

NightBench User's Guide



0890514-060
September 2000

Copyright 2000 by Concurrent Computer Corporation. All rights reserved. This publication or any part thereof is intended for use with Concurrent Computer Corporation products by Concurrent Computer Corporation personnel, customers, and end-users. It may not be reproduced in any form without the written permission of the publisher.

The information contained in this document is believed to be correct at the time of publication. It is subject to change without notice. Concurrent Computer Corporation makes no warranties, expressed or implied, concerning the information contained in this document.

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Computer Corporation, 2881 Gateway Drive, Pompano Beach, FL 33069-4324. Mark the envelope **“Attention: Publications Department.”** This publication may not be reproduced for any other reason in any form without written permission of the publisher.

NightBench, NightView, MAXAda and PowerMAX OS are trademarks of Concurrent Computer Corporation.

Power Hawk is a trademark of Concurrent Computer Corporation.

Élan License Manager is a trademark of Élan Computer Group, Inc.

OSF/Motif is a registered trademark of The Open Group.

X Window System and X are trademarks of The Open Group.

UNIX is a registered trademark, licensed exclusively by X/Open Company Ltd.

HyperHelp is a trademark of Brisol Technology Inc.

The NightBench Program Development Environment includes the XmpTable widget, which carries the following notice:

Copyright 1990, 1991, 1992, 1993, 1994 David E. Smyth

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of David E. Smyth not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

Printed in U. S. A.

Revision History:	Level:	Effective With:
Original Release -- November 1997	000	NightBench 1.0
Current Release -- September 2000	060	NightBench 2.1

General Information

The *NightBench User's Guide* documents the NightBench Program Development Environment. NightBench is a set of graphical user interface (GUI) tools for developing software with the Concurrent MAXAda™ and C/C++ compiler toolsets on Concurrent computers running under PowerMAX OS™.

Scope of Manual

This manual is a reference document and user's guide for the NightBench Program Development Environment.

Structure of Manual

This manual consists of 7 chapters and a glossary. A brief description of the contents of each of the chapters of the manual is described as follows.

- Chapter 1 is an “Introduction to NightBench”. It gives an overview of the different components of NightBench and their functions within the system. It also gives information on starting NightBench and the effect of the various command-line options. Finally, it gives specific information about some of the common NightBench dialogs.
- Chapter 2 is a tutorial on “Using NightBench with Ada”. It goes step-by-step from creating an Ada environment to building a partition. Many of the common tasks performed in NightBench are discussed here.
- Chapter 3 is a tutorial on “Using NightBench with C/C++”. It goes step-by-step from creating a C++ environment to building a partition. Many of the common tasks performed in NightBench are discussed here.
- Chapter 4 documents “NightBench Project”, a central location for access to the other NightBench components and environments. This chapter gives a description of each of the items found in this window and the dialogs associated with it.
- Chapter 5 talks about the “NightBench Development”, the window in which environments are created, units and source files are managed, and partitions are defined.
- Chapter 6 details the “NightBench Builder”, the mechanism used for compilation and linking of Ada units and partitions.

- Chapter 7 discusses the “Options Editors”, the various settings contained within them, and their relationship to the units and partitions contained within the environment.

Syntax Notation

The following notation is used throughout this guide:

<i>italic</i>	Books, reference cards, and items that the user must specify appear in <i>italic</i> type. Special terms and comments in code may also appear in <i>italic</i> .
list bold	User input appears in list bold type and must be entered exactly as shown. Names of directories, files, commands, options and man page references also appear in list bold type.
list	Operating system and program output such as prompts and messages and listings of files and programs appears in list type. Keywords also appear in list type.
<u>emphasis</u>	Words or phrases that require extra emphasis use <u>emphasis</u> type.
window	Keyboard sequences and window features such as push buttons, radio buttons, menu items, labels, and titles appear in window type.
[]	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify such option or arguments.
{ }	Braces enclose mutually exclusive choices separated by the pipe () character, where one choice must be selected. You do not type the braces or the pipe character with the choice.
...	An ellipsis follows an item that can be repeated.
::=	This symbol means <i>is defined as</i> in Backus-Naur Form (BNF).

Referenced Publications

The following publications are referenced in this document:

0890395	<i>NightView™ User's Guide</i>
0890398	<i>NightTrace™ Manual</i>
0890423	<i>PowerMAX OS™ Programming Guide</i>
0890516	<i>MAXAda™ Reference Manual</i>
0891082	<i>Real-Time Clock and Interrupt Module User's Guide</i>

Contents

Chapter 1 Introduction to NightBench

NightBench Features	1-1
NightBench Components	1-1
Internal Components	1-2
X Resource Names	1-2
Starting NightBench	1-3
NightBench Help	1-3
NightBench Project	1-3
NightBench Development	1-3
Additional NightBench Development Options	1-4
NightBench Builder	1-4
Additional Builder Options	1-5
Additional NightBench Options	1-6
Configuring NightBench	1-7
Preferences	1-8
Preferences - Viewer	1-8
Preferences - Editor	1-9
Preferences - Terminal	1-11
Preferences - Transcripts	1-12
Preferences - Language	1-13
Preferences - Start	1-14
Using the Emacs Editor - Additional Considerations	1-16
Resizing columns	1-17
Command Log	1-18
Tools Menu	1-19
Help Menu	1-21

Chapter 2 Using NightBench with Ada

Starting NightBench	2-1
Creating a new environment	2-2
Introducing units into the environment	2-3
Defining a partition	2-6
Including units in the partition	2-6
Building a partition	2-8
Running the program	2-9
What options do I have?	2-11
Setting environment-wide compile options	2-11
Setting unit compile options	2-12
Determining the effective options	2-14
Copying the temporary options to the permanent set	2-15
Hello Galaxy - The Example Continues.....	2-16
Setting up another environment	2-16
Introducing units into the galaxy	2-17
Modifying an existing unit	2-20

Building a partition with references outside the local environment	2-21
Adding an environment to the Environment Search Path	2-23
Making contact!!!	2-25
Who resides on earth?	2-26
Hello Again - Ambiguous Units	2-28
No more ambiguities!!!	2-30
Using NightBench - Conclusion	2-32

Chapter 3 Using NightBench with C/C++

Starting NightBench	3-1
Creating a new environment	3-2
Introducing units into the environment	3-3
Defining a partition	3-7
Including units in the partition	3-8
Building a partition	3-9
Running the program	3-10
What options do I have?	3-12
Setting environment-wide compile options	3-12
Setting unit compile options	3-13
Determining the effective options	3-14
Copying the temporary options to the permanent set	3-15
Using NightBench - Conclusion	3-15

Chapter 4 NightBench Project

NightBench Project Menu Bar	4-1
NightBench Menu	4-2
Options Menu	4-3
NightBench Project	4-4
New Environment	4-6
Add Environment to List	4-9

Chapter 5 NightBench Development

Structure	5-1
NightBench Development Menu Bar	5-3
Development Menu	5-3
New Environment	5-5
Open Environment	5-8
Remove Environment	5-11
Select Menu	5-13
Options Menu	5-14
Accelerated Item Selection	5-16
Development - Settings	5-18
Change Frozen Status	5-25
Change Permissions	5-27
Add Environment Search Path Element	5-28
Replace Environment Search Path Element	5-30
Development - Source Files	5-31
Source File Commands - (Ada Only)	5-35
Introduce Source Files	5-37
Commands	5-43

Edit File Name	5-47
Edit Unit Name	5-48
Edit Make Command	5-49
Analyze Source Files - (Ada Only)	5-51
Analyze Errors	5-53
Remove Source Files	5-56
Save List to File	5-57
Introduce from File.	5-59
Development - Units	5-61
Commands	5-68
Hidden Units	5-72
Resolve Ambiguity	5-74
Hide Status	5-76
Magnet Status	5-78
Manual Instantiation Resolution	5-80
Select Real Units	5-80
Select Artificial Units.	5-83
Unit Information.	5-87
Development - Dependencies - (Ada Only)	5-93
Development - Partitions - (Ada)	5-96
Create Partition	5-103
Partition File Commands	5-105
Save Partition Definitions to File	5-106
Create Partitions from File	5-107
Partitions - General	5-110
Partitions - Units	5-113
Included Units	5-114
Excluded Units.	5-118
Add Included Unit	5-121
Add Excluded Unit	5-122
Partitions - Link Control	5-124
Dependent Partitions	5-124
Link Rule.	5-126
Implicitly-Included Libraries	5-130
Link Rule Examples.	5-131
Add Dependent Partition	5-131
Partitions - Link Options	5-133
Partitions - Tracing	5-138
Partitions - Expert	5-142
Development - Partitions - (C/C++)	5-147
Create Partition	5-154
Partitions - General	5-156
Partitions - Units	5-161
Included Units	5-162
Excluded Units.	5-164
Add Included Unit	5-166
Add Excluded Unit	5-166
Partitions - Partitions	5-168
Dependent Partitions	5-168
Add Dependent Partition	5-171
Partitions - Libraries	5-172
Ordering	5-172
Library Paths (-L)	5-173
Libraries (-l)	5-174

Partitions - Link Options 5-176
Partitions - Expert 5-180

Chapter 6 NightBench Builder

Structure 6-1
Builder Menu Bar 6-2
 Builder Menu 6-2
 Open Environment 6-3
 Build Transcripts 6-6
 Save Transcript 6-9
 Options Menu 6-12
Builder - Build 6-13
 Run Context 6-19
 New Run Context 6-22
 Rename Run Context 6-23
 Run Context - Common 6-24
 Run Context - Debug 6-27
 Run Context - Run 6-29
 Run Context - Preference 6-32
 Search Transcript 6-34
Builder - Targets 6-36
Builder - Settings 6-38
Builder - Notification 6-41
Builder - Expert 6-44

Chapter 7 Options Editors

Effective Options 7-1
Tri-state Checkboxes 7-1
Environment Compile Options Editor 7-3
 Button Functionality 7-3
 Ada Environment Compile Options Editor 7-4
 Ada Environment Compile Options Editor - General 7-4
 Ada Environment Compile Options Editor - Inlining 7-6
 Ada Environment Compile Options Editor - Optimization 7-7
 Ada Environment Compile Options Editor - Expert 7-8
 C/C++ Environment Compile Options Editor 7-9
 C/C++ Environment Compile Options Editor - General 7-9
 C/C++ Environment Compile Options Editor - Macros 7-11
 C/C++ Environment Compile Options Editor - Includes 7-13
 C/C++ Environment Compile Options Editor - Optimizer 7-16
 C/C++ Environment Compile Options Editor - Language 7-18
 C/C++ Environment Compile Options Editor - Dialects 7-20
 C/C++ Environment Compile Options Editor - Expert 7-22
 Unit Compile Options Editor 7-24
 Ada Unit Compile Options Editor 7-24
 Button Functionality 7-24
 Ada Unit Compile Options Editor - General 7-26
 Ada Unit Compile Options Editor - Inlining 7-27
 Ada Unit Compile Options Editor - Optimization 7-28
 Ada Unit Compile Options Editor - Expert 7-29
 C/C++ Unit Compile Options Editor 7-31

C/C++ Unit Compile Options Editor - General	7-31
C/C++ Unit Compile Options Editor - Macros	7-33
C/C++ Unit Compile Options Editor - Includes	7-36
C/C++ Unit Compile Options Editor - Optimizer	7-40
C/C++ Unit Compile Options Editor - Language	7-42
C/C++ Unit Compile Options Editor - Dialects.	7-44
C/C++ Unit Compile Options Editor - Expert	7-46
Compile Options	7-47
Ada Options	7-47
Compiler Error Output	7-47
Debug Information	7-48
Growth Limit.	7-49
Inline Line Count	7-49
Inline Nesting Depth	7-49
Inline Statement Limit.	7-49
Inlines Per Compilation.	7-50
Invert Divides	7-50
Loops to Optimize.	7-50
Maximum (“benchmark”) Optimization	7-51
Optimization Class	7-51
Optimization Level	7-51
Optimization Size Limit	7-52
Optimize for Space	7-52
Quiet Compiler Info Msgs.	7-52
Quiet Compiler Warnings	7-53
Share Mode	7-53
Suppress Runtime Checks.	7-54
Sync After Volatile References	7-54
Unroll Limit.	7-54
Use Instruction Scheduler	7-54
Use Multiply-Add Instructions	7-55
Variables to Optimize	7-55
C/C++ Options	7-55
64 Bit long long type.	7-55
Accept Alternate Tokens	7-56
Accept the bool Keyword	7-56
Accept the typename Keyword	7-56
Accept the wchar_t Keyword	7-56
Allow Function Inlining	7-57
Ansi Compliance Enforcement	7-57
Cfront 2.1 Compatible.	7-57
Cfront 3.0 Compatible.	7-57
Chars Signed by Default	7-58
Check for Embedded C++ Subset.	7-58
Constant Loop Unroll Limit	7-58
Debug Information	7-58
Enable Intrinsics	7-59
Error Limit.	7-59
Exclude Standard Includes	7-59
Float Arguments Single.	7-59
Float Constants Single.	7-60
Full Debug Information.	7-60
Growth Limit.	7-61
Initialized Vars Readonly	7-61

Loops to Optimize	7-61
Old K&R Language (C Only)	7-61
Optimization Class	7-62
Optimization Level	7-62
Optimize for Space	7-63
Position Independent Code	7-63
Short Temp Lifetimes	7-63
Show Remarks (Nitpick)	7-63
Suppress Warnings.	7-64
SVR4 Compatibility (C Only).	7-64
Total Alias Object Limit	7-64
Use Exceptions	7-64
Use Long Preserving Rules (C Only)	7-65
Use namespaces.	7-65
Use Old For Loop Declarations.	7-65
Use Run Time Type Identification.	7-65
Variable Loop Unroll Limit.	7-66
Variables to Optimize	7-66
Warn About Old For Loops	7-66
NightBench Project	A-1
NightBench Development	A-1
NightBench Builder	A-2

Screens

Screen 2-1. Starting NightBench.	2-1
Screen 2-2. Source file world.a containing hello unit	2-4
Screen 2-3. Source file planet.a containing alien unit	2-18
Screen 2-4. Reference the alien unit within the hello unit	2-20
Screen 2-5. Source file newunit.a containing ambiguous hello unit.	2-28
Screen 3-1. Starting NightBench.	3-1
Screen 3-2. Sample source file - world.c	3-4
Screen 5-1. Sample Makefile for pebbles.c Make Command	5-49

Illustrations

Figure 1-1. Preferences - Viewer	1-8
Figure 1-2. Preferences - Editor	1-10
Figure 1-3. Preferences - Terminal	1-11
Figure 1-4. Preferences - Transcripts	1-12
Figure 1-5. Preferences - Language	1-13
Figure 1-6. Preferences - Start	1-14
Figure 1-7. Resizing columns	1-17
Figure 1-8. Command Log window	1-18
Figure 1-9. Tools menu	1-19
Figure 1-10. Help menu	1-21
Figure 2-1. NightBench Project	2-2
Figure 2-2. Selecting the language	2-2
Figure 2-3. Selecting the directory	2-3
Figure 2-4. Introducing a source file	2-4
Figure 2-5. Source file, world.a - newly introduced	2-5
Figure 2-6. Including units in a newly defined partition	2-7
Figure 2-7. Builder window - Build page for hello partition	2-9

Figure 2-8. Ada Environment Compile Options Editor	2-11
Figure 2-9. Ada Unit Compile Options Editor	2-13
Figure 2-10. Ada Unit Compile Options Editor - Effective Options	2-14
Figure 2-11. NightBench Project	2-17
Figure 2-12. Analyze Errors dialog - mistake in unit being introduced	2-19
Figure 2-13. Builder window - Build page for hello partition with errors	2-22
Figure 2-14. Environment Search Path for the earth environment	2-23
Figure 2-15. Updated Environment Search Path for the earth environment	2-24
Figure 2-16. Units page with Visa List Format	2-27
Figure 2-17. Analyze Errors dialog - ambiguous units	2-29
Figure 2-18. Resolve Ambiguity dialog	2-29
Figure 2-19. Source file, newunit.a , newly introduced	2-30
Figure 2-20. Running the program	2-32
Figure 3-1. NightBench Project	3-2
Figure 3-2. Selecting a language for a new environment	3-2
Figure 3-3. Selecting a directory for a new environment	3-3
Figure 3-4. Introducing a source file	3-4
Figure 3-5. Source Files page	3-5
Figure 3-6. Defining an executable partition	3-7
Figure 3-7. Including units in a newly defined partition	3-8
Figure 3-8. Builder window - Build page	3-10
Figure 3-9. Unit Compile Options dialog - General Page - Effective Options	3-14
Figure 4-1. NightBench menu	4-2
Figure 4-2. Options menu	4-3
Figure 4-3. NightBench Project	4-4
Figure 4-4. New Environment - Language Selection	4-6
Figure 4-5. New Environment - Directory Selection	4-7
Figure 4-6. Add Environment to List - Directory Selection	4-9
Figure 4-7. Add Environment to List - Language Selection	4-10
Figure 5-1. Development menu	5-3
Figure 5-2. New Environment - Language Selection	5-6
Figure 5-3. New Environment - Directory Selection	5-7
Figure 5-4. Open Environment - Directory Selection	5-8
Figure 5-5. Open Environment - Language Selection	5-9
Figure 5-6. Remove Environment - Directory Selection	5-11
Figure 5-7. Remove Environment - Language Selection	5-12
Figure 5-8. Select menu	5-13
Figure 5-9. Options menu	5-14
Figure 5-10. Accelerated Item Selection	5-16
Figure 5-11. NightBench Development - Settings page (Ada)	5-18
Figure 5-12. NightBench Development - Settings page (C/C++)	5-19
Figure 5-13. Change Frozen Status dialog	5-25
Figure 5-14. Change Permissions dialog	5-27
Figure 5-15. Add Environment Search Path Element dialog	5-28
Figure 5-16. Replace Environment Search Path Element dialog	5-30
Figure 5-17. NightBench Development - Source Files page	5-31
Figure 5-18. Source File Commands menu	5-35
Figure 5-19. Introduce Source Files dialog	5-37
Figure 5-20. Introduce Source Files - Commands menu	5-43
Figure 5-21. Intro Edit File Name dialog	5-47
Figure 5-22. Intro Edit Unit Name dialog	5-48
Figure 5-23. Intro Edit Make Command dialog	5-49
Figure 5-24. Analyze Source Files dialog	5-51
Figure 5-25. Analyze Errors dialog	5-54

Figure 5-26. Remove Source Files dialog	5-56
Figure 5-27. Save List to File dialog	5-57
Figure 5-28. Introduce from File dialog	5-59
Figure 5-29. NightBench Development - Units page	5-61
Figure 5-30. Commands menu	5-68
Figure 5-31. Hidden Units dialog	5-72
Figure 5-32. Resolve Ambiguity dialog	5-74
Figure 5-33. Hide Status dialog	5-76
Figure 5-34. Magnet Status dialog	5-78
Figure 5-35. Manual Instantiation Resolution - Select Real Units	5-81
Figure 5-36. Manual Instantiation Resolution - Select Artificial Units	5-84
Figure 5-37. NightBench Development - Dependencies page	5-93
Figure 5-38. NightBench Development - Partitions page	5-97
Figure 5-39. Create Partition dialog	5-103
Figure 5-40. Partition File Commands menu	5-105
Figure 5-41. Save Partitions to File dialog	5-106
Figure 5-42. Create Partitions from File dialog	5-108
Figure 5-43. Partitions - General settings	5-110
Figure 5-44. Partitions - Units settings	5-113
Figure 5-45. Partitions - Included Units - active partition	5-114
Figure 5-46. Partitions - Included Units - non-active partition	5-115
Figure 5-47. Partitions - Excluded Units	5-118
Figure 5-48. Partitions - Add Included Unit dialog	5-121
Figure 5-49. Partitions - Add Excluded Unit dialog	5-123
Figure 5-50. Partitions - Link Control settings	5-124
Figure 5-51. Partitions - Add Dependent Partition dialog	5-132
Figure 5-52. Partitions - Link Options settings	5-133
Figure 5-53. Partitions - Tracing settings	5-138
Figure 5-54. Partitions - Expert settings	5-142
Figure 5-55. NightBench Development - Partitions page	5-148
Figure 5-56. Create Partition dialog	5-154
Figure 5-57. Partitions - General settings	5-156
Figure 5-58. Partitions - Units settings	5-161
Figure 5-59. Partitions - Included Units	5-162
Figure 5-60. Partitions - Excluded Units	5-164
Figure 5-61. Partitions - Add Included Unit dialog	5-166
Figure 5-62. Partitions - Add Excluded Unit dialog	5-167
Figure 5-63. Partitions - Partitions settings	5-168
Figure 5-64. Partitions - Add Dependent Partition dialog	5-171
Figure 5-65. Partitions - Libraries settings	5-172
Figure 5-66. Partitions - Link Options settings	5-176
Figure 5-67. Partitions - Expert settings	5-180
Figure 6-1. Builder menu	6-2
Figure 6-2. Open Environment dialog	6-4
Figure 6-3. Open Environment - Language Selection	6-4
Figure 6-4. Build Transcripts dialog	6-6
Figure 6-5. Save Transcript dialog	6-9
Figure 6-6. Options menu	6-12
Figure 6-7. NightBench Builder - Build page	6-13
Figure 6-8. Run Context dialog	6-19
Figure 6-9. New Run Context dialog	6-22
Figure 6-10. Rename Run Context dialog	6-23
Figure 6-11. Run Context - Common page	6-24
Figure 6-12. Run Context - Debug page	6-27

Figure 6-13. Run Context - Run page	6-29
Figure 6-14. Run Context - Preference page	6-32
Figure 6-15. Search Transcript dialog	6-34
Figure 6-16. NightBench Builder - Targets page	6-36
Figure 6-17. NightBench Builder - Settings page	6-38
Figure 6-18. NightBench Builder - Notification page	6-41
Figure 6-19. NightBench Builder - Expert page	6-44
Figure 7-1. Ada Environment Compile Options dialog - General Page	7-4
Figure 7-2. Ada Environment Compile Options dialog - Inlining Page	7-6
Figure 7-3. Ada Environment Compile Options dialog - Optimization Page	7-7
Figure 7-4. Ada Environment Compile Options dialog - Expert Page	7-8
Figure 7-5. C/C++ Environment Compile Options dialog - General Page	7-9
Figure 7-6. C/C++ Environment Compile Options dialog - Macros Page	7-11
Figure 7-7. C/C++ Environment Compile Options dialog - Includes Page	7-13
Figure 7-8. C/C++ Environment Compile Options dialog - Optimizer Page	7-16
Figure 7-9. C/C++ Environment Compile Options dialog - Language Page	7-18
Figure 7-10. C/C++ Environment Compile Options dialog - Dialects Page	7-20
Figure 7-11. C/C++ Environment Compile Options dialog - Expert Page	7-22
Figure 7-12. Ada Unit Compile Options dialog - General Page	7-26
Figure 7-13. Ada Unit Compile Options dialog - Inlining Page	7-27
Figure 7-14. Ada Unit Compile Options dialog - Optimization Page	7-28
Figure 7-15. Ada Unit Compile Options dialog - Expert Page	7-29
Figure 7-16. C/C++ Unit Compile Options dialog - General Page	7-31
Figure 7-17. C/C++ Unit Compile Options dialog - Macros Page	7-33
Figure 7-18. C/C++ Unit Compile Options dialog - Includes Page	7-36
Figure 7-19. C/C++ Unit Compile Options dialog - Optimizer Page	7-40
Figure 7-20. C/C++ Unit Compile Options dialog - Language Page	7-42
Figure 7-21. C/C++ Unit Compile Options dialog - Dialects Page	7-44
Figure 7-22. C/C++ Unit Compile Options dialog - Expert Page	7-46

Introduction to NightBench

The *NightBench User's Guide* documents the NightBench Program Development Environment. NightBench is a set of graphical user interface (GUI) tools for developing software with the Concurrent MAXAda™ and C/C++ compiler toolsets on Concurrent computers running under PowerMAX OS™.

This manual describes the components of the graphical user interface and their respective functions. The windows, dialogs, and menus of NightBench are provided throughout the manual, complete with descriptions of the various items contained within each. In addition, a glossary of terms is provided.

NightBench Features

In addition to the compiler toolsets, NightBench also incorporates NightView™, a fully-symbolic debugger, and NEdit, a fully-integrated text editor, bringing together a collection of tools crucial to special development - all within a consistent framework.

Addressing the various aspects of the software development process, NightBench provides a complete solution, handling the intricate details involved in consistent program generation. With its automated build facility, enhanced error processing, and persistence of compilation and other build options, the NightBench architecture assures reproducibility of programs within a given build environment.

NightBench supplies these tools in one unified graphical user interface based on OSF/Motif™ and the X Window System™ standards, providing the user with a work environment in which to develop Ada, C, or C++ programs quickly and easily.

See also:

- *MAXAda Reference Manual* (0890516)
- *NightView User's Guide* (0890395)

NightBench Components

The NightBench Program Development Environment is divided into three main components:

- NightBench Project - Chapter 4
- NightBench Development - Chapter 5

- NightBench Builder - Chapter 6

Internal Components

A number of processes are used by the the NightBench Program Development Environment. These processes run internally and are not directly referenced by the average NightBench user. The list of these internal components is provided for informational purposes only.

nb.bench	command issued by the user to start NightBench. See “Starting NightBench” on page 1-3 for more information.
nb.proj	handles all the NightBench Project user interactions
nb.dev	handles all of the NightBench Development user interactions
nb.build	handles all of the NightBench Builder user interactions
nb.serv	coordinates NightBench-specific activities
nb.term	handles communication between NightBench and programs running in terminal windows (such as text editors)
nb.error	manages communication of error messages between NightBench and the underlying MAXAda utilities
ktserv	manages general communications

X Resource Names

Setting the X resource names for the NightBench tools may be accomplished by using the application class names instead of the program names (which contain the ‘.’ character and therefore cause confusion).

Therefore, use the following application class names when modifying the X resources for the appropriate NightBench tools:

- NbDev - NightBench Development
- NbBuild - NightBench Builder
- NbProj - NightBench Project

Starting NightBench

A number of command line options may be specified when starting NightBench, allowing the user to open a particular NightBench component, gain access to the online *NightBench User's Guide*, send options to the Builder tool, among other such actions. These options and their respective meanings are listed here:

NightBench Help

nbench -H

Prints the usage and options help screen to standard output and exits.

nbench -man

Opens the online *NightBench User's Guide*.

NightBench Project

nbench

Brings up the NightBench Project window.

NightBench Development

nbench -dev

Brings up the NightBench Development window. If the current directory is already a NightBench environment, that environment will automatically be loaded into a new Development window. Otherwise, an empty Development window will be opened.

The **-lang** *language* option may be used to specify the language the environment will use (*language* may be either `ada` or `c/c++` - these arguments are case-insensitive and may be abbreviated).

nbench -env *directory*

Brings up the NightBench Development window, automatically opening the environment in the specified *directory*. If the environment is not valid, an empty Development window will be opened.

The **-lang** *language* option may also be used to specify the language of the environment to open when multiple environments exist in the same directory (*language* may be either `ada` or `c/c++` - these arguments are case-insensitive and may be abbreviated).

nbench -mkenv -lang language [-f] [-env directory] [-rel release]

This option indicates that a new environment should be created in the current working directory and opened in the **Development** window.

The **-lang language** option is required and specifies which language the new environment will use (*language* may be either `ada` or `c/c++` - these arguments are case-insensitive and may be abbreviated).

The **-f** option will force creation of the environment even if it or some portion of it already exists.

The **-env** option can be added to specify creation of the environment in a *directory* other than the current working directory.

The **-rel** option can be added to indicate which *release* of the compiler to use when creating the environment.

nbench -ada

This option is equivalent to **nbench -dev -lang ada** and is provided for backward compatibility with previous versions of NightBench.

Additional NightBench Development Options

-log

Opens a Command Log window when opening the NightBench Development window.

NOTE

You may also set the `NbAda*showLogWindow` resource to `True`. See “X Resource Names” on page 1-2 for related information.

NightBench Builder

nbench -build

Brings up the NightBench Builder window. If the current directory is already a NightBench environment, that environment will automatically be loaded into the Builder window. Otherwise, an empty **Builder** window will be opened.

nbench -build -env directory

Brings up the NightBench Builder window, automatically opening the environment in the specified *directory*. If the environment is not valid, an empty **Builder** window will be opened.

The **-lang** *language* option may also be used to specify the language of the environment to open when multiple environments exist in the same directory (*language* may be either *ada* or *c/c++* - these arguments are case-insensitive and may be abbreviated).

Additional Builder Options

-log

Opens a Command Log window when opening the NightBench Builder window.

The following options specify build targets for the NightBench Builder window.

-allparts

Sets the build targets to all partitions defined within the environment.

-parts *part1* ...

Sets the build targets to those partitions listed with this option. Multiple partitions, separated by spaces, may be specified to this option.

Partition names may be specified directly if no units are specified. For instance,

```
nbench -build -parts hello
```

is equivalent to:

```
nbench -build hello
```

-units *unit1* ...

Sets the build targets to those units listed with this option. Multiple units, separated by spaces, may be specified to this option.

A unit name of **all** may be specified to indicate all units should be built.

-requnits *unit1* ...

Sets the build targets to those units listed with this option. In addition, this option indicates that all units which depend upon the listed units should be built. Multiple units, separated by spaces, may be specified to this option.

-start

Immediately starts the build when the NightBench Builder window is opened. If no build targets are specified, all units and all partitions in the environment will be built by default.

NOTE

You may also specify the **Automatically Start Builds** option on the **Options** menu of either the NightBench Project or the NightBench Development window for this behavior. See “Automatically Start Builds” on page 4-3 or “Automatically Start Builds” on page 5-14.

Additional NightBench Options

-Xoption

NightBench accepts all of the standard X Toolkit command-line options (see **x(1)**).

Configuring NightBench

NightBench allows the user to configure certain aspects of its functionality. Using the **Preferences** dialog, the user can set which editor should be used for viewing and editing files under NightBench and which terminal emulator program is preferred.

In addition, there exists a mechanism for changing the size of columns in the lists that appear throughout the NightBench Program Development Environment.

See also:

- “Preferences” on page 1-8
- “Resizing columns” on page 1-17

Preferences

The **Preferences** dialog allows the user to configure which file viewer, file editor, and terminal program that NightBench uses. In addition, options with respect to build transcripts, language-specific settings, and activities when starting certain tools are presented in this dialog.

The following areas can be configured using the **Preferences** dialog:

- “Preferences - Viewer” on page 1-8
- “Preferences - Editor” on page 1-9
- “Preferences - Terminal” on page 1-11
- “Preferences - Transcripts” on page 1-12
- “Preferences - Language” on page 1-13
- “Preferences - Start” on page 1-14

Preferences - Viewer

Specifies which program should be used to view files.

