them, or place them in a source file and compile it to create pre-compiled MODULE definition files. Precompiled MODULE files are distinguished with the .mod extension given to them by the compiler when they are created. The advantage of pre-compiled MODULE definition files is the increased speed of compilation of files that USE them.

The compiler will create a separate MODULE definition file for each MODULE encountered in the source file. The name of the precompiled MODULE file will be the name of the MODULE as specified in the source with an extension of .mod. MODULE definition files in the current working directory will automatically be scanned. Use the -p option (see the section Module File(s) in the chapter Using the Compilers) if the MODULE definition file resides in a different directory. Precompiled MODULE definition files are only opened and scanned if a USE statement is encountered in the source program being compiled and the source program does not contain a definition of the referenced MODULE.
MODULE definition files may be compiled at the same time as program units using them provided that they are presented to the compiler on the command line before the source file containing program unit(s) using them.

If a MODULE definition file is compiled separately and it contains subprograms, an object file will be produced with the same name as the module definition source file and an extension of .obj. Only one .obj file is produced for each source file. The object file name produced by this compilation must be presented to the linker in order for the program to link properly.

If a MODULE definition file is compiled separately and it contains data initialization (DATA statements, Fortran 90/95 pointers, or allocatable arrays), an object file will be produced with the same name as the module definition source file and an extension of .obj. Only one .obj file is produced for each source file. The object file name produced by this compilation must be presented to the linker in order for the data to be initialized properly. Note that if the object file name is not presented to the linker, the program will link, but the initialization(s) will not take place.
CHAPTER 7
Windows Programming

This chapter covers the broad topic of programming for Microsoft Windows using both MRWE and the Win32 API. It covers additional topics of interest such as interfacing with other languages. Where appropriate, sample programs in the \texttt{C:\Absoft\Examples} and \texttt{C:\Absoft\Examples\Mrwe} directories are referenced.

USING MRWE

The MRWE application framework is a special feature of Pro Fortran that automatically give your program a Microsoft Windows interface with menus, a text window, and a graphics window. Usually, when you want to create applications with windows and menus, you need to know how to use the Win32 API and the Microsoft user-interface guidelines. When you create an application using MRWE, your program will automatically exhibit these features without the need for intensive Windows programming. MRWE is convenient and flexible to use, and can be modified to suit your needs.

MRWE is a collection of pre-compiled FORTRAN routines contained in the library file \texttt{mrwe.lib} that you can link with your program. Applications generated by an Absoft compiler are linked with this library when you select \texttt{MRWE Application} as shown in the dialog box below. The FORTRAN source code for MRWE is in the \texttt{Examples\Mrwe} folder and can be used as an example of how to call Win32 API routines.

The Absoft Developer Tools Interface allows you to select the format of the executable file. A \texttt{Target Type} setting of \texttt{MRWE Application} will cause your program to be built as a stand-alone, double-clickable application with MRWE providing a window and menus interface. If your FORTRAN code includes its own Windows message loop and Win32 API calls, you should disable MRWE by choosing \texttt{Console Application} for the \texttt{Target Type} of the executable file.
The MRWE source code is an excellent example of programming Windows using the Win32 API. It shows how to use multiple windows, scroll bars, menus, text, graphics, the mouse, dialog boxes, and message handling; how to print and save files; and everything else needed in a standard Windows application. The `main.f` program provided in the MRWE folder shows how to use all of MRWE capabilities; type `amake -f makefile.dev` from a command prompt to build it. Feel free to use portions of the MRWE source code in applications of your own that are compiled with Absoft Pro Fortran.

**The MRWE Window**

When you start 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 MRWE Application Framework

Standard input and output directed to the MRWE window are preconnected to the FORTRAN I/O units 5, 6, 9 and *. Any of these except * may be connected instead to an actual 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.

Additional MRWE text windows files may be opened and connected to Fortran units with extensions to the OPEN statement described later in this chapter in the section Working With Text in MRWE.

How Your Program And MRWE Work Together

When your program is started, control is first passed to MRWE, which sets up a Windows environment with menus and the standard input and output window. After the text window appears, the application begins executing your FORTRAN program. By default, you cannot pull down menus, save the output window, or terminate the application until an I/O statement occurs. Your application is completely dedicated to executing your FORTRAN program at maximum speed.

During any input and output statements (i.e. READ, WRITE, ENCODE, etc.), you can use the menus and their commands. When the program is waiting for you to type standard input, you also have complete control of the menus and normal mouse actions. If you want the ability to save the output window or terminate your application while it is calculating,
select the MRWE Events check box (the -N9 compiler option) before compiling the program.

The -N9 compiler option turns on the timer interval feature. Your program will be interrupted at timed intervals to check for Windows messages such as menu selections and mouse activities. The default interval is one-tenth second, which can be changed as described later in this chapter in the Modifying MRWE section. Without the Timer Interval option activated, checking for Windows messages will still occur, but only during FORTRAN input and output statements.

If you did not include the -N9 compiler option when compiling, and your program is executing lengthy calculations that you want to stop, you can regain control by typing the Ctrl, Alt, and Delete keys simultaneously. You will be given an opportunity to close the program. [Caution: Data in files that were open may be lost during this procedure.]

Working With Text in MRWE

An MRWE application is an MDI (Multiple Document Interface) type of application. This means that there is application frame window that can support multiple document windows within that frame. MRWE currently supports two document window types—text and graphics. The application frame window that appears is named after the FORTRAN application. The default text window connected to standard input and output is initially titled Input/Output, but changes when the text is saved to a file.

You can only type into a text window when a READ statement is currently active for the window, and only 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. Tab characters (ASCII 9) read from formatted files are expanded modulo TABSIZE where TABSIZE is an environment variable. If TABSIZE is not set, tabs are expanded modulo 8. If TABSIZE is set to 0, the tab is passed unmolested to the application.

Additional text windows can be created with the Fortran OPEN statement setting the ACCESS= specifier to:

"window [,height, width]"

The optional arguments, height and width, are integers which specify the dimensions of the window in pixels. The window title will be the argument of the FILE= specifier of the OPEN statement.

An MRWE window has a text limit of 262144 characters in 32768 lines. See the Modifying MRWE section for information changing these parameters.
mrwe_clear function

The mrwe_clear subroutine can be used to clear an MRWE text buffer of accumulated output, clear the window and redraw it, and reposition the insertion caret in the upper left corner of the window. The calling sequence is:

```
subroutine mrwe_clear(n)
```

where \( n \) specifies the unit number connected to the window. Use \( n=0 \) for the default text window connected to standard input and output.

Working With Graphics in MRWE

By default, the graphics window is not displayed when an MRWE application begins. You must explicitly open the graphics window by referencing the mrwe_Opengraphics function if you wish to use it. The argument list for the mrwe_O.opengraphics function is:

```
logical function mrwe_O.opengraphics(x, y, title, mode, draw)
```

where:

- \( x \) is an integer expression specifying the initial width of the window. If \( x = 0 \), the default is 600.
- \( y \) is an integer expression specifying the initial height of the window. If \( y = 0 \), the default is 300.
- \( title \) is a character expression specifying the title of the window. If \( title \) consists of only spaces, the window title is set to “Graphics”.
- \( mode \) is an integer expression specifying the mapping mode to be used in the Device Context (see below). If \( mode = 0 \), the mapping mode is set to MM_TEXT.
- \( draw \) is the name of the subroutine that will be called when MRWE receives an MM_PAINT message (see below). If \( draw = 0 \), your program will not respond to MM_PAINT messages.

A Windows Device Context provides a device independent graphics context for your program to draw in. When you draw into a device context, your program does need to be concerned with whether that device is a monitor, a printer, or a plotter – the device context handles the necessary transformations.

The mapping mode controls the conversion of logical units to device units (pixels on a monitor, “dots” on a printer). The origin is the upper left corner of the device and positive \( x \) is to the right – the direction of positive \( y \) depends on the mapping mode. The following logical unit mapping modes are available:

- **MM_TEXT**
  - one logical unit is equal to one device unit; positive \( y \) is down.
- **MM_HIENGLISH**
  - one logical unit is equal to 0.001 inch; positive \( y \) is up.
MM_LOENGLISH  one logical unit is equal to 0.01 inch; positive y is up.

MM_HIMETRIC  one logical unit is equal to 0.01 millimeter; positive y is up.

MM_LOMETRIC  one logical unit is equal to 0.1 millimeter; positive y is up.

MM_TWIPS  one logical unit is equal to one twentieth of a printer’s point (1/1440 inch); positive y is up.

MM_ANISOTROPIC  logical units are mapped to arbitrary units with arbitrary scaled axes. Use SetWindowExtEx and SetViewportExtEx to specify units, scaling, and orientation.

MM_ISOTROPIC  logical units are mapped to arbitrary units with equally scaled axes. Use SetWindowExtEx and SetViewportExtEx to specify units, scaling, and orientation.

In Windows, you normally perform your drawing operations in response to a WM_PAINT message (for more information on Windows messages, see Program Organization: FORTRAN vs. Windows, later in this chapter). The WM_PAINT message is issued to your program whenever an event transpires that requires your window to be redrawn, such as resizing it or uncovering it. To have your program respond to WM_PAINT messages, register your drawing subroutine with MRWE as the last argument in the reference to mrwe_OpenGraphics:

```fortran
external OnDraw

logical function mrwe_OpenGraphics(0, 0, ",", MM_HIENGLISH, OnDraw)
```

Begin your OnDraw function as follows:

```fortran
subroutine OnDraw(dc)

integer, INTENT(IN):: dc
```

When the MRWE message loop receives a WM_PAINT message, it will perform the necessary drawing initialization and call your subroutine with the handle of the device context. Your subroutine can begin drawing immediately. For example:

```fortran
integer x, y
logical:: result
record /POINT/ point

result = MoveToEx(dc, x, y, loc(point))
result = LineTo(dc, x, y)
```
Adding Menu Commands to MRWE

MRWE provides a number of functions for adding menu commands, providing routines that respond to user menu selections, and modifying menu items. New menus and menu commands are added to the MRWE default menu (described later).

**mrwe_InsertMenu function**

Use this function insert a new menu on the menu bar or insert a menu command in an existing menu:

```fortran
logical function mrwe_InsertMenu(menu,item,flags,title,routine)
```

**where:**

- `menu` is an integer expression specifying the id of the menu. Menu ids are numbered from 1 on the far left. If `item` (described next) is 0, indicating a new menu is to be inserted, then `menu` indicates the id where the new menu will be inserted.

- `item` is an integer expression specifying the insertion point of the command in the menu identified by `menu`. The first menu command is 1. If `item` is 0, the `mrwe_InsertMenu` function will insert a new menu at the position indicated in `menu`.

- `flags` is an integer expression specifying any special characteristics initially associated with the menu command. See below for a description of the possible values that may be used.

- `title` is a character expression that specifies the title of the new menu or menu command.

- `routine` is the name of the subroutine that will be called when this menu command is chosen.

**flags** may be any of the following:

- **MF_CHECKED** Places a check mark next to the menu item.
- **MF_UNCHECKED** Does not place a check mark next to the menu item. (default)
- **MF_ENABLED** Enables the menu item so that it can be selected. (default)
- **MF_DISABLED** Disables the menu item so that it cannot be selected, but does not gray it.
- **MF_GRAYED** Disables the menu item so that it cannot be selected and grays it.
- **MF_SEPARATOR** Draws a horizontal dividing line.
mrwe_AppendMenu function

To append a new command to the end of an existing menu, use the function:

```fortran
logical function mrwe_AppendMenu(menu, flags, title, routine)
```

where:  
- `menu` is an integer expression specifying the id of the menu. Menu ids are numbered from 1 on the far left.

- `flags` is an integer expression specifying any special characteristics initially associated with the menu command. See the mrwe_InsertMenu Function section for a description of the possible values that may be used.

- `title` is a character expression that specifies the title of the new menu or menu command.

- `routine` is the name of the subroutine that will be called when this menu command is chosen.

mrwe_CheckMenuItem subroutine

The mrwe_CheckMenuItem subroutine can be used to set or clear a check. The calling sequence is:

```fortran
subroutine mrwe_CheckMenuItem (menu, item, flags)
```

where:  
- `menu` is an integer expression specifying the id of the menu. Menu ids are numbered from 1 on the far left.

- `item` is an integer expression specifying the command in the menu identified by `menu`. The menu itself is identified with a 0 and the first menu command is 1.

- `flags` is an integer expression specifying either MF_CHECKED or MF_UNCHECKED. See the mrwe_InsertMenu Function section for a description of the possible values that may be used.

mrwe_EnableMenuItem subroutine

The mrwe_EnableMenuItem subroutine can be used to enable or disable a command. The calling sequence is:

```fortran
subroutine mrwe_EnableMenuItem (menu, item, flags)
```

where:  
- `menu` is an integer expression specifying the id of the menu. Menu ids are numbered from 1 on the far left.