See also:

- “Preferences” on page 1-8

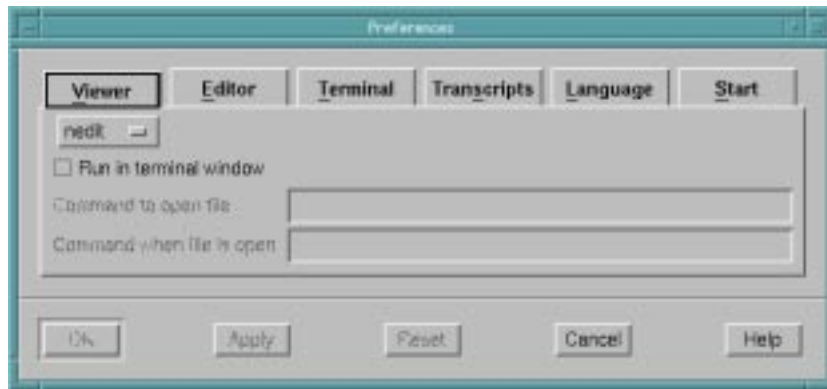


Figure 1-1. Preferences - Viewer

nedit	Either the NEdit or Emacs editor can be used to view files. Emacs will be used if it is running and has been configured for use with NightBench. Otherwise, NEdit will be used.
	See “Using the Emacs Editor - Additional Considerations” on page 1-16 for information on configuring Emacs for use with NightBench.
view	Use the view command to read files (see vi(1)).
more	Use the more(1) command to view files.
custom	Select this item if a file viewer other than those listed in this menu is desired.

Run in terminal window

The user should check this if the program chosen as the file viewer needs a terminal window (e.g. **xterm**) in which to run. This is particularly important if the **custom** option is selected from the **Viewer** menu.

When the **more** option is chosen from the **Viewer** menu, this checkbox is automatically checked because the **more(1)** command needs a terminal window in which to run.

Also, since neither the NEdit or Emacs editors need a terminal window (they create their own window when started), this checkbox remains unchecked when the **nedit** option is selected from the **Viewer** menu.

Command to open file

Enter the name of the program to be used as the file viewer.

This field is active only when the **custom** menu item is selected from the **Viewer** menu.

Command when file is open

This field is active only when the **custom** menu item is selected from the **Viewer** menu.

Preferences - Editor

Specifies which program should be used to edit files.

See also:

- “Preferences” on page 1-8

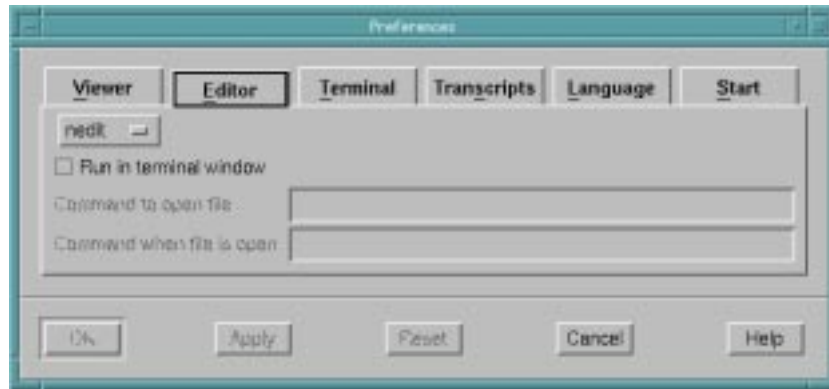


Figure 1-2. Preferences - Editor

nedit	Either the NEdit or Emacs editor can be used to edit files. Emacs will be used if it is running and has been configured for use with NightBench. Otherwise, NEdit will be used.
	See “Using the Emacs Editor - Additional Considerations” on page 1-16 for information on configuring Emacs for use with NightBench.
vi	Uses the vi(1) editor to edit files.
custom	Select this item if a file editor other than those listed in this menu is desired.

Run in terminal window

The user should check this if the program chosen as the file editor needs a terminal window in which to run. This is particularly important if the **custom** option is selected from the **Editor** menu.

When the **vi** option is chosen from the **Editor** menu, this checkbox is automatically checked because the **vi(1)** command needs a terminal window in which to run.

Also, since neither the NEdit or Emacs editors need a terminal window (they create their own window when started), this checkbox remains unchecked when the **nedit** option is selected from the **Editor** menu.

Command to open file

Enter the path and filename of the program to use as the file editor.

This field is active only when the **custom** menu item is selected from the **Editor** menu.

Command when file is open

This field is active only when the `custom` menu item is selected from the Editor menu.

Preferences - Terminal

Specifies which program should be used as a terminal emulator.

See also:

- “Preferences” on page 1-8

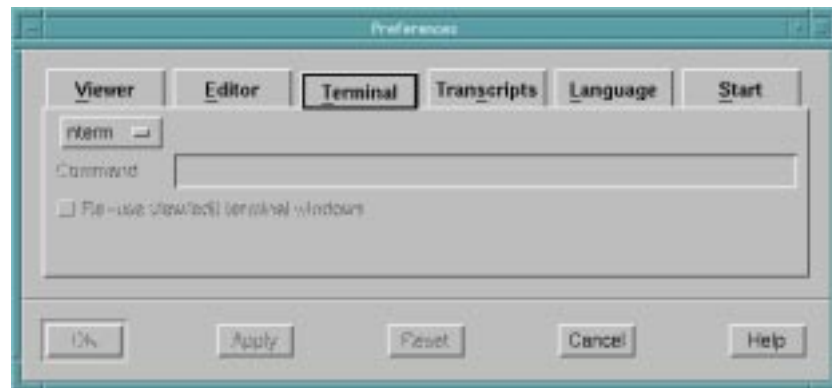


Figure 1-3. Preferences - Terminal

<code>nterm</code>	Uses the <code>nterm(1)</code> terminal emulator.
<code>xterm</code>	Uses the <code>xterm(1)</code> terminal emulator.
<code>custom</code>	Select this item if a terminal program other than those listed in this menu is desired.

Command

Enter the path and filename of the program to use as the terminal emulator. This command must accept `xterm(1)` command-line arguments.

This field is active only when the `custom` menu item is selected.

Re-use view/edit terminal windows

This option will leave the terminal window open when the user is finished viewing or editing a file, allowing it to be re-used the next time a file is viewed/edited. This will alleviate the need to create and destroy windows each time a file is opened by NightBench for this purpose.

If this option is not checked, the terminal window opened for a particular file will be destroyed when the user is finished with that file and a new terminal window will be created the next time a file is viewed or edited by NightBench.

If neither the viewer or editor has been configured to run in a terminal window (e.g. NEdit), this checkbox will be deactivated.

See also:

- “Preferences - Viewer” on page 1-8
- “Preferences - Editor” on page 1-9

Preferences - Transcripts

The following selections allow the user to specify the number of build transcripts, if any, to retain when the Builder exits the current environment and whether or not verification is required before removal of any transcripts.

The user may also remove individual transcripts manually using the Build Transcripts dialog.

See also:

- “Build Transcripts” on page 6-6
- “Preferences” on page 1-8

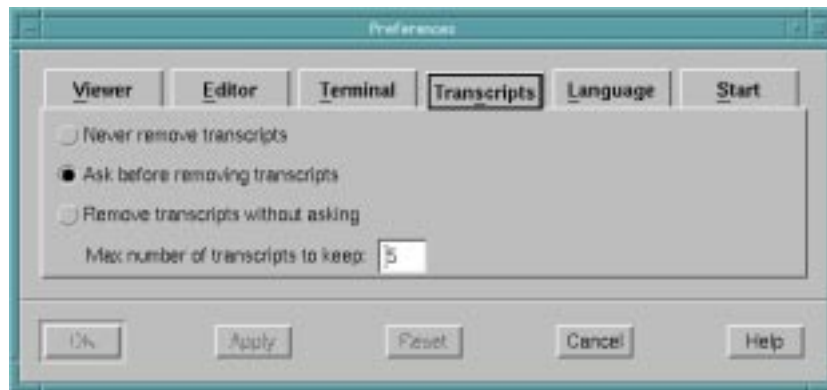


Figure 1-4. Preferences - Transcripts

Never remove transcripts

No build transcripts are removed when the Builder exits the current environment.

Ask before removing transcripts

When the Builder exits the current environment, the user is presented with a dialog verifying that the number of transcripts in excess of the specified maximum should be deleted.

Remove transcripts without asking

When the Builder exits the current environment, it removes the number of transcripts in excess of the specified maximum without verification.

Max number of transcripts to keep

This specifies the number of transcripts to retain when the Builder exits the current environment, deleting those transcripts in excess. The default is 5.

Preferences - Language

By default, when a new environment is created or an existing environment is opened, NightBench will prompt the user as to the language that will be used for development in that environment. The options on this page allow the user to specify a particular language, circumventing the query each time an environment is created or opened. The user may also choose to be prompted every time an environment is created or opened.

See also:

- NightBench Project - “New Environment” on page 4-6
- NightBench Project - “Open” on page 4-5
- NightBench Development - “New Environment” on page 5-5
- NightBench Development - “Open Environment” on page 5-8
- “Preferences” on page 1-8

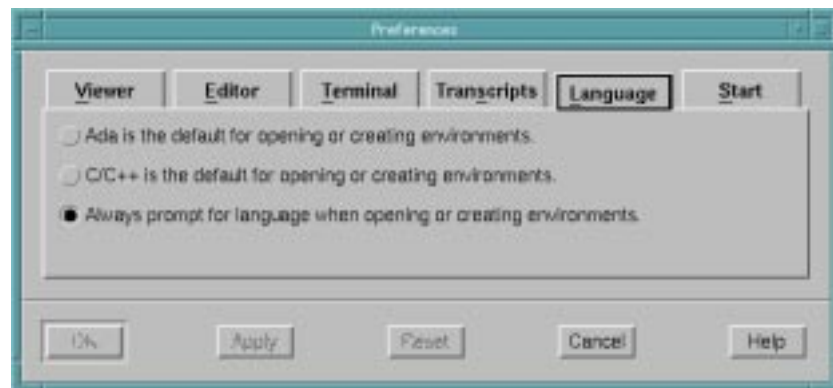


Figure 1-5. Preferences - Language

Ada is the default for opening or creating environments

When this option is selected, a NightBench Development window configured for Ada program development will be opened automatically when a new environment is created or an existing environment is opened.

C/C++ is the default for opening or creating environments

When this option is selected, a NightBench Development window configured for C or C++ program development will be opened automatically when a new environment is created or an existing environment is opened.

Always prompt for language when opening or creating environments

When this option is selected, NightBench will prompt the user as to the language that will be used for development in this environment every time a new environment is created or an existing environment is opened.

Preferences - Start

This page contains options that allow the user to configure NightBench so that a build is automatically started when the user presses the **Build** button from either the NightBench Development or NightBench Project windows.

In addition, this page also contains options that allow the user to specify that a **Command Log** window is opened automatically when either a NightBench Development or NightBench Builder window is opened.

See also:

- “Command Log” on page 1-18
- “Preferences” on page 1-8

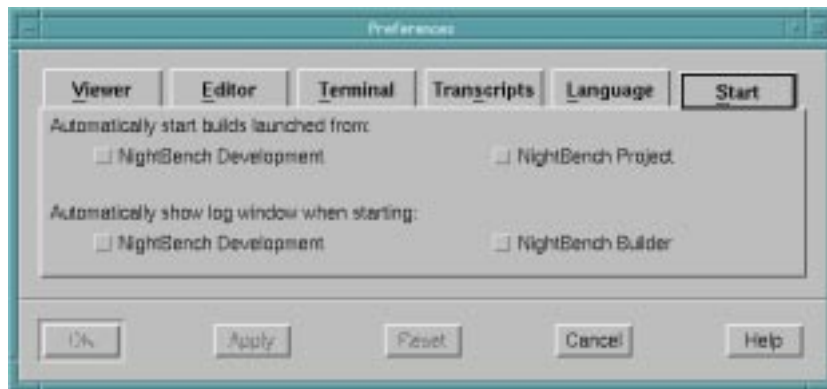


Figure 1-6. Preferences - Start

Automatically start builds launched from

By default, when the **Build** button is pressed from either the NightBench Development window or the NightBench Project window, a NightBench Builder window is opened for that environment but the build does not start until the user presses the **Start Build** button in the Builder. The **Options** menu for each of these tools has an **Automatically Start Builds** menu item that the user may use to override these preferences.

See also:

- NightBench Builder - Chapter 6
- NightBench Development - “Options Menu” on page 5-14
- NightBench Project - “Options Menu” on page 4-3

NightBench Development

When this item is selected, a build will automatically be started when the user presses the **Build** button from either the **Units** page or the **Partitions** page in the NightBench Development window.

See also:

- NightBench Development - Units page - “Build” on page 5-65
- NightBench Development (*Ada*) - Partitions page - “Build” on page 5-100
- NightBench Development (*C/C++*) - Partitions page - “Build” on page 5-150

NightBench Project

When this item is selected, a build will automatically be started when the user presses the **Build** button from the NightBench Project window.

See also:

- NightBench Project - “Build” on page 4-5

Automatically show log window when starting

The NightBench Program Development Environment provides a history window from both the NightBench Development window and the NightBench Builder window. This command log contains a list of all the commands issued by NightBench to the underlying compiler toolsets in the carrying out of its tasks.

See also:

- “Command Log” on page 1-18

NightBench Development

When this option is selected, a **Command Log** window is opened automatically when a NightBench Development window is opened.

In addition, the **Show Command Log** item on the **Options** menu of the Development tool controls the visibility of the **Command Log** window.

See also:

- “Show Command Log” on page 5-15

- NightBench Development - Chapter 5

NightBench Builder

When this option is selected, a **Command Log** window is opened automatically when a NightBench Builder window is opened.

In addition, the **Show Command Log** item on the **Options** menu of the Builder tool controls the visibility of the **Command Log** window.

See also:

- “Show Command Log” on page 6-12
- NightBench Builder - Chapter 6

Using the Emacs Editor - Additional Considerations

NightBench can use the Emacs editor to both view and edit files. However, certain preliminary setup must be done in order for NightBench to communicate with Emacs.

First, copy the file `/usr/lib/X11/kt-emacs.el` into your Emacs installation's site `elisp` directory (e.g. `/usr/tools/lag/share/emacs/site-lisp`). If desired, you can byte-compile the file for better performance.

Next, the following elisp commands need to be run in your Emacs session. You may wish to add these commands to your Emacs startup file (usually `~/.emacs`) so that Emacs will be always be ready to communicate with NightBench.

```
(require 'kt-emacs)
(ktalk-start)
```

If successful, these commands will start up the command `/usr/bin/X11/kt-emacs` which is the program that handles communication between NightBench and Emacs.

Once this is running, NightBench edit and view operations can be directed to your running Emacs session.

NOTE

If using these techniques, Emacs must have an established session running before an **Edit** or **View** command is applied to a NightBench file. Otherwise, NEdit will be launched to edit/view the desired file(s).

Resizing columns

NightBench allows the user to resize columns in the various lists that appear throughout the interface.

To adjust the columns in any of the lists presented in the NightBench Program Development Environment, simply drag the divider that appears between the columns in the heading area at the top of the list.

For example, to make the **Name** column wider on the **Units** page of the NightBench Development window, drag the divider that appears between the **Name** and **Part** column titles to the right until the desired width is achieved.

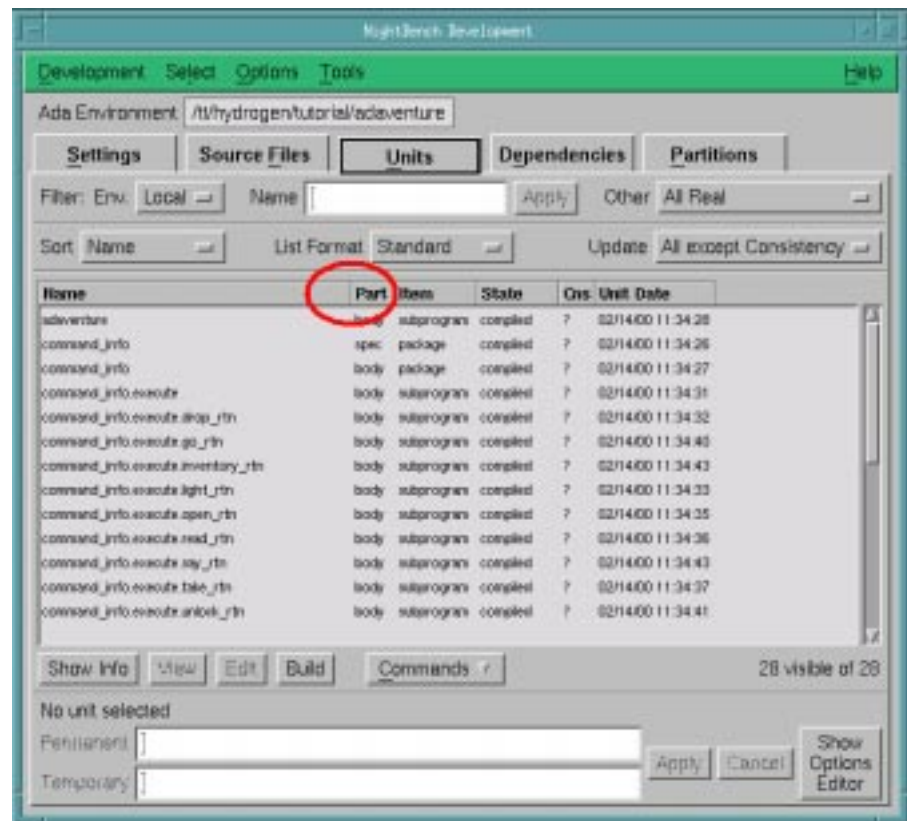


Figure 1-7. Resizing columns

Command Log

The NightBench Program Development Environment provides a history window from both the NightBench Development window and the NightBench Builder window. This command log contains a list of all the commands issued by NightBench to the underlying compiler toolsets in the carrying out of its tasks.

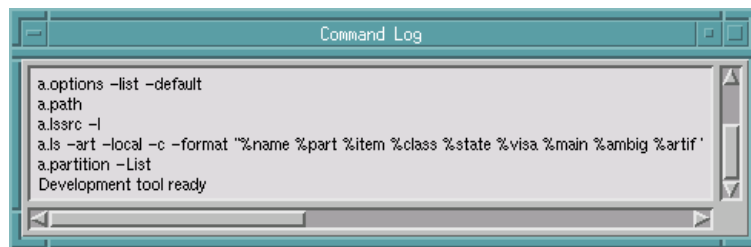


Figure 1-8. Command Log window

The Preferences dialog contains options that allow the user to specify that a Command Log window is opened automatically when either a NightBench Development or NightBench Builder window is opened.

See also:

- *MAXAda Reference Manual* (0890516)
- NightBench Development - Chapter 5
- NightBench Builder - Chapter 6
- “Preferences - Start” on page 1-14

Tools Menu

Every window in the NightBench Program Development Environment has a **TOOLS** menu on its menu bar in which to access the various NightBench components. Also, menu items are provided on the **TOOLS** menu to view or edit files.

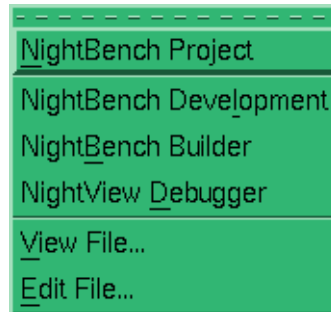


Figure 1-9. Tools menu

NightBench Project

Opens the NightBench Project window.

(This menu item does not appear on the **TOOLS** menu of the NightBench Project window.)

See also:

- NightBench Project - Chapter 4

NightBench Development

Opens the NightBench Development window.

See also:

- NightBench Development - Chapter 5

NightBench Builder

Opens the NightBench Builder window.

See also:

- NightBench Builder - Chapter 6

NightView Debugger

Opens the NightView debugger.

See also:

- *NightView User's Guide* (0890395)

View File...

Loads the specified file into the editor for viewing only. The user selects the desired file using the standard file dialog.

See also:

- “Preferences - Viewer” on page 1-8

Edit File...

Loads the specified file into the editor, allowing changes to be made to the file. The user selects the desired file using the standard file dialog.

See also:

- “Preferences - Editor” on page 1-9

Help Menu

Every window in the NightBench Program Development Environment has a **Help** menu on its menu bar.

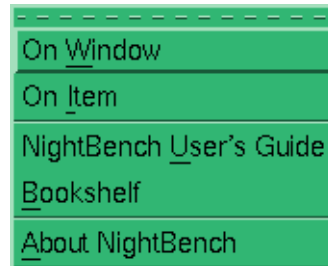


Figure 1-10. Help menu

On Window

Opens the help topic for the current window.

On Item

Gives context-sensitive help on the various menu options, dialogs, or other parts of the user interface.

Help for a particular item is obtained by first choosing this menu option, then clicking the mouse pointer on the object for which help is desired (the mouse pointer will become a floating question mark when the **On Item** menu item is selected).

In addition, context-sensitive help may be obtained for the currently highlighted option by pressing the F1 key. The HyperHelp viewer will open with the appropriate topic displayed.

NightBench User's Guide

Opens the online *NightBench User's Guide*.

Bookshelf

Opens a HyperHelp window that lists all of the currently available HyperHelp publications.

About NightBench

Displays version and copyright information for the NightBench product.

Using NightBench with Ada

As a starting point into the world of NightBench, an example will be given. This example will traverse step-by-step through the various NightBench windows and dialogs, from creating an environment, to introducing units and managing source files, defining a partition, and finally, building that partition.

It will go over some of NightBench's error handling capabilities and present some of the dialogs that the user will encounter.

In addition, it will present the user with more detailed information about some NightBench-specific topics such as the Environment Search Path and the Options Editors.

Starting NightBench

We will begin by opening the NightBench Project window, our launching pad for this example. From the command line, type the following command.

```
$ nbench
```

Screen 2-1. Starting NightBench

Note that we have not provided **nbench** with any parameters, indicating that we want to open the NightBench Project window. **nbench** accepts a number of command line options, allowing the user to open a particular NightBench component or to provide start-up information to NightBench. See "Starting NightBench" on page 1-3 for more information about these options.

The NightBench Project window will appear, listing the environments with which it has interacted. If no other environments have been created under NightBench, this list will be empty. See Chapter 4 for more information about the NightBench Project window.

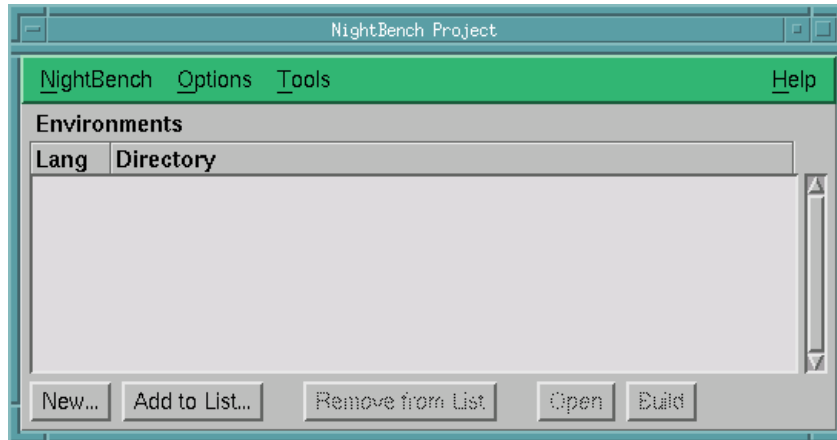


Figure 2-1. NightBench Project

Creating a new environment

One of the first steps we must take in order to use NightBench for Ada program development is to create an *environment*. Environments are used as the basic structure of organization within the NightBench Program Development Environment.

To create a new environment from the NightBench Project window:

- On the NightBench Project window, press the button marked **New...** so we can create our new environment. This will open the New Environment Language Selection dialog. Select the language that will be used in this environment (Ada). Press the **Next>** button.

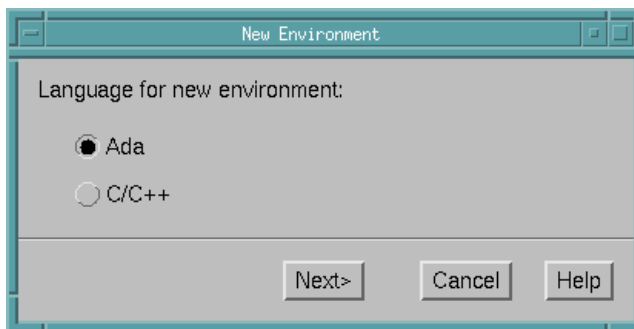


Figure 2-2. Selecting the language

- The next dialog presented is the New Environment Directory Selection dialog:

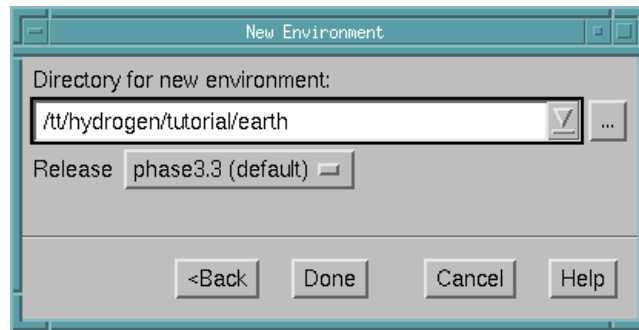


Figure 2-3. Selecting the directory

- Type the directory name in the **Directory for new environment** field where you want NightBench to create the new environment. This can be the name of an existing directory or NightBench can create the directory for you. (Note that NightBench can only create a subdirectory of an existing directory.) The directory in which our new environment is to be created will be called **earth** and shall be created under the existing directory **/tt/hydrogen/tutorial**.
- You may also select a **Release** if you have more than one release of MAXAda installed on your system. If you have only one release of MAXAda installed on your system, it will appear here.
- Press **Done**.

This will add the new environment to the list of **Environments** in the NightBench Project window. NightBench will also open the new environment in its own NightBench Development window.

See also:

- “New Environment” on page 4-6
- NightBench Development - Chapter 5

Introducing units into the environment

Our next step is to populate the environment with *units*. Units are the basic building blocks for Ada programs in NightBench. They are contained within source files and it is through these source files that they are introduced into their intended environments.

Source files may already have been created outside the NightBench environment or you may use the editing features of NightBench to create a new file. For this example, we will assume we have already created a file named **world.a** and that it contains one unit, **hello**. This file appears in Screen 2-2. (The source file is just an ordinary text file.)

```
with ada.text_io;  
procedure hello is  
begin  
    ada.text_io.put_line ("Hello World!!!");  
end hello;
```

Screen 2-2. Source file world.a containing hello unit

To introduce an existing source file into an environment:

- Click on the Source Files tab of the NightBench Development window.
- Press the Introduce/Create... button. This will open the Introduce Source Files dialog so we can introduce our source files (and the units contained within) into the new environment.

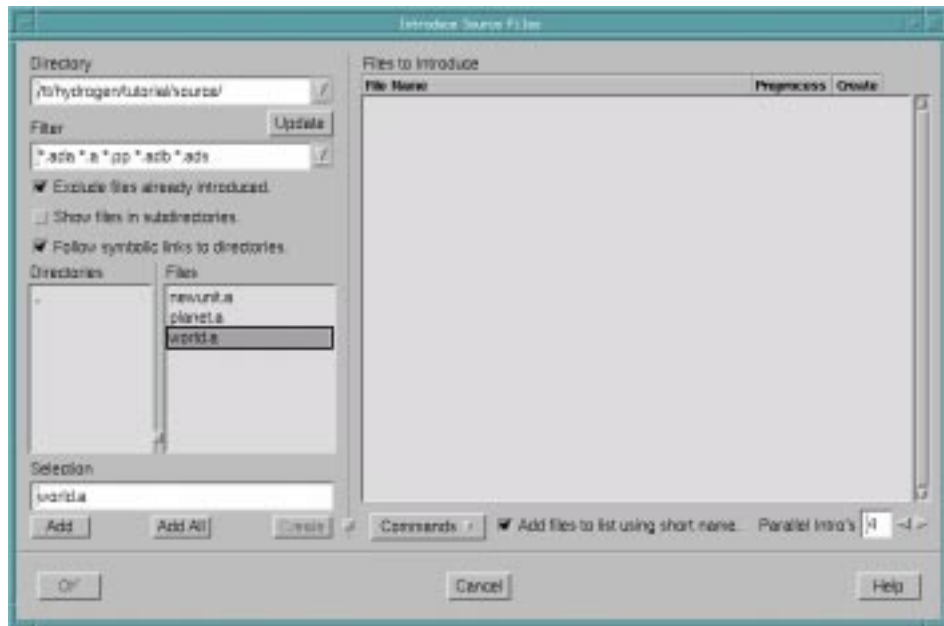


Figure 2-4. Introducing a source file

- Maneuver to the directory in which the **world.a** source file is contained. You may type the path to the directory name in the **Directory** field or use the entries in the **Directories** list to navigate to the desired directory.
- When you have arrived at the desired directory and the **world.a** source file name appears in the **Files** list, select it by clicking the name with the mouse. It will then appear in the **Selection** field.

- Press the **Add** button to add this file to the list of **Files to Introduce**. (You may introduce any number of files by adding them here but for our example we will just introduce this one file.)
- Press the **OK** button to introduce the source file into the environment.

The unit `hello` that was contained in the source file `world.a` is now a part of the environment `earth`.

The source file now appears in the list of files on the **Source Files** page of the Night-Bench Development window and the unit contained within now appears on the **Units** page.

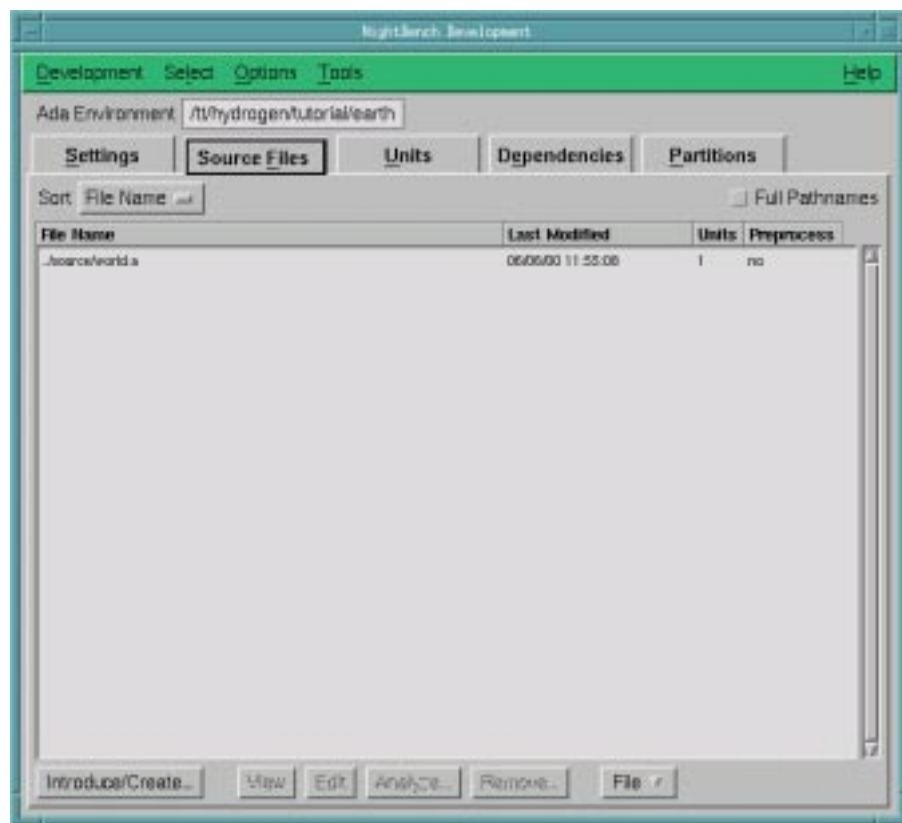


Figure 2-5. Source file, `world.a` - newly introduced

See also:

- “Development - Source Files” on page 5-31
- “Introduce Source Files” on page 5-37

Defining a partition

In order to use the units introduced into NightBench, we must include them in a *partition*. NightBench defines three types of Ada partitions:

- *active*
- *archive*
- *shared object*

For our example, we want to include our `hello` unit in an executable program so we will be defining an *active* partition. Although we may name this partition anything we want, we will name this particular partition **hello**, the same as the unit which will function as our *main subprogram*.