- `item` is an integer expression specifying the command in the menu identified by `menu`. The menu itself is identified with a 0 and the first menu command is 1.
flags is an integer expression specifying any special characteristics to be associated with the menu command. See the mrwe_InsertMenu Function section for a description of the possible values that may be used.

mrwe_DeleteMenu function

The mrwe_DeleteMenu function can be used to delete an entire menu or a single menu command. The function reference arguments are:

```fortran
logical function mrwe_DeleteMenu(menu, item)
where: menu is an integer expression specifying the id of the menu. Menu ids are numbered from 1 on the far left.
      item is an integer expression specifying the command in the menu identified by menu. The menu itself is identified with a 0 and the first menu command is 1.
```

Using Timer Functions in MRWE

Timers allow you to execute sections of your program at timed intervals. With timers, you can animate your program, you can save important data to files at regular intervals, and so on. Timers work by sending a WM_TIMER message to the message handler in MRWE. Since MRWE receives all of the messages, your program must be compiled with -N9 compiler option to allow the message loop to be entered periodically.

To install a timer, use the mrwe_SetTimer function:

```fortran
logical function mrwe_SetTimer(id, timeout, routine)
where: id is an integer expression specifying the id of the timer. This value is also used with mrwe_KillTimer to remove the timer.
      timeout is an integer expression specifying the number of milliseconds between timer messages.
      routine is the name of the subroutine in your program that will be called when the timeout interval has elapsed.
```

To remove a timer, use the subroutine:

```fortran
subroutine mrwe_KillTimer(id)
where: id is an integer expression specifying the id of the timer. This must the same value used with mrwe_SetTimer to install the timer.
```
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 the text in the Input/Output window to a file, choosing printers and printer options, printing the text, and exiting the application.

Save

This command saves the MRWE Input/Output window to a text file. The first time the text is saved, the window title is changed to the file name.

Save As…

This command saves the MRWE Input/Output window to a text file. It displays a standard file dialog prompting for a file name in which to save the text. If the file already exists, you will be prompted to overwrite it. After the text is saved, the window title is changed to the file name.

Print…

To print the contents of the window to the printer, use this menu item. A dialog box is displayed allowing you to choose the printer and set various other print options.

About…

This dialog box displays various parameters of the program execution environment including page size, minimum and maximum memory addresses, and CPU type.

Exit

This command is used to exit the application.

Edit Menu

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

Undo (Ctrl+Z)

This command is not used by MRWE and is always disabled (grayed).
Cut (Ctrl+X)

This command removes the selected text from the window and places it on the Clipboard. Text in the Clipboard may be pasted into other applications. Like other editing commands, this command is only available during a READ statement unless the program was compiled with -N9 option.

Copy (Ctrl+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. Like other editing commands, this command is only available during a READ statement unless the program was compiled with -N9 option.

Paste (Ctrl+V)

This command replaces the selected text in the window with text from 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 unless the program was compiled with -N9 option.

Delete

This command clears the selected text.

Format Menu

The Format menu commands allow you change the font style and color used to display text in MRWE text windows.

Font...

This command opens a standard font dialog.

Color

This command opens a standard color dialog.

Window Menu

The Window menu contains the standard editing commands for cutting, pasting, and copying text.
Cascade (Shift+F5)

The **Cascade** command is used to arrange multiple open windows in an overlapped or cascaded manner.

Tile (Shift+F6)

Use this command to tile multiple windows within the MRWE frame.

Arrange Icons

Use the **Arrange Icons** command to order minimized MRWE window icons.

Modifying MRWE

MRWE is written entirely in FORTRAN and the source code is provided in the directory: `C:\Absoft\Examples\Mrwe`. There is a `makefile` supplied to assist you in rebuilding the library if you choose to modify it. To change the maximum number of characters or lines that MRWE can accommodate, edit the `mrwe.inc` include file and change the `PARAMETER` statements for the two symbolic constants `MAX_CHARS` and `MAX_LINES`. Then rebuild the library with the `amake` command.

A program linked with the MRWE library normally only checks for Windows messages during Input/Output statements. If you compile your program with the `-N9` option (the **MRWE Events** check box in the compiler graphical interface), the compiler inserts code throughout your program to check if a timer has gone off. If the timer has gone off, a Windows message loop is entered to process any pending messages and then control is returned to your program. You can fine tune the timer for your specific application by including the following `COMMON` block declaration in your program:

```fortran
INTEGER mrwe_flag ! check for messages flag
INTEGER mrwe_delay ! delay in milliseconds between checks
INTEGER mrwe_resolution ! resolution of the timer

COMMON /call_mrwe_timer/ mrwe_flag, mrwe_delay, mrwe_resolution
```

The flag is usually set by the timer interrupt routine, but you can force a check of the message queue by setting `mrwe_flag` to 1. The default delay is 1/10 of a second (100 milliseconds) and can be changed with the `mrwe_delay` variable. `mrwe_resolution` controls the resolution of the interrupt timer.

USING THE WIN32 API

This section describes how to access the routines available in the Win32 API. For a complete explanation of all the Windows systems routines, refer to *Microsoft Win32 Programmers Reference, Volumes 1-5*, listed in the bibliography. Written by Microsoft, the *Programmers Reference* is the definitive resource for Windows programming and is essential if you plan to use system calls directly.
You will also find a reference to the Win32 API in the Absoft Pro Fortran menu selection labeled Win32 SDK Help.

**Program Organization: FORTRAN VS. WINDOWS**

Usually a FORTRAN 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.

Windows, on the other hand, presents a very different, highly interactive environment to the user. The interface provided on Windows 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. Windows communicates user requests to the program through a mechanism known as a *message* 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 a message it processes it, carrying out whatever action is required.

These actions are usually directed by a *message loop*, in which the program requests from Windows any messages pending for the program, carries out the directed action, and then repeats the process (see figure above). When no messages are pending, the program will carry out whatever other procedures it was designed to do. Some programs may 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 Windows 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 Windows look and feel to it. MRWE can add most of these features to your program without delving into the details of Windows programming.

Note that MRWE provides a more than adequate Windows look and feel to the typical ported FORTRAN program. However, you cannot avoid using the Win32 API to access many of the features of Windows. Given the proper reference materials (such as the Programmers Reference mentioned above) programming Windows need not be as difficult as the above discussion may have led you to believe—you may even find it rewarding.

**How Absoft Fortran 90 Interfaces with Windows**

A group of FORTRAN module files with the extension “.mod” allows Absoft Fortran 77 to interface with the Win32 API for Windows. These files contain FORTRAN declarations for every routine and most of the structures, types, and constants defined in Microsoft Win32 Programmers Reference, Volumes 1-5. To include one of these files, use a statement such as:

```fortran
use windows
```

This USE statement will define all the GDI information for a single program unit.

**Locating the Interface Files**

The “.mod” files are located in the directory C:\Absoft\F90INC. This is the default directory for the Absoft FORTRAN 90 system module files so there is no need to completely specify the full path—the file name alone is sufficient.

If, for any reason, you move the system MODULE files to another directory, you must inform the compiler of the new location by using the -P compiler option (Module File Paths). See the chapter, Using the Compilers for more information.

**Passing Arguments**

By language definition, FORTRAN passes all actual arguments and receives all dummy arguments by reference. Absoft Fortran 77 and Fortran 90 can also pass arguments by value using the VAL() function and can receive dummy arguments by value using the VALUE statement.

Standard FORTRAN 77 and Fortran 90 pass all arguments by reference. This means that the memory address containing the argument is sent to the called subprogram, thereby allowing the subprogram to return information to the calling program as well as to use the value passed. Other languages, such as Pascal and C, allow a choice of passing by reference or passing by value. Passing by value is a one way communication. A copy is created of the argument and passed to the subprogram. This copy may be used and modified but the actual argument is not changed.
Absoft Fortran 77 and Fortran 90 are designed to interface directly to the C programming language enabling FORTRAN programs to call Win32 API routines with the only requirement being careful attention to argument data types. You must be careful to pass the proper kind and type of argument to the Win32 API routines. Failure to do so can cause incorrect results or even program crashes. Passing by reference is simple because it is the standard FORTRAN convention. Passing by value requires using the FORTRAN VAL function corresponding to the parameter size (1, 2, or 4 bytes). The VAL function must be used to pass all pointers and literal values, whether variables or numeric constants, to any C language argument that is not declared as a pointer type. See the section Value Parameters for C later in this chapter for further details. To determine the correct argument order and type, refer to the Programmer's Reference and to the appropriate “.inc” files containing the routines you want to call. The following FORTRAN 77 code calls the LineTo routine with three “pass by value” arguments:

```fortran
CALL LineTo(val(hdc),val(50),val(50)) ! x=50, y=50
```

The Absoft Fortran 90 VALUE keyword can be associated with a dummy argument and indicates that the argument is passed by value rather than by reference. The following FORTRAN 90 code shows this relationship for the LineTo routine:

```fortran
INTERFACE
 STDCALL FUNCTION LineTo(dummy1, dummy2, dummy3) &
 & RESULT(FUNCTION RESULT)
 INTEGER(4), VALUE :: dummy1
 INTEGER(4), VALUE :: dummy2
 INTEGER(4), VALUE :: dummy3
 INTEGER(4) :: FUNCTION_RESULT
 END FUNCTION LineTo
ENDINTERFACE

INTEGER :: fresult
fresult = LineTo(hdc, 50, 50)
```

The C declaration for LineTo (from the Microsoft Win32 Programmer's Reference, Volume 4):

```c
BOOL LineTo(hdc,nXEnd,nYEnd)
HDC hdc; /* device context handle */
int nXEnd; /* x-coordinate of line's ending point */
int nYEnd; /* y-coordinate of line's ending point */
```

All variables passed must have the same size as the corresponding routine parameters. Passing more complex parameter types is discussed later in this chapter.

All pointers and handles are 32 bits and must be declared as INTEGER*4 values or as POINTERS and passed as arguments with the VAL4 function. These calls pass a pointer to the GetCaretPos routine:

```fortran
INTEGER*4 p_point
CALL GetCaretPos (VAL4(p_point))
```
150 Windows Programming

or

    RECORD /POINT/ point  
    POINTER (p_point, point)  
    CALL GetCaretPos (VAL4(p_point))

or

    RECORD /POINT/ point  
    POINTER (p_point, point)  
    CALL GetCaretPos (point)

For Absoft FORTRAN 90 this might look like the following:

    TYPE (POINT) p  
    POINTER (p_point, p)  
    INTEGER fresult  
    fresult = GetCaretPos(p_point)

CHARACTER Arguments

The binary interface to the Windows system functions on computers using Intel processors does not adhere to the same CALL/RETURN sequence that FORTRAN and C use. This discrepancy is resolved by the include files for each compiler which make a special declaration for all of the Win32 API functions: STDCALL. This declaration directs the compiler to generate the special code sequences required to interface to the system.

Normally, you do not need to be concerned about this difference since proper use of the include files insures that the calls are made correctly. However, CHARACTER arguments need special attention when calling Win32 API functions that require pointers to strings as parameters. The Absoft Fortran 77 compiler adds extra arguments to subroutine and function argument lists which specify the length of CHARACTER arguments. The extra arguments are placed after the formal parameter list and are used by the called procedure to determine the size of CHARACTER arguments declared with the *(*) length specifier. Unfortunately, these length arguments are not recognized by the Windows functions on Intel based systems and you cannot pass CHARACTER arguments in the normal manner. There are three ways to successfully pass pointers to FORTRAN CHARACTER arguments that are equivalent to:

    CHARACTER*256 WindowName  
    INTEGER hWnd  
    .  
    .  
    .  
    call SetWindowText(val(hWnd),WindowName)

but do not pass the additional hidden length argument.

1) Take the address of the argument and pass it by value:

    CHARACTER*256 WindowName
INTEGER hWnd
.
.
call SetWindowText(val(hWnd),VAL(LOC(WindowName)))

2) Create a pointer to the argument and pass the pointer

CHARACTER*256 WindowName
POINTER (p_WindowName,WindowName)
INTEGER hWnd
.
.
call SetWindowText(val(hWnd),VAL(p_WindowName))

3) Using CARG with Absoft FORTRAN 90

CHARACTER*256 WindowName
INTEGER hWnd, fresult

Fresult = SetWindowText(hWnd, CARG(WindowName))

IMPLICIT NONE Statement

Use of the IMPLICIT NONE statement is strongly recommended when using “.inc” files. The IMPLICIT NONE statement informs the compiler to issue warnings for all undeclared variables, thereby forcing explicit declaration of all variables used. Placing this statement at the beginning of each program unit is the easiest way to avoid difficulties due to undeclared or misspelled variable names. The following code fragment is an example of using IMPLICIT NONE:

stdcall function mrwe_AbortDlgProc (hDlg, message, wParam, lParam)
use windows
implicit none
logical mrwe_AbortDlgProc
integer, value :: hDlg
integer, value :: message
integer, value :: wParam
integer, value :: lParam

include "print.inc"

if (message == WM_COMMAND) then
  mrwe_AbortDlgProc = .TRUE.
  print_flag        = .FALSE.
else
  mrwe_AbortDlgProc = .FALSE.
end if
end
More details about the `IMPLICIT` statement can be found in the *FORTRAN 77 Language Reference Manual* and the *Fortran 90 Concise Reference*.