To define a partition in the environment:

- Click on the **Partitions** tab of the NightBench Development window to get to the **Partitions** page.
- Press the **Create...** button. This will open the Create Partition dialog so we can define a partition in the new environment.
 - Type the name `hello` in the **Name** field.
 - Verify that the **active** radiobutton is selected for the **Kind of partition** we are creating.
 - Press **OK**.

See also:

- “Create Partition” on page 5-103

Including units in the partition

Once the partition is defined, we may add the units we want to be included in the partition.

We do this on the **Units** subpage of the **Partitions** page. Click on the **Units** tab on the **Partitions** page to get to this area.

You will see the figure shown in Figure 2-6:

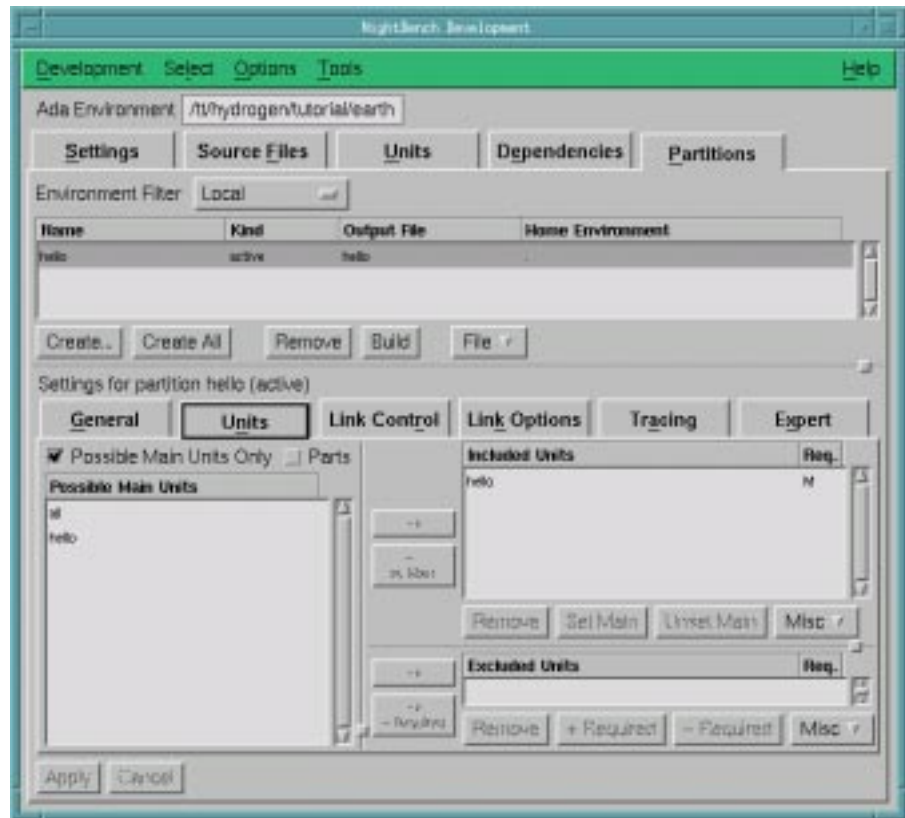


Figure 2-6. Including units in a newly defined partition

As you can see in Figure 2-6, the unit `hello` has been automatically added to the list of Included Units for the `hello` partition.

NOTE

When the partition has the same name as a library subprogram in the environment, that subprogram is assumed to be the *main unit*. (as indicated by the M under the Req. column in the list of Included Units. Otherwise, no main subprogram is assumed and must be manually included.

See also:

- “Partitions - Units” on page 5-113

Building a partition

At this point, we have an environment, **earth**, that has within it the definition for the partition, **hello**, made up of a main unit, `hello`, that was introduced from the source file, `world.a`.

We can now build this partition. We do this using the NightBench Builder.

To build the partition:

- In the list of partitions on the **Partitions** page, make sure the partition **hello** is selected.
- Press the button marked **Build**. This will open the NightBench Builder window so we can build our new partition.

In Figure 2-7, you will see that partition **hello** has been automatically entered in the **Targets** field on the **Build** page. This is because it was selected on the **Partitions** page when the **Build** button was pressed.

- Press **Start Build**.

The **Build Progress** bar shows the number of actions (compilations and links) left to perform in the current build as the **Transcript** window details each step taken during the build.

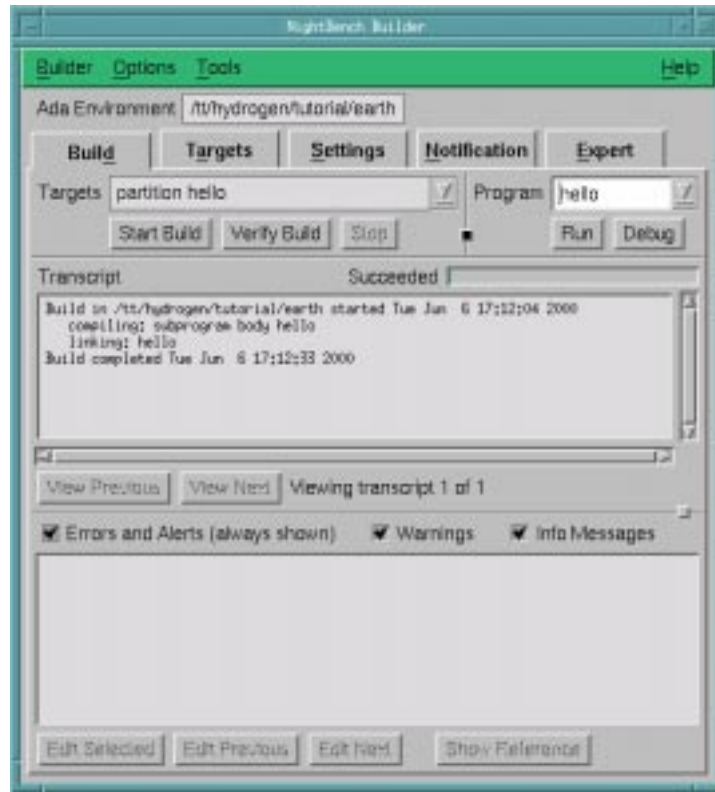


Figure 2-7. Builder window - Build page for hello partition

When the build is completed, a Build Completed dialog notifies the user.

NOTE

The notification operations can be changed on the Notification page (see “Builder - Notification” on page 6-41).

See also:

- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13

Running the program

All that’s left to do is run the program. This is easily accomplished in the NightBench Builder window.

To run the program:

- On the **Build** page of the NightBench Builder window in the **earth** environment, make sure the program name, **hello**, appears correctly in the **Run Context** field. (By default, a *run context* is created for every active partition.)
- Press the button marked **Run**.

This will open a terminal window so our executable can run. (The type of terminal window used is configured on the **Terminal** page of the **Preferences** dialog which may be accessed from the **Options** menu - see “Preferences - Terminal” on page 1-11.)

You should see the following output:

```
[running hello]

Hello World!!!

[program exited with status 0]
```

- Close the terminal window when you are finished viewing the output.

See also:

- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13

What options do I have?

NightBench uses the concept of persistent compile options. These options are specified through the Options Editors and are “remembered” at compilation time. They can apply to any of three areas: *environment-wide compile options* (which apply to all units within the environment), *permanent unit compile options* and *temporary unit compile options* (both of which apply and are unique to specific units).

Let’s manipulate the options in our example to get an idea of how these options work.

Setting environment-wide compile options

First, we will consider the *environment-wide compile options*. These apply to all units within the environment. Since we only have one unit right now, it will apply to that. However, if we add any other units later, they will “inherit” these options automatically.

To change the environment-wide compile options:

- Click on the Settings tab of the NightBench Development window.

You’ll see that the Default Compile Options field is empty. That’s because we haven’t set anything yet. So let’s set them to something and see what happens.

- Click on the button marked Show Options Editor. This brings up the Environment Compile Options Editor.

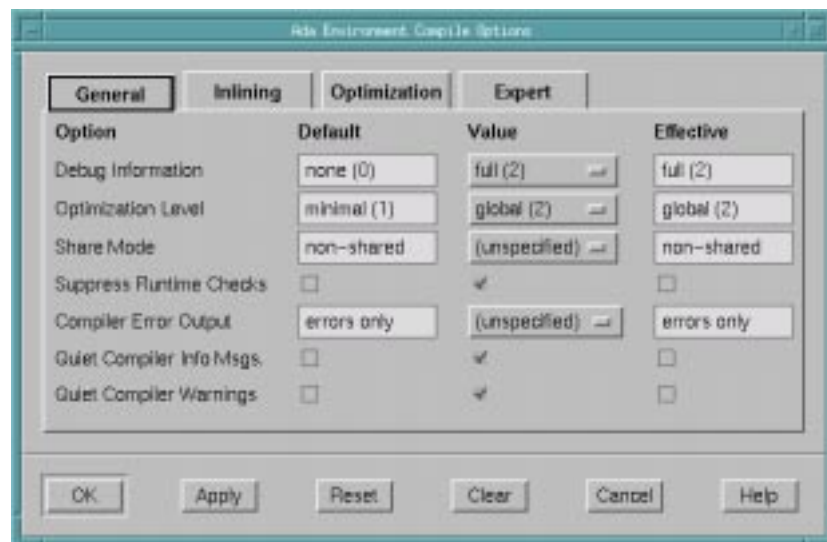


Figure 2-8. Ada Environment Compile Options Editor

- Click on the General tab to get to the Ada Environment Compile Options Editor - General page.

- Change the Value of Debug Information to full (2) (using the drop-down list).
- Change the Value of Optimization Level to global (2) (using the drop-down list).
- Press OK to apply the changes and close the editor dialog.

You will see `-O -g` in the Default Compile Options field, corresponding to the changes we just made.

Remember, these options apply to all units in the environment and will be “inherited” by any units we add to this environment.

See also:

- Options Editors - Chapter 7
- “Environment Compile Options Editor” on page 7-3

Setting unit compile options

If we'd like to set particular options for a specific unit, we can use the *permanent unit compile options* for that unit. They're set in much the same way as *environment-wide compile options*, except that we need to specify the units to which they apply.

To change the permanent unit compile options:

- Click on the Units tab of the NightBench Development window.
- Select the unit `hello` from the list of units.

You'll see that the Permanent field is empty. That's because we haven't set anything yet. So let's set these options and see what happens.

- Click on the button marked Show Options Editor. This brings up the Ada Unit Compile Options Editor.

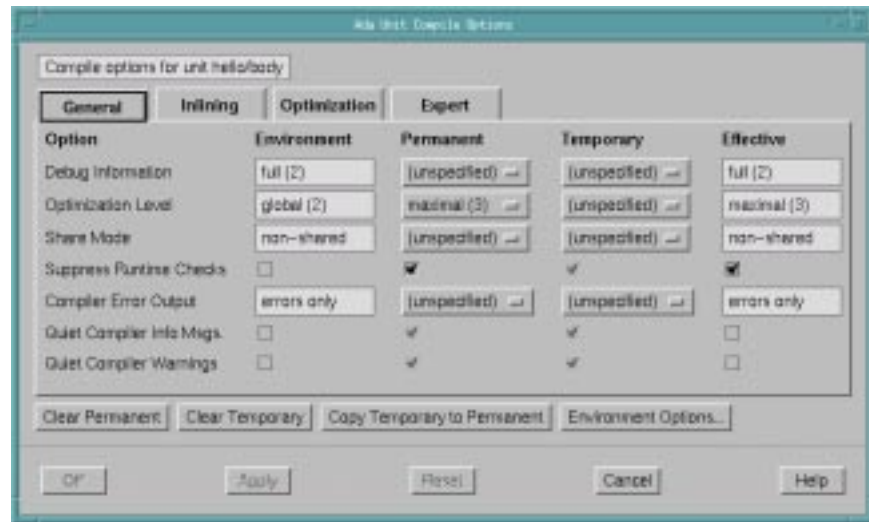


Figure 2-9. Ada Unit Compile Options Editor

- Click on the **General** tab to get to the Ada Unit Compile Options Editor - General page.
- Change the **Permanent** value of **Optimization Level** to **maximal (3)** (using the drop-down list).
- Change the **Permanent** value of **Suppress Runtime Checks** so that a solid checkmark appears in the box. (See “Tri-state Checkboxes” on page 7-1 for more information.)
- Press **Apply** so that the changes take effect but the Ada Unit Compile Options Editor remains open.

We may decide that in addition to the specified options, we may want to “try out” some options or change particular options for a specific compilation but only “temporarily”. The *temporary unit compile options* are a separate set of options maintained for this purpose. They are persistent in the same way that permanent options are persistent, however, they may be cleared easily without altering the permanent options.

Let’s suppose we don’t want to produce any debug information for our `hello` unit for this particular compilation. We can set a temporary compile option for that.

To change the temporary unit compile options:

- Change the **Temporary** value of **Debug Information** to **none (0)** (using the drop-down list).
- Press **Apply** so that the changes take effect but the Ada Unit Compile Options Editor remains open.

In addition, let’s assume that for this compilation, we only want minimal optimization for the `hello` unit. Set that option as well.

- Change the Temporary value of Optimization Label to minimal (1) (using the drop-down list).
- Press Apply so that the changes take effect but the Ada Unit Compile Options Editor remains open.

See also:

- Options Editors - Chapter 7
- “Ada Unit Compile Options Editor” on page 7-24

Determining the effective options

The *environment-wide compile options*, *permanent unit compile options*, and *temporary unit compile options* have a hierarchical relationship to one another which means the environment-wide compile options are overridden by the permanent unit options, which are, in turn, overridden by the temporary unit options. This relationship forms the *effective compile options* for the unit, which the compiler will use during compilation.

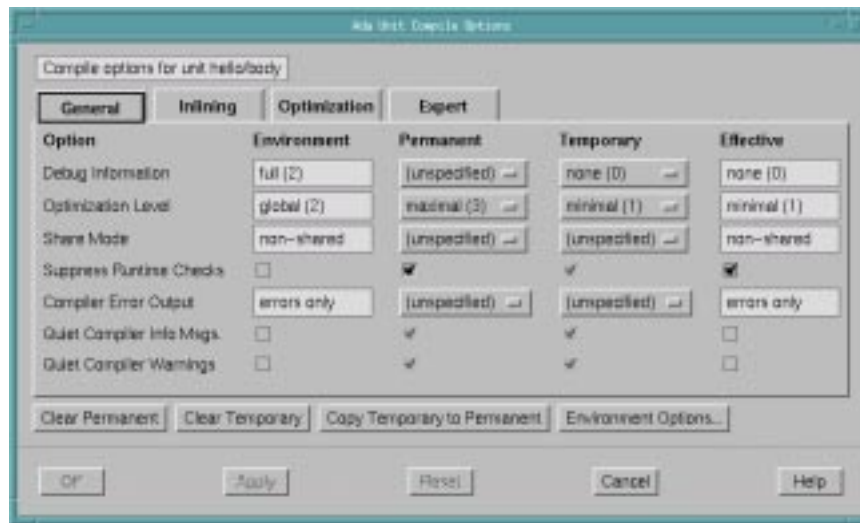


Figure 2-10. Ada Unit Compile Options Editor - Effective Options

As you can see in Figure 2-10, the effective options are derived from the environment, permanent, and temporary compile options. In this example, the Effective value for Debug Information comes from the Temporary value set which takes precedence over the Environment and Permanent compile options. In the case of Suppress Runtime Checks, since the Temporary value is unspecified, the Effective value comes from the Permanent value. If neither the Permanent nor the Temporary value is set, the Effective option comes from the Environment value, as is the case with the Effective value for Compiler Error Output.

The effective values may change if the environment, permanent, or temporary option values change. We can see this if we change the Temporary value of Debug Information.

Deleting a temporary unit compile option:

- Change the **Temporary** value of **Debug Information** to **(unspecified)** (using the drop-down list).
- Press **Apply** so that the changes take effect but the **Unit Compile Options Editor** remains open.

You can see that the **Effective** value for **Debug Information** has changed to **full (2)**, the **Environment** value for this option.

See also:

- Options Editors - Chapter 7
- “Unit Compile Options Editor” on page 7-24

Copying the temporary options to the permanent set

If we like the other temporary options so much that we’d like to make them permanent, NightBench provides the **Copy Temporary to Permanent** button to propagate all the temporary options for a particular unit to the permanent option set for that same unit.

Propagating the temporary options to the permanent set:

- Press the **Copy Temporary to Permanent** button on the **Ada Unit Compile Options Editor**.

All values specified as **Temporary** options have been propagated to the **Permanent** options set and reset to unspecified values. In our example, the **minimal (1)** value specified as a **Temporary** option for **Optimization Level** has become a **Permanent** option.

- Press **OK** to apply the changes and close the **Ada Unit Compile Options Editor**.

See also:

- Options Editors - Chapter 7
- “Unit Compile Options Editor” on page 7-24

Hello Galaxy - The Example Continues...

Setting up another environment

Let's set up another environment with a function which our `hello` unit can call. The last time we created an environment, we did so from the NightBench Project window. This time, we will create it from the NightBench Development window.

To create a new environment from the NightBench Development window:

- From the **Development** menu of the NightBench Development window, select the item marked **New Environment...** This will open the New Environment Language Selection dialog so we can create our new environment. Select the language that will be used in this environment (**Ada**). Press the **Next>** button.
- The next dialog presented is the New Environment Directory Selection dialog:
 - Type the directory name in the **Directory for new environment** field where you want NightBench to create the new environment. This can be the name of an existing directory or NightBench can create the directory for you. (Note that NightBench can only create a subdirectory of an existing directory.) The directory in which our new environment is to be created will be called **galaxy** and shall be created under the existing directory **/tt/hydrogen/tutorial**.
 - You may also select a **Release** if you have more than one release of MAXAda installed on your system. If you have only one release of MAXAda installed on your system, it will appear here.
 - This dialog also gives the user the choice to open the new environment in the current NightBench Development window. For our example, we *uncheck* the **Open in current window** option so that our new environment will be opened in its own NightBench Development window.
 - Press **Done**.

This will add the new environment to the list of **Environments** in the NightBench Project window. NightBench will also open the new environment in its own NightBench Development window.

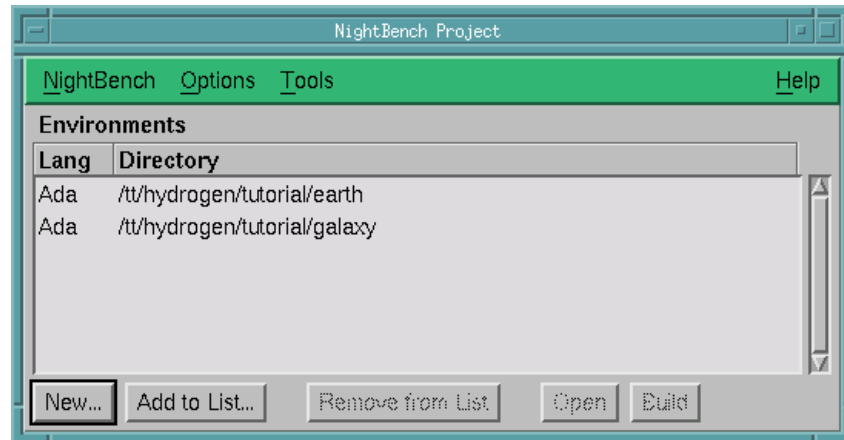


Figure 2-11. NightBench Project

See also:

- “New Environment” on page 5-5
- NightBench Development - Chapter 5

Introducing units into the galaxy

Let’s create a unit in our new environment. Suppose this unit does not already exist in a source file as in the earlier example. You can use NightBench to create a new source file and have it introduced automatically. Furthermore, NightBench will analyze the file before it introduces it to determine the units contained within and their dependencies. If NightBench finds any problems during this analysis, it will notify the user before proceeding. This next section will demonstrate that very situation.

To create a new unit and introduce it into the environment:

- Click on the **Source Files** tab of the NightBench Development window containing the `/tt/hydrogen/tutorial/galaxy` environment.
- Press the **Introduce/Create...** button. This will open the **Introduce Source Files** dialog so we can create our source file and introduce it into the environment.
 - Type in the name of the source file in the **Selection** field. We will name this source file `planet.a`.
 - Press the **Create** button. The file will be added to the list of **Files to Introduce**.
 - Press **OK** to close the **Introduce Source Files** dialog.
- The program configured as the file editor for NightBench (see “Preferences - Editor” on page 1-9) will be opened. Enter the following text into the editor and save the file:

(Note the mistake in the last line. This was intentional in order to show NightBench's error handling capabilities.)

```
with ada.text_io;  
procedure alien is  
begin  
  ada.text_io.put_line("Greetings from Outer Space!!!");  
end mistake
```

Screen 2-3. Source file `planet.a` containing alien unit

- Close the file in the editor or exit the editor completely.
- The **Analyze Errors** dialog is presented showing the error on the last line of `planet.a`. NightBench analyzes any file that is introduced into an Ada environment and if errors are found that will prohibit NightBench from introducing the file, it raises this dialog to allow the user to fix the errors before continuing.

In our example, the last line of the file:

```
end mistake
```

does not have a semi-colon. Because of this, NightBench will not be able determine the unit contained within this source file. It presents the **Analyze Errors** dialog so that we can fix the mistake.



Figure 2-12. Analyze Errors dialog - mistake in unit being introduced

- Select the error by pressing the **Edit Next** button (or you may click on the error itself). This will bring up the file editor configured in the **Preferences** dialog (see “Preferences - Editor” on page 1-9). The editor will be positioned at the location of the error.
- Repair the last line by replacing:


```
end mistake
```

 with


```
end alien;
```

 and save the file.
- Close the file in the editor or exit the editor completely.
- Press **OK** on the **Analyze Errors** dialog.

The unit **alien** that was contained in the source file **planet.a** will now be introduced into the environment **galaxy**.

The source file now appears in the list of files on the **Source Files** page of the NightBench Development window and the unit contained within now appears on the **Units** page of the NightBench Development window.

NOTE

We have not compiled this unit nor have we created a partition and included the unit in the partition to be built. This was intentional to demonstrate a point later in the example.

See also:

- “Preferences” on page 1-8
- “Analyze Errors” on page 5-53

Modifying an existing unit

Let's go back to the NightBench Development window that contains our **earth** environment.

We will now update the unit `hello` so that it references the new `alien` unit.

To modify an existing source file within an environment:

- Click on the **Units** tab of the NightBench Development window that contains the **earth** environment.
- Select the unit `hello` by clicking on it.
- Press the **Edit** button. This brings up the source file containing the unit `hello` into the file editor configured in the **Preferences** dialog (see “Preferences - Editor” on page 1-9).
- Make the following changes to the `hello` unit (the changes appear in **bold**).

```
with ada.text_io ;  
with alien;  
procedure hello is  
begin  
    ada.text_io.put_line ("Hello World!!!") ;  
    alien;  
end hello ;
```

Screen 2-4. Reference the `alien` unit within the `hello` unit

and save the changes to the file.

- Close the file in the editor or exit the editor completely.

See also:

- “Development - Units” on page 5-61

- “Preferences” on page 1-8

Building a partition with references outside the local environment

Now let’s try to build it.

Building a partition with references outside the local environment:

- Select the **NightBench Builder** menu item from the **Tools** menu of the NightBench Development window that contains the **earth** environment.

This will open the NightBench Builder window that we were using earlier.

- Press **Start Build**.

When the build is completed, a **Build Completed** dialog notifies the user.

NOTE

The notification operations can be changed on the **Notification** page (see “Builder - Notification” on page 6-41).

The **Transcript** and **Errors** windows will show the errors encountered during the most recent build.

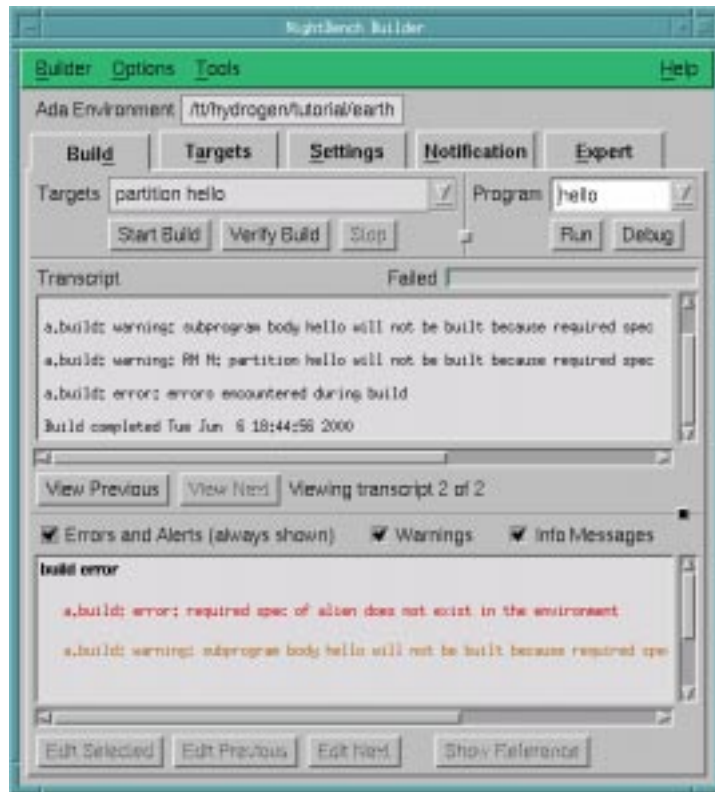


Figure 2-13. Builder window - Build page for hello partition with errors

Because the `alien` unit does not exist in the current environment AND because we have not manually added it to our *Environment Search Path*, the Builder cannot find it and therefore issues the following error:

```
a.build: error: required spec of alien does not exist in
the environment
```

This is easily remedied by adding the new environment's path to the Environment Search Path for the current environment.

See also:

- "Development - Partitions - (Ada)" on page 5-96
- NightBench Builder - Chapter 6
- "Builder - Build" on page 6-13

Adding an environment to the Environment Search Path

To add an environment to the Environment Search Path:

- Click on the **Settings** tab of the NightBench Development window that contains the **earth** environment.

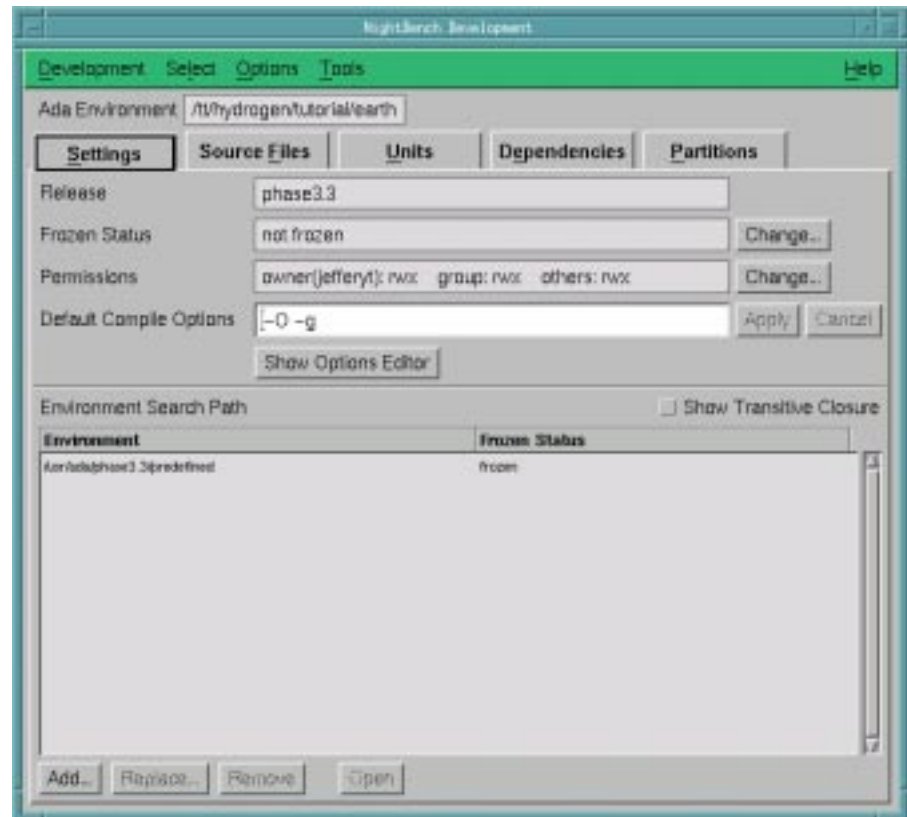


Figure 2-14. Environment Search Path for the earth environment

Every NightBench environment has an *Environment Search Path* associated with it. The Environment Search Path is your gateway to other environments. Upon creation of an Ada environment, NightBench defines the Environment Search Path so that you have access to the Predefined Language Environment, as specified in Annex A of the *Ada 95 Reference Manual*.

If you look at the Environment Search Path for the **earth** environment, you will see the path to the **predefined** environment.

NOTE

The Environment Search Path was the mechanism that made `ada.text_io` visible to the unit `hello`.

Using the Environment Search Path, you can use units that exist in foreign environments. All you need to do is add the foreign environment's path to your Environment Search Path. It's as simple as that!

And that is precisely how we will make the `alien` unit visible to our `hello` unit.

- Press the **Add...** button. This brings up the **Add Environment Search Path Element** dialog (see “Add Environment Search Path Element” on page 5-28).
 - In the **Environment pathname or keyword to add:** field, type in the directory name of the environment you want added to the Environment Search Path. You can manually type in the pathname to the desired directory or you may maneuver to it using the **...** button. We will add the pathname `/tt/hydrogen/tutorial/galaxy`.
 - Select the radiobutton where you want the environment to be added to the Environment Search Path.

We will select **At end of search path** (the default) for this option.

- Press **OK**.

You will see the `galaxy` environment added to the Environment Search Path for the `earth` environment on the **Settings** page.