**Using Derived TYPES, STRUCTUREs and RECORDs with Win32**

Some of the Win32 routines require structures as parameters. Although standard FORTRAN 77 does not have structures, Absoft Fortran 77 supports VAX-compatible structures that may be interchanged with C structures.

Here is an example of the `SYSTEM_INFO` structure described in Volume 5 of the *Microsoft Win32 Programmers Reference* and defined in the `winbase.h` C header file:

```c
typedef struct SYSTEM_INFO {
    DWORD dwOemId;
    DWORD swPageSize;
    LPVOID lpMinimumApplicationAddress;
    LPVOID lpMaximumApplicationAddress;
    DWORD dwActiveProcessorMask;
    DWORD dwNumberOfProcessors;
    DWORD dwProcessorType;
    DWORD dwReserved1;
    DWORD dwReserved2;
} SYSTEM_INFO *LPSYSTEM_INFO;
```

The equivalent FORTRAN 77 structure is in the `INCLUDE` file `winbase.inc`:

```fortran
STRUCTURE /SYSTEM_INFO/
    INTEGER*4 dwOemId
    INTEGER*4 swPageSize
    INTEGER lpMinimumApplicationAddress
    INTEGER lpMaximumApplicationAddress
    INTEGER*4 dwActiveProcessorMask
    INTEGER*4 dwNumberOfProcessors
    INTEGER*4 dwProcessorType
    INTEGER*4 dwReserved1
    INTEGER*4 dwReserved2
END STRUCTURE
```

The equivalent FORTRAN 90 type as referenced by the window module is in the `INCLUDE` file `windows.inc`:

```fortran
TYPE SYSTEM_INFO
    SEQUENCE
        TYPE(TSTRUCTRBwindows) :: genfieldS
        INTEGER(4) :: dwPageSize
        INTEGER(4) :: lpMinimumApplicationAddress
        INTEGER(4) :: lpMaximumApplicationAddress
        INTEGER(4) :: dwActiveProcessorMask
        INTEGER(4) :: dwNumberOfProcessors
        INTEGER(4) :: dwProcessorType
        INTEGER(4) :: dwAllocationGranularity
        INTEGER(2) :: wProcessorLevel
        INTEGER(2) :: wProcessorRevision
ENDTYPE SYSTEM_INFO
```
This FORTRAN 77 sample program uses the SYSTEM_INFO structure:

```fortran
PROGRAM ProcessorType
INCLUDE "windows.inc"
CHARACTER*7 processor
RECORD /SYSTEM_INFO/ sinf

CALL GetSystemInfo(sinf)
IF (((sinf.dwProcessorType == 386) .OR. +
   (sinf.dwProcessorType == 486) .OR. +
   (sinf.dwProcessorType == 586) THEN
   processor = "Intel"
ELSE IF (sinf.dwProcessorType == 601) THEN
   processor = "IBM"
ELSE
   processor = "Unknown"
END IF
PRINT *, processor, sinf.dwProcessorType
END
```

The CALL to the routine GetSystemInfo passes the address of the record.

**Passing Procedure Pointers to Win32 Routines**

The binary interface to the operating system functions on Intel based Windows systems does not conform to the procedure interface used by FORTRAN and C. The Absoft Fortran 77 and Absoft Fortran 90 compilers must be informed when these Windows routines are referenced so that it can set up the CALL/RETURN sequence properly for the operating system. This is accomplished with the STDCALL declaration statement. For example, for Fortran 77:

```fortran
INTEGER Ellipse
STDCALL EXTERNAL Ellipse
```

A similar example for Fortran 90 might look like the following:

```fortran
logical, external:: mrwe_AbortProc, mrwe_AbortDlgProc
stdcall mrwe_AbortProc, mrwe_AbortDlgProc
```

When you use the INCLUDE or MODULE files supplied with the compiler, these functions are all declared correctly for you. However, certain Win32 API routines require pointers to other routines as parameters. These may be routines that you write to customize a standard feature or they may be handlers for messages that you want to act on. When you write a routine that will be called by the operating system you must always use the STDCALL specification statement. For example, a dialog handler might start with:
stdcall function AboutDlgProc (hDlg, message, wParam, lParam)
use windows
implicit none
logical AboutDlgProc
integer hDlg;    value hDlg
integer message; value message
integer wParam;  value wParam
integer lParam;  value lParam

For this function, the parameters are passed by value and are declared accordingly.

**Note:** Although the **stdcall** mechanism is not required for PowerPC based systems, the compiler ignores it making it easier to move programs between the two systems.

**Handles**

The Windows memory manager can move certain blocks of memory at any time to a new location and pointers referencing data in those blocks will be left “dangling”, or pointing to an incorrect address. To avoid this, a data type called a handle is used. Handles use one level of indirection to access the data they reference. The indirection is via a **master pointer** which exists for each moveable block. When the block is moved, the memory manager properly updates its master pointer and all handles are still valid because the master pointer has not moved. In other words, handles are pointers to pointers.

A handle points to a pointer to a memory block
Handles are dealt with in much the same manner as pointers. You will rarely need to dereference handles to access the pointer to the data directly. If you do need to, you should use the Win32 API GlobalLock and GlobalUnlock functions to return a safe pointer. The following FORTRAN 90 example copies up to 1024 bytes of text from the Clipboard:

```fortran
INTEGER*1 text(1024)
INTEGER*1 memory(1)
POINTER (p_memory,memory)

INTEGER i, n, h_memory, result

IF (IsClipboardFormatAvailable(CF_TEXT) .eq. 0) RETURN
IF (OpenClipboard(0) .eq. 0) RETURN

h_memory = GetClipboardData(CF_TEXT)
IF (h_memory == 0) RETURN

p_memory = GlobalLock(h_memory)

DO i=1,1024
  IF (memory(i) == 0) EXIT
  text(i) = memory(i)
END DO

CALL GlobalUnlock(h_memory)

END
```

**INTERFACING WITH C**

Absoft FORTRAN 77 is designed to be fully compatible with the Microsoft C/C++ compilers. 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 and some changes to the linking procedure must be made. All of these rules are detailed below. Be sure to follow them closely, or the results will be both unpredictable and invalid.
FORTRAN Data Types in C

Declarations for FORTRAN data types and the equivalent declarations in C are as follows:

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGICAL*1 l</td>
<td>unsigned char l;</td>
</tr>
<tr>
<td>LOGICAL*2 m</td>
<td>unsigned short m;</td>
</tr>
<tr>
<td>LOGICAL*4 n</td>
<td>unsigned long n;</td>
</tr>
<tr>
<td>CHARACTER*n c</td>
<td>char c[n];</td>
</tr>
<tr>
<td>INTEGER*1 i</td>
<td>char i;</td>
</tr>
<tr>
<td>INTEGER*2 j</td>
<td>short j;</td>
</tr>
<tr>
<td>INTEGER*4 k</td>
<td>int k;</td>
</tr>
<tr>
<td>INTEGER*8 l</td>
<td>long long l;</td>
</tr>
<tr>
<td>REAL*4 a</td>
<td>float a;</td>
</tr>
<tr>
<td>REAL*8 d</td>
<td>double d;</td>
</tr>
<tr>
<td>COMPLEX*8 c</td>
<td>struct complx {</td>
</tr>
<tr>
<td></td>
<td>float x;</td>
</tr>
<tr>
<td></td>
<td>float y;</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td></td>
<td>struct complx c;</td>
</tr>
<tr>
<td>COMPLEX*16 d</td>
<td>struct dcomp {</td>
</tr>
<tr>
<td></td>
<td>double x;</td>
</tr>
<tr>
<td></td>
<td>double y;</td>
</tr>
<tr>
<td></td>
<td>};</td>
</tr>
<tr>
<td></td>
<td>struct dcomp d;</td>
</tr>
<tr>
<td>RECORD ...</td>
<td>struct ...</td>
</tr>
</tbody>
</table>

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 cautions when passing FORTRAN strings to C routines. See Passing Strings to C later in this chapter for more information.

Passing arguments Between C and FORTRAN

The Absoft FORTRAN 77 compiler uses the calling conventions of the C 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 in the previous section.
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:

```fortran
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:

```fortran
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:

```c
main()
{
    float a_actual;
    int i_actual;

    a_actual = 3.3;
    i_actual = 9;
    sub(&a_actual, &i_actual);
}
```

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:

```fortran
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 FORTRAN 77 provides the intrinsic function `VAL` for passing value parameters. Although there is generally no need to pass a value directly to a FORTRAN procedure, these functions may be used to pass a value to a C function:

```fortran
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 declared as `VALUE`.

```c
void c_fun()
{
    void fortran_sub();
    int i;

    fortran_sub(i);
}
```

```fortran
SUBROUTINE fortran_sub(i)
    VALUE i
    ...  
END
```

Note that C will pass all floating point data as double precision by default.

**Indirection (the LOC Function)**

The `[%]LOC` function is provided to give one level of indirection. The argument to `[%]LOC` must be a scalar name, an array name or the name of an external procedure. The function returns the address of its argument as a 32-bit integer.

This example illustrates the use of `LOC` to pass an array. Note that this is a one-dimensional array. Due to the different ordering used by C and FORTRAN for arrays, multi-dimensional arrays cannot be freely passed and indexed between the languages:

```c
void c_fun(i)
int *i[10];
{
    int j;

    for(j=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`, `DOUBLE PRECISION`, `RECORD`, or `POINTER`. All other data types must be returned in reference parameters. The following are examples of the passing of function results between FORTRAN and C.
A Call to C from FORTRAN

```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
```

```c
int Cmax (x,y)
int *x,*y;
{
    return( (*x >= *y) ? *x : *y );
}
```

A Call to FORTRAN from C

```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));
}
```

```fortran
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 or array of characters terminated by a null character. Therefore, when passing FORTRAN strings to C routines, eliminate the extra blanks and terminate them with a null character. The following FORTRAN expression will properly pass the FORTRAN string `anystring` to the C routine `CPrint`:

```fortran
PROGRAM cstringcall
character*255 string
string = 'Moscow on the Hudson'
CALL CPrint(TRIM(string)//CHAR(0))
END
```

```c
void CPrint (anystring)
char *anystring[];
{
    printf ("%s\n",anystring);
}
```
This example will neatly output “Moscow on the Hudson”. If the TRIM function was not used, the same string would be printed, but followed by 235 blanks. If the CHAR(0) was omitted, C would print characters until a null character was encountered, whenever that might be.

In Absoft FORTRAN, the -K option may be used to allow embedded escape sequences in strings. The sequence for a null “\0” may be used to pass string constants as argument:

```fortran
character*15 Fstring
CALL CPrint("string constant\0") ! null terminated string

Fstring = "string constant" ! blank padded string
CALL CPrint(TRIM(Fstring)//"\0") ! append a null
```

You can also take advantage of the method Absoft Fortran compilers employ to pass the lengths of CHARACTER arguments on the stack. After the end of the formal argument list, the lengths of any CHARACTER arguments in the list are passed by value as 32-bit integers. The lengths are passed in the order they appear in the argument list and only CHARACTER argument lengths are passed. For example:

```fortran
main()
{
  int i = 1;
  double d = 3.0;
  void fortran_sub();
      fortran_sub(&i, "two", &d, 3);
}
```

```fortran
SUBROUTINE fortran_sub(i, s, d)
  INTEGER i
  CHARACTER*(*) s
  DOUBLE PRECISION d
  PRINT *, i, s, d
END
```

### Calling FORTRAN Math Routines

All of the FORTRAN intrinsic math functions that return values recognized by the C language can be called directly from C as long as the FORTRAN runtime libraries, FIO.LIB and FMATH.LIB, is linked to the application. All arguments are passed by value. Taking the specific 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);
}

Calling FORTRAN I/O Routines

There are no restrictions on calling FORTRAN subroutines which perform I/O from C. The FORTRAN I/O library is completely re-entrant and requires no initialization. The only requirement is that all files opened in FORTRAN must be explicitly closed in FORTRAN before exiting to the system. FORTRAN unit numbers may not be used as C file descriptors and vice versa. Refer to the Input/Output and FORMAT Specification chapter in the FORTRAN 77 Language Reference Manual for a description of I/O in Absoft FORTRAN 77.