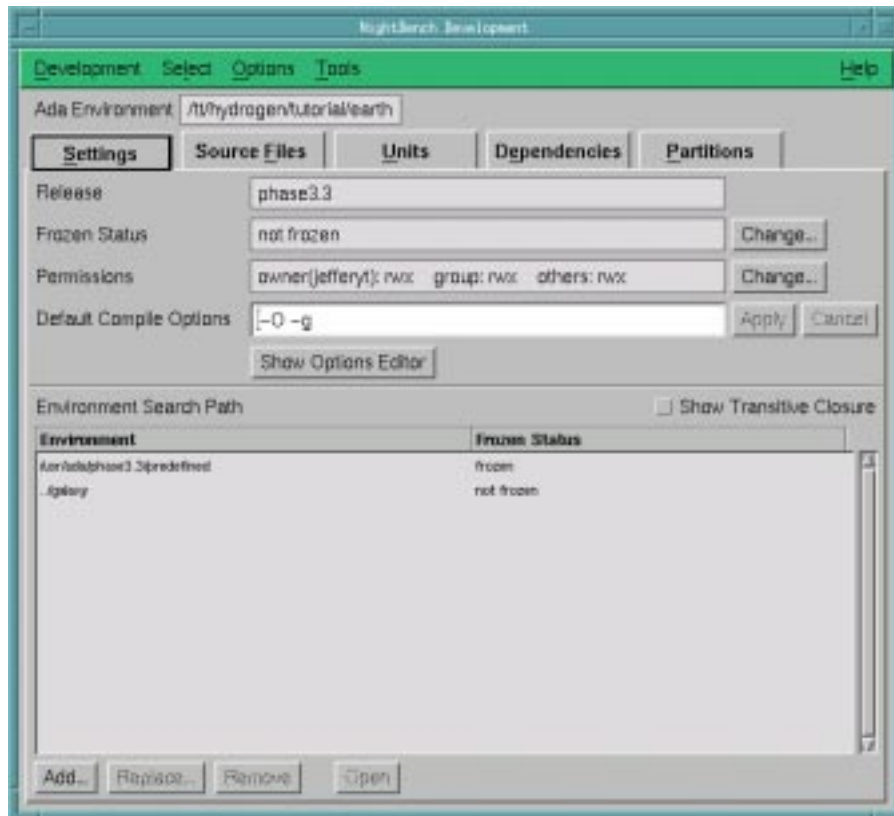


Figure 2-15. Updated Environment Search Path for the `earth` environment

See also:

- “Development - Settings” on page 5-18
- *Ada 95 Reference Manual*

Making contact!!!

Let’s try to build it again.

Building a partition with references on the Environment Search Path:

- Select the NightBench Builder menu item from the Tools menu of the NightBench Development window that contains the **earth** environment.

This will open the NightBench Builder window that we were using earlier.

- Press Start Build.

You will notice that the subprogram body of `alien` and the subprogram body of `earth` are compiled and linked together.

When the build is completed, a Build Completed dialog notifies the user.

NOTE

The notification operations can be changed on the Notification page (see “Builder - Notification” on page 6-41).

To run the program:

- On the Build page of the NightBench Builder window in the **earth** environment, make sure the program name, **hello**, appears correctly in the Run Context field. (By default, a *run context* is created for every active partition.)
- Press the button marked Run.

This will open a terminal window so our executable can run. (The type of terminal window used is configured on the Terminal page of the Preferences dialog which may be accessed from the Options menu - see “Preferences - Terminal” on page 1-11.)

You should see the following output:

```
[running hello]
```

```
Hello World!!!  
Greetings from Outer Space!!!
```

```
[program exited with status 0]
```

- Close the terminal window when you are finished viewing the output.

See also:

- “Development - Partitions - (Ada)” on page 5-96
- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13
- “Preferences” on page 1-8

Who resides on earth?

The **Units** page offers different formats in which to view the units which reside in an environment. The format of the list of units can be changed by choosing a different format from the **List Format** menu.

Let's look at the “nationality” of the units:

Listing the visa for each unit in our environment:

- Click on the **Units** tab of the NightBench Development window that contains the **earth** environment. We can see that there are two units in the environment, **alien** and **hello**.
- Choose **Visa** from the **List Format** menu.

You will see the following:

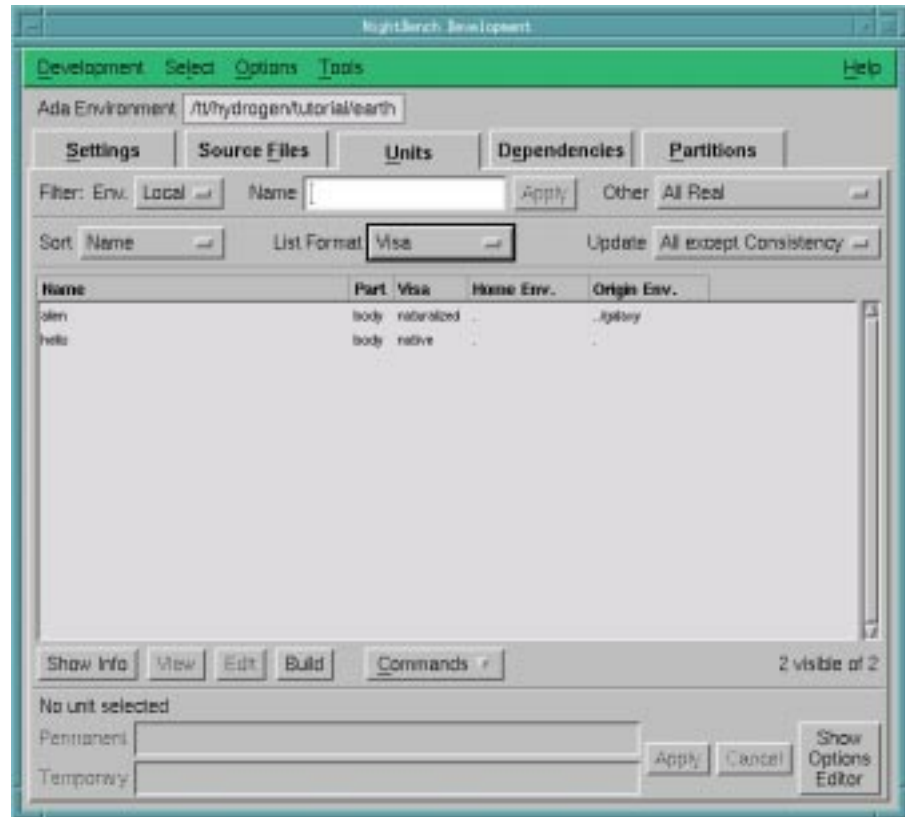


Figure 2-16. Units page with Visa List Format

- The unit `hello` was introduced directly into the environment, therefore it is regarded as a *native* unit.
- The unit `alien`, however, was never formally introduced into the current environment. It was found on the *Environment Search Path*.

Now, remember that the `alien` unit was not compiled in its original foreign environment. The NightBench Builder, when run in this local environment (**earth**), could not find a compiled form of the `alien` unit on the Environment Search Path and had to do something in order to build the partition. It therefore compiled the `alien` unit in the local environment.

This compiled form of a foreign unit within the local environment is considered *naturalized* by the system.

NOTE

If the `alien` unit had been compiled in its own foreign environment, NightBench Builder would have found that compiled form on the Environment Search Path and would have used that when linking the `hello` executable together. In that case, the Units page would have only shown the local unit `hello` as before.

Hello Again - Ambiguous Units

Let's see what happens when we introduce a unit having the same name as one already introduced into our environment.

We'll create a source file, `newunit.a`, in our `earth` environment, containing a unit named `hello`:

To create an ambiguous unit and introduce it into the environment:

- Click on the **Source Files** tab of the NightBench Development window for the `earth` environment.
- Press the **Introduce/Create...** button. This will open the **Introduce Source Files** dialog so we can create our source file and introduce it into the environment.
 - Type in the name of the source file in the **Selection** field. We will name this source file `newunit.a`.
 - Press the **Create** button. The file will be added to the list of **Files to Introduce**.
 - Press **OK** to close the **Introduce Source Files** dialog.
- The program configured as the file editor for NightBench (see "Preferences - Editor" on page 1-9) will be opened. Enter the following text into the editor and save the file:

(Note that this procedure is named `hello`, the same name as a unit already introduced into the `earth` environment. This was intentional in order to show NightBench's error handling capabilities.)

```
with ada.text_io ;
procedure hello is
begin
  ada.text_io.put_line ("I am a new unit - Hello!!!") ;
end hello ;
```

Screen 2-5. Source file `newunit.a` containing ambiguous `hello` unit

- Close the file in the editor or exit the editor completely.
- The **Analyze Errors** dialog is presented showing the ambiguity (see "Analyze Errors" on page 5-53). NightBench analyzes any file that is introduced into an Ada environment and if errors are found that will prohibit NightBench from introducing the file, it raises this dialog to allow the user to fix the errors before continuing.

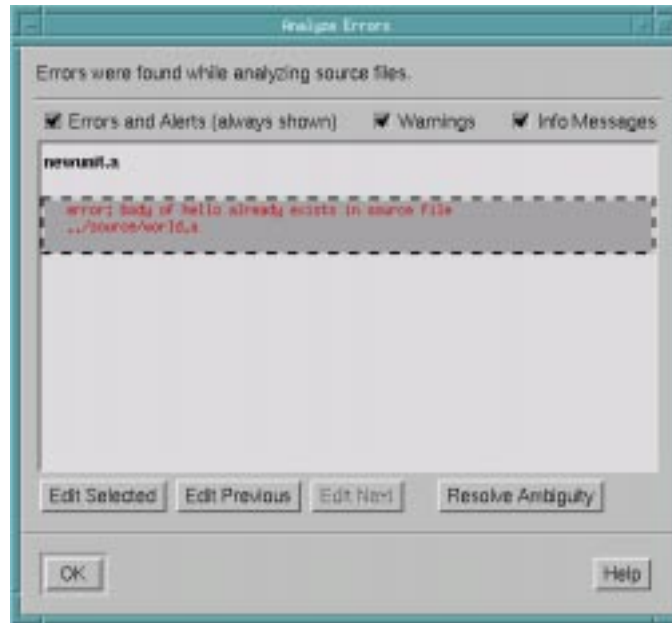


Figure 2-17. Analyze Errors dialog - ambiguous units

This error occurs because the new unit appears within another source file owned by the environment. This is easily resolved using the **Resolve Ambiguity** dialog (see “Resolve Ambiguity” on page 5-74).

- Select the error by clicking on the error itself.
- Press the **Resolve Ambiguity** button. This will bring up the **Resolve Ambiguity** dialog. This lists the files in which the ambiguous unit appears and allows the user to choose which source file NightBench should use for the unit.

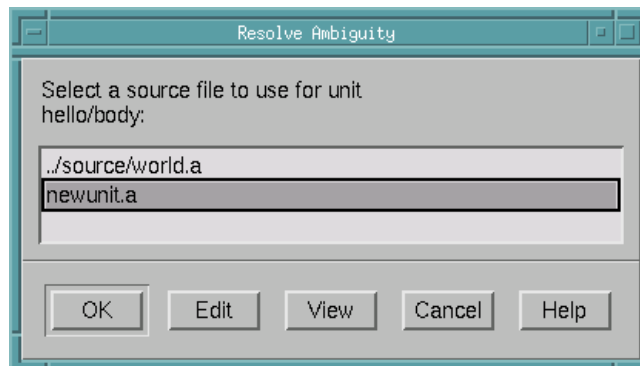


Figure 2-18. Resolve Ambiguity dialog

- Select the source file named **newunit.a**.
- Press **OK** on the **Resolve Ambiguity** dialog.

- Press OK on the Analyze Errors dialog.

The new unit `hello` that was contained in the source file `newunit.a` is now a part of the environment `earth`. The old unit `hello` that was contained in the source file `world.a` is now hidden from the `earth` environment. It can be unhidden at any time (although this will raise the same ambiguous unit situation that we just resolved) using the `Unhide...` selection from the `Commands` menu (see “Hidden Units” on page 5-72 for more information).

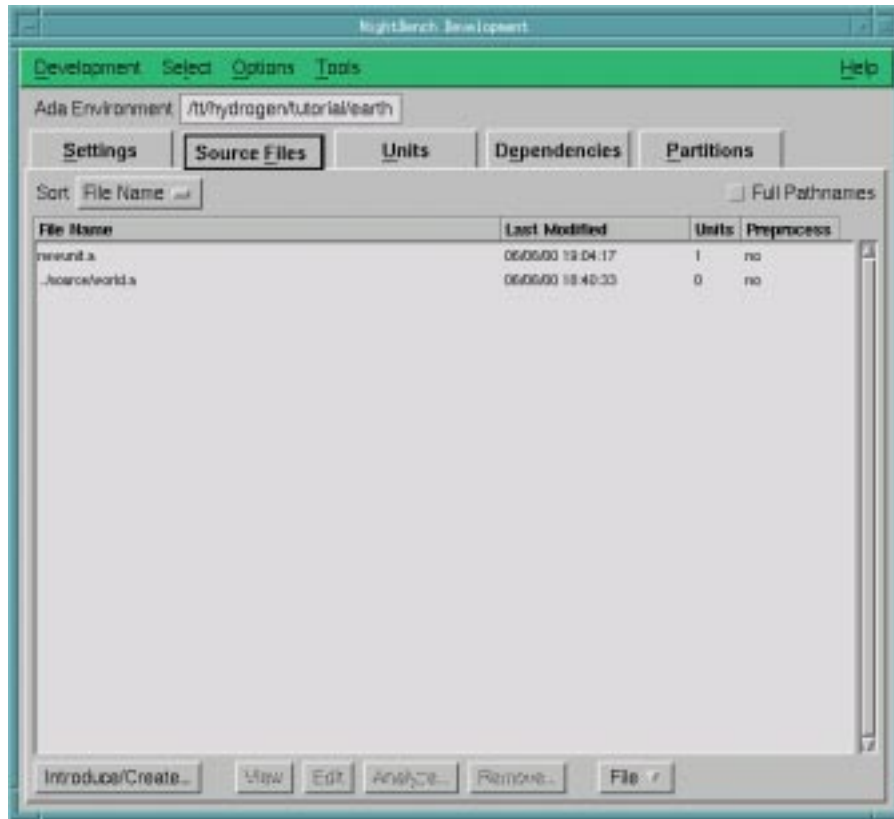


Figure 2-19. Source file, `newunit.a`, newly introduced

Notice that the file `world.a` shows 0 units because its `hello` unit has been hidden in order to resolve the ambiguity.

See also:

- “Development - Source Files” on page 5-31

No more ambiguities!!!

Let's try to build it again now that the ambiguities are resolved and execute the file to see the results.

Building a partition with ambiguities resolved:

- On the **Partitions** page of the **earth** environment, make sure the partition **hello** is selected in the list of partitions.
- Press the button marked **Build**. This will open the NightBench Builder window so we can build our partition.

You will see that partition **hello** has been automatically entered in the **Targets** field on the **Build** page. This is because it was selected on the **Partitions** page when the **Build** button was pressed.

- Press **Start Build**.

When the build is completed, a **Build Completed** dialog notifies the user.

NOTE

The notification operations can be changed on the **Notification** page (see “Builder - Notification” on page 6-41).

To run the program:

- On the **Build** page of the NightBench Builder window in the **earth** environment, make sure the program name, **hello**, appears correctly in the **Run Context** field. (By default, a *run context* is created for every active partition.)
- Press the button marked **Run**.

This will open a terminal window so our executable can run. (The type of terminal window used is configured on the **Terminal** page of the **Preferences** dialog which may be accessed from the **Options** menu - see “Preferences - Terminal” on page 1-11.)

You should see the following output:

```
I am a new unit - Hello!!!
```



Figure 2-20. Running the program

NOTE

When two ambiguous units are in conflict, the compilation state of the original unit remains intact in the environment. This is useful in case the original unit is selected to resolve the ambiguity. If this is the case, the original unit (and all units dependent on it) would not have to be recompiled.

In this example, however, we have chosen the newly added unit, so the unit must be compiled in order for the partition to be built.

- Close the terminal window when you are finished viewing the output.

See also:

- “Development - Partitions - (Ada)” on page 5-96
- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13
- “Builder - Notification” on page 6-41
- “Preferences” on page 1-8

Using NightBench - Conclusion

This concludes the Ada tutorial for NightBench. You should now be familiar enough with the NightBench interface and its usage to explore the additional functionality not covered

in this tutorial. You may investigate the other sections of this manual to get detailed explanations of each of the NightBench windows and their contents.

If any questions arise while you are using NightBench, you may use the online help system by selecting the **On Window** or **On Item** menu items from the **Help** menu (see “Help Menu” on page 1-21). In addition, context-sensitive help may be obtained for the currently highlighted option by pressing the **F1** key. The HyperHelp viewer will open with the appropriate topic displayed.

Using NightBench with C/C++

As a starting point into the world of NightBench, an example will be given. This example will traverse step-by-step through the various NightBench windows and dialogs, from creating an environment, to introducing units and managing source files, defining a partition, and finally, building that partition.

Starting NightBench

We will begin by opening the NightBench Project window, our launching pad for this example. From the command line, type the following command.

```
$ nbench
```

Screen 3-1. Starting NightBench

Note that we have not provided **nbench** with any parameters, indicating that we want to open the NightBench Project window. **nbench** accepts a number of command line options, allowing the user to open a particular NightBench component or to provide start-up information to NightBench. See “Starting NightBench” on page 1-3 for more information about these options.

The NightBench Project window will appear, listing the environments with which it has interacted. If no other environments have been created under NightBench, this list will be empty. See Chapter 4 for more information about the NightBench Project window.

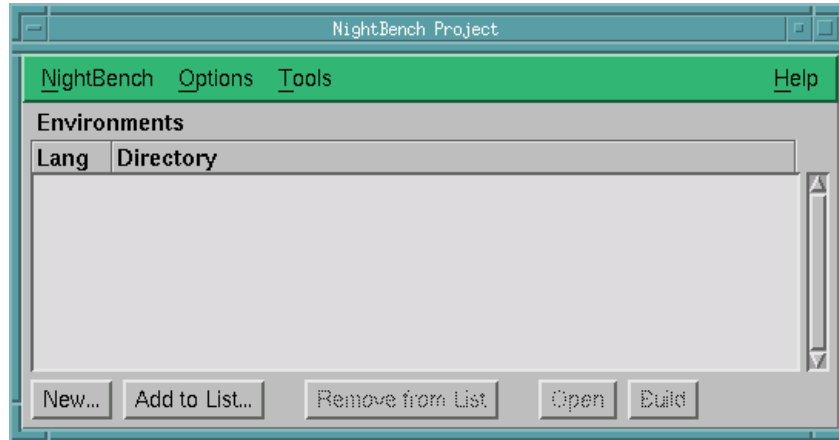


Figure 3-1. NightBench Project

Creating a new environment

One of the first steps we must take in order to use NightBench for C or C++ program development is to create an *environment*. Environments are used as the basic structure of organization within the NightBench Program Development Environment.

To create a new environment from the NightBench Project window:

- On the NightBench Project window, press the button marked **New...** so we can create our new environment. This will open the New Environment Language Selection dialog. Select the language that will be used in this environment (C/C++). Press the **Next>** button.

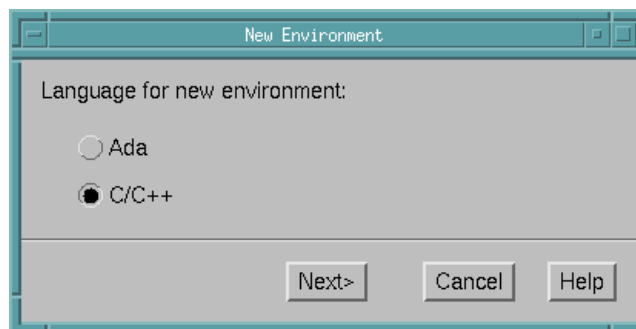


Figure 3-2. Selecting a language for a new environment

- The next dialog presented is the New Environment Directory Selection dialog:
 - Type the directory name in the Directory for new environment field where you want NightBench to create the new environment.

This can be the name of an existing directory or NightBench can create the directory for you. (Note that NightBench can only create a subdirectory of an existing directory.) The directory in which our new environment is to be created will be called **c-world** and shall be created under the existing directory **/tt/hydrogen/tutorial**.

- You may also select a **Release** if you have more than one release of Concurrent C/C++ installed on your system. If you have only one release of Concurrent C/C++ installed on your system, it will appear here.
- Press **Done**.

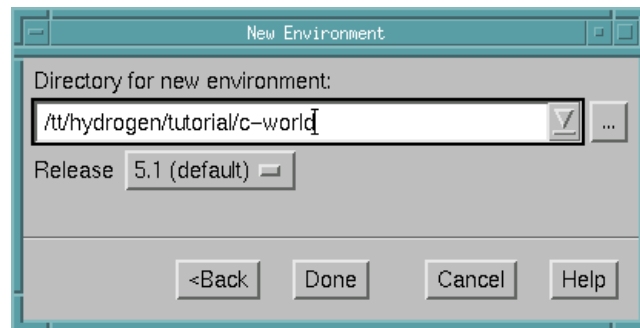


Figure 3-3. Selecting a directory for a new environment

This will add the new environment to the list of **Environments** in the NightBench Project window. NightBench will also open the new environment in its own NightBench Development window.

See also:

- “New Environment” on page 4-6
- NightBench Development - Chapter 5

Introducing units into the environment

Our next step is to populate the environment with *units*. Units are the basic building blocks for C or C++ programs in NightBench. They are contained within source files and it is through these source files that they are introduced into their intended environments.

Source files may already have been created outside the NightBench environment or you may use the editing features of NightBench to create a new file. For this example, we will assume we have already created a file named **world.c**. In C and C++ environments, source files are usually associated with one unit which, by default, is named the same as the source file without the extension (in our example, our source file **world.c** contains one unit, **world**). This file appears in Screen 3-2. (The source file is just an ordinary text file.)

```
#include <stdio.h>

int main (int argc, void **argv) {
    printf("Hello World!!!\n");
    return 0;
}
```

Screen 3-2. Sample source file - world.c

To introduce an existing source file into an environment:

- Click on the Source Files tab of the NightBench Development window.
- Press the Introduce/Create... button. This will open the Introduce Source Files dialog so we can introduce our source files (and the units contained within) into the new environment.

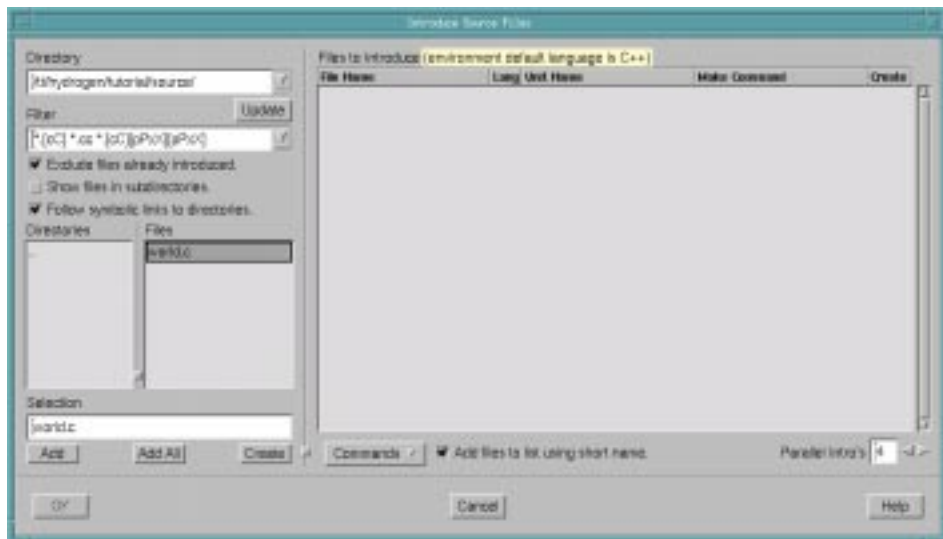


Figure 3-4. Introducing a source file

- Manuever to the directory in which the **world.c** source file is contained. You may type the path to the directory name in the **Directory** field or use the entries in the **Directories** list to navigate to the desired directory.
- When you have arrived at the desired directory and the **world.c** source file name appears in the **Files** list, select it by clicking the name with the mouse. It will then appear in the **Selection** field.
- Press the **Add** button to add this file to the list of **Files to Introduce**. (You may introduce any number of files by adding them here but for our example we will just introduce this one file.)

- Because the default language for compiling source files is set to C++, and our source file is C, we need to change the language setting for this file before it is introduced:
 - Select the source file in the list of **Files to Introduce**.
 - Click on the **Commands** menu.
 - Click on the **Set language** menu item and select **C** from the list of choices.
- Press the **OK** button to introduce the source file into the environment.

The unit `world` that was contained in the source file `world.c` is now a part of the environment `c-world`.

The source file now appears in the list of files on the **Source Files** page of the NightBench Development window and its associated unit now appears on the **Units** page.

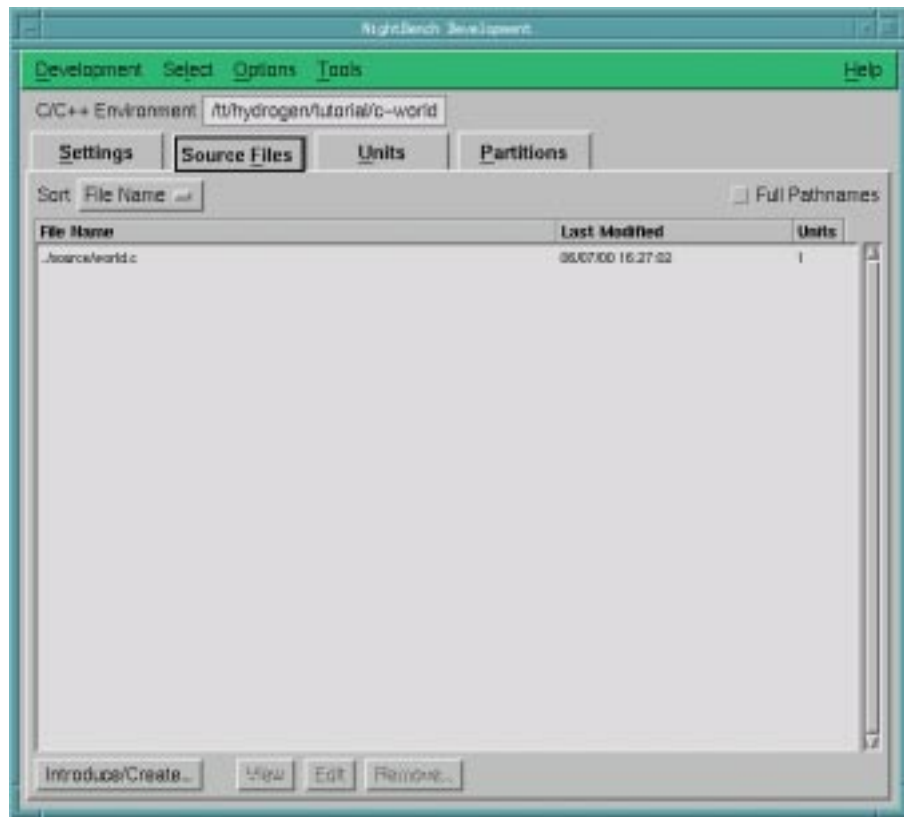


Figure 3-5. Source Files page

See also:

- “Development - Source Files” on page 5-31

- “Introduce Source Files” on page 5-37

Defining a partition

In order to use the units introduced into NightBench, we must include them in a *partition*. NightBench defines four types of C/C++ partitions:

- *executable*
- *archive*
- *shared object*
- *object*

For our example, we want to include the unit `world` in an executable program so we will be defining an *executable* partition.

To define a partition in the environment:

- Click on the **Partitions** tab of the NightBench Development window to get to the **Partitions** page.
- Press the **Create...** button. This will open the Create Partition dialog so we can define a partition in the new environment.

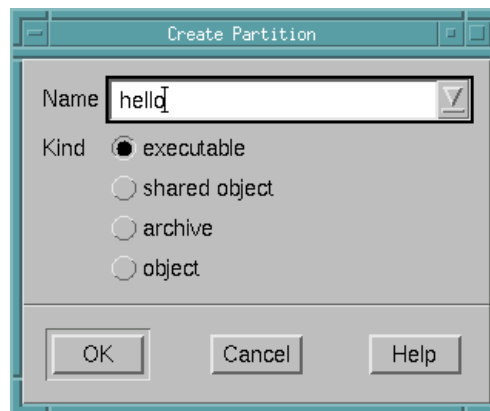


Figure 3-6. Defining an executable partition

- Type the name `hello` in the Name field.
- Verify that the `executable` radiobutton is selected for the Kind of partition we are creating.
- Press OK.

See also:

- “Create Partition” on page 5-154

Including units in the partition

Once the partition is defined, we may add the units we want to be included in the partition.

We do this on the Units subpage of the Partitions page.

To include a unit in a partition definition:

- Click on the Units tab on the Partitions page of the NightBench Development window.
- Select the unit `world` from the list of Units in Environment.
- Press the `->` button to add it to the list of Included Units.

The `+` in the Req. column indicates that all units required by the unit `world` are to be included when linking this partition.

- Press Apply so that your changes will take effect.

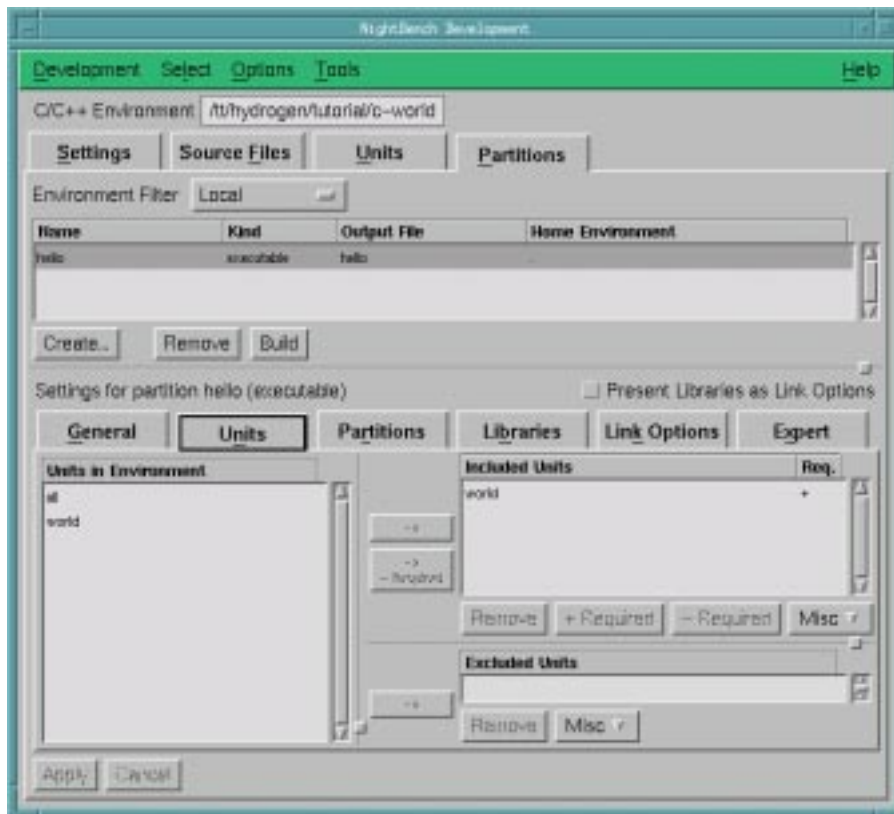


Figure 3-7. Including units in a newly defined partition

As you can see in Figure 3-7, the unit `world` has been added to the list of Included Units for the `hello` partition.

See also:

- “Partitions - Units” on page 5-161

Building a partition

At this point, we have an environment, **c-world**, that has within it the definition for the partition, **hello**, made up of a the unit **world**, which was introduced from the source file, **world.c**.

We can now build this partition. We do this using the NightBench Builder.