Naming Conventions

Global names in FORTRAN include procedure names and COMMON block names, both of which are significant to 31 characters. All global names are case sensitive, meaning the compiler recognizes the difference between upper and lower case characters. Use of the -f option will fold global names to lower case, while the -N109 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.

Procedure Names

Names of functions and subroutines in FORTRAN programs will appear in the assembly language source output or object file records as they were typed in the source code with a period prefix character attached. Symbolic names in the C language are case sensitive, distinguishing between upper and lower case characters. To make FORTRAN code compatible with C, avoid using the -f or -N109 options when compiling the FORTRAN source code.

Accessing COMMON Blocks from C

COMMON block names are formed in Absoft FORTRAN 77 by adding the characters "_C" to the beginning of the name of the COMMON block. COMMON block names are case sensitive unless either the -f or -N109 options are specified at compile-time. The elements of the COMMON block can be accessed from the C language by declaring an external structure using this name.
For example:

```fortran
COMMON /comm/ a, b, c
```

can be accessed with the global declaration:

```fortran
extern struct {
    float a;
    float b;
    float c;
} _Ccomm;
```

### Declaring C Structures In 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:

**C**  
```c
struct str {
    char c;
    long l;
    float f;
    double d;
};
struct str my_struct;
```

**FORTRAN**  
```fortran
STRUCTURE /str/
    CHARACTER c
    INTEGER*4 l
    REAL*4 f
    REAL*8 d
END STRUCTURE
RECORD /str/ my_struct
```

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.
Appendix A

Absoft Compiler Option Guide

This appendix summarizes the options for the Absoft Fortran 90/95, FORTRAN 77, and C/C++ compilers. Refer to the chapter, Using the Compilers for detailed descriptions of the options.

ABSOFT COMPILER OPTIONS

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>suppresses creation of an executable file — leaves compiled files in object code format</td>
</tr>
<tr>
<td>-g</td>
<td>generates symbol information for Fx™.</td>
</tr>
<tr>
<td>-l</td>
<td>used to supply a comma separated list of directory paths which are prepended to file names used with the INCLUDE statement</td>
</tr>
<tr>
<td>-O1</td>
<td>enables a group of basic optimizations which cause most code to run faster and enables optimizations that do not rearrange your program</td>
</tr>
<tr>
<td>-O2</td>
<td>enables a group of moderate optimizers that can rearrange the code generated for a program</td>
</tr>
<tr>
<td>-O3</td>
<td>enables a group of advanced optimizers including the IPA (Inter-Procedural Analyzer) linker that can substantially rearrange the code generated for a program</td>
</tr>
<tr>
<td>-O4</td>
<td>enables a group of advanced optimizers that can substantially rearrange the code generated for a program</td>
</tr>
<tr>
<td>-o name</td>
<td>directs the compiler to produce an executable file called name where name is a Windows file name</td>
</tr>
<tr>
<td>-cpu:type</td>
<td>specifies the target processor where type is one of 486, p5, p6, p7, athlon, or host</td>
</tr>
</tbody>
</table>

FLOATING POINT UNIT CONTROL OPTIONS

-OPT=roundoff changes the rounding mode of the FPU to roundoff.
-TENV=exception enables FPU exception trapping for exception.

FORTRAN 90/95 GENERAL OPTIONS

-en causes the compiler to issue a warning whenever the source code contains an extension to the Fortran 90/95 standard.
-w suppresses listing of all compile-time warning messages.
-Zn suppresses messages by message number.
-q suppresses any messages printed to standard output during the compilation process.
-v directs the compiler to print status information as the compilation process proceeds.

-zn suppresses messages by message level.

-dq continue compilation if more than 100 errors are encountered.

-ea causes the f95 compiler to abort the compilation process on the first error that it encounters.

-V causes the f95 compiler to display its version number.

-eR default recursion

-Tn changes the number of handles used internally by the compiler.

-tn this option increases the default temporary string size to $1024 \times 10^n$ bytes.

-YDEALLOC= Controls underlying cache management of ALLOCATE and DEALLOCATE statements.

-p path specify module search path

**FORTRAN 90/95 COMPATIBILITY OPTIONS**

-xdirective disable compiler directive in the source file.

-in changes the default storage length of INTEGER data types to $n$ bytes which can be either 2 or 8.

-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.

-N113 changes REAL(KIND=4) and COMPLEX(KIND=4) data types without explicit length declaration to REAL(KIND=8) and COMPLEX(KIND=8)

-s allocate local variables statically

-Rb enables array boundary checking

-Rc enables array conformance checking

-Rs enables substring checking

-Rp enables null pointer checking

-YCFRL= controls how CHARACTER arguments are passed to FUNCTION and SUBROUTINE programs

-YPEI= allows Cray-style pointers to be equivalent to integers

**FORTRAN 90/95 FORMAT OPTIONS**

-YEXT_NAMES controls the case of external procedure names

-YEXT_PFX= establishes the prefix of external procedure names

-YEXT_SFX= establishes the suffix of external procedure names

-fform sets the form of the source file to fixed, free, alt_fixed.

-Wn sets the line length of source statements accepted by the compiler in Fixed-Form source format

-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
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-YCSLASH=</td>
<td>directs the compiler to transform certain escape sequences marked with a</td>
</tr>
<tr>
<td></td>
<td>‘\’ embedded in character constants</td>
</tr>
<tr>
<td>-YNDFP=</td>
<td>disallow the use of a ‘.’ as a structure field separator. The default value is 0 or false</td>
</tr>
<tr>
<td>-TMS7D</td>
<td>recognize Microsoft style compiler directives beginning with a ‘$’ in</td>
</tr>
<tr>
<td></td>
<td>column 1</td>
</tr>
</tbody>
</table>

**FORTRAN 90/95 MISCELLANEOUS OPTIONS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-YCOM_NAMES</td>
<td>controls the case of COMMON block names</td>
</tr>
<tr>
<td>-COM_PFX=</td>
<td>establishes the prefix of COMMON names</td>
</tr>
<tr>
<td>-YCM_SFX=</td>
<td>establishes the suffix of COMMON names</td>
</tr>
<tr>
<td>-h, -H, -U</td>
<td>loop unrolling control options</td>
</tr>
<tr>
<td>-safefp</td>
<td>disable floating point optimizations in numerically sensitive codes</td>
</tr>
</tbody>
</table>

**FORTRAN 90/95 OTHER OPTIONS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-YIL=</td>
<td>Controls DLL calling mechanism and name mangling convention</td>
</tr>
</tbody>
</table>

**FORTRAN 77 GENERAL OPTIONS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w</td>
<td>suppresses listing of all compile-time warning messages</td>
</tr>
<tr>
<td>-N32</td>
<td>directs the compiler to issue a warning whenever the source code contains an extension to the ANSI FORTRAN 77 standard</td>
</tr>
<tr>
<td>-q</td>
<td>suppresses any messages printed to standard output during the compilation process</td>
</tr>
<tr>
<td>-v</td>
<td>directs the compiler to print status information as the compilation process proceeds</td>
</tr>
<tr>
<td>-C</td>
<td>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</td>
</tr>
<tr>
<td>-x</td>
<td>replaces any occurrence of X or D in column one with a blank character: allows a restricted form of conditional compilation</td>
</tr>
<tr>
<td>-T</td>
<td>used to change the number of handles used internally by the compiler.</td>
</tr>
<tr>
<td>-t</td>
<td>modifies the default temporary string size to nn bytes from the default of 1024 bytes</td>
</tr>
</tbody>
</table>

**FORTRAN 77 CONTROL OPTIONS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-D</td>
<td>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</td>
</tr>
</tbody>
</table>

**FORTRAN 77 COMPATIBILITY OPTIONS**
-f \hspace{1em} folds all symbolic names to lower case
-s \hspace{1em} forces all program storage to be treated as static
-N109 \hspace{1em} folds all symbolic names to UPPER CASE
-ln \hspace{1em} changes the default storage length of INTEGER data types to $n$ bytes which can be either 2 or 8
-N15 \hspace{1em} causes the compiler to define SUBROUTINE and FUNCTION names with a trailing underscore
-N113 \hspace{1em} changes REAL and COMPLEX data types without explicit length declaration to DOUBLE PRECISION and DOUBLE COMPLEX

**FORTRAN 77 FORMAT OPTIONS**

-8 \hspace{1em} directs the compiler to accept source code written in Fortran 90 Free Source Form
-V \hspace{1em} directs the compiler to accept VAX Tab-Format source code
-W \hspace{1em} directs the compiler to accept statements which extend beyond column 72 up to column 132
-N26 \hspace{1em} force the compiler to consider the byte ordering of all unformatted files to be big-endian by default
-N27 \hspace{1em} force the compiler to consider the byte ordering of all unformatted files to be little-endian by default
-K \hspace{1em} directs the compiler to transform certain escape sequences marked with a "\" embedded in character constants

**FORTRAN 77 COMMON OPTIONS**

-N22 \hspace{1em} append trailing underscore to COMMON block names
-N25 \hspace{1em} export COMMON block names in DLLs
-N110 \hspace{1em} do not mangle COMMON block names with leading "_c"

**FORTRAN 77 OTHER OPTIONS**

-h, -H, -U \hspace{1em} loop unrolling control options
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:

- **IEEE_DATATYPE**: IEEE data types are available
- **IEEE_DENORMAL**: IEEE denormalized values are supported
- **IEEE_DIVIDE**: IEEE division to the required precision is supported
- **IEEE_HALTING**: control of halting is supported
- **IEEE_INEXACT_FLAG**: inexact exceptions are supported.
- **IEEE_INF**: IEEE infinities (positive and negative) are supported
- **IEEE_INVALID_FLAG**: invalid exceptions are supported
- **IEEE_NAN**: IEEE NaN (Not a Number) values are supported
- **IEEE_ROUNDING**: all IEEE rounding modes are supported
- **IEEE_SQRT**: SQRT is supported to the IEEE standard
- **IEEE_UNDERFLOW_FLAG**: 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:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE_SIGNALING_NAN</td>
<td>NaN (Not a Number)</td>
</tr>
<tr>
<td>IEEE QUIET_NAN</td>
<td>NaN (Not a Number)</td>
</tr>
<tr>
<td>IEEE NEGATIVE_INF</td>
<td>negative infinity</td>
</tr>
<tr>
<td>IEEE NEGATIVE_NORMAL</td>
<td>negative number</td>
</tr>
<tr>
<td>IEEE NEGATIVE_DENORMAL</td>
<td>negative number smaller than the normal representation</td>
</tr>
<tr>
<td>IEEE NEGATIVE_ZERO</td>
<td>negative zero</td>
</tr>
<tr>
<td>IEEE_POSITIVE_ZERO</td>
<td>zero</td>
</tr>
<tr>
<td>IEEE_POSITIVE_DENORMAL</td>
<td>positive number smaller than the normal representation</td>
</tr>
<tr>
<td>IEEE_POSITIVE_NORMAL</td>
<td>positive number</td>
</tr>
<tr>
<td>IEEE_POSITIVE_INF</td>
<td>positive infinity</td>
</tr>
</tbody>
</table>
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:

- \( \text{IEEE\_NEGATIVE\_NORMAL} \)
- \( \text{IEEE\_NEGATIVE\_DENORMAL} \)
- \( \text{IEEE\_NEGATIVE\_ZERO} \)
- \( \text{IEEE\_NEGATIVE\_INFINITY} \)

elemental real function \( \text{IEEE\_LOGB}(x) \)
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 \( \text{IEEE\_DIVIDE\_BY\_ZERO} \) is signaled.

elemental real function \( \text{IEEE\_NEXT\_AFTER}(x, y) \)
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 \( \text{IEEE\_REM}(x, y) \)
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 \( |n-x/y| = \frac{1}{2} \), \( n \) is even. The kind of the result is the same as \( x \).

elemental real function \( \text{IEEE\_RINT}(x) \)
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 \( \text{IEEE\_SCALB}(x, i) \)
real(kind=any), intent(in) :: x
integer (kind=4), intent(in) :: i

returns the floating-point value of \( x*2^i \). The kind of the result is the same as \( x \).

elemental logical function \( \text{IEEE\_UNORDERED}(x, y) \)
real(kind=any), intent(in) :: x, y

returns if either \( x \) or \( y \) is a NaN.

elemental real function \( \text{IEEE\_VALUE}(x, \text{class}) \)
real(kind=any), intent(in) :: x
type(IEEE\_CLASS\_TYPE), intent(in) :: class

returns a value of the specified IEEE\_CLASS\_TYPE.

The \( \text{IEEE\_ARITHMETIC} \) 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 of 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:

```fortran
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..
```

```fortran
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:

```fortran
subroutine IEEE_GET_STATUS(status_value)
  type(IEEE_STATUS_TYPE) :: status_value

  retrieves the state of the floating-point environment.
```

```fortran
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.
```

```fortran
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.
```

```fortran
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.

```fortran
subroutine safe_divide(a, b, c, fail)
use IEEE_EXCEPTIONS
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
call IEEE_GET_FLAG(IEEE_DIVIDE_BY_ZERO, fail)
call IEEE_SET_STATUS(value)

end safe_divide
```
## ASCII Table

ASCII codes 0 through 31 are control codes that may or may not have meaning on Windows. They are listed for 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. Most of these characters may be typed with keystrokes; use the Key Caps desk accessory to determine which keystrokes to use. The Dec, Oct, and Hex columns refer to the decimal, octal, and hexadecimal numerical representations.

<table>
<thead>
<tr>
<th>Character</th>
<th>Dec</th>
<th>Oct</th>
<th>Hex</th>
<th>Description</th>
<th>Character</th>
<th>Dec</th>
<th>Oct</th>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>0</td>
<td>000</td>
<td>00</td>
<td>null</td>
<td>!</td>
<td>32</td>
<td>040</td>
<td>20</td>
<td>space</td>
</tr>
<tr>
<td>SOH</td>
<td>1</td>
<td>001</td>
<td>01</td>
<td>start of heading</td>
<td>&quot;</td>
<td>34</td>
<td>042</td>
<td>22</td>
<td>exclamation</td>
</tr>
<tr>
<td>STX</td>
<td>2</td>
<td>002</td>
<td>02</td>
<td>start of text</td>
<td>&quot;</td>
<td>34</td>
<td>042</td>
<td>22</td>
<td>quotation mark</td>
</tr>
<tr>
<td>ETX</td>
<td>3</td>
<td>003</td>
<td>03</td>
<td>end of text</td>
<td>#</td>
<td>35</td>
<td>043</td>
<td>23</td>
<td>number sign</td>
</tr>
<tr>
<td>ECT</td>
<td>4</td>
<td>004</td>
<td>04</td>
<td>end of trans</td>
<td>$</td>
<td>36</td>
<td>044</td>
<td>24</td>
<td>dollar sign</td>
</tr>
<tr>
<td>ENQ</td>
<td>5</td>
<td>005</td>
<td>05</td>
<td>enquiry</td>
<td>%</td>
<td>37</td>
<td>045</td>
<td>25</td>
<td>percent sign</td>
</tr>
<tr>
<td>ACK</td>
<td>6</td>
<td>006</td>
<td>06</td>
<td>acknowledge</td>
<td>&amp;</td>
<td>38</td>
<td>046</td>
<td>26</td>
<td>ampersand</td>
</tr>
<tr>
<td>BEL</td>
<td>7</td>
<td>007</td>
<td>07</td>
<td>bell code</td>
<td>'</td>
<td>39</td>
<td>047</td>
<td>27</td>
<td>apostrophe</td>
</tr>
<tr>
<td>BS</td>
<td>8</td>
<td>010</td>
<td>08</td>
<td>back space</td>
<td>(</td>
<td>40</td>
<td>050</td>
<td>28</td>
<td>opening paren</td>
</tr>
<tr>
<td>HT</td>
<td>9</td>
<td>011</td>
<td>09</td>
<td>horizontal tab</td>
<td>)</td>
<td>41</td>
<td>051</td>
<td>29</td>
<td>closing paren</td>
</tr>
<tr>
<td>LF</td>
<td>10</td>
<td>012</td>
<td>0A</td>
<td>line feed</td>
<td>*</td>
<td>42</td>
<td>052</td>
<td>2A</td>
<td>asterisk</td>
</tr>
<tr>
<td>VT</td>
<td>11</td>
<td>013</td>
<td>0B</td>
<td>vertical tab</td>
<td>+</td>
<td>43</td>
<td>053</td>
<td>2B</td>
<td>plus</td>
</tr>
<tr>
<td>FF</td>
<td>12</td>
<td>014</td>
<td>0C</td>
<td>form feed</td>
<td>,</td>
<td>44</td>
<td>054</td>
<td>2C</td>
<td>comma</td>
</tr>
<tr>
<td>CR</td>
<td>13</td>
<td>015</td>
<td>0D</td>
<td>carriage return</td>
<td>-</td>
<td>45</td>
<td>055</td>
<td>2D</td>
<td>minus</td>
</tr>
<tr>
<td>SO</td>
<td>14</td>
<td>016</td>
<td>0E</td>
<td>shift out</td>
<td>.</td>
<td>46</td>
<td>056</td>
<td>2E</td>
<td>period</td>
</tr>
<tr>
<td>SI</td>
<td>15</td>
<td>017</td>
<td>0F</td>
<td>shift in</td>
<td>/</td>
<td>47</td>
<td>057</td>
<td>2F</td>
<td>slash</td>
</tr>
<tr>
<td>DLE</td>
<td>16</td>
<td>020</td>
<td>10</td>
<td>data link escape</td>
<td>0</td>
<td>48</td>
<td>060</td>
<td>30</td>
<td>zero</td>
</tr>
<tr>
<td>DC1</td>
<td>17</td>
<td>021</td>
<td>11</td>
<td>device control 1</td>
<td>1</td>
<td>49</td>
<td>061</td>
<td>31</td>
<td>one</td>
</tr>
<tr>
<td>DC2</td>
<td>18</td>
<td>022</td>
<td>12</td>
<td>device control 2</td>
<td>2</td>
<td>50</td>
<td>062</td>
<td>32</td>
<td>two</td>
</tr>
<tr>
<td>DC3</td>
<td>19</td>
<td>023</td>
<td>13</td>
<td>device control 3</td>
<td>3</td>
<td>51</td>
<td>063</td>
<td>33</td>
<td>three</td>
</tr>
<tr>
<td>DC4</td>
<td>20</td>
<td>024</td>
<td>14</td>
<td>device control 4</td>
<td>4</td>
<td>52</td>
<td>064</td>
<td>34</td>
<td>four</td>
</tr>
<tr>
<td>NAK</td>
<td>21</td>
<td>025</td>
<td>15</td>
<td>negative ack</td>
<td>5</td>
<td>53</td>
<td>065</td>
<td>35</td>
<td>five</td>
</tr>
<tr>
<td>SYN</td>
<td>22</td>
<td>026</td>
<td>16</td>
<td>synch idle</td>
<td>6</td>
<td>54</td>
<td>066</td>
<td>36</td>
<td>six</td>
</tr>
<tr>
<td>ETB</td>
<td>23</td>
<td>027</td>
<td>17</td>
<td>end of trans blk</td>
<td>7</td>
<td>55</td>
<td>067</td>
<td>37</td>
<td>seven</td>
</tr>
<tr>
<td>CAN</td>
<td>24</td>
<td>030</td>
<td>18</td>
<td>cancel</td>
<td>8</td>
<td>56</td>
<td>070</td>
<td>38</td>
<td>eight</td>
</tr>
<tr>
<td>EM</td>
<td>25</td>
<td>031</td>
<td>19</td>
<td>end of medium</td>
<td>9</td>
<td>57</td>
<td>071</td>
<td>39</td>
<td>nine</td>
</tr>
<tr>
<td>SS</td>
<td>26</td>
<td>032</td>
<td>1A</td>
<td>special sequence</td>
<td>:</td>
<td>58</td>
<td>072</td>
<td>3A</td>
<td>colon</td>
</tr>
<tr>
<td>ESC</td>
<td>27</td>
<td>033</td>
<td>1B</td>
<td>escape</td>
<td>;</td>
<td>59</td>
<td>073</td>
<td>3B</td>
<td>semicolon</td>
</tr>
<tr>
<td>FS</td>
<td>28</td>
<td>034</td>
<td>1C</td>
<td>file separator</td>
<td>&lt;</td>
<td>60</td>
<td>074</td>
<td>3C</td>
<td>less than</td>
</tr>
<tr>
<td>GS</td>
<td>29</td>
<td>035</td>
<td>1D</td>
<td>group separator</td>
<td>=</td>
<td>61</td>
<td>075</td>
<td>3D</td>
<td>equal</td>
</tr>
<tr>
<td>RS</td>
<td>30</td>
<td>036</td>
<td>1E</td>
<td>record separator</td>
<td>&gt;</td>
<td>62</td>
<td>076</td>
<td>3E</td>
<td>greater than</td>
</tr>
<tr>
<td>US</td>
<td>31</td>
<td>037</td>
<td>1F</td>
<td>unit separator</td>
<td>?</td>
<td>63</td>
<td>077</td>
<td>3F</td>
<td>question mark</td>
</tr>
</tbody>
</table>