To build the partition:

- In the list of partitions on the **Partitions** page, make sure the partition **hello** is selected.
- Press the button marked **Build**. This will open the NightBench Builder window so we can build our new partition.

In Figure 3-8, you will see that **partition hello** has been automatically entered in the **Targets** field on the **Build** page. This is because it was selected on the **Partitions** page when the **Build** button was pressed.

- Press **Start Build**.

The **Build Progress** bar shows the number of actions (compilations and links) left to perform in the current build as the **Transcript** window details each step taken during the build.

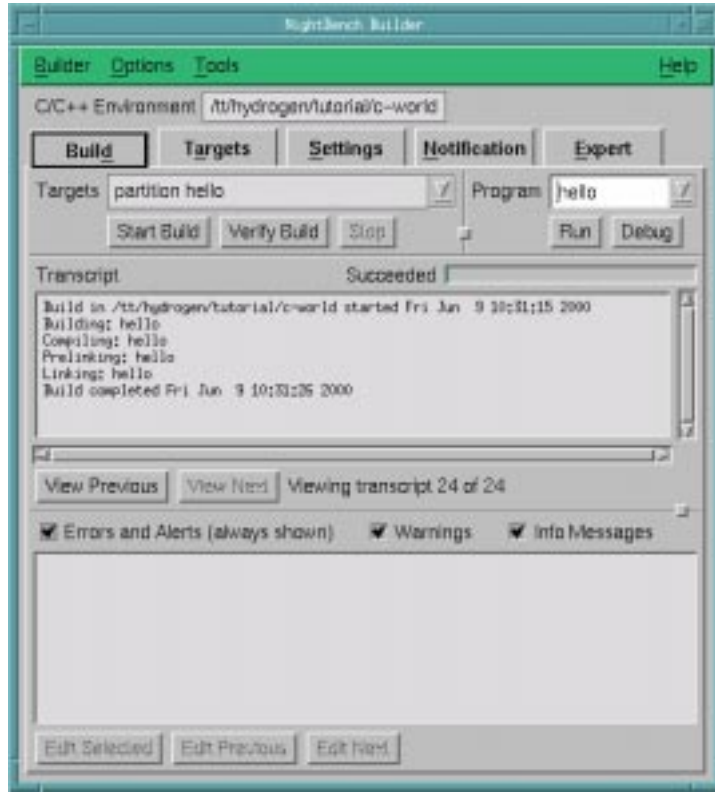


Figure 3-8. Builder window - Build page

When the build is completed, a Build Completed dialog notifies the user.

NOTE

The notification operations can be changed on the Notification page (see “Builder - Notification” on page 6-41).

See also:

- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13

Running the program

All that’s left to do is run the program. This is easily accomplished in the NightBench Builder window.

To run the program:

- On the **Build** page of the NightBench Builder window in the **c-world** environment, make sure the program name, **hello**, appears correctly in the **Run Context** field. (By default, a *run context* is created for every active partition.)
- Press the button marked **Run**.

This will open a terminal window so our executable can run. (The type of terminal window used is configured on the **Terminal** page of the **Preferences** dialog which may be accessed from the **Options** menu - see “Preferences - Terminal” on page 1-11.)

You should see the following output:

```
[running hello]

Hello World!!!

[program exited with status 0]
```

- Close the terminal window when you are finished viewing the output.

See also:

- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13

What options do I have?

NightBench uses the concept of persistent compile options. These options are specified through the Options Editors and are “remembered” at compilation time. They can apply to any of three areas: *environment-wide compile options* (which apply to all units within the environment), *permanent unit compile options* and *temporary unit compile options* (both of which apply and are unique to specific units).

Let's manipulate the options in our example to get an idea of how these options work.

Setting environment-wide compile options

First, we will consider the *environment-wide compile options*. These apply to all units within the environment. Since we only have one unit right now, it will apply to that. However, if we add any other units later, they will “inherit” these options automatically.

To change the environment-wide compile options:

- Click on the **Settings** tab of the NightBench Development window.

You'll see that the **Permanent Options** field is empty. That's because we haven't set anything yet. So let's set them to something and see what happens.

- Click on the button marked **Show Options Editor**. This brings up the C/C++ Environment Compile Options Editor.
 - Click on the **General** tab to get to the C/C++ Environment Compile Options Editor - General page.
 - Change the **Permanent** column of the **Debug Information** row so that a solid checkmark appears in the box. (See “Tri-state Checkboxes” on page 7-1 for more information.)
 - Click on the **Optimizer** tab to get to the C/C++ Environment Compile Options Editor - Optimizer page.
 - Change the **Permanent** value of **Optimization Level** to **global (2)** (using the drop-down list).
 - Press **OK** to apply the changes and close the editor dialog.

You will see **-O2 -g** in the **Permanent Options** field, corresponding to the changes we just made.

Remember, these options apply to all units in the environment and will be “inherited” by any units we add to this environment.

See also:

- Options Editors - Chapter 7
- “Environment Compile Options Editor” on page 7-3

Setting unit compile options

If we'd like to set particular options for a specific unit, we can use the *permanent unit compile options* for that unit. They're set in much the same way as *environment-wide compile options*, except that we need to specify the units to which they apply.

To change the permanent unit compile options:

- Click on the Units tab of the NightBench Development window.
- Select the unit `world` from the list of units.

You'll see that the Permanent field is empty. That's because we haven't set anything yet. So let's set these options and see what happens.

- Click on the button marked Show Options Editor. This brings up the C/C++ Unit Compile Options Editor.
 - Click on the General tab of the C/C++ Unit Compile Options Editor.
 - Change the Permanent value of Suppress Warnings so that a solid checkmark appears in the box. (See "Tri-state Checkboxes" on page 7-1 for more information.)
 - Press Apply so that the changes take effect but the C/C++ Unit Compile Options Editor remains open.

We may decide that in addition to the specified options, we may want to "try out" some options or change particular options for a specific compilation but only "temporarily". The *temporary unit compile options* are a separate set of options maintained for this purpose. They are persistent in the same way that permanent options are persistent, however, they may be cleared easily without altering the permanent options.

Let's suppose we don't want to produce any debug information for our `hello` unit for this particular compilation. We can set a temporary compile option for that.

To change the temporary unit compile options:

- Click on the General tab of the C/C++ Unit Compile Options Editor.
- Change the Temporary column of the Debug Information row so that no checkmark appears in the box. (See "Tri-state Checkboxes" on page 7-1 for more information.)
- Change the Temporary value of Chars Signed by Default so that a solid checkmark appears in the box.
- Press Apply so that the changes take effect but the C/C++ Unit Compile Options Editor remains open.

See also:

- Options Editors - Chapter 7
- "Ada Unit Compile Options Editor" on page 7-24

Determining the effective options

The *environment-wide compile options*, *permanent unit compile options*, and *temporary unit compile options* have a hierarchical relationship to one another which means the environment-wide compile options are overridden by the permanent unit options, which are, in turn, overridden by the temporary unit options. This relationship forms the *effective compile options* for the unit, which the compiler will use during compilation.

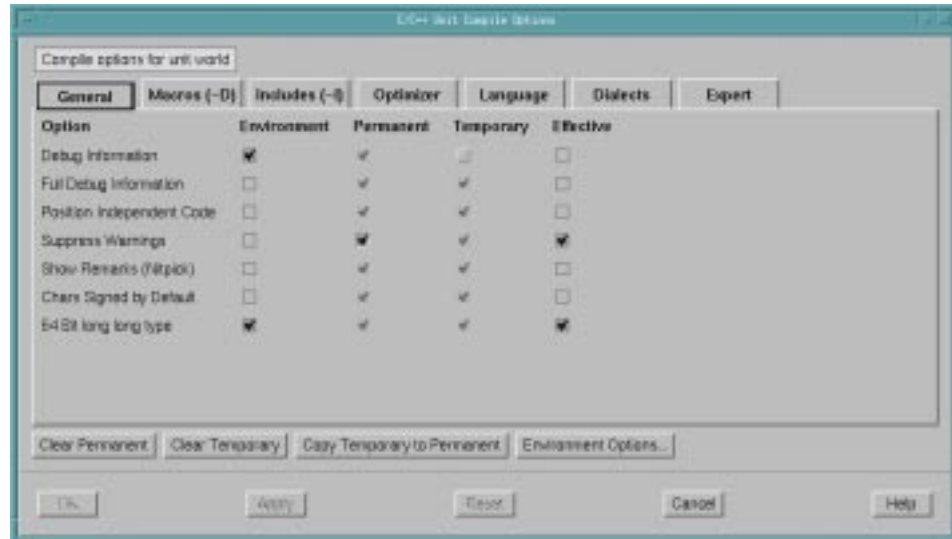


Figure 3-9. Unit Compile Options dialog - General Page - Effective Options

As you can see in Figure 3-9, the effective options are derived from the environment, permanent, and temporary compile options. In this example, the **Effective** values for both **Debug Information** and **Chars Signed by Default** come from the **Temporary** value set which takes precedence over the **Environment** and **Permanent** compile options. In the case of **Suppress Warnings**, since the **Temporary** value is unspecified, the **Effective** value comes from the **Permanent** value. If neither the **Permanent** nor the **Temporary** value is set, the **Effective** option comes from the **Environment** option, as is the case with the **Effective** value for **64 Bit long long type**.

The effective values may change if the environment, permanent, or temporary option values change. We can see this if we change the **Temporary** value of **Debug Information**.

Deleting a temporary unit compile option:

- Change the **Temporary** value of **Debug Information** to (unspecified) (a grayed-out checkbox).
- Press **Apply** so that the changes take effect but the C/C++ Unit Compile Options Editor remains open.

You can see that the **Effective** value for **Debug Information** has changed to a solid checkmark, the **Environment** value for this option.

See also:

- Options Editors - Chapter 7
- “Unit Compile Options Editor” on page 7-24

Copying the temporary options to the permanent set

If we like the other temporary options so much that we'd like to make them permanent, NightBench provides the **Copy Temporary to Permanent** button to propagate all the temporary options for a particular unit to the permanent option set for that same unit.

Propagating the temporary options to the permanent set:

- Press the **Copy Temporary to Permanent** button on the C/C++ Unit Compile Options Editor.

All values specified as **Temporary** options have been propagated to the **Permanent** options set and reset to unspecified values. In our example, the value specified as a **Temporary** option for **Chars Signed by Default** has become a **Permanent** option.

- Press **OK** to apply the changes and close the C/C++ Unit Compile Options Editor.

See also:

- Options Editors - Chapter 7
- “Unit Compile Options Editor” on page 7-24

Using NightBench - Conclusion

This concludes the C/C++ tutorial for NightBench. You should now be familiar enough with the NightBench interface and its usage to explore the additional functionality not covered in this tutorial. You may investigate the other sections of this manual to get detailed explanations of each of the NightBench windows and their contents.

If any questions arise while you are using NightBench, you may use the online help system by selecting the **On Window** or **On Item** menu items from the **Help** menu (see “Help Menu” on page 1-21). In addition, context-sensitive help may be obtained for the currently highlighted option by pressing the **F1** key. The **HyperHelp** viewer will open with the appropriate topic displayed.

NightBench Project

NightBench Project is the central location for accessing the various NightBench tools and environments. It maintains a list of the user's NightBench environments and allows the user to add or remove environments from this list. It acts as a launching pad to any of the environments listed, permitting the user to open any of these environments in the NightBench Development window or the NightBench Builder.

In addition, other functionality can be found on the NightBench Project menu bar.

See also:

- NightBench Development - Chapter 5
- NightBench Builder - Chapter 6

NightBench Project Menu Bar

The NightBench Project menu bar consists of the following items:

- “NightBench Menu” on page 4-2
- “Options Menu” on page 4-3
- “Tools Menu” on page 1-19
- “Help Menu” on page 1-21

NightBench Menu

The NightBench menu appears on the NightBench Project menu bar.

See also:

- “NightBench Project Menu Bar” on page 4-1

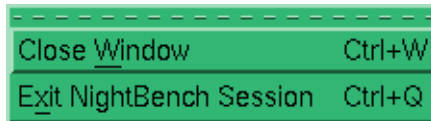


Figure 4-1. NightBench menu

Close Window

Closes the NightBench Project window, leaving any other NightBench windows or internal utilities running.

Exit NightBench Session

Closes all of the NightBench windows and shuts down all NightBench servers and internal utilities.

Options Menu

The Options menu appears on the NightBench Project menu bar.

See also:

- “NightBench Project Menu Bar” on page 4-1

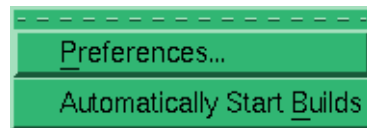


Figure 4-2. Options menu

Preferences...

Opens the Preferences dialog so that the user may configure which file viewer, file editor, and terminal program NightBench should use.

See also:

- “Preferences” on page 1-8

Automatically Start Builds

If this item is checked, pressing the **Build** button on the **Units** page or the **Partitions** page will start up a NightBench Builder tool, then immediately start the build.

There is a setting in the **Preferences** dialog which controls the value of this when a new Project window is opened. Note, however, that the setting of this particular menu item overrides the setting in the **Preferences** dialog and dictates the appropriate behavior.

See also:

- “Units - Build” on 5-65
- “Partitions - Build” (*Ada*) on 5-100
- “Partitions - Build” (*C/C++*) on 5-150
- NightBench Builder - Chapter 6
- “Preferences - Start” on page 1-14

NightBench Project

NightBench Project is the central location for accessing the various NightBench tools and environments. It maintains a list of the user's NightBench environments and allows the user to add or remove environments from this list. It acts as a launching pad to any of the environments listed, permitting the user to open any of these environments in the NightBench Development window or the NightBench Builder.

In addition, other functionality can be found on the NightBench Project menu bar (see page 4-1).

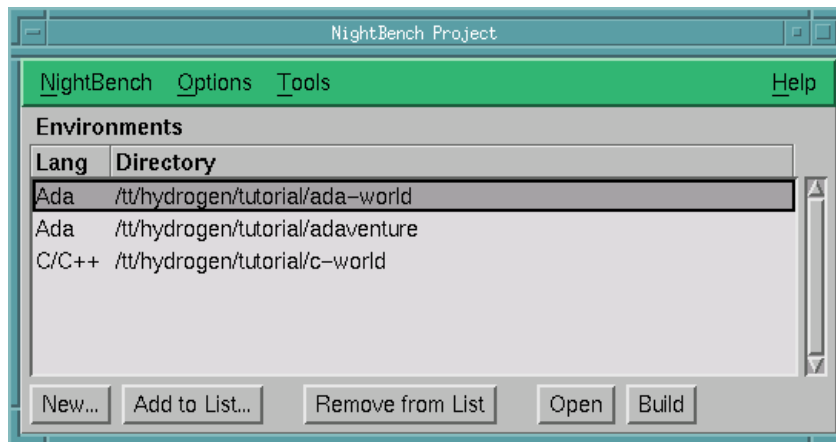


Figure 4-3. NightBench Project

Environments

Lists the environments previously used by NightBench.

In addition to the directory in which the environment is found, the programming language used for development in this environment is also listed.

You may double-click on an environment in this list to open it in its own NightBench Development window. If the environment is already opened, it will bring that NightBench Development window to the top.

See also:

- NightBench Development - Chapter 5
- “Preferences - Language” on page 1-13

New...

Creates a new NightBench environment, adds the environment name to the list in the NightBench Project window, and opens the new environment in an NightBench Development window.

If the directory for the new environment does not exist, NightBench will create this directory. NightBench can only create a subdirectory under an existing directory.

See also:

- “New Environment” on page 4-6
- NightBench Development - Chapter 5

Add to List...

Adds an existing NightBench environment to the list in the NightBench Project window.

See also:

- “Add Environment to List” on page 4-9

Remove from List

Remove the selected NightBench environment from the list in the NightBench Project window.

Open

Open the selected NightBench environment in an NightBench Development window.

See also:

- NightBench Development - Chapter 5

Build

Open the NightBench Builder window for the selected NightBench environment.

See also:

- NightBench Builder - Chapter 6

New Environment

Creates a new NightBench environment, adds the environment to the list in the NightBench Project window, and opens the new environment in a NightBench Development window.

If the directory for the new environment does not exist, NightBench will create this directory. NightBench can only create a subdirectory under an existing directory.

See also:

- NightBench Development - Chapter 5
- “NightBench Project” on page 4-4

The creation of a new environment is presented as a series of dialogs. The first dialog is shown below.

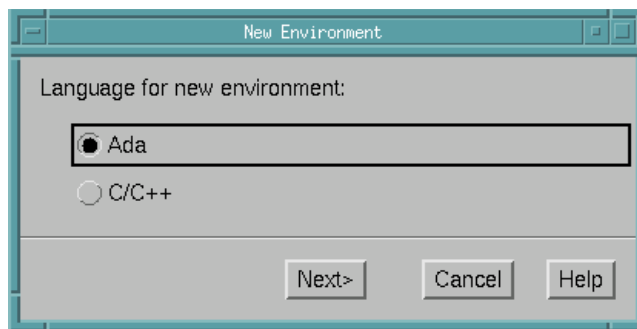


Figure 4-4. New Environment - Language Selection

NOTE

This dialog is bypassed if a default language has been selected in the **Languages** section of the **Preferences** dialog. (See “Preferences - Language” on page 1-13.)

Language for new environment

This dialog allows the user to choose the programming language which will be used in this new environment.

Ada

Create a new environment for the development of Ada programs.

C/C++

Create a new environment for the development of C or C++ programs.

The next dialog allows the user to select which directory the environment should be created in and the release of the compiler toolset to be used.

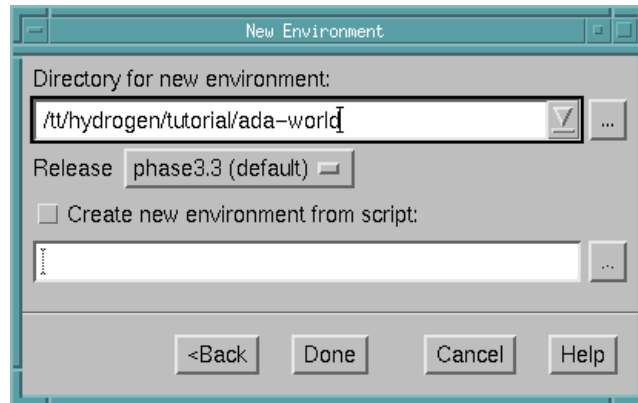


Figure 4-5. New Environment - Directory Selection

Directory for new environment

The directory in which the new environment is to be created. This can be specified either by manually entering the pathname or by using the file selection dialog (...) to navigate to it.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the **Directory for new environment** field.

Release

The release of the compiler toolset associated with the current environment. If more than one release is installed on the system, the release names will appear in the drop-down list associated with this field. The names of the releases in this list will be based on either the language selected in the previous dialog (see Figure 4-4 on page 4-6) or the choice selected in the **Languages** section of the **Preferences** dialog.

See also:

- “Preferences - Language” on page 1-13
- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)

Create new environment from script

Specifies the pathname of a previously-saved script of commands which will be used to create this new environment.

This script will create the environment, populate the environment with units, set the compile options, populate the environment with partitions, and set up dependencies between partitions, as dictated by the script.

See also:

- “Save Environment...” on page 5-4
- “Load Environment...” on page 5-4



Use this to popup a dialog and browse for the script you want. When finished, the pathname of the script is entered in the **Create new environment from script** field.

Add Environment to List

This dialog adds a preexisting NightBench environment to the list in the NightBench Project window.

See also:

- NightBench Development - Chapter 5
- “NightBench Project” on page 4-4

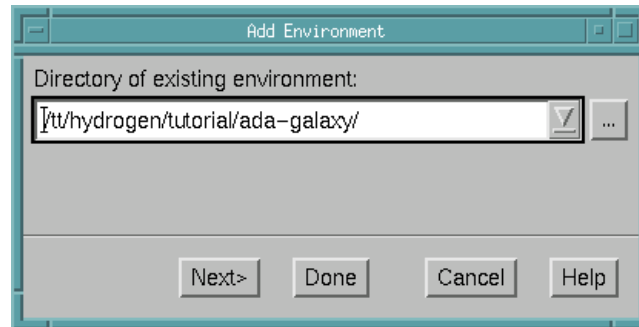


Figure 4-6. Add Environment to List - Directory Selection

Directory of existing environment

The environment to be added to the list of environments in the NightBench Project window is specified in this field, either by manually entering the pathname or by using the file selection dialog (...) to navigate to it.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the Directory of existing environment field.

If more than one environment exists in the directory specified, the user is then asked to provide the language used for program development in the specified environment.

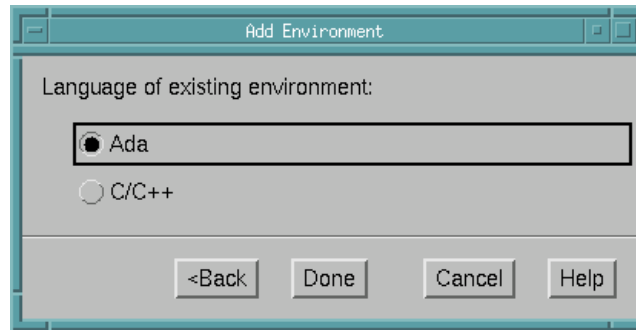


Figure 4-7. Add Environment to List - Language Selection

NOTE

This dialog is bypassed if a default language has been selected in the Languages section of the Preferences dialog. (See "Preferences - Language" on page 1-13.)

Language of existing environment

This dialog allows the user to select the programming language used in this environment.

Ada

Select this if the specified environment is used for the development of Ada programs.

C/C++

Select this if the specified environment is used for the development of C or C++ programs.

NightBench Development

The NightBench Development window contains the resources necessary to establish and maintain a NightBench environment. Environments are used as the basic method of organization in NightBench. These environments take advantage of the various NightBench utilities to manipulate the various *units* that form any number of *partitions*.

Environments may include:

- units that have been introduced
- partitions that have been defined
- *Environment Search Paths*
- references to source files (which generally contain units)
- other information used internally by NightBench

The mechanisms for program generation, including compilation and linking, as well as access to debugging tools can be found in the NightBench Builder (see Chapter 6).

Structure

The NightBench Development window is composed of the following five pages:

- **Settings** (see page 5-18)
- **Source Files** (see page 5-31)
- **Units** (see page 5-61)
- **Dependencies** - (*Ada Only*) (see page 5-93)
- **Partitions** - (*Ada*) (see page 5-96) or **Partitions** - (*C/C++*) (see page 5-147)

At the top of the NightBench Development window is listed:

Ada Environment / C/C++ Environment

The name of the environment currently being operated upon in the NightBench Development window.

In addition, other functionality can be found on the NightBench Development menu bar (see page 5-3).

NightBench Development Menu Bar

The NightBench Development menu bar consists of the following items:

- “Development Menu” on page 5-3
- “Select Menu” on page 5-13
- “Options Menu” on page 5-14
- “Tools Menu” on page 1-19
- “Help Menu” on page 1-21

Development Menu

The Development menu appears on the NightBench Development menu bar.

See also:

- “NightBench Development Menu Bar” on page 5-3



Figure 5-1. Development menu

New Environment...

Creates a new NightBench environment.

See also:

- “New Environment” on page 5-5

Open Environment...

Opens an existing NightBench environment.

See also:

- “Open Environment” on page 5-8

Refresh Environment Info

Updates all the information in the NightBench Development window for the current environment.

This is useful for informing NightBench when an environment changes due to an external source (for instance, the command-line interface or another NightBench session).

Save Environment...

Creates a script of commands which can be used to reproduce the current environment. This script will create the environment, populate the environment with units, set the compile options, populate the environment with partitions, and set up dependencies between partitions, as dictated by the script.

This script can either be used when creating a new environment or can be loaded into an existing environment.

See also:

- “New Environment” on page 4-6
- “New Environment” on page 5-5
- “Load Environment...” on page 5-4
- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)

Load Environment...

Removes the existing environment and recreates it using the specified script. This script will create the environment, populate the environment with units, set the compile options, populate the environment with partitions, and set up dependencies between partitions, as dictated by the script.

NOTE

It will be necessary to rebuild any existing units and partitions.

See also:

- “Save Environment...” on page 5-4
- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)

Run Context...

Opens a dialog to edit *run contexts*.

See also:

- “Run Context” on page 6-19

Remove Environment...

Removes an existing NightBench environment.

See also:

- “Remove Environment” on page 5-11

Close Window

Closes the NightBench Development window, leaving any other NightBench windows or internal utilities running.

Exit NightBench Session

Closes all of the NightBench windows and shuts down all NightBench servers and internal utilities.

New Environment

The **New Environment** dialog allows the user to create a new environment in the specified directory using a particular release. This environment can be opened in the current NightBench Development window or in a new NightBench Development window.

See also:

- NightBench Development - Chapter 5
- “Development Menu” on page 5-3

The creation of a new environment is presented as a series of dialogs. The first dialog is shown below.

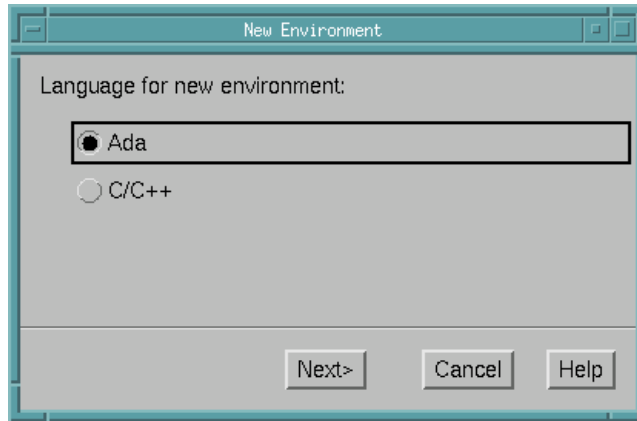


Figure 5-2. New Environment - Language Selection

NOTE

This dialog is bypassed if a default language has been selected in the **Languages** section of the **Preferences** dialog. (See “Preferences - Language” on page 1-13.)

Language for new environment

This dialog allows the user to choose the programming language which will be used in this new environment.

Ada

Create a new environment for the development of Ada programs.

C/C++

Create a new environment for the development of C or C++ programs.

The next dialog allows the user to select which directory the environment should be created in and the release of the compiler toolset to be used. The user may also specify a previously-saved script from which to create the environment.

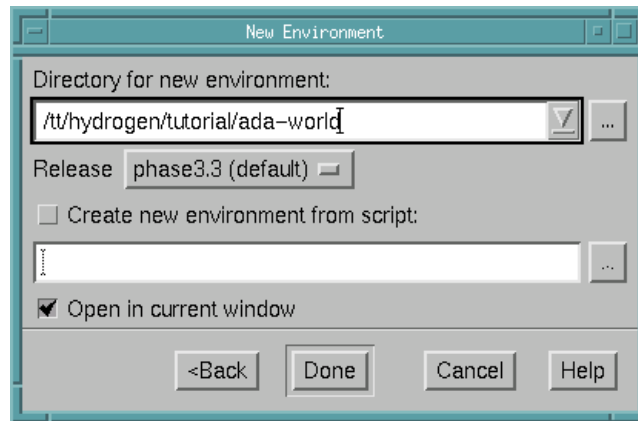


Figure 5-3. New Environment - Directory Selection

Directory for new environment

The directory in which this environment is to be created. This can be specified either by manually entering the pathname or by using the file selection dialog (...) to navigate to it.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the **Directory for new environment** field.

Release

The release of the compiler toolset associated with the current environment. If more than one release is installed on the system, the release names will appear in the drop-down list associated with this field. The names of the releases in this list will be based on either the language selected in the previous dialog (see Figure 5-2 on page 5-6) or the choice selected in the **Languages** tab of the **Preferences** dialog.

See also:

- “Preferences - Language” on page 1-13
- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)

Create new environment from script

Specifies the pathname of a previously-saved script of commands which will be used to create this new environment.

This script will create the environment, populate the environment with units, set the compile options, populate the environment with partitions, and set up dependencies between partitions, as dictated by the script.

See also:

- “Save Environment...” on page 5-4
- “Load Environment...” on page 5-4



Use this to popup a dialog and browse for the script you want. When finished, the pathname of the script is entered in the **Create new environment from script** field.

Open in current window

If this option is checked, the new environment will be displayed in the current NightBench Development window. If this option is not checked, a new NightBench Development window will be opened for this new environment.

Open Environment

The Open Environment dialog allows the user to open a preexisting environment either in the current NightBench Development window or a new NightBench Development window.

See also:

- NightBench Development - Chapter 5
- “Development Menu” on page 5-3

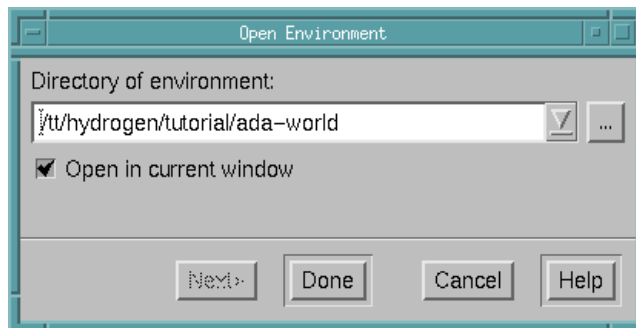


Figure 5-4. Open Environment - Directory Selection

Directory of environment

The environment to be opened is specified in this field, either by manually entering the environment name or by using the file selection dialog (...) to navigate to it.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the Directory of environment field.

Open in current window

If this option is checked, the selected environment will be displayed in the current NightBench Development window. If this option is not checked, a new NightBench Development window will be opened for the specified environment.

If more than one environment exists in the directory specified, the **Next** button will be activated to allow you to specify the language of the environment to open:

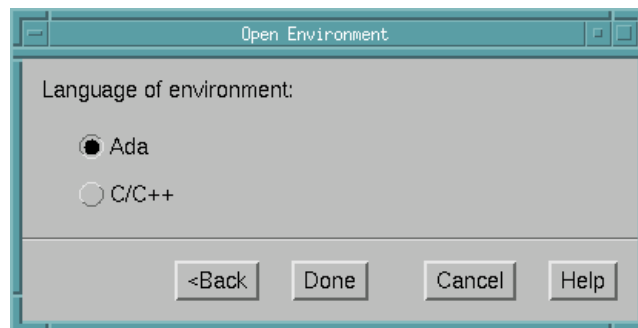


Figure 5-5. Open Environment - Language Selection

NOTE

This dialog is bypassed if a default language has been selected in the **Languages** section of the **Preferences** dialog. (See “Preferences - Language” on page 1-13.)

Language of environment

This dialog allows the user to specify the programming language of the environment you wish to open.

Ada

Open the environment used for the development of Ada programs.

C/C++

Open the environment used for the development of C or C++ programs.

Remove Environment

The **Remove Environment** dialog allows the user to remove a preexisting environment, including all units, their state information, and any partition definitions. The source files and any built partitions are left intact after this operation.