Fortran User Guide
<table>
<thead>
<tr>
<th>Character</th>
<th>Dec</th>
<th>Oct</th>
<th>Hex</th>
<th>Description</th>
<th>Character</th>
<th>Dec</th>
<th>Oct</th>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>64</td>
<td>100</td>
<td>40</td>
<td>commercial at</td>
<td>{</td>
<td>123</td>
<td>173</td>
<td>7B</td>
<td>opening brace</td>
</tr>
<tr>
<td>A</td>
<td>65</td>
<td>101</td>
<td>41</td>
<td>upper case letter</td>
<td></td>
<td>124</td>
<td>174</td>
<td>7C</td>
<td>vertical bar</td>
</tr>
<tr>
<td>B</td>
<td>66</td>
<td>102</td>
<td>42</td>
<td>upper case letter</td>
<td>}</td>
<td>125</td>
<td>175</td>
<td>7D</td>
<td>closing brace</td>
</tr>
<tr>
<td>C</td>
<td>67</td>
<td>103</td>
<td>43</td>
<td>upper case letter</td>
<td>~</td>
<td>126</td>
<td>176</td>
<td>7E</td>
<td>tilda</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
<td>104</td>
<td>44</td>
<td>upper case letter</td>
<td></td>
<td>127</td>
<td>177</td>
<td>7F</td>
<td>delete</td>
</tr>
<tr>
<td>E</td>
<td>69</td>
<td>105</td>
<td>45</td>
<td>upper case letter</td>
<td></td>
<td>128</td>
<td>200</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>70</td>
<td>106</td>
<td>46</td>
<td>upper case letter</td>
<td></td>
<td>129</td>
<td>201</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>71</td>
<td>107</td>
<td>47</td>
<td>upper case letter</td>
<td>,</td>
<td>130</td>
<td>202</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>72</td>
<td>110</td>
<td>48</td>
<td>upper case letter</td>
<td>}</td>
<td>131</td>
<td>203</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>73</td>
<td>111</td>
<td>49</td>
<td>upper case letter</td>
<td>&quot;</td>
<td>132</td>
<td>204</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>74</td>
<td>112</td>
<td>4A</td>
<td>upper case letter</td>
<td>...</td>
<td>133</td>
<td>205</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>75</td>
<td>113</td>
<td>4B</td>
<td>upper case letter</td>
<td>†</td>
<td>134</td>
<td>206</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>76</td>
<td>114</td>
<td>4C</td>
<td>upper case letter</td>
<td>‡</td>
<td>135</td>
<td>207</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>77</td>
<td>115</td>
<td>4D</td>
<td>upper case letter</td>
<td>&quot;</td>
<td>136</td>
<td>210</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>78</td>
<td>116</td>
<td>4E</td>
<td>upper case letter</td>
<td>%</td>
<td>137</td>
<td>211</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>79</td>
<td>117</td>
<td>4F</td>
<td>upper case letter</td>
<td>§</td>
<td>138</td>
<td>212</td>
<td>8A</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>80</td>
<td>120</td>
<td>50</td>
<td>upper case letter</td>
<td>†</td>
<td>139</td>
<td>213</td>
<td>8B</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>81</td>
<td>121</td>
<td>51</td>
<td>upper case letter</td>
<td>Ø</td>
<td>140</td>
<td>214</td>
<td>8C</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>82</td>
<td>122</td>
<td>52</td>
<td>upper case letter</td>
<td>µ</td>
<td>141</td>
<td>215</td>
<td>8D</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>83</td>
<td>123</td>
<td>53</td>
<td>upper case letter</td>
<td></td>
<td>142</td>
<td>216</td>
<td>8E</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>84</td>
<td>124</td>
<td>54</td>
<td>upper case letter</td>
<td></td>
<td>143</td>
<td>217</td>
<td>8F</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>85</td>
<td>125</td>
<td>55</td>
<td>upper case letter</td>
<td></td>
<td>144</td>
<td>220</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>86</td>
<td>126</td>
<td>56</td>
<td>upper case letter</td>
<td>.</td>
<td>145</td>
<td>221</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>87</td>
<td>127</td>
<td>57</td>
<td>upper case letter</td>
<td>,</td>
<td>146</td>
<td>222</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>88</td>
<td>130</td>
<td>58</td>
<td>upper case letter</td>
<td>&quot;</td>
<td>147</td>
<td>223</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>131</td>
<td>59</td>
<td>upper case letter</td>
<td>&quot;</td>
<td>148</td>
<td>224</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
<td>132</td>
<td>5A</td>
<td>upper case letter</td>
<td>•</td>
<td>149</td>
<td>225</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>[</td>
<td>91</td>
<td>133</td>
<td>5B</td>
<td>opening bracket</td>
<td></td>
<td>150</td>
<td>226</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>92</td>
<td>134</td>
<td>5C</td>
<td>back slash</td>
<td></td>
<td>151</td>
<td>227</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>]</td>
<td>93</td>
<td>135</td>
<td>5D</td>
<td>closing bracket</td>
<td></td>
<td>152</td>
<td>230</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>94</td>
<td>136</td>
<td>5E</td>
<td>circumflex</td>
<td></td>
<td>153</td>
<td>231</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>_</td>
<td>95</td>
<td>137</td>
<td>5F</td>
<td>underscore</td>
<td></td>
<td>154</td>
<td>232</td>
<td>9A</td>
<td></td>
</tr>
<tr>
<td>`</td>
<td>96</td>
<td>140</td>
<td>60</td>
<td>grave accent</td>
<td></td>
<td>155</td>
<td>233</td>
<td>9B</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>141</td>
<td>61</td>
<td>lower case letter</td>
<td>œ</td>
<td>156</td>
<td>234</td>
<td>9C</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>98</td>
<td>142</td>
<td>62</td>
<td>lower case letter</td>
<td></td>
<td>157</td>
<td>235</td>
<td>9D</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>99</td>
<td>143</td>
<td>63</td>
<td>lower case letter</td>
<td></td>
<td>158</td>
<td>236</td>
<td>9E</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>100</td>
<td>144</td>
<td>64</td>
<td>lower case letter</td>
<td>Y</td>
<td>159</td>
<td>237</td>
<td>9F</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>145</td>
<td>65</td>
<td>lower case letter</td>
<td></td>
<td>160</td>
<td>240</td>
<td>A0</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>102</td>
<td>146</td>
<td>66</td>
<td>lower case letter</td>
<td>I</td>
<td>161</td>
<td>241</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>103</td>
<td>147</td>
<td>67</td>
<td>lower case letter</td>
<td></td>
<td>162</td>
<td>242</td>
<td>A2</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>140</td>
<td>68</td>
<td>lower case letter</td>
<td>L</td>
<td>163</td>
<td>243</td>
<td>A3</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>105</td>
<td>151</td>
<td>69</td>
<td>lower case letter</td>
<td></td>
<td>164</td>
<td>244</td>
<td>A4</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>106</td>
<td>152</td>
<td>6A</td>
<td>lower case letter</td>
<td></td>
<td>165</td>
<td>245</td>
<td>A5</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>153</td>
<td>6B</td>
<td>lower case letter</td>
<td></td>
<td>166</td>
<td>246</td>
<td>A6</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>108</td>
<td>154</td>
<td>6C</td>
<td>lower case letter</td>
<td></td>
<td>167</td>
<td>247</td>
<td>A7</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>109</td>
<td>155</td>
<td>6D</td>
<td>lower case letter</td>
<td></td>
<td>168</td>
<td>250</td>
<td>A8</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>110</td>
<td>156</td>
<td>6E</td>
<td>lower case letter</td>
<td>©</td>
<td>169</td>
<td>251</td>
<td>A9</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>157</td>
<td>6F</td>
<td>lower case letter</td>
<td></td>
<td>170</td>
<td>252</td>
<td>AA</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>160</td>
<td>70</td>
<td>lower case letter</td>
<td></td>
<td>171</td>
<td>253</td>
<td>AB</td>
<td></td>
</tr>
<tr>
<td>q</td>
<td>113</td>
<td>161</td>
<td>71</td>
<td>lower case letter</td>
<td></td>
<td>172</td>
<td>254</td>
<td>AC</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>162</td>
<td>72</td>
<td>lower case letter</td>
<td></td>
<td>173</td>
<td>255</td>
<td>AD</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>163</td>
<td>73</td>
<td>lower case letter</td>
<td></td>
<td>174</td>
<td>256</td>
<td>AE</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>116</td>
<td>164</td>
<td>74</td>
<td>lower case letter</td>
<td></td>
<td>175</td>
<td>257</td>
<td>AF</td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>117</td>
<td>165</td>
<td>75</td>
<td>lower case letter</td>
<td></td>
<td>176</td>
<td>260</td>
<td>B0</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>118</td>
<td>166</td>
<td>76</td>
<td>lower case letter</td>
<td></td>
<td>177</td>
<td>261</td>
<td>B1</td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>119</td>
<td>167</td>
<td>77</td>
<td>lower case letter</td>
<td></td>
<td>178</td>
<td>262</td>
<td>B2</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>120</td>
<td>170</td>
<td>78</td>
<td>lower case letter</td>
<td></td>
<td>179</td>
<td>263</td>
<td>B3</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>121</td>
<td>171</td>
<td>79</td>
<td>lower case letter</td>
<td></td>
<td>180</td>
<td>264</td>
<td>B4</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>122</td>
<td>172</td>
<td>7A</td>
<td>lower case letter</td>
<td></td>
<td>181</td>
<td>265</td>
<td>B5</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Dec</td>
<td>Oct</td>
<td>Hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¶</td>
<td>182</td>
<td>266</td>
<td>B6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>·</td>
<td>183</td>
<td>267</td>
<td>B7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>184</td>
<td>270</td>
<td>B8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'</td>
<td>185</td>
<td>271</td>
<td>B9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>º</td>
<td>186</td>
<td>272</td>
<td>BA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>»</td>
<td>187</td>
<td>273</td>
<td>BB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¥</td>
<td>188</td>
<td>274</td>
<td>BC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¥</td>
<td>189</td>
<td>275</td>
<td>BD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¥</td>
<td>190</td>
<td>276</td>
<td>BE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¤</td>
<td>191</td>
<td>277</td>
<td>BF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>À</td>
<td>192</td>
<td>300</td>
<td>C0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Á</td>
<td>193</td>
<td>301</td>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Å</td>
<td>194</td>
<td>302</td>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Å</td>
<td>195</td>
<td>303</td>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Å</td>
<td>196</td>
<td>304</td>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Å</td>
<td>197</td>
<td>305</td>
<td>C5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Æ</td>
<td>198</td>
<td>306</td>
<td>C6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ç</td>
<td>199</td>
<td>307</td>
<td>C7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>É</td>
<td>200</td>
<td>310</td>
<td>C8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>É</td>
<td>201</td>
<td>311</td>
<td>C9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>È</td>
<td>202</td>
<td>312</td>
<td>CA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ê</td>
<td>203</td>
<td>313</td>
<td>CB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ë</td>
<td>204</td>
<td>314</td>
<td>CC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ë</td>
<td>205</td>
<td>315</td>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ë</td>
<td>206</td>
<td>316</td>
<td>CE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ë</td>
<td>207</td>
<td>317</td>
<td>CF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ë</td>
<td>208</td>
<td>320</td>
<td>D0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ë</td>
<td>209</td>
<td>321</td>
<td>D1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ô</td>
<td>210</td>
<td>322</td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ô</td>
<td>211</td>
<td>323</td>
<td>D3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ô</td>
<td>212</td>
<td>324</td>
<td>D4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ô</td>
<td>213</td>
<td>325</td>
<td>D5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ô</td>
<td>214</td>
<td>326</td>
<td>D6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td>215</td>
<td>327</td>
<td>D7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ø</td>
<td>216</td>
<td>330</td>
<td>D8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ú</td>
<td>217</td>
<td>331</td>
<td>D9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ú</td>
<td>218</td>
<td>332</td>
<td>DA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ú</td>
<td>219</td>
<td>333</td>
<td>DB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ú</td>
<td>220</td>
<td>334</td>
<td>DC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ý</td>
<td>221</td>
<td>335</td>
<td>DD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ß</td>
<td>222</td>
<td>336</td>
<td>DE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ß</td>
<td>223</td>
<td>337</td>
<td>DF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Dec</td>
<td>Oct</td>
<td>Hex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>â</td>
<td>224</td>
<td>340</td>
<td>E0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>á</td>
<td>225</td>
<td>341</td>
<td>E1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>â</td>
<td>226</td>
<td>342</td>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>à</td>
<td>227</td>
<td>343</td>
<td>E3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ä</td>
<td>228</td>
<td>344</td>
<td>E4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>â</td>
<td>229</td>
<td>345</td>
<td>E5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>æ</td>
<td>230</td>
<td>346</td>
<td>E6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ç</td>
<td>231</td>
<td>347</td>
<td>E7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>é</td>
<td>232</td>
<td>350</td>
<td>E8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>é</td>
<td>233</td>
<td>351</td>
<td>E9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>è</td>
<td>234</td>
<td>352</td>
<td>EA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ë</td>
<td>235</td>
<td>353</td>
<td>EB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>236</td>
<td>354</td>
<td>EC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>í</td>
<td>237</td>
<td>355</td>
<td>ED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ì</td>
<td>238</td>
<td>356</td>
<td>EE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ï</td>
<td>239</td>
<td>357</td>
<td>EF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ð</td>
<td>240</td>
<td>360</td>
<td>F0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ñ</td>
<td>241</td>
<td>361</td>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ò</td>
<td>242</td>
<td>362</td>
<td>F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ó</td>
<td>243</td>
<td>363</td>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ò</td>
<td>244</td>
<td>364</td>
<td>F4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ô</td>
<td>245</td>
<td>365</td>
<td>F5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>õ</td>
<td>246</td>
<td>366</td>
<td>F6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>÷</td>
<td>247</td>
<td>367</td>
<td>F7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ø</td>
<td>248</td>
<td>370</td>
<td>F8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ù</td>
<td>249</td>
<td>371</td>
<td>F9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ú</td>
<td>250</td>
<td>372</td>
<td>FA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ū</td>
<td>251</td>
<td>373</td>
<td>FB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>û</td>
<td>252</td>
<td>374</td>
<td>FC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ý</td>
<td>253</td>
<td>375</td>
<td>FD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>þ</td>
<td>254</td>
<td>376</td>
<td>FE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ý</td>
<td>255</td>
<td>377</td>
<td>FF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Bibliography

REFERENCES ON THE FORTRAN 90/95 LANGUAGE

Michael Metcalf and John Reid, *FORTRAN 90/95 explained*, Oxford University Press (1996)


REFERENCES ON THE FORTRAN 77 LANGUAGE

These books and manuals are useful references for the FORTRAN language and the floating point math format used by Absoft Fortran 77 on Windows.


Harry Katzan, Jr., *FORTRAN 77*, Van Nostrand Reinhold Company (1978)


Fortran Forum, Association for Computing Machinery. Phone: 1-212-869-7440.


**REFERENCES ON THE C/C++ PROGRAMMING LANGUAGES**


**REFERENCES ON WINDOWS PROGRAMMING**

These books are suggested reading for learning how to program in the Win32 API for Windows. Most of these books are available in book stores.

Charles Petzold, Programming Windows 3.1, Microsoft Press (1992)

Appendix E

Technical Support

The Absoft Technical Support Group will provide technical assistance to all registered users. They will not answer general questions about operating systems, operating system interfaces, graphical user interfaces, or teach the FORTRAN language. 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 Fortran User Guide to make sure your problem is not covered here. Specifically, look at the chapter Using The Compilers in the Fortran User Guide and the Error Messages appendices of both manuals. 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) reproducible program or step-by-step example if possible.

Contacting Technical Support:

Address: Absoft Corporation
Attn: Technical Support
2781 Bond Street
Rochester Hills, MI 48309
<table>
<thead>
<tr>
<th>Service</th>
<th>Contact Information</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Support</td>
<td>(248) 853-0095</td>
<td>9am - 3pm EST</td>
</tr>
<tr>
<td>FAX</td>
<td>(248) 853-0108</td>
<td>24 Hours</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:support@absoft.com">support@absoft.com</a></td>
<td>24 Hours</td>
</tr>
<tr>
<td>World Wide Web</td>
<td><a href="http://www.absoft.com">http://www.absoft.com</a></td>
<td></td>
</tr>
</tbody>
</table>
This appendix describes how to use the Absoft Fortran compiler and linker to create DLLs that are callable from Microsoft Visual Basic™. A DLL, Dynamic Link Library, is a library of routines that are callable at runtime from any application that conforms to the Windows API.

You must be using a 32-bit version of Visual Basic, such as Visual Basic V6.0, to call Absoft Fortran DLLs.