See also:

- NightBench Development - Chapter 5
- “Development Menu” on page 5-3

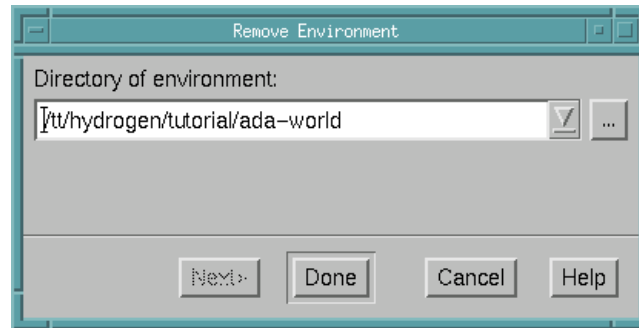


Figure 5-6. Remove Environment - Directory Selection

Directory of environment

The environment which is to be removed is specified in this field, either by manually entering the environment name or by using the file selection dialog (...) to navigate to it.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the **Directory of environment** field.

If more than one environment exists in the directory specified, the **Next** button will be activated to allow you to specify the language of the environment to remove:

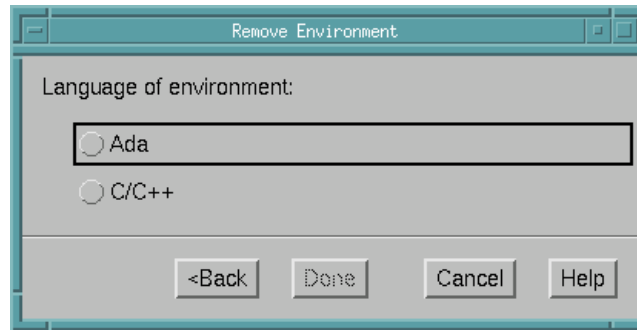


Figure 5-7. Remove Environment - Language Selection

NOTE

This dialog is bypassed if a default language has been selected in the Languages section of the Preferences dialog. (See "Preferences - Language" on page 1-13.)

Language of environment

This dialog allows the user to specify the programming language of the environment you wish to remove.

Ada

Remove the environment used for the development of Ada programs.

C/C++

Remove the environment used for the development of C or C++ programs.

Select Menu

The Select menu appears on the NightBench Development menu bar.

See also:

- “NightBench Development Menu Bar” on page 5-3

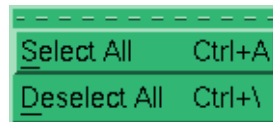


Figure 5-8. Select menu

Select All

For pages in the NightBench Development window which contain a list of selectable items, this selects all items in the list.

For example, if applied on the Units page, this action selects all units within the current environment.

Deselect All

For pages in the NightBench Development window which contain a list of selectable items, this deselects all items in the list.

Options Menu

The Options menu appears on the NightBench Development menu bar.

See also:

- “NightBench Development Menu Bar” on page 5-3



Figure 5-9. Options menu

Preferences...

Opens the Preferences dialog so that the user may configure which file viewer, file editor, and terminal program NightBench should use.

See also:

- “Preferences” on page 1-8

Automatically Start Builds

If this item is checked, pressing the Build button on the Units page or the Partitions page will start up a NightBench Builder window, then immediately start the build.

There is a setting in the Preferences dialog which controls the value of this when a new Development window is opened. Note, however, that the setting of this particular menu item overrides the setting in the Preferences dialog and dictates the appropriate behavior.

See also:

- “Units - Build” on page 5-65
- “Partitions - Build” (*Ada*) on page 5-100
- “Partitions - Build” (*C/C++*) on page 5-150
- NightBench Builder - Chapter 6
- “Preferences - Start” on page 1-14

Long Time Stamps

The user has the choice of displaying time stamps with accuracy either in seconds or in microseconds.

If this item is unchecked, time stamps are shown in *hh:mm:ss* format. If this item is checked, they are shown in *hh:mm:ss.uuuuuu* format.

Show Command Log

This opens a window which displays the specific NightBench Program Development Environment commands that are being performed for each action specified and contains a history of these commands since this NightBench Development window was opened.

There is a setting in the **Preferences** dialog which controls the value of this when a new Development window is opened.

See also:

- *MAXAda Reference Manual* (0890516)
- “Command Log” on page 1-18
- “Preferences - Start” on page 1-14

Accelerated Item Selection

On each page of the NightBench Development window, NightBench lists information relating to that section in a tabular format. For instance, the **Units** page contains a list of all the units contained in the current environment. The user may use the scrollbar to manually search through this list. However, NightBench supports an accelerated form of selection by allowing users to navigate this list using their keyboards.

When focus is placed on the list, users may type in the letters of the item for which they are searching. For instance, a user may type in “com” to find the first unit in the list that begins with those letters. If an item matches those letters, its row is selected and the matching letters appear in bold. This feedback can be seen in Figure 5-10. As the user enters more letters that match items in the list, the selection may change and the feedback will appropriately reflect the matching letters. Any keystrokes entered that do not result in a match in the list are simply discarded. Furthermore, pressing the **Backspace** or **Delete** key removes the last character from the set of matching letters and changes the current selection accordingly. The **ESC** key is used to clear the set of matching letters completely.

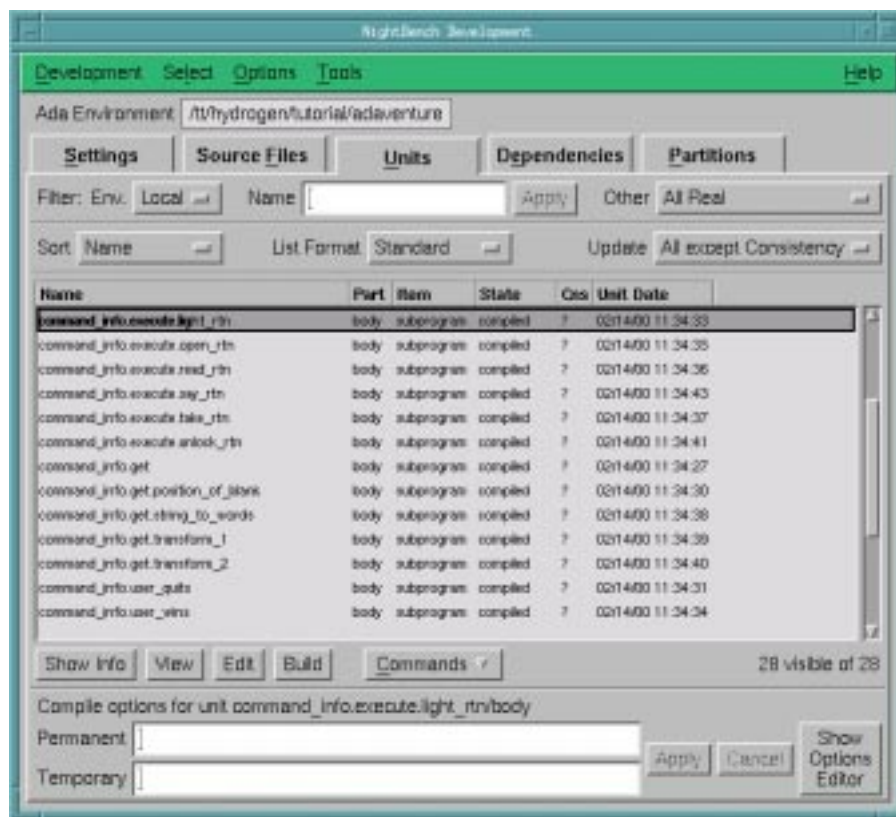


Figure 5-10. Accelerated Item Selection

The space bar acts as a completion mechanism, extending the current set of matching letters to the longest unique string that begins with those letters. The following example illustrates this feature.

If a list contains the following items:

```
cat
dog
mongoose
mongrel
monkey
parakeet
```

and the user enters “m”, the line containing the item “mongoose” will be selected and the letter “**m**” will appear bold. Pressing the space bar will extend the set of matching letters to “**mon**” because it is the longest unique string that begins with the “**m**” that had already been entered.

If the user then extends the selection by typing a “g” and then an “r”, the selection will change to the line containing the item “mongrel” and “**mongr**” will be highlighted. If the user then presses the space bar, the set of matching letters will be extended to “**mon-grel**” because it is the longest unique item in the list that begins with “**mongr**”.

Development - Settings

The Settings page displays general information about the current environment such as the release associated with this environment and the default compile options to be applied to all units introduced into this environment. Other settings such as the *frozen status* and permissions of entities within the current environment are displayed and utilities to change these are supplied. In addition, all environments on the Environment Search Path for the current environment are shown in the order in which they reside on this path. Finally, utilities to manipulate the *Environment Search Path* are provided.

See also:

- NightBench Development - Chapter 5

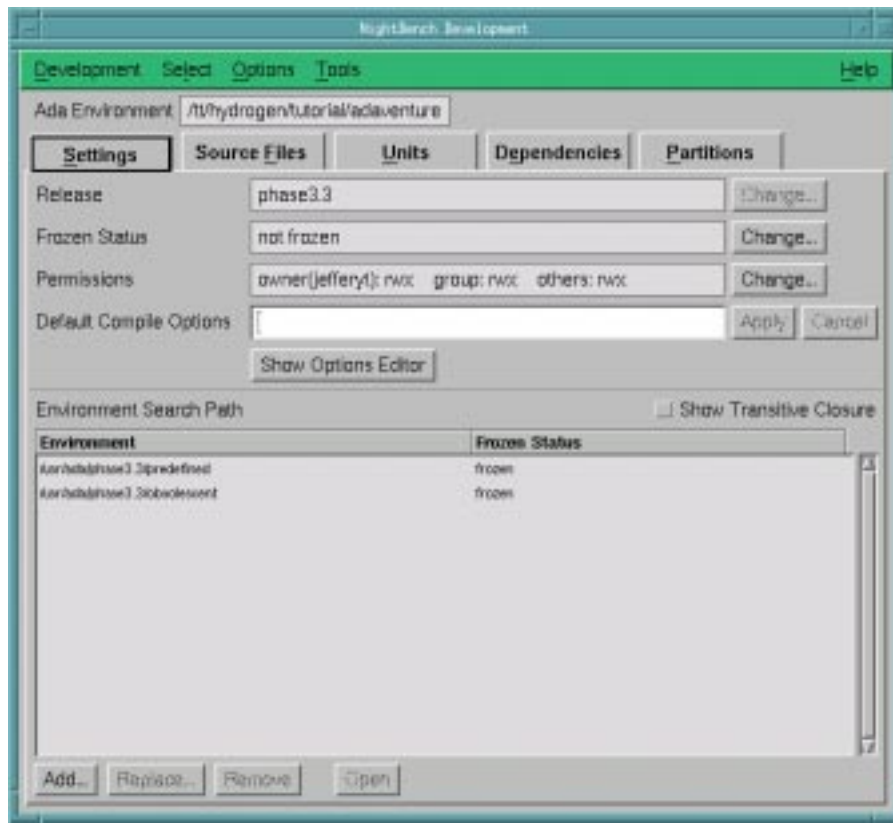


Figure 5-11. NightBench Development - Settings page (Ada)

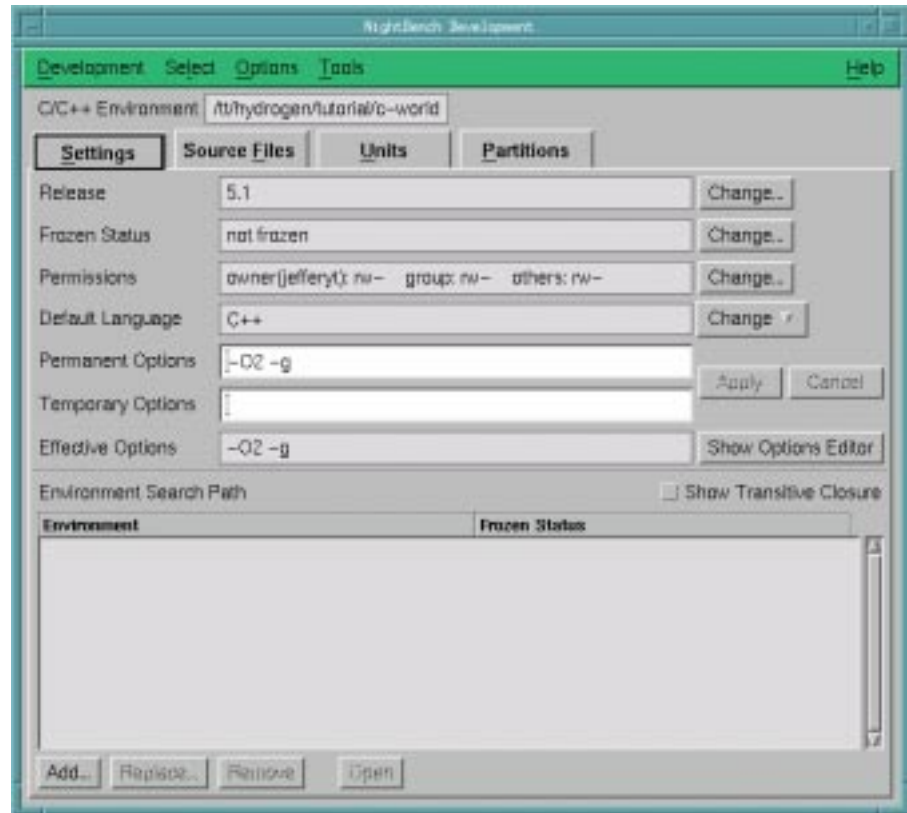


Figure 5-12. NightBench Development - Settings page (C/C++)

Release

The version of the current NightBench Program Development Environment installation associated with the current environment. The release is specified at the time the environment is created.

See also:

- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)

Change...

Changes the release of the current environment. This requires removing the current environment and rebuilding it completely.

NOTE

Any existing units or partitions will need to be rebuilt after this action.

Frozen Status

An environment can be *frozen*. Doing so makes the environment unalterable unless it is subsequently thawed. The advantage of freezing an environment is that the consistency of the contents of the environment is ascertained and subsequently stored. Access to the contents of a frozen environment from other, usually unfrozen, environments is substantially faster since the consistency information does not have to be determined.

Any environment which will not be changed for a significant period of time and which will be used by other environments is a good candidate to be frozen in order to improve compilation performance.

An environment may only be frozen if all of its required environments (those environments on its Environment Search Path) are also frozen.

Change...

Change the *frozen status* of the current environment.

See also:

- “Change Frozen Status” on page 5-25

Permissions

These permissions apply to all the files within the current environment. The permissions are specified in **chmod (1)** format, and are used to allow greater control over those files in the current environment.

Change...

Use this button to change the permissions of the files associated with the current environment.

See also:

- “Change Permissions” on page 5-27

Default Language

(C/C++ only)

Allows the user to specify the default language associated with the source files introduced into the current environment.

C	specifies that source files introduced into the environment will be compiled with the C compiler
C++	specifies that source files introduced into the environment will be compiled with the C++ compiler

Change

Select the default language associated with files introduced into the current environment.

Default Compile Options

(Ada only)

The default compile options (also known as *environment-wide compile options*) apply to all units within the current environment. All compilations within this environment observe these compile options unless overridden. The default compile options can be overridden by the temporary or permanent unit compile options or by pragmas in the source of the units themselves.

The default compile options can be entered directly in this field and applied to the environment or they may be manipulated by using the Environment Compile Options Editor.

In addition, compile options can be set for specific units by selecting the unit from the **Units** page and bringing up the Unit Compile Options Editor for that unit. Compile options set for a specific unit override those specified for the environment.

See also:

- “Ada Environment Compile Options Editor” on page 7-4
- “Ada Unit Compile Options Editor” on page 7-24
- *MAXAda Reference Manual* (0890516)

Apply

Set the options specified in the **Default Compile Options** field as the current environment-wide options.

Cancel

Reset the Default Compile Options to the set of environment-wide options previously applied to the current environment, disregarding any changes made since.

Permanent Options

(C/C++ only)

Each C/C++ environment has its own set of options permanently associated with it that override the defaults specified for the environment. These options apply to all units in the environment. Permanent options can be specified here or by using the Environment Compile Options Editor.

In addition, compile options can be set for specific units by selecting the unit from the **Units** page and bringing up the Unit Compile Options Editor for that unit. Compile options set for a specific unit override those specified for the environment.

See also:

- “C/C++ Environment Compile Options Editor” on page 7-9
- “C/C++ Unit Compile Options Editor” on page 7-31

Temporary Options

(C/C++ only)

Each C/C++ environment also has a set of options that may be temporarily associated with it that override those that are permanently associated with it. These options apply to all units in the environment. Temporary options can be specified here or by using the Environment Compile Options Editor.

Temporary options allow users to “try out” some options or change particular options for a specific compilation but only “temporarily”. These options are a separate set of options maintained for this purpose. They are persistent in the same way that permanent options are persistent, however, they may be cleared easily without altering the permanent options.

In addition, compile options can be set for specific units by selecting the unit from the **Units** page and bringing up the Unit Compile Options Editor for that unit. Compile options set for a specific unit override those specified for the environment.

See also:

- “C/C++ Environment Compile Options Editor” on page 7-9
- “C/C++ Unit Compile Options Editor” on page 7-31

Apply

Set the options specified in the **Permanent Options** and **Temporary Options** field as the current environment-wide options.

Cancel

Reset the **Permanent Options** and **Temporary Options** to those options previously applied to the current environment, disregarding any changes made since.

Effective Options

(C/C++ only)

Displays the set of *effective options* associated with this environment.

Show Options Editor

Bring up the Environment Compile Options Editor so that the environment-wide options can be changed.

See also:

- “Environment Compile Options Editor” on page 7-3

Environment Search Path

A unit will be found in the current environment if it exists there. If not, it will be found in the first environment in the *Environment Search Path* in which it exists. Each environment has its own Environment Search Path. This area lists the environments in the order they are found on the Environment Search Path for the given environment.

By placing an environment on the Environment Search Path, all the units from that environment are conceptually added to the local environment unless that would involve either:

- replacing a unit which was introduced manually into the local environment by a user, or
- replacing a unit found in an environment whose position precedes the new environment on the Environment Search Path.

Particular attention to placement, therefore, should be made when modifying the Environment Search Path.

See also:

- “Add Environment Search Path Element” on page 5-28
- “Replace Environment Search Path Element” on page 5-30
- “Accelerated Item Selection” on page 5-16

Show Transitive Closure

Show all environments that are directly or indirectly required by the current environment.

Environments indirectly required by the current *Environment Search Path* (i.e. those environments that are not mentioned in the current environment's Environment Search Path, but which are mentioned in the Environment Search Path of some environment mentioned in the current environment's Environment Search Path, possibly transitively) are searched for needed units and partitions but only after any explicitly mentioned environment in the Environment Search Path.

When viewing the Environment Search Path with the transitive closure option selected, the Environment Search Path cannot be manipulated in any way. The transitive closure option must be disabled before making any changes to the Environment Search Path.

Add...

Add a new environment to the Environment Search Path for the current environment.

See also:

- “Add Environment Search Path Element” on page 5-28

Replace...

Replace the selected environment in the Environment Search Path with another environment.

See also:

- “Replace Environment Search Path Element” on page 5-30

Remove

Remove the selected environment from the Environment Search Path.

Open

Open the selected environment in a new NightBench Development window.

Change Frozen Status

The frozen status of the current environment is manipulated using this dialog.

An environment can be *frozen*, making it unalterable unless it is subsequently thawed. The advantage of freezing an environment is that the consistency of the contents of the environment is ascertained and subsequently stored. Access to the contents of a frozen environment from other, usually unfrozen, environments is substantially faster since the consistency information does not have to be determined.

Any environment which will not be changed for a significant period of time and which will be used by other environments is a good candidate to be frozen in order to improve compilation performance.

An environment may only be frozen if all of its required environments (those environments on its Environment Search Path) are also frozen. If any of those required environments changes its state to *not frozen*, the state of the currently frozen environment will change to *invalidly frozen*. If an environment is invalidly frozen, it will behave as if it is not frozen until all its required environments are frozen again, whereupon the invalidly frozen environment must be re-frozen.

See also:

- NightBench Development - Chapter 5
- “Development - Settings” on page 5-18

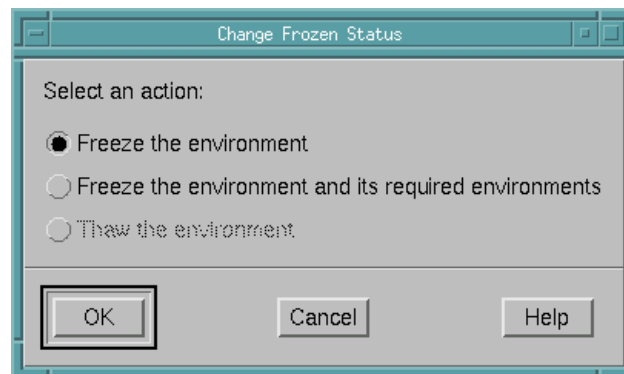


Figure 5-13. Change Frozen Status dialog

Freeze the environment

Change the state of the selected environment so that it is unalterable. This may only be done if all its required environments (those on the Environment Search Path for the current environment) are frozen.

See also:

- “Frozen Status” on page 5-20

Freeze the environment and its required environments

Change the state of the current environment and all environments on its Environment Search Path so that they are unalterable. An environment may only be frozen if all its required environments (those on its Environment Search Path) are also frozen.

See also:

- “Frozen Status” on page 5-20
- “Environment Search Path” on page 5-23

Thaw the environment

Change the state of the current environment so that it may be altered.

This option is only available to environments that are already frozen.

See also:

- “Frozen Status” on page 5-20

Change Permissions

The Change Permissions dialog allows the user to modify the permissions of all files associated with the current NightBench environment.

See also:

- NightBench Development - Chapter 5
- “Development - Settings” on page 5-18

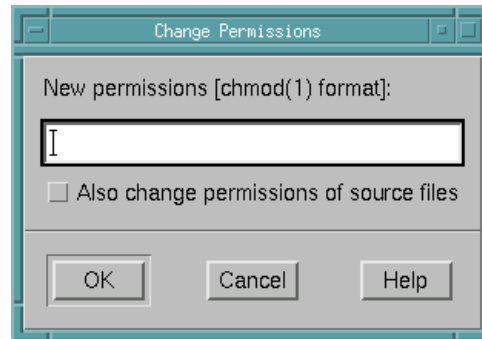


Figure 5-14. Change Permissions dialog

New permissions [chmod(1) format]

The permissions specified here will be applied to all files associated with the current NightBench environment.

Also change permissions of source files

In addition to the internal NightBench files for this environment, the specified permissions will be applied to source files and partition targets as well.

Add Environment Search Path Element

New environments are added to the Environment Search Path using this dialog. Positioning of the environments on the Environment Search Path can play an important role as to which units will be used when the Builder is invoked. Units not contained within the current environment will be sought by traversing the Environment Search Path. The environments are searched in the order they appear on the Environment Search Path until the desired unit is found.

See also:

- NightBench Development - Chapter 5
- “Development - Settings” on page 5-18

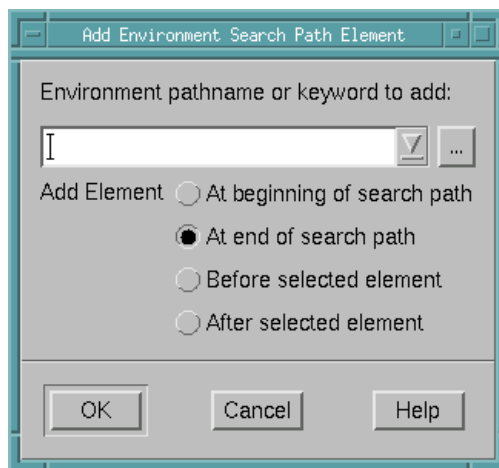


Figure 5-15. Add Environment Search Path Element dialog

Environment pathname or keyword to add

Enter the pathname of the environment to be added to the Environment Search Path. This can be a full pathname, a relative pathname, or a *keyword*.

Keywords are simply shorthand for *environments*. The drop-down menu provides a list of the available keywords.

...

Use this to select the environment pathname. A standard file dialog is displayed in order to navigate to the directory of the desired environment.

At beginning of search path

The environment specified is added to the beginning of the Environment Search Path.

At end of search path

The environment specified is added to the end of the Environment Search Path.

This is the default selection.

Before selected element

The environment specified in this dialog will be added to the Environment Search Path before the environment selected on the **Settings** page.

After selected element

The environment specified in this dialog will be added to the Environment Search Path after the environment selected on the **Settings** page.

Replace Environment Search Path Element

Environments already on the Environment Search Path may be replaced by a different environment using this dialog. The new environment is placed at the position previously held by the environment which is being replaced.

See also:

- NightBench Development - Chapter 5
- “Development - Settings” on page 5-18

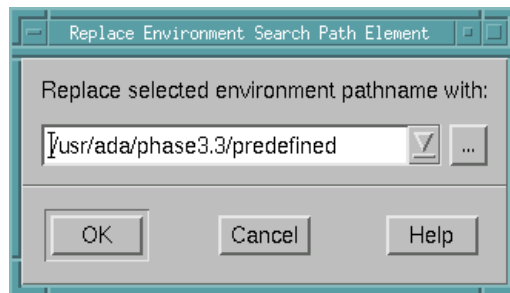


Figure 5-16. Replace Environment Search Path Element dialog

Replace selected environment pathname with

The environment specified in this dialog will replace the environment selected on the Settings page in the Environment Search Path.

...

Use this to select the environment pathname. A standard file dialog is displayed in order to navigate to the directory of the desired environment.

Development - Source Files

The Source Files page shows information about all the source files which contain units in the current environment. This page lists each source file name, the date it was last modified, the number of units contained within it, and whether or not it is to be preprocessed. This list may be sorted in various ways. Commands are supplied to introduce new source files, remove existing source files, or replace current source files with other source files. Also, a utility is provided to introduce a number of source files at once, using a file that contains a list of the names of those source files.

See also:

- NightBench Development - Chapter 5

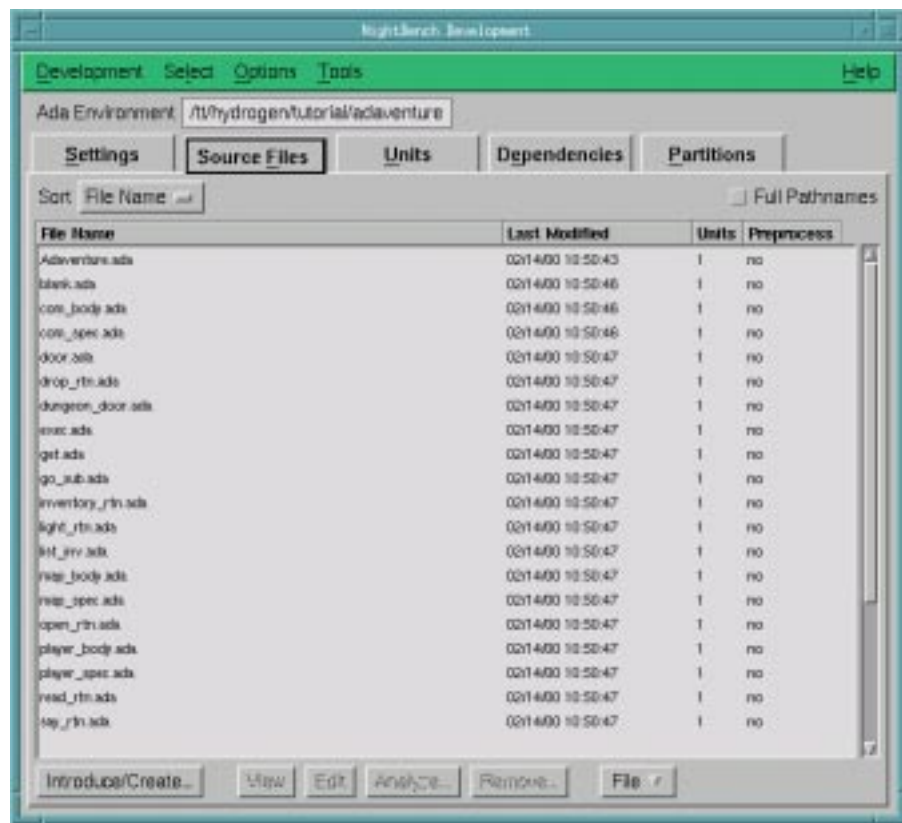


Figure 5-17. NightBench Development - Source Files page

Sort

NightBench allows for sorting the list of source files either alphabetically by file name or chronologically by the date last modified.

Full Pathnames

When this option is selected, the absolute pathname of each of the source files is shown. Otherwise, a relative pathname is displayed.

Source Files List

A listing of all the source files that have been introduced into the current environment.

See also:

- “Accelerated Item Selection” on page 5-16

File Name

The name of the source file and, if **Full Pathnames** is selected, its corresponding path.

Last Modified

The date and time this file was last modified.

Units

Displays the number of units contained within the given source file.

If the file contains only *configuration pragmas*, “C.P.” will be displayed in this column.

Preprocess

(Ada only)

Indicates whether or not a particular unit is to be preprocessed.

Introduce/Create...

Source files, and the *units* therein, must be introduced into the environment before they can be used. This opens a dialog in which the files to be introduced can be selected.

This dialog may also be used to create new files to introduce into the environment.

(Ada only) Ada source files will be analyzed for errors before they are introduced into the environment. If errors are encountered, the **Analyze Errors** dialog will be presented, showing the errors found.

See also:

- “Introduce Source Files” on page 5-37
- “Preferences - Editor” on page 1-9

- “Analyze Errors” on page 5-53

View

Loads the selected files into the configured file viewer.

See also:

- “Preferences - Viewer” on page 1-8

Edit

Loads the selected files into the configured file editor, allowing changes to be made to the source files.

The source file will be analyzed for errors after it has been saved. If errors are encountered, the **Analyze Errors** dialog will be presented, showing the errors found.

See also:

- “Preferences - Editor” on page 1-9
- “Analyze Errors” on page 5-53

Analyze...

(Ada only)

Analyzes the source file for large-scale changes (e.g. removal or addition of units, removal or addition of dependencies) and makes appropriate adjustments to the NightBench internal mechanisms.

See also:

- “Analyze Source Files - (Ada Only)” on page 5-51

Remove...

Removes knowledge of the selected source (and units therein) from the current environment. These source files may be re-introduced at a later time (but will be recreated in an uncompiled state).

An option to remove the actual source files is also given, thereby eliminating the physical source files from the system.

See also:

- “Remove Source Files” on page 5-56

File

(Ada only)

Contains a drop-down list of utilities for multiple file processing.

See also:

- “Source File Commands - (Ada Only)” on page 5-35

Source File Commands - (Ada Only)

The Source File Commands menu appears on the Source Files page of the NightBench Development window. It is labeled on the Source Files page as File and contains a drop-down list of utilities for multiple file processing.

See also:

- NightBench Development - Chapter 5
- “Development - Source Files” on page 5-31

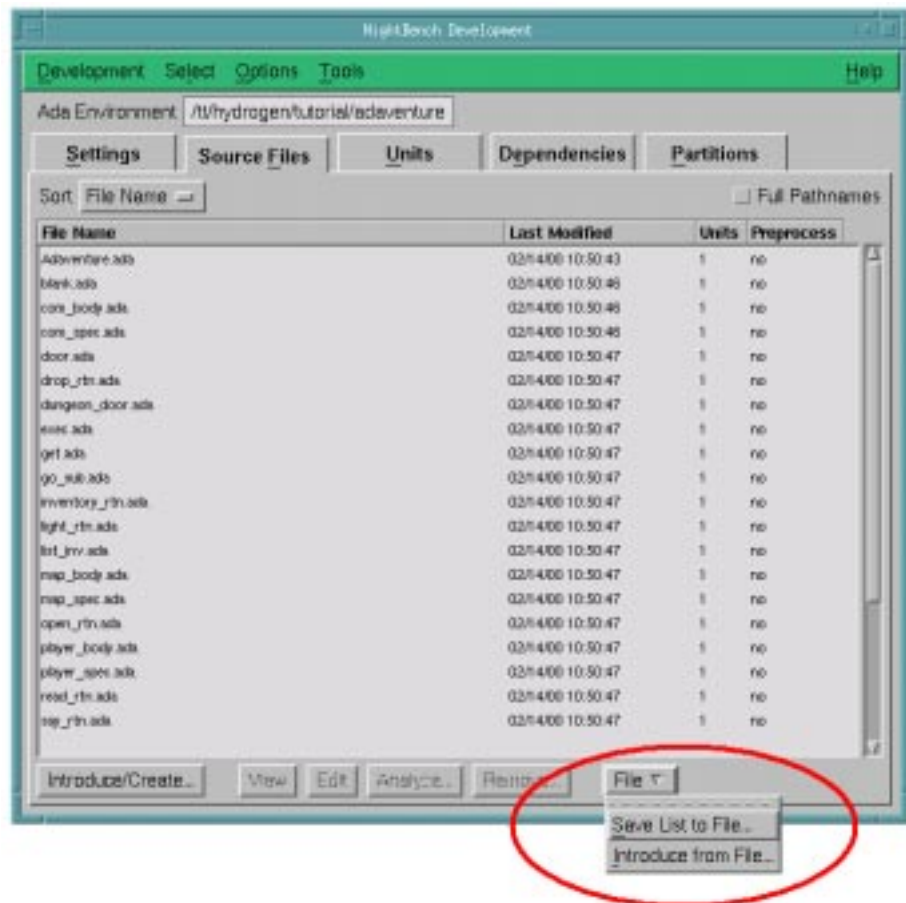


Figure 5-18. Source File Commands menu

Save List to File...

Use this to save the list of source files contained within the current environment to a file. This file can then be used by the Introduce from File... option at a later time.

See also:

- “Save List to File” on page 5-57

Introduce from File...

Use this to introduce a number of source files at once. This requires a file that contains the relative or absolute pathnames of each of the source files to be introduced. A file of this format is created using the **Save List to File...** option.

See also:

- “Introduce from File” on page 5-59

Introduce Source Files

Source files are introduced to NightBench using this dialog. Source files must be introduced into NightBench before any actions can be performed on the units that are contained within them.

This dialog displays the list of files within the current directory. A filter is supplied to narrow the selection. Selected files are shown in the **Selection** field, whereupon they may be added to the list of **Files to Introduce**. Once all the desired source files are added to the **Files to Introduce** list, they can be introduced en masse.

To create a new file to be introduced into the environment, enter the name of the new file in the **Selection** field and press the **Create** button.

See also:

- NightBench Development - Chapter 5
- “Development - Source Files” on page 5-31

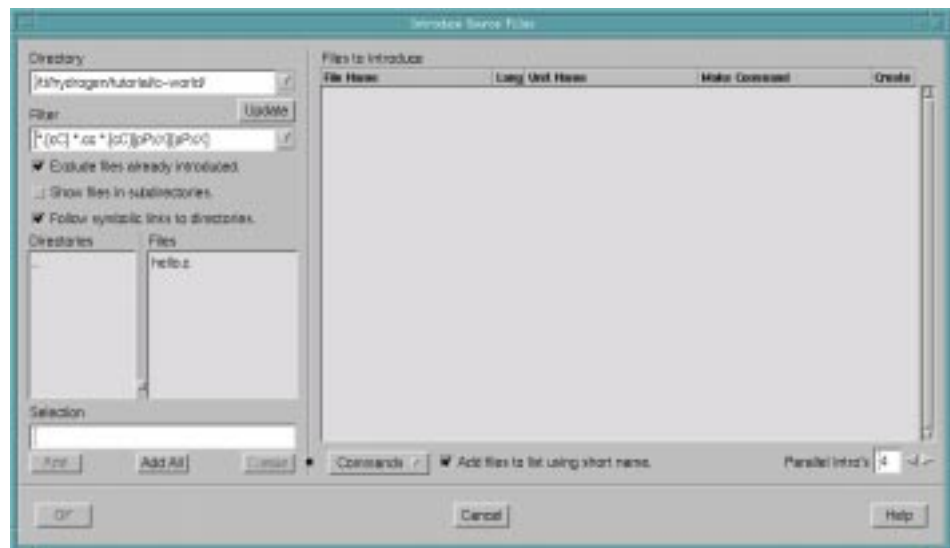


Figure 5-19. Introduce Source Files dialog

Directory

The current directory in which to look for source files.

The user may select a recently visited directory using the drop-down list for this field.

Alternately, the user may type the name of a directory into this field directly. In this case, the user must press the **Update** button to refresh the items on this dialog.

Update

Allows the list of **Files** to be updated with respect to the current **Directory** and **Filter** and other related settings in this dialog.

For instance, if a subdirectory in the **Directories** list is selected, performing an update will display in the **Files** list those files within that directory that match the **Filter**.

Filter

Of all the files contained in the current **Directory**, display only those that match the specified filter. After a filter is selected, the list of **Files** is automatically updated.

The default filter for Ada includes all files with the following suffixes:

- **.a**
- **.ada**
- **.adb**
- **.ads**
- **.pp**

The default filter for C and C++ files includes all files with the following suffixes:

- **.c**
- **.C**
- **.cc**
- **.cpp**
- **.CPP**
- **.cxx**
- **.CXX**

Exclude files already introduced

When this checkbox is selected, the names of those files already introduced into the current environment will not be displayed in the list of **Files**.

Show files in subdirectories

When this checkbox is selected, all files that match the specified **Filter** within the directory specified by the **Directory** field and *all its subdirectories* (recursively) will be displayed in the list of **Files**. All such files will be displayed with their pathnames relative to the **Directory**.

Follow symbolic links to directories

When this checkbox is selected, those directories that are symbolic links will also appear in the the Directories list.

Directories

Contains a list of all the subdirectories within the current directory. Selecting any of these changes the current **Directory** to that subdirectory. Double-clicking on any of these directory names will change to that directory and update the **Files** list accordingly.

Files

Within the current **Directory**, this lists all files that match the specified **Filter**. Any of these files can be selected. When selected, the filename appears in the **Selection** field.

You may use the **Exclude files already introduced**, **Show files in subdirectories**, and **Follow symbolic links to directories** checkboxes to modify the content of this list.

Double-clicking a file in this list adds it to the **Files to Introduce** list.

Multiple files may be selected by dragging the mouse pointer over contiguous files, or by holding the <CTRL> key down while selecting the files. When multiple files are selected from the this list, the text **Multiple files selected** will be displayed in the **Selection** field.

Selection

The current file to be added to the list of **Files to Introduce**.

Selections made in the **Files** list will show up here. If multiple files are selected from the **Files** list, the text **Multiple files selected** will be displayed.

Only one filename at a time can be manually entered into this field before it is added to the list of **Files to Introduce**.

Add

Adds the current files within the **Selection** field to the **Files to Introduce** list.

Add All

Adds all files contained within the **Files** list to the **Files to Introduce** list.

Create

Adds the name of the file specified in the **Selection** field to the **Files to Introduce** list and a “**Yes**” in the **Create** column of this list. Before this file is introduced, the configured file editor is opened to create the file.

NOTE

You must specify the name of the file you wish to create *before* you press this button.

See also:

- “Preferences - Editor” on page 1-9

Files to Introduce

Contains a list of all the files selected for introduction into the current environment.

File Name

The name of the source file that is to be introduced into the current environment.

See also:

- “Edit file name” on page 5-45
- “Edit File Name” on page 5-47

Preprocess

(Ada only)

Indicates that every source file in the list of Files to Introduce should be preprocessed.

See also:

- “Preprocessing” on page 5-45

Lang

(C/C++ only)

Specifies whether the source file will be compiled with the C or the C++ compiler.

See also:

- “Set language” on page 5-46

Unit Name

(C/C++ only)

The name of the unit associated with this source file.

See also:

- “Edit unit name” on page 5-46
- “Edit Unit Name” on page 5-48

Make Command

(C/C++ only)

The command invoked before building the unit associated with this source file.

See also:

- “Edit make command” on page 5-45
- “Edit Make Command” on page 5-49

Create

Displays the text “**Yes**” if this file needs to be created before it is introduced into the environment. The configured file editor will be opened for each file with this designation before it is introduced.

See also:

- “Preferences - Editor” on page 1-9
- “Create” on page 5-39

Commands

This menu provides various actions that may be performed on the units selected in the Files to Introduce list before they are introduced into the environment.

See also:

- “Commands” on page 5-43

Add files to list using short name

This option adds the file specified in the Selection field to the list of Files to Introduce using a heuristic based on how “far away” the file is from the environment. It introduces files “near” the environment using relative pathnames and files “far” from the environment using absolute pathnames.

Once added to the list of Files to Introduce, the pathname may be modified by selecting the desired file from the list of Files to Introduce and using either the Use absolute file name (see “Use absolute file name” on page 5-44) or Use relative file name (see “Use relative file name” on page 5-44) option from the Commands menu (see “Commands” on page 5-41). The Use shortest file name (see “Use shortest file name” on page 5-43) modifies the pathname of the selected file using the same heuristic as the Add files to list using short name option.

Parallel Intro's

Units can be introduced in parallel, thereby better utilizing the available CPUs. The number of parallel introductions defaults to the number of CPUs on the system.

NOTE

In practice, specifying a number larger than twice the number of CPUs on the system is probably counterproductive.

Commands

The following commands operate on the files selected in the list of **Files to Introduce** before they are introduced into the environment.

See also:

- “Introduce Source Files” on page 5-37
- “Development - Source Files” on page 5-31
- NightBench Development - Chapter 5

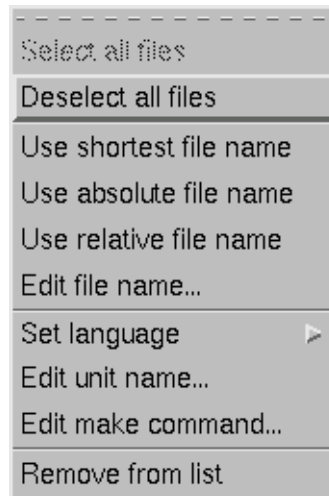


Figure 5-20. Introduce Source Files - Commands menu

Select all files

Selects all the files in the list of **Files to Introduce**. The items in the **Commands** menu operate on the items selected in the **Files to Introduce** list.

Deselect all files

Deselects all the files in the list of **Files to Introduce**.

Use shortest file name

NightBench keeps track of whether you introduce files with fully-specified pathnames or with relative ones. This option determines how “far away” each of the selected items within the list of **Files to Introduce** is and will introduce “near” ones with relative pathnames and “far” ones with absolute pathnames.

In most cases, relative pathnames work well for files “near” the environment (for example if the source files are in the same directory as the environment, or fairly close by - in a subdirectory or a parent directory, for instance).

Conversely, fully-rooted pathnames are useful for files that are "far away" from the environment. In such cases, relative pathnames may not be particularly helpful. For instance, a relative pathname of:

```
../../../../this/directory/is/far/away
```

might be more clearly stated using an absolute pathname.

You may also use either the **Use absolute file name** or the **Use relative file name** option to directly specify your choice.

In addition, the **Add files to list using short name** checkbox allows NightBench to select the shorter by default when adding files to the list of **Files to Introduce**.

Use relative file name

NightBench keeps track of whether you introduce files with fully-specified pathnames or with relative ones.

This option makes the pathnames for the selected items within the list of **Files to Introduce** relative to the directory containing the current environment.

In most cases, relative pathnames work well for files "near" the environment (for example if the source files are in the same directory as the environment, or fairly close by - in a subdirectory or a parent directory, for instance).

This may be desirable so that the source files and the environment can be copied to a different location at some time in the future, whereupon the copied environment will reference the copied source files instead of the originals.

You may also use either the **Use absolute file name** to specify that NightBench uses the fully-rooted pathname when introducing the files selected in the list of **Files to Introduce**. You may also use the **Use shortest file name** option to let NightBench decide which should be used.

In addition, the **Add files to list using short name** checkbox allows NightBench to select the shorter by default when adding files to the list of **Files to Introduce**.

Use absolute file name

NightBench keeps track of whether you introduce files with fully-specified pathnames or with relative ones.

This option introduces the selected items within the list of **Files to Introduce** with absolute pathnames.

This may be desirable if the source files reside in a location so far away from the environment that relative pathnames would be ineffective (see the example below). Also, when using absolute pathnames, the environment can be copied to a different location at some time in the future and still reference the source files in their original location.

Fully-rooted pathnames are useful for files that are "far away" from the environment. In such cases, relative pathnames may not be particularly helpful. For instance, a relative pathname of:

```
../../../../this/directory/is/far/away
```

might be more clearly stated using an absolute pathname.

You may also use either the **Use relative file name** to specify that NightBench uses the relative pathname when introducing the files selected in the list of **Files to Introduce**. You may also use the **Use shortest file name** option to let NightBench decide which should be used.

In addition, the **Add files to list using short name** checkbox allows NightBench to select the shorter by default when adding files to the list of **Files to Introduce**.

Edit file name

Allows the user to change the name of the file selected in the list of **Files to Introduce** before it is introduced.

See also:

- “Edit File Name” on page 5-47

Preprocessing

(Ada only)

Allows preprocessing of source files before introduction into the environment.

Auto	specifies that only source files with a .pp suffix are pre-processed
Yes	specifies that every source file in the list of Files to Introduce should be preprocessed
No	specifies that preprocessing should not be performed for any source file in the list of Files to Introduce

Edit make command

A "Make Command" may be associated with a source file when it is introduced. The specified command will be invoked before building the unit associated with this source file. The timestamp of the source file is then checked to determine if the unit(s) built from the source file need to be (re)compiled.

See also:

- “Edit Make Command” on page 5-49

Set language

(C/C++ only)

Allows the user to specify the language for the source files selected in the list of **Files to Introduce** so that the appropriate compiler will be used.

Default	The Default Language associated with files is specified on the Settings page (see “Default Language” on page 5-20)
C	Specifies that the source files selected in the list of Files to Introduce will be compiled with the C compiler
C++	Specifies that the source files selected in the list of Files to Introduce will be compiled with the C++ compiler

Edit unit name

(C/C++ only)

Allows the user to change the name of the unit associated with the selected file before it is introduced.

By default, the unit name corresponds directly to the name of the file being introduced. For instance, the unit corresponding to a source file `hello.c` would be named `hello`. However, the user may specify a different unit name when introducing a source file.

Multiple uniquely-named units may be associated with a particular source file. For instance, in a particular environment, it might be useful to have different versions of the same source file each with a different set of compilation options.

See also:

- “Edit Unit Name” on page 5-48

Remove from list

Removes the selected files from the list of **Files to Introduce**.

Edit File Name

Allows the user to change the name of the file selected in the list of Files to Introduce before it is introduced.

NOTE

This dialog does not rename existing files. For existing files that the user wishes to Add to the environment (as opposed to those that the user uses the Introduce Source Files dialog to Create), a file with the name specified in this dialog must already exist or NightBench will generate an error.

See also:

- “Commands” on page 5-43
- “Introduce Source Files” on page 5-37
- “Development - Source Files” on page 5-31
- NightBench Development - Chapter 5

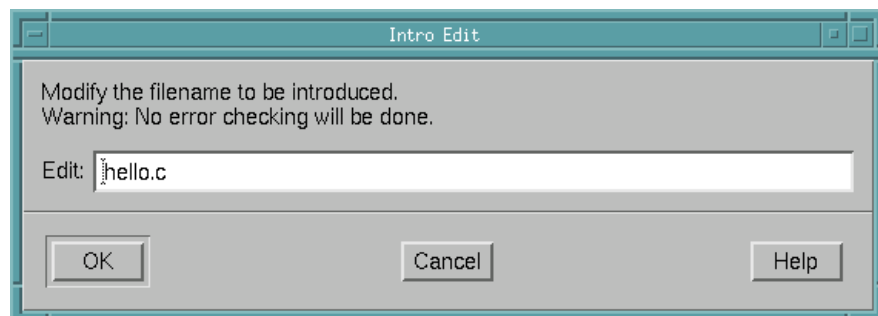


Figure 5-21. Intro Edit File Name dialog

Edit Unit Name

Allows the user to change the name of the unit associated with the selected file before it is introduced.

By default, the unit name corresponds directly to the name of the file being introduced. For instance, the unit corresponding to a source file `hello.c` would be named `hello`. However, the user may specify a different unit name when introducing a source file.

Multiple uniquely-named units may be associated with a particular source file. For instance, in a particular environment, it might be useful to have different versions of the same source file each with a different set of compilation options.

See also:

- “Commands” on page 5-43
- “Introduce Source Files” on page 5-37
- “Development - Source Files” on page 5-31
- NightBench Development - Chapter 5

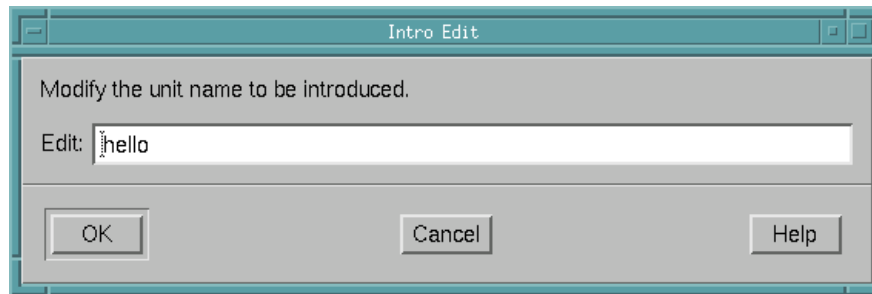


Figure 5-22. Intro Edit Unit Name dialog

Edit Make Command

A "Make Command" may be associated with a source file when it is introduced. The specified command will be invoked before building the unit associated with this source file. The timestamp of the source file is then checked to determine if the unit(s) built from the source file need to be (re)compiled.

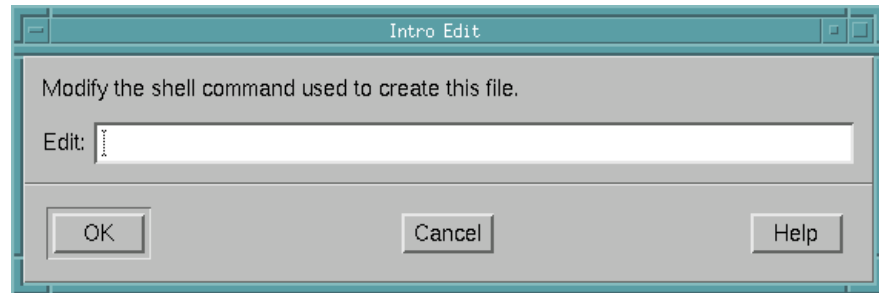


Figure 5-23. Intro Edit Make Command dialog

For instance, a "Make Command" of:

```
make pebbles.c
```

may be specified using the Edit Make Command dialog when creating the source file **pebbles.c** from the Introduce Source Files dialog.

This "Make Command" will then be associated with the unit **pebbles** (the default name of the unit associated with the source file **pebbles.c**) when the source file is introduced into the environment.

Assume the following **Makefile** exists in the environment:

```
pebbles.c: fred.c wilma.c
cat fred.c wilma.c > pebbles.c
```

Screen 5-1. Sample Makefile for pebbles.c Make Command

When the unit **pebbles** is built, the "Make Command" associated with it is invoked (in this example, the command **make pebbles.c**) before the unit is even attempted to be built. Since the **Makefile** in this environment specifies that if **pebbles.c** is out-of-date, the files **fred.c** and **wilma.c** will be concatenated together to create the file **pebbles.c**.

The first time the unit is built, **pebbles.c** will be created and the unit **pebbles** will be built by NightBench.

If the build is attempted again for the unit **pebbles**, the "Make Command" associated with it is invoked, which announces that **pebbles.c** is already up to date, and NightBench will not attempt to build the unit since the source file associated with that unit was unchanged.

If anything occurs which brings **pebbles.c** out of date (e.g. **fred.c** or **wilma.c** changes, **pebbles.c** is erased, etc.), the “Make Command” will create a new **pebbles.c** and the unit `pebbles` will be built again.

See also:

- “Commands” on page 5-43
- “Introduce Source Files” on page 5-37
- “Development - Source Files” on page 5-31
- NightBench Development - Chapter 5

Analyze Source Files - (Ada Only)

This dialog is used to analyze source files in order to determine the units that exist in that source file and the dependencies related to those units. The effect is very similar to the introduction of a source file, except that it applies to source files that have been introduced previously. If errors are encountered during this analysis, the following dialog is presented showing the errors encountered. The user is given an opportunity to repair the errors directly from this dialog by selecting the error, bringing its corresponding source file into the configured file editor, and fixing the code.

See also:

- “Preferences” on page 1-8
- “Analyze Errors” on page 5-53
- NightBench Development - Chapter 5
- “Development - Source Files” on page 5-31

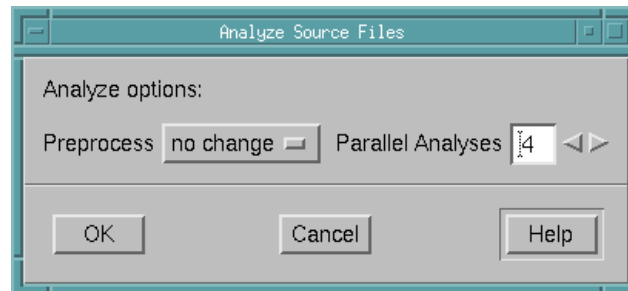


Figure 5-24. Analyze Source Files dialog

Preprocess

Allows preprocessing during analysis of source files.

no change	preprocess if the file has previously been preprocessed
yes	preprocess this file before analysis
no	preprocessing should not be performed on this source file before analysis

Parallel Analyses

Units can be analyzed in parallel, thereby better utilizing the available CPUs. The number of parallel introductions defaults to the number of CPUs on the system.

NOTE

In practice, specifying a number larger than twice the number of CPUs on the system is probably counterproductive.

Analyze Errors

When a source file is first introduced into the NightBench Program Development Environment, it is analyzed to determine the units that exist in that source file and the dependencies related to those units. In addition, every time a source file or unit is edited and subsequently saved using the NightBench configured file editor, the file is analyzed before NightBench will accept it. Somewhat forgiving, some syntax errors may be ignored at this time if NightBench can determine the information it requires. During compilation, a more complete analysis is done.

If errors are encountered during the analysis that is performed when the source file is first introduced or when a unit is edited, the following dialog is presented showing the errors encountered. The user is given an opportunity to repair the errors directly from this dialog by selecting the error, bringing its corresponding source file into the configured file editor, and fixing the code.

HINT

If a number of errors are presented, it is recommended to start at the last error and use the **Edit Previous** button to traverse through the errors. Otherwise, fixing errors that appear earlier in the file by deleting or adding lines may result in improper positioning when the **Edit Next** button is subsequently used.

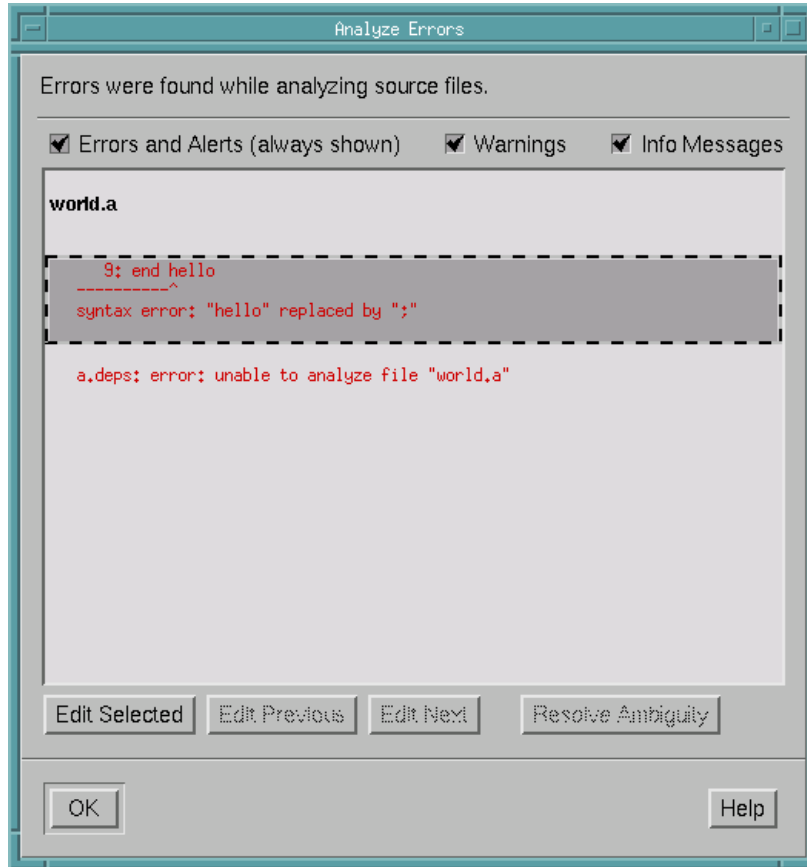


Figure 5-25. Analyze Errors dialog

Errors and Alerts (always shown)

Errors and alerts encountered during the analysis are always shown in the Errors Window.

Warnings

When selected, warning messages that occurred during the analysis are displayed.

Info Messages

When selected, info messages that occurred during the analysis are displayed.

Errors Window

The Errors Window shows errors, alerts, warnings and info messages that were found during the analysis.

Edit Selected

Opens the configured file editor to the location where the selected error occurred.

See also:

- “Preferences - Editor” on page 1-9

Edit Previous

Allows the user to step backward through the errors in the **Errors Window**.

Opens the source file in the configured file editor to the location where the error previous to the currently selected error occurred.

See also:

- “Preferences - Editor” on page 1-9

Edit Next

Allows the user to step forward through the errors in the **Errors Window**.

Opens the source file in the configured file editor to the location where the error following the currently selected error occurred.

See also:

- “Preferences - Editor” on page 1-9

Resolve Ambiguity

Brings up the **Resolve Ambiguity** dialog in order to choose which of the source files containing *ambiguous units* should be used.

This button is activated when a file is analyzed and it is found that it contains a unit having the same name as another unit in the current environment.

See also:

- “Resolve Ambiguity” on page 5-74

Remove Source Files

This action is performed on source files that are no longer to be considered part of the current environment. All information about the selected source files (and all units contained therein) will be removed from NightBench.

See also:

- NightBench Development - Chapter 5
- “Development - Source Files” on page 5-31

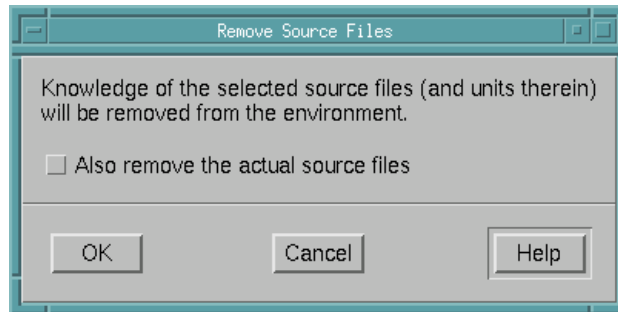


Figure 5-26. Remove Source Files dialog

Also remove the actual source files

Delete the physical source files in addition to any information used internally by NightBench about the selected source files (and the units contained therein).

Save List to File

The pathnames of all the source files contained within the current environment are saved to the file specified by this dialog. This file can later be used by the **Introduce from File** dialog to introduce all of these source files into an environment at one time.

See also:

- “Introduce from File” on page 5-59
- NightBench Development - Chapter 5
- “Development - Source Files” on page 5-31

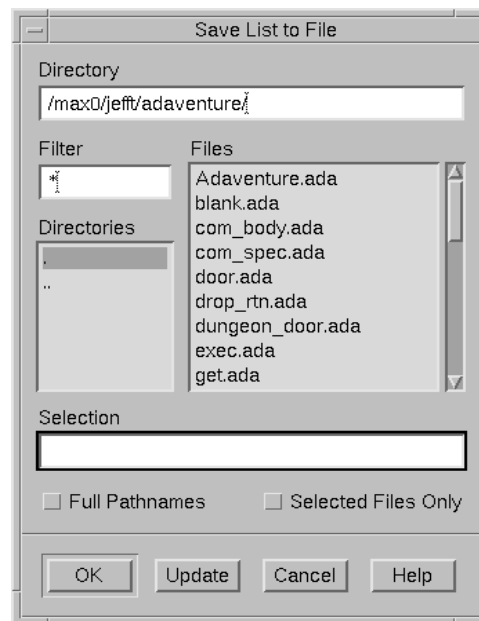


Figure 5-27. Save List to File dialog

Directory

The directory in which to place the file containing the list of source files within the current environment.

The user may type the name of a directory into this field directly. In this case, the user must press the **Update** button to refresh the items on this dialog.

Filter

Of all the files contained in the current **Directory**, display only those that match the specified filter.

After a filter is entered, the **Update** button should be pressed to update the list of **Files**.

Directories

Contains a list of all the subdirectories within the current directory. Selecting any of these changes the current **Directory** to that subdirectory. Double-clicking on any of these directory names will change to that directory and update the **Files** list accordingly.

Files

Within the current **Directory**, this is a list of the files that match the specified **Filter**. Any of these filenames can be selected. When selected, the filename appears in the **Selection** field.

Selection

The current file in which to save the list of source files contained within the current environment.

A selection made in the **Files** list will show up here.

The actual filename can be manually entered into this field.

Full Pathnames

Specifies whether the full pathname should also be saved to the file in addition to the name of the source file.

Selected Files Only

Only save the names of the files currently selected in the list on the **Source Files** page.

See also:

- “Development - Source Files” on page 5-31

Update

Update the list of **Files** based on the information entered in the **Directory** and **Filter** fields.

Introduce from File

A number of files may be introduced into NightBench at one time by using this dialog. The pathnames of the files to be introduced are contained within a file (most likely created by the **Save List to File** option).

See also:

- “Save List to File” on page 5-57
- NightBench Development - Chapter 5
- “Development - Source Files” on page 5-31

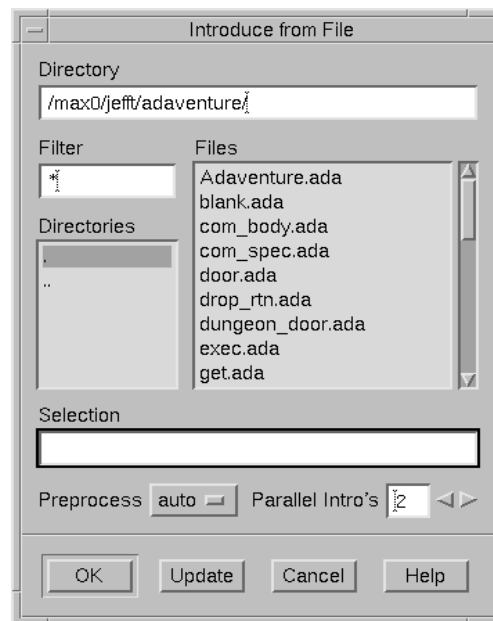


Figure 5-28. Introduce from File dialog

Directory

The current directory in which to look for the file containing the list of files to be introduced into the current environment.