CREATING THE FORTRAN DLL

Write your FORTRAN source code in the usual manner, declaring the program unit as either a subroutine or a function. Insert the additional keyword _stdcall_ before each _SUBROUTINE_ or _FUNCTION_ keyword. Adhere to the normal FORTRAN _CALL/RETURN_ sequences and argument passing rules. For example, consider the following subroutine

```fortran
_stDCALL subroutine DegreeSin(input, output)
implicit none

double precision input
double precision output

output = dsind(input)
return
end
```

This subroutine simply computes the double precision sin in degrees of the input argument and returns the result in the output argument. Compile the file using the _-c_ option, which instructs the compiler to produce an object file only. If you are using f90, use the option "-_YEXT_NAMES=ASIS_" so that the case of the routine name is preserved. The f77 compiler preserves case by default.

Next, create a new linker control file containing the names of the program units that will be placed in the DLL. This file should contain only those names in the library that will be externally visible. If the DLL contains any programs units that are used only by the routines within the DLL, they should not be placed in this file. In the case of the above example, we would create the file, "DegreeSin.xps", containing the single line:

```plaintext
DegreeSin
```
You must also create a second new linker control file containing the aliases for the internal names that the compiler generated for your subprogram names. The compiler creates internal names to avoid conflicts with other global symbolic names. It does this by prepending an underscore to the name and, in the case of _STDCALL_ subprograms, appending the size of the call stack to the name. The size of the call stack will be equal to 4 times the argument list count. In the case of the _DegreeSin_ subroutine, this value will be 8. This value is separated from the subprogram name with a commercial at sign (@) since the subprogram name may legally end with a number. In the case of the above example, we would create the file, “DegreeSin.als”, containing the single line:

```
_DegreeSin@8 DegreeSin
```

which says that the internal name, _DegreeSin@8_, will be known externally as _DegreeSin_.

To create the DLL, we would enter the following line on the MS-DOS Prompt command line:

```
lnk -dll DegreeSin.obj -exports:DegreeSin.xps absRT0.lib fio.lib fmath.lib libac.lib kernel32.lib -aliases:DegreeSin.als
```

The four .lib file references allow the linker to resolve any runtime library routines that may needed. That is, your DLL may itself reference other DLLs.

**CREATING THE VISUAL BASIC CODE**

**Note:** The default argument passing method for Visual Basic .NET is different from what was in previous versions. Arguments are now passed _ByVal_ by default. Note below that _ByRef_ must be specified.

Complete documentation about calling Fortran DLLs from Microsoft Visual Basic can be found in the _Microsoft Visual Basic Programmer’s Guide_. This section will describe the basics of referencing a Fortran Subroutine or Function.

The first step is to declare the Fortran subprogram in your Basic program. The declaration for the subroutine discussed in the previous section would be:

```vbnet
Imports System.Runtime.InteropServices
Imports System.Text

Public Class Form1

...

<DllImport("c:\...\DegreeSin.dll")> _
Public Shared Sub DegreeSin( _
    ByVal inval As System.Double, ByVal outval As System.Double)
End Sub
```

**Fortran User Guide**
The string following the Lib keyword should be the path to the DLL.

The actual reference to the Fortran subroutine is:

```
Dim inval As Double
Dim outval As Double
.
.
Call DegreeSin(inval, outval)
```

**PASSING VISUAL BASIC ARRAYS TO A DLL**

The Microsoft Visual Basic Programmer's Guide describes how to pass arrays to DLLs. Visual Basic passes entire arrays using OLE Automation argument protocols. Absoft F77/F95/C/C++ expect CDECL arguments. The Visual Basic manual section explains how to pass the address of the first argument of the array. Basically:

```
Declare ... lParam as Any

Dim array(100)

Call DLL(array(0))
```

**PASSING VISUAL BASIC STRINGS TO A DLL**

Visual Basic strings are maintained in a data structured referred to as a BSTR which is not compatible with other languages in a DLL. However, you can pass a null terminated, C programming language string in Visual Basic. The declaration and usage would take the following form:

```
Imports System.Runtime.InteropServices
Imports System.Text

Public Class Form1
...

<DllImport("c:\test\vbstring\test.dll")>
    Public Shared Sub getString(ByVal text As System.String)
End Sub

Private Sub Button1_Click()
    Dim text As String
    text = "hello, world"
    Call getString(text)
End Sub
```

The key is to pass the string by value (ByVal).
At the FORTRAN end:

```fortran
stdcall subroutine getString(p_theString)
use windows
implicit none

C FORTRAN expects the string lengths to be passed
C after the formal argument list as values. Since
C Visual Basic does not do this and because this is
C a STDCALL procedure, we need to accept the argument
C as a general pointer to a string passed by value.

    integer p_theString; value p_theString

C Local variables
    character*1024 string ! longer than expected pointer (p_string, string)
    pointer (p_string, string)
    character*1024 temp, title
    integer length, p_temp, p_title, i

C The Visual Basic string is passed as a null terminated
C C string. The first thing we have to do is find the
C null to determine the length of the string.
    p_string = p_theString
    length = index(string, char(0))
    if ((length .lt. 2) .or. (length .gt. 1024)) return

C Copy the string to a local (and safe) variable.
C Initialize the message box title string
    temp = string(1:length-1)
    title = "FORTRAN DLL"

C Null terminate the strings for the call to "Message Box".
    temp = trim(temp)//char(0)
    title = trim(title)//char(0)

C Create pointers to the strings so they can be
C passed by value to the Win32 API function "MessageBox".
    p_temp = loc(temp)
    p_title = loc(title)
    i = MessageBox(0, p_temp, p_title, MB_OK)

C Null terminate the string so that we can get the length
    temp ="Goodbye World"//char(0)

C Get the length of the string
    length = index(temp, char(0))

C Copy the string character by character
    do i=1,length-1
        string(i:i)=temp(i:i)
    end do
    string(length:length)=char(0)
end
```

Fortran User Guide
Use the following commands to build the DLL (assuming the FORTRAN source file is "test.f"):

```plaintext
f95 -c test.f
lnk -lib test.obj -exports:test.xps
lnk -dll test.obj -exports:test.xps absRT0.lib kernel32.lib user32.lib \ fio.lib fmath.lib libac.lib -aliases:unicode.als
```
132 column source code................................. 60, 70
About Editor menu command........................................ 33
Absoft Editor commands........................................... 24
  Edit menu.................................................. 16
  Help menu.................................................. 33
  Tools menu.................................................. 31
  View menu.................................................. 28
  Window menu............................................... 33
ABSOFT_RT_FLAGS.............................................. 95
ACCESS ............................................................ 130, 131
advanced optimizations.......................................... 39
amake tool....................................................... 113
Array menu command............................................. 33, 51
array boundary checking.................................. 54, 67
ASCII table............................................................... 181
Atlas............................................................... 49
basic optimizations........................................... 39
bibliography...................................................... 186
BLAS libraries...................................................... 84
BLOCKSIZE ...................................................... 130, 131
bookmarks...................................................... 12, 14, 25
books, reference.................................................. 185, 186
C function results............................................ 159
structures.......................................................... 163
C/C++ options.................................................. 78
CARRIAGECONTROL ............................................. 131
CDECL ............................................................... 85
CHARACTER length arguments.............................. 161
Character Map application.................................. 181
Check Syntax menu command.................................. 31
Close All menu command...................................... 17
Close menu command......................................... 17
colon, in path names........................................... 94
COMMON blocks
  from C......................................................... 162
COMMON, aligning data......................................... 61
Compile menu command........................................... 31
compiler directives........................................... 76
  FIXED directive........................................... 77
  FREE directive ........................................... 77
  NAME directive ........................................... 77
  PACKOFF directive......................................... 130
  PACKON directive.......................................... 129
  STACK directive........................................... 77
compiler options............................................. 165
  -traps, FPU exception handling........................ 42
  -8, Fortran 90............................................. 60
  -C, check boundaries.................................... 54
  -cpu, CPU specific optimization........................ 39
  -D, define compiler variable........................... 56
  -i, define compiler variable........................... 56
  -o, one trip DO loops................................... 58
  -ea, stop on error........................................ 64
  -ej, one trip DO loops................................... 67
  -en, non-standard usage................................ 63
  -ep, demote Double Precision........................... 67
  -eq, allow greater than 100 errors.................... 64
  -ER, default recursion.................................... 64
  -et, exception traceback.................................. 38
  -f, case fold............................................... 57, 87
  -f, case folding.......................................... 162
  -f, fixed source form.................................. 70
  -f, free source form.................................... 70
  -g, Fx debugging........................................... 87
  -H, operations to unroll.................................. 73
  -h, unroll count.......................................... 73
  -heap, heap size........................................... 80
  -i, integer sizes.......................................... 58, 67
  -I, MODULE file ............................................ 133
  -K, escape sequences..................................... 60, 71
  -K, escape sequences in strings........................ 161
  -link -exports
    name, exports file........................................ 81
  -link -map
    name, link map file...................................... 80
  -MS7D, Microsoft directives............................ 71
  -N1, static storage........................................ 58
  -N109, case fold........................................... 58
  -N110, no Common mangling................................ 62
  -N13, floating point sizes............................... 58, 67
  -N15, append underscore................................... 58
  -N22, set Common........................................... 61
  -N26, set big-endian....................................... 60, 70
  -N27, set little-endian..................................... 60, 70
  -N32, non-ANSI............................................... 54
  -N34, align COMMON......................................... 61
  -N9, MRWE Events............................................ 38
  -nodefaultmod, MODULE path................................ 74
  -noesse2, disable SSE2 instructions..................... 74
  -O1, basic optimizations.................................. 39
  -O2, normal optimizations................................ 39
  -O3, advanced optimizations............................. 39
  -O4, IPA optimizations..................................... 39
  -OPT
    round=, FPU rounding mode................................ 41
    -p, module files.......................................... 65
    -q, quiet.................................................. 54, 64
    -Ra, check argument interface.......................... 74
    -Rb, check array conformance........................... 68
    -Rb, check boundaries.................................... 67
    -Rn, check argument count................................ 74
    -Rp, check pointers....................................... 68
    -Rs, check substrings..................................... 68
    -s, static storage........................................ 58, 67, 87
    -safefp, safe floating-point............................ 73, 74
    -sse2, enable SSE2 instructions........................ 74
    -stack, stack size........................................ 80
    -T, max internal handle................................... 55, 65
    -t, temporary strings..................................... 55, 65
    -U, loop unrolling........................................ 73
    -v, show progress.......................................... 54, 64
    -V, show version............................................ 64
    -V, VAX Tab-Format......................................... 60
    -W, line length............................................ 70
    -w, suppress compiler warnings........................ 54, 63
    -W, wide format.......................................... 60
    -x, disable compiler directive............................ 66
Index

-x, conditional compilation........................................... 54
-YALL_NAMES, symbolic names .................................. 76
-YCFRL=1, CHARACTER argument parameters ....................... 68
-YCOM_NAMES, COMMON block case .................................. 72
-YCOM_PFX, COMMON block prefix .................................. 72
-YCOM_SFX, COMMON block suffix .................................. 72
-YDEALLOC, cache control ............................................. 65
-YEXT_NAMES, external names ........................................ 70
-YEXT_PFX, external symbol prefix .................................. 71
-YEXT_SFX, external symbol suffix .................................. 71
-YMSFT_GLB_PFX, Add Microsoft GLOBAL prefix ................. 73
-YNDFP, type elements ............................................... 71
--YNOCDE, ignore CDEC$ directives ............................... 76
-YPEI, pointers/integers .............................................. 68
-YVAR_NAMES, variable names ....................................... 75
-YPEF_CHAR, DVF/CVF Character .................................... 69
-YX, conditional compilation ........................................ 74
-z, suppress messages ............................................... 64
-Z, suppress warning number ........................................ 63
compiler version ....................................................... 64
compiling FORTRAN source code .................................... 5
conditional compilation ............................................. 54, 74
conditional compilation variables ................................... 56
contacting Absoft ..................................................... 190
Contents menu command .............................................. 33
continuation lines ...................................................... 89
collections used in the manual ...................................... 2
Copy menu command ................................................. 24
Customize Toolbar menu command .................................... 32
Cut menu command .................................................... 24
DATE subroutine ....................................................... 88
diagnosis of errors .................................................... 98
diagnostic messages ................................................... 98
DLL creating ............................................................ 84
linking to ............................................................. 84
DLLs ................................................................. 191
DO loops one trip ...................................................... 58
DO Loops .................................................................. 67
documentation conventions .......................................... 2
Edit menu .................................................................. 24
elemental functions ..................................................... 174
elemental subroutines .................................................. 178
enabling the exception ............................................... 98
END MAP statement ................................................... 132
END UNION statement ............................................... 132
Environment Preferences ............................................. 20
errors ....................................................................... 64
exceptions divide by zero .............................................. 98
operand error .......................................................... 98
overflow ................................................................... 98
Exit menu command ................................................. 23
extensions key Microsoft FORTRAN .............................. 90
key Sun FORTRAN ones .............................................. 92
key VAX FORTRAN ones ............................................. 88
key VS FORTRAN ones ............................................... 90
f77.exe .................................................................. 100
file names ............................................................... 94
File menu .................................................................. 16
Find menu command ................................................... 28
Find Next menu command ........................................... 29
FIXED directive ......................................................... 77
FLEN ..................................................................... 131
floating point unit ...................................................... 97
floating point unit exception handling .................................. 98
rounding direction ...................................................... 98
FORM="BINARY" specifier ......................................... 91
Format menu ............................................................. 25
Format Preferences ...................................................... 18
Color ....................................................................... 18
Language .................................................................. 18
Fortran 77 options ..................................................... 53, 62, 79
Fortran 77 extensions .................................................. 74
conditional compilation ............................................. 74
Fortran 77 Fixed Source Form ...................................... 59
Fortran 90 ............................................................... 129
Fortran 90 Fixed Source Form .................................... 70
Fortran 90 Free Source Form ....................................... 60, 70
Fortran Forum ......................................................... 186
FPU exception handling .............................................. 42
FPU rounding mode; ................................................... 41
FREE directive .......................................................... 77
Fsplit utility tool ......................................................... 101
global symbols ........................................................ 162
Go to Line menu command .......................................... 30
handle ..................................................................... 154
Help menu ............................................................... 33, 52
IBM RS/6000, porting from ........................................... 92
IDATE subroutine ..................................................... 88
IEEE floating point math ............................................. 97
IEEE_ALL ............................................................ 177
IEEE_CLASS .......................................................... 174
IEEE_CLASS_TYPE ............................................... 175
IEEE_COPY_SIGN ................................................... 174
IEEE_DATATYPE ..................................................... 170
IEEE_DENORMAL ..................................................... 170
IEEE_DIVIDE .......................................................... 170
IEEE_DIVIDE_BY_ZERO ........................................... 175, 177
IEEE_DOWN ............................................................ 172
IEEE_FEATURES_TYPE ........................................... 170, 171
IEEE_FLAG_TYPE .................................................... 177
IEEE_GET_FLAG ....................................................... 177, 178
IEEE_GET_HALTING_MODE ....................................... 178
IEEE_GET_ROUNDING_MODE ..................................... 176
IEEE_GET_STATUS ................................................... 178
IEEE_GET_UNDERFLOW_MODE ..................................... 176
IEEE_HALTING ......................................................... 170
IEEE_INEXACT ......................................................... 177
IEEE_INEXACT_FLAG ............................................. 170

Fortran User Guide
IEEE_INF ......................................................... 170
IEEE_INVALID .................................................. 177
IEEE_INVALID_FLAG .......................................... 170
IEEE_IS_FINITE ................................................ 174
IEEE_IS_NAN .................................................... 174
IEEE_IS_NEGATIVE .......................................... 174
IEEE_IS_NORMAL .............................................. 174
IEEE_LOGB ..................................................... 175
IEEE_NAN ....................................................... 170
IEEE_NEAREST ................................................ 172
IEEE_NEGATIVE_DENORMAL .................................. 171
IEEE_NEGATIVE_NORMAL .................................... 171
IEEE_NEGATIVE_ZERO ....................................... 171
IEEE_NEXT_AFTER ........................................... 175
IEEE_OVERFLOW ............................................... 177
IEEE_POSITIVE_DENORMAL .................................. 171
IEEE_POSITIVE_INF ......................................... 171
IEEE_POSITIVE_NORMAL .................................... 171
IEEE_POSITIVE_ZERO ....................................... 171
IEEE_QUIET_NAN ............................................. 171
IEEE_RINT ..................................................... 175
IEEE_ROUND_TYPE ......................................... 171
IEEE_ROUNDING ............................................. 170
IEEE_SCALB ................................................... 175
IEEE_SELECTED_REAL_KIND ................................ 176
IEEE_SET_FLAG ............................................... 178
IEEE_SET_HALTING_MODE .................................. 178
IEEE_SET_ROUNDING_MODE ................................ 176
IEEE_SET_STATUS ............................................ 178
IEEE_SET_UNDERFLOW_MODE ................................ 176
IEEE_SIGNALING_NAN ....................................... 171
IEEE_SQRT .................................................... 170
IEEE_STATUS_TYPE .......................................... 177
IEEE_SUPPORT_DATATYPE .................................. 172, 174
IEEE_SUPPORT_DENORMAL .................................. 172
IEEE_SUPPORT_DIVIDE ..................................... 172
IEEE_SUPPORT_INF ......................................... 172
IEEE_SUPPORT_IO ............................................ 173
IEEE_SUPPORT_NAN ......................................... 173
IEEE_SUPPORT_ROUNDING ................................ 173
IEEE_SUPPORT_SQRT ........................................ 173
IEEE_SUPPORT_STANDARD .................................. 173
IEEE_SUPPORT_UNDERFLOW ................................ 174
IEEE_SUPPORT_UNDERFLOW_CONTROL ..................... 176
IEEE_TO_ZERO ............................................... 172
IEEE_UNDERFLOW .......................................... 177
IEEE_UNDERFLOW_FLAG ..................................... 170
IEEE_UNORDERED ............................................ 175
IEEE_UP ....................................................... 172
IEEE_USUAL ................................................... 177
IEEE_VALUE ..................................................... 175
IMPLICIT NONE statement .................................. 151
IMSL libraries .................................................. 83
inquiry functions ............................................. 172
Insert Continuation menu command ...................... 30
italicized text, defined ..................................... 2
key equivalent ................................................. 11
kind function ................................................... 176
Language Systems Fortran .................................. 93
LAPACK libraries .............................................. 84
lb tool ......................................................... 110
lib tool ......................................................... 109
license for redistribution ................................... 94
link tool ......................................................... 102
LOC intrinsic function ..................................... 159
loop unrolling ................................................ 73
MAP statement ............................................... 132
master pointer ................................................. 154
Match Brackets menu command ........................... 30
memory management ........................................ 94
menus
Edit ............................................................. 24
File .............................................................. 16
Help ............................................................ 33, 52
Tools ............................................................ 31
View ............................................................. 28
Window .......................................................... 33
metacommads, Microsoft FORTRAN ..................... 91
Microsoft FORTRAN
metacommads ................................................. 91
porting from .................................................. 90
Microsoft Win32 Programmers Reference, Volumes I-
5: To include one of these files, use a statement
such as......................................................... 148
modifier keys ................................................... 2
MODULE .......................................................... 133
MODULE files ................................................... 133
module files .................................................... 65
MODULE files
locating .......................................................... 148
search path ..................................................... 148
modules .......................................................... 133
MRWE .......................................................... 135
modifying ....................................................... 146
mrwe graphics .................................................. 139
mrwe text ....................................................... 138
mrwe.lib file ................................................... 135
MS-DOS, porting from ....................................... 90
-N9 .............................................................. 138, 143
NAME directive .............................................. 77
naming conventions .......................................... 162
New menu command ........................................ 16
New Window menu command ............................... 33
Next Error menu command ................................ 30
Next Error menu command ................................ 30
NeXT, porting from .......................................... 92
non ANSI warnings .......................................... 54
non-elemental subroutines ................................ 175, 178
normal optimizations ........................................ 39
one trip DO ..................................................... 58
Open menu command ........................................ 16
operand error exceptions .................................... 98
optimization ................................................... 38
optimizations
loop unrolling ................................................ 73
options ......................................................... 165
options, manual convention ................................ 2
other porting issues ......................................... 94
overflow exceptions ......................................... 98
overstrike ....................................................... 12
PACKOFF directive .......................................... 130
PACKON directive .......................................... 129
Index

Published by  

Fortran User Guide

Page Setup menu command .................................. 22, 23
partial path name ............................................... 95
Paste menu command ........................................... 24
path names ......................................................... 94
phone number, technical support .......................... 190
PLplot ............................................................... 84
Plug-Ins............................................................ 81
porting code ..................................................... 87
  from IBM RS/6000 ........................................... 92
  from Microsoft FORTRAN .............................. 90
  from MS-DOS ................................................. 90
  from NeXT ................................................... 92
  from Sparc ................................................... 92
  from Sun FORTRAN ......................................... 92
  from VAX FORTRAN ....................................... 87
  from VS FORTRAN ......................................... 90
preconnected units ........................................... 137
Preferences menu command .................................. 17
Previous Error menu command ................................ 30
Print menu command ........................................... 22
Print Preview menu command ................................. 22
problems, technical support for ............................ 189
procedure naming conventions .............................. 162
Properties menu command .................................... 26
qualifiers, VAX FORTRAN .................................... 89
RAN function .................................................... 88
rc, resource compiler ........................................ 100
Recent File menu command .................................... 23
RECORD statement ............................................ 152, 163
Redo menu command ........................................... 24
Refresh menu command ....................................... 30
Replace menu command ....................................... 29
Rever menu command ......................................... 17
road maps ........................................................ 2
RS/6000, porting from ........................................ 92
running compiled applications .............................. 5
Save All menu command ....................................... 17
Save As menu command ........................................ 17
Save menu command ........................................... 17
search path, for MODULE ..................................... 148
SECNDS subroutine ............................................ 88
Select All menu command ..................................... 24
setting tab size ............................................... 95
show compiler progress ....................................... 95
tab character .................................................... 95
tab size
  compile time ................................................... 95
  runtime .......................................................... 95
TABSIZE variable ............................................. 102
target file, updating ......................................... 114
technical support ............................................. 189
text selection .................................................... 12
Tile Horizontally menu command ........................... 33, 51
Tile Vertically menu command ................................ 33, 51
TIME subroutine ............................................... 88
Toolbar menu command ....................................... 28
Tools menu ....................................................... 31
Tools Preferences ............................................... 22
tutorial
  books for beginners ....................................... 185, 186
  underlined text, defined ................................... 2
Undo menu command ........................................... 24
UNION statement .............................................. 132
updating target file .......................................... 114
using compilers ............................................... 12
Using Help menu command ................................... 33
VAL function ..................................................... 148
VALUE statement ............................................. 148, 159
VAST pre-processor .......................................... 83
VAX FORTRAN
  porting from ............................................... 87
  qualifiers ...................................................... 89
VAX Tab-Format source ...................................... 60
View menu ........................................................ 28
Visual Basic .................................................... 191, 192
VS FORTRAN, porting from ................................ 90
wide source format .......................................... 60
Window menu ................................................... 33
Windows menu command ...................................... 33, 52
Windows System Services .................................... 3
Y2K bug .......................................................... 3
static storage .................................................. 67
Status Bar menu command .................................... 28
STDCALL .......................................................... 85
STDCALL statement ......................................... 150, 153
Stop menu command ........................................... 31
string length ................................................... 65
STRUCTURE statement ...................................... 152
Sun FORTRAN, porting from ................................ 92
suppress list of compiler warning messages ............. 63
SYSTEM_INFO structure ..................................... 153
tab character .................................................... 95
"tab size"
  compile time ................................................... 95
  runtime .......................................................... 95
TABSIZE variable ............................................. 102
target file, updating ......................................... 114
technical support ............................................. 189
text selection .................................................... 12
Tile Horizontally menu command ........................... 33, 51
Tile Vertically menu command ................................ 33, 51
TIME subroutine ............................................... 88
Toolbar menu command ....................................... 28
Tools menu ....................................................... 31
Tools Preferences ............................................... 22
tutorial
  books for beginners ....................................... 185, 186
  underlined text, defined ................................... 2
Undo menu command ........................................... 24
UNION statement .............................................. 132
updating target file .......................................... 114
using compilers ............................................... 12
Using Help menu command ................................... 33
VAL function ..................................................... 148
VALUE statement ............................................. 148, 159
VAST pre-processor .......................................... 83
VAX FORTRAN
  porting from ............................................... 87
  qualifiers ...................................................... 89
VAX Tab-Format source ...................................... 60
View menu ........................................................ 28
Visual Basic .................................................... 191, 192
VS FORTRAN, porting from ................................ 90
wide source format .......................................... 60
Window menu ................................................... 33
Windows menu command ...................................... 33, 52
Windows System Services .................................... 3
Y2K bug .......................................................... 3
static storage .................................................. 67
Status Bar menu command .................................... 28
STDCALL .......................................................... 85
STDCALL statement ......................................... 150, 153
Stop menu command ........................................... 31
string length ................................................... 65
STRUCTURE statement ...................................... 152
Sun FORTRAN, porting from ................................ 92
suppress list of compiler warning messages ............. 63
SYSTEM_INFO structure ..................................... 153
Y2K bug .......................................................... 3