The user may type the name of a directory into this field directly. In this case, the user must press the **Update** button to refresh the items on this dialog.

Filter

Of all the files contained in the current **Directory**, display only those that match the specified filter.

After a filter is entered, the **Update** button should be pressed to update the list of **Files**.

Directories

Contains a list of all the subdirectories within the current directory. Selecting any of these changes the current **Directory** to that subdirectory. Double-clicking on any of these directory names will change to that directory and update the **Files** list accordingly.

Files

Within the current **Directory**, this is a list of the files that match the specified **Filter**. Any of these filenames can be selected. When selected, the filename appears in the **Selection** field.

Selection

The current file from which to obtain the list of source files to be introduced into the current environment.

A selection made in the **Files** list will show up here.

The actual filename may be manually entered into this field.

Preprocess

Allows preprocessing of source files before introduction into NightBench.

Auto	specifies that only source files with a .pp suffix are pre-processed
Yes	specifies that every source file in the list should be pre-processed
No	specifies that preprocessing should not be performed for any source file in the list

Parallel Intro's

Units can be introduced in parallel, thereby better utilizing the available CPUs. The number of parallel introductions defaults to the number of CPUs on the system.

NOTE

In practice, specifying a number larger than twice the number of CPUs on the system is probably counterproductive.

Update

Update the list of **Files** based on the information entered in the **Directory** and **Filter** fields.

Development - Units

The Units page shows a list of all units contained within the current environment and information about them. This list may be sorted and filters may be applied to show only a particular set of units. In addition, the Units page allows access to the various commands which may be applied to selected units. Permanent and temporary unit options are displayed and can be manipulated either directly on the Units page or through the Options Editor.

See also:

- NightBench Development - Chapter 5
- “Unit Compile Options Editor” on page 7-24

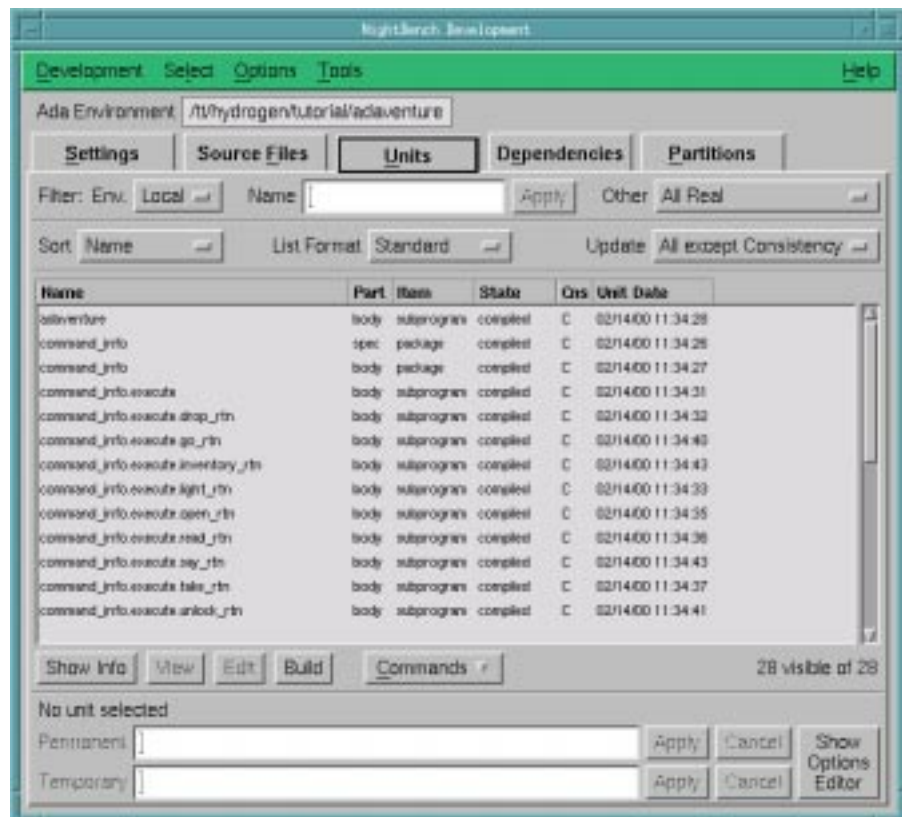


Figure 5-29. NightBench Development - Units page

Env.

Display in the Unit Information area only those units specified by the selected environment filter:

Local	display units in the current environment
User	display units in the current environment and those on the Environment Search Path, excluding units in environments shipped with MAXAda
All	display units in the current environment and those on the Environment Search Path, including units in environments shipped with MAXAda

Name

Display units in the Unit Information area that begin with the characters in this field. Press <RETURN> or Apply after entering the desired characters.

Apply

Apply the filter entered in the **Name** field to the Unit Information area, displaying only those units that begin with the characters in that field.

Other

Filter the Unit Information area by one of the following criteria:

All Real	display all units introduced to the environment
Consistently Compiled	display only those compiled units which are also consistent (C) or were consistent (C?)
Not Consist. Compiled	display only those units which are compiled but which are inconsistent (Inc), were inconsistent (Inc?), or have unknown consistency (?)
Spec	<i>(Ada only)</i> display only units which are <i>specifications</i>
Body	<i>(Ada only)</i> display only units which are <i>bodies</i>
All (including Artificial)	display all units in the environment including those created by the environment related to instances of <i>generics</i>

Sort

Sort the Unit Information area by one of the following criteria:

Name	Sort the units alphabetically by unit name
Unit Date	Sort the units chronologically by the date and time the unit last changed compilation state
Source Date	Sort the units based on the date of the last modification to their respective source files
Include Date	<i>(C/C++ only)</i> Sort the units based on the latest timestamp of the primary source file and any files that it includes.
Source File	Sort the units alphabetically based on the names of their respective source files
Language	<i>(C/C++ only)</i> Sort the units by the language in which they were written (C, C++)
State	Sort the units by compilation state
Part	<i>(Ada only)</i> Sort the units into the categories of spec and body
Item	<i>(Ada only)</i> Sort the units into the categories subprogram, package, task, protected
Class	<i>(Ada only)</i> Sort the units into groupings of library, subunit, and nested
Generic Info	<i>(Ada only)</i> Sort the units into groupings of generic , instance-of-... , generic-corresponding-to... , and n/a
Hidden	<i>(C/C++ only)</i> Sort the units based on their <i>hidden</i> status. Hidden units are those units that will not be considered for resolving <i>instantiation</i> requests.
Magnet	<i>(C/C++ only)</i> Sort the units based on their <i>magnet</i> status. Units specified as magnets will be preferred over others to host <i>artificial units</i> for <i>instantiations</i> .
Visa	Sort the units into categories of native , naturalized , fetchd , and foreign
Home Env.	<i>(Ada only)</i> Sort the units based on the environment in which the unit was compiled (including any naturalized copies)
Origin Env.	Sort the units based on the environment into which the unit was originally introduced (ignoring any naturalized copies)

List Format

In addition to the **Name** and **Part**, list the following in the Unit Information area:

Standard	display Item, State, and Unit Date
Source Files	display Source File
Options	display Permanent Options, Temporary Options, and Effective Options
Visa	display Visa, Home Env., and Origin Env.
Verbose	display Item, Class, State, Unit Date, Source File, Source Date, and Generic Info
Everything	display Item, Class, State, Unit Date, Source File, Source Date, Generic Info, Visa, Home Environment, Origin Environment, Permanent Options, Temporary Options, and Effective Options
Unit Only	display only the Unit Name and Portion

See also:

- “Unit Information” on page 5-87 for information about these fields

Update

This menu controls the extent to which NightBench keeps the information in the Unit Information area up-to-date.

All except Consistency	<i>(Ada only)</i> All information except the consistency column (Cns.) is kept up-to-date. This is the default because attempting to keep the consistency column (Cns.) up-to-date is very time-consuming.
All Information	All information is kept up-to-date. Any change to a unit may possibly affect its consistency or the consistency of other units in the environment. NightBench's performance may therefore be affected when this option is selected, since it will refresh the information whenever any changes are made.
None	No attempt is made to keep the list information up-to-date. This option is useful if you wish to issue a number of commands in a row and don't want NightBench to take the time to update until you're finished.

Unit Information

This area of the **Units** page displays the unit information as designated by the **Env.**, **Name**, and **Other** filters as well as the values specified by the **Sort**, **List Format**, and **Update** menus.

NOTE

A unit that is marked as *ambiguous* is designated by a different background color across the line for this unit.

See also:

- “Unit Information” on page 5-87
- “List Format” on page 5-64
- “Accelerated Item Selection” on page 5-16

Show Info

Brings up a window that displays the complete unit information for a selected unit. This will only show information for one unit at a time.

See also:

- “Unit Information” on page 5-87

View

Loads the files containing the selected units into the configured file viewer.

See also:

- “Preferences - Viewer” on page 1-8

Edit

Loads the files containing the selected units into the configured file editor.

See also:

- “Preferences - Editor” on page 1-9

Build

Opens the NightBench Builder with the selected units as the targets. The build may automatically start if either the **Automatically Start Builds** option is selected from the **Options** menu of the NightBench Development window or if this is specified in the **Preferences** dialog.

See also:

- NightBench Builder - Chapter 6
- “Automatically Start Builds” on page 5-14
- “Preferences - Start” on page 1-14

Commands

Displays a menu listing the actions that can be applied to the selected units.

See also:

- “Commands” on page 5-68

Permanent

Each unit has its own set of options permanently associated with it that override those specified for the environment. Permanent options can be specified here or by using the Unit Compile Options Editor.

See also:

- “Show Options Editor” on page 5-67
- “Unit Compile Options Editor” on page 7-24

Temporary

Each unit also has a set of options that may be temporarily associated with it that override those that are permanently associated with it. Temporary options can be specified here or by using the Unit Compile Options Editor.

Temporary options allow users to “try out” some options or change particular options for a specific compilation but only “temporarily”. The *temporary unit compile options* are a separate set of options maintained for this purpose. They are persistent in the same way that permanent options are persistent, however, they may be cleared easily without altering the permanent options.

See also:

- “Show Options Editor” on page 5-67
- “Unit Compile Options Editor” on page 7-24

Apply

Changes the unit options set to reflect any modifications made to the Permanent and Temporary fields.

Cancel

Resets the Permanent and Temporary options to their respective set of options previously applied to the selected units, disregarding any changes made since.

Show Options Editor

Opens the Unit Compile Options Editor.

See also:

- “Unit Compile Options Editor” on page 7-24

Commands

The **Commands** menu lists commands which may not be used as frequently but may still be applied to the selected units. However, items in this menu that are grayed out are not applicable to the currently selected units.

See also:

- NightBench Development - Chapter 5
- “Development - Units” on page 5-61



Figure 5-30. Commands menu

Invalidate -t

(C/C++ only)

Forces a unit to be inconsistent and clears automatic instantiations assigned to it, thus requiring the unit (and any units that depend on it) to be recompiled and the reselection of a unit to host the instantiation of those entities no longer instantiated by this unit.

See also:

- “Instantiations” on page 5-71

Invalidate

Forces a unit to be inconsistent thus requiring it and any units that depend on it to be recompiled.

See also:

- “Touch” on page 5-69 for the opposite functionality

Touch

Makes the environment consider a unit consistent with its source file's timestamp. This is usually done to keep it from being automatically rebuilt because of a semantically irrelevant source file change.

See also:

- “Invalidate” on page 5-68 for the opposite functionality

Set Language

(C/C++ only)

Sets the language for the particular unit to either C or C++ so that the appropriate compiler will be used.

C	compile the selected unit with the C compiler
C++	compile the selected unit with the C++ compiler

Resolve Ambiguity...

(Ada only)

NightBench detects the case when two versions of the same unit appear among all the source files introduced to the environment.

Upon introducing a unit having the same name and part (specification or body) as a previously introduced unit, NightBench labels both units as *ambiguous*. These units appear highlighted on the Units page of the NightBench Development window.

NightBench will then refuse to perform any operations on either of the two versions, or on any units depending on the ambiguous unit. The user will be forced to choose which of the two units should actually exist in the environment by “removing” the other.

See also:

- “Resolve Ambiguity” on page 5-74

Hide

(Ada only)

Marks the unit as being persistently hidden in the environment.

There are times when a source file may contain units other than those the user would like introduced into the environment. All units contained within a particular source file are automatically introduced into the environment (unless they have previously been hidden). In order to “remove” any unwanted units from the environment, this command is provided. After this action is performed on a unit, it is no longer visible to the environment.

See also:

- “Unhide...” on page 5-70 for the opposite functionality

Unhide...

(Ada only)

Bring up the **Hidden Units** dialog if there are hidden units in the current environment.

See also:

- “Hidden Units” on page 5-72
- “Hide” on page 5-69 for the opposite functionality

Fetch

Obtain the compiled form of a unit from another environment.

It may be desirable for users to be able to force copies of specified units from other environments into the current environment. This may be necessary in cases where the visibility to a particular unit is blocked by a different unit on the Environment Search Path. One such reason might arise out of a conflict with dependencies. Fetching a unit forces visibility to that particular unit because its compiled form will then reside in the current environment.

A fetched unit in the current environment is distinct from the original from which it was copied. In most respects, it is as though the unit was introduced directly to the current environment. In particular, it derives its compilation options from the current environment's *environment-wide compile options* set. The only perceptible differences from a directly introduced unit are that its visa is **fetched** and that it can be expelled.

See also:

- “Expel” on page 5-70 for the opposite functionality

Expel

Removes a *fetched* or *naturalized* unit from the local environment, thus restoring visibility to the foreign version.

See also:

- “Fetch” on page 5-70 for the opposite functionality

Update Consistency

(Ada only)

Determines the *consistency* of the selected units or of all units if none are selected.

Instantiations

(C/C++ only)

The C++ language includes the concept of *templates*. A template is a description of a class or function that is a model for a family of related classes or functions independent of the type of data being manipulated. From a single source description of the template, the compiler can create *instantiations* of the template for each of the types specified.

The Concurrent C++ compiler provides *automatic instantiation* during the prelinking phase of a C++ program that assigns instantiations of instantiatable entities (such as template functions or extern inlines) to specific compilation units in the program. The following dialogs provide different methods of controlling this process.

See also:

- “Invalidate -t” on page 5-68

Hide Status

Hide or unhide selected units from being considered for resolving *instantiation* requests.

See also:

- “Hide Status” on page 5-76

Magnet Status

Change the *magnet* status of selected units, specifying those units which will be preferred over others to host *artificial units* for *instantiations*.

See also:

- “Magnet Status” on page 5-78

Manual Control

Manually assign *instantiations* of instantiatable entities (such as template functions or extern inlines) to selected units.

See also:

- “Manual Instantiation Resolution” on page 5-80

Hidden Units

The Hidden Units dialog allows the user to unhide any previously hidden units. Units can become hidden by selecting the **Hide** option on the **Commands** menu of the **Units** Page or may become hidden when resolving ambiguities.

See also:

- “Hide” on page 5-69
- “Resolve Ambiguity...” on page 5-69
- NightBench Development - Chapter 5
- “Development - Units” on page 5-61

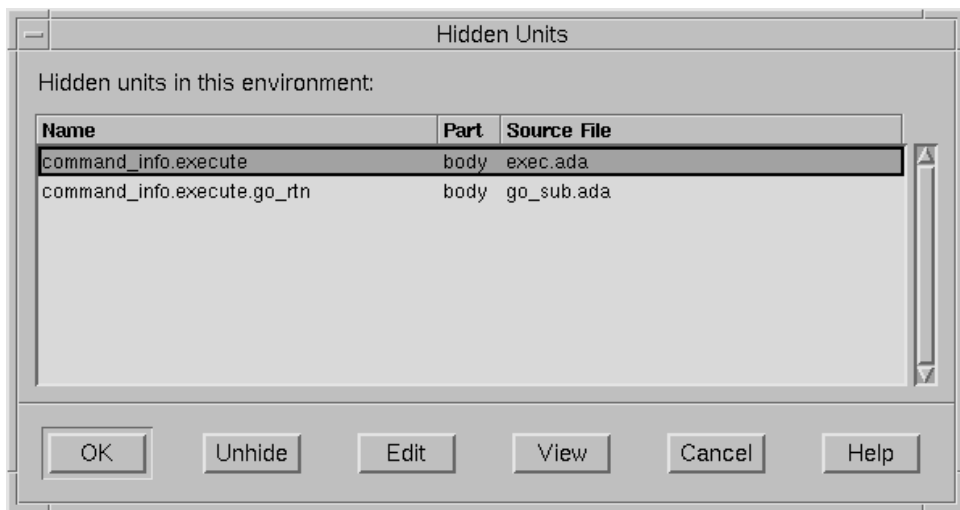


Figure 5-31. Hidden Units dialog

Name

The name of the unit.

Part

Units are identified by their name and by their *part*. The part can be designated as either **spec** or **body**.

See also:

- “Unit Information” on page 5-87

Source File

The name of the file in which this unit exists.

Double-clicking a list item will open the corresponding source file in the user's configured file editor.

See also:

- “Preferences - Editor” on page 1-9

OK

The dialog is closed after selected units are made visible to the environment.

Unhide

Selected units are made visible to the environment, leaving the dialog open.

Edit

Loads the files containing the selected units into the configured file editor.

See also:

- “Preferences - Editor” on page 1-9

View

Loads the files containing the selected units into the configured file viewer.

See also:

- “Preferences - Viewer” on page 1-8

Cancel

Closes the dialog without changing the visibility of the hidden units listed. Units previously made visible using the **Unhide** button in this dialog will remain visible.

Resolve Ambiguity

The **Resolve Ambiguity** dialog allows the user to select from which source file a unit should be chosen in order to resolve a situation involving *ambiguous units*. Once chosen, the other units are “hidden” from the environment so that there is not any further conflict between these units.

See also:

- NightBench Development - Chapter 5
- “Development - Units” on page 5-61

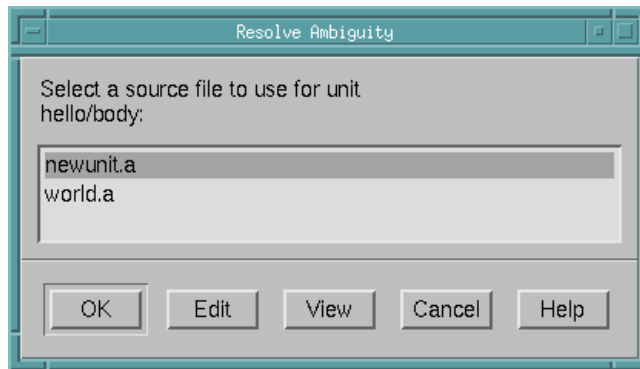


Figure 5-32. Resolve Ambiguity dialog

Source file list

Lists the source files in which the ambiguous units appear. Select one of these files to resolve the ambiguity.

Double-clicking a list item will open that file in the user’s configured file editor.

See also:

- “Preferences - Editor” on page 1-9

OK

Uses the unit from the selected source file in order to resolve the ambiguity. Other units are “hidden” from the environment so that there is not any further conflict between these units.

Edit

Loads the files containing the selected units into the configured file editor.

See also:

- “Preferences - Editor” on page 1-9

View

Loads the files containing the selected units into the configured file viewer.

See also:

- “Preferences - Viewer” on page 1-8

Cancel

Closes the dialog without selecting any source file to resolve the ambiguity.

Hide Status

This dialog allows the user to specify units which should not be considered for resolving *instantiation* requests.

Changing the hide status of a unit does not alter instantiation assignments already made within the environment. The `Invalidate -t` command may be used (see “Invalidate -t” on page 5-68) to clear instantiation assignments from particular units so that the environment may take these changes into account during *automatic instantiation*.

See also:

- “Instantiations” on page 5-71
- “Development - Units” on page 5-61

NOTE

The information displayed in this dialog will correspond to those units selected on the **Units** page when this dialog was opened. To see the hidden status of *all* units in the environment, you should **Deselect All** units on the **Units** page before opening this dialog (see “Deselect All” on page 5-13).

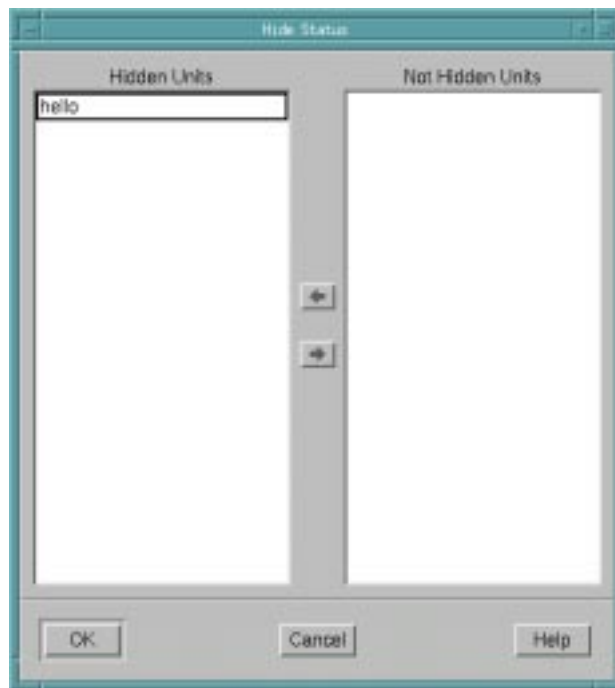


Figure 5-33. Hide Status dialog

Hidden Units

Unit(s) that will not be considered for resolving *instantiation* requests.



Moves the selected unit from the Not Hidden Units list to the Hidden Units list.



Moves the selected unit from the Hidden Units list to the Not Hidden Units list.

Not Hidden Units

Unit(s) that will be considered for resolving *instantiation* requests.

Magnet Status

This dialog allows the user to change the *magnet* status of a unit, specifying those units which will be preferred over others to host *artificial units* for *instantiations*.

Changing the magnet status of a unit does not alter instantiation assignments already made within the environment. The `Invalidate -t` command may be used (see “Invalidate -t” on page 5-68) to clear instantiation assignments from particular units so that the environment may take these changes into account during *automatic instantiation*.

See also:

- “Instantiations” on page 5-71
- “Development - Units” on page 5-61

NOTE

The information displayed in this dialog will correspond to those units selected on the **Units** page when this dialog was opened. To see the magnet status of *all* units in the environment, you should **Deselect All** units on the **Units** page before opening this dialog (see “Deselect All” on page 5-13).

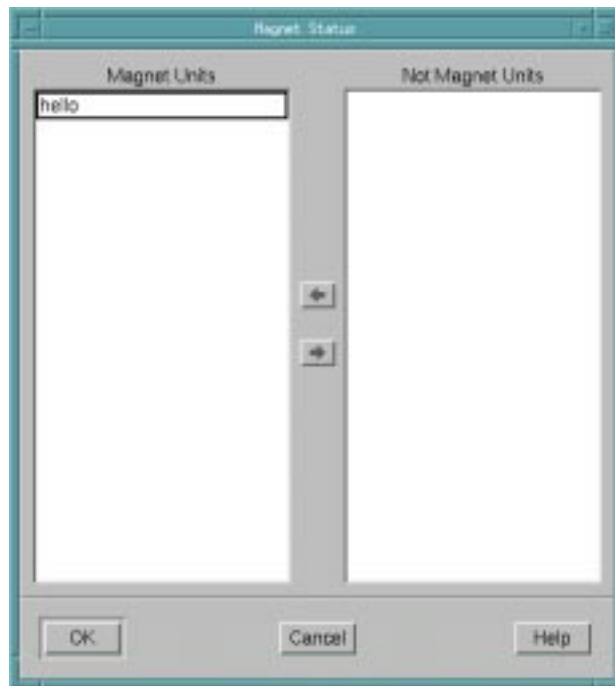


Figure 5-34. Magnet Status dialog

Magnet Units

Units which will be preferred over others to host *artificial units* for *instantiations*.



Moves the selected unit from the Not Magnet Units list to the Magnet Units list.



Moves the selected unit from the Magnet Units list to the Not Magnet Units list.

Not Magnet Units

Units that have no preferred status over others to host *artificial units* for *instantiations*.

Manual Instantiation Resolution

The Manual Instantiation Resolution dialog allows the user to manually assign *instantiations* of entities (such as template functions or extern inlines) to specific compilation units in the program. The user is allowed to make these associations by:

- selecting a compilation unit and manually assigning the instantiation of specific entities to that unit (see “Select Real Units” on page 5-80), or
- selecting an entity and manually assigning the instantiation of it to a specific compilation unit in the environment (see “Select Artificial Units” on page 5-83)

See also:

- “Instantiations” on page 5-71

Decode artificial unit names

By default, each instantiatable entity is listed by its unique internal name generated by the compiler. This checkbox allows the user to view each entity in a more human-readable form.

Select Real Units

This page of the Manual Instantiation Resolution dialog allows the user to select a compilation unit and manually assign the *instantiation* of specific entities (such as template functions or extern inlines) to that unit, possibly overriding the choices made by the compiler and prelinker during *automatic instantiation*. The **Select Artificial Units** page of this dialog provides the opposite functionality.

See also:

- “Manual Instantiation Resolution” on page 5-80
- “Select Artificial Units” on page 5-83



Figure 5-35. Manual Instantiation Resolution - Select Real Units

Real Units

The units in this list consist of those units selected on the Units page (see “Development - Units” on page 5-61) which contain entities that may be *instantiated*. To list *all* units in the environment that contain instantiatable entities, you must **Deselect All** units on the Units page before opening this dialog (see “Deselect All” on page 5-13).

Can Instantiate

This list contains every entity that could be *instantiated* by the compilation unit selected in the Real Units list. The user may explicitly assign the instantiation of this entity to the selected unit by moving the selected template entity to the Will Instantiate list.

By default, each instantiatable entity is listed by its unique internal name generated by the compiler. The Decode artificial unit names checkbox allows the user to view each entity in a more human-readable form (see “Decode artificial unit names” on page 5-80).



This button allows the user to *explicitly assign* the *instantiation* of the entity selected in the **Will Instantiate** list to the compilation unit selected in the **Real Units** list.

The instantiatable entity in the **Will Instantiate** list may have already been assigned to the unit either during *automatic instantiation* or due to an explicit instantiation in the source (such as through the `instantiate` pragma). This button will cause that assignment to persist even if automatic instantiations are cleared or explicit instantiations are removed from the source.



Moves the selected unit from the **Can Instantiate** list to the **Will Instantiate** list.



Moves the selected unit from the **Will Instantiate** list to the **Can Instantiate** list.

Will Instantiate

This list consists of every entity that *will* be *instantiated* by the compilation unit selected in the **Real Units** list.

By default, each instantiatable entity is listed by its unique internal name generated by the compiler. The **Decode artificial unit names** checkbox allows the user to view each entity in a more human-readable form (see “Decode artificial unit names” on page 5-80).


Each entity listed may be preceded by a special symbol that indicates the manner in which it was assigned to the selected unit. No symbol indicates that the instantiation

of the entity was assigned to the particular compilation unit during *automatic instantiation*.

* indicates that the entity has been specifically chosen by the user to be instantiated by the selected compilation unit.

The user may have either:

- moved the template entity from the **Can Instantiate** list to the **Will Instantiate** list, or

- used the  button to *explicitly assign* the *instantiation* of an entity already assigned to that unit. This entity may have been previously assigned to that unit during *automatic instantiation* or by an explicit instantiation in the source (such as through the `instantiate` pragma).

indicates that the instantiation was requested explicitly in the source (such as through the `instantiate` pragma)

The user may remove the explicit assignment by moving the selected entity to the **Can Instantiate** list. This cannot be done, however, for those entities requested explicitly in the source of a unit.

Select Artificial Units

This page of the **Manual Instantiation Resolution** dialog allows the user to select an entity and manually assign the *instantiation* of it to a specific compilation unit in the environment, possibly overriding the choices made by the compiler and prelinker during *automatic instantiation*. The **Select Real Units** page of this dialog provides the opposite functionality.

See also:

- “Manual Instantiation Resolution” on page 5-80
- “Select Real Units” on page 5-80



Figure 5-36. Manual Instantiation Resolution - Select Artificial Units

Artificial Units

The items in this list consist of entities in the environment which may be *instantiated*. These instantiatable entities are contained within those units selected on the Units page (see “Development - Units” on page 5-61). To list instantiatable entities for *all* units in the environment, you must **Deselect All** units on the Units page before opening this dialog (see “Deselect All” on page 5-13).

By default, each instantiatable entity is listed by its unique internal name generated by the compiler. The **Decode artificial unit names** checkbox allows the user to view each entity in a more human-readable form (see “Decode artificial unit names” on page 5-80).

Can be instantiated by

This list contains every compilation unit that could instantiate the entity selected in the Artificial Units list. The user may explicitly assign the *instantiation* of this template instance to the selected unit by moving the selected compilation unit to the Will be instantiated by list.



This button allows the user to *explicitly assign* the compilation unit selected in the Will be instantiated by list to instantiate the template entity selected in the Artificial Units list.

The entity may have already been assigned to the unit either during *automatic instantiation* or due to an explicit instantiation in the source (such as through the `instantiate` pragma). This button will cause that assignment to persist even if automatic instantiations are cleared or explicit instantiations are removed from the source.



Moves the selected unit from the Can be instantiated by list to the Will be instantiated by list.



Moves the selected unit from the Will be instantiated by list to the Can be instantiated by list.

Will be instantiated by

This list contains the compilation units that *will instantiate* the entity selected in the Artificial Units list.

NOTE

It is quite legitimate for more than one unit to instantiate a given entity. Under the current release, the instantiable entity's object is bundled in the same object file as the unit that instantiates it. Different programs in the same environment may need a particular entity but may include different units which instantiate that entity. In such cases, each unit must instantiate the entity for the program in which it is included.


The listed compilation unit may be preceded by a special symbol that indicates how it was assigned to instantiate the given entity. No symbol indicates that the assign-

ment of this unit to instantiate the selected entity occurred during *automatic instantiation*.

* indicates that this compilation unit has been specifically chosen by the user to instantiate the selected entity.

The user may have either:

- moved the compilation unit from the **Can be instantiated by list** to the **Will be instantiated by list**, or

- used the  button to *explicitly assign* the compilation unit to instantiate the selected entity. This entity may have been previously assigned to that unit during *automatic instantiation* or due to an explicit instantiation in the source (such as through the `instantiate` pragma).

indicates that the instantiation was requested explicitly in the source (such as through the `instantiate` pragma)

The user may remove the explicit assignment by moving the selected compilation unit to the **Can be instantiated by list**. This cannot be done, however, for entities requested explicitly in the source of that unit.

Unit Information

The following is a list of the fields that comprise the information for a unit. Each field detailed below corresponds to a column heading in the list that appears on the **Units** page.

This information can also be seen using the **Show Info** button on the **Units** page. The information displayed by **Show Info** differs between Ada and C/C++. The following table shows the information displayed by **Show Info** depending on the language used for development in that environment:

Ada	C/C++
Name	Name
Part	Language
Source File	Source File
Item	State
Class	Consistency (Cns.)
State	Artificial
Consistency (Cns.)	Hidden
Unit Date	Magnet
Source Date	Unit Date
Generic Info	Source Date
Visa	Include(s) Date
Home Environment	Visa
Origin Environment	Origin Environment
Permanent Options	Permanent Options
Temporary Options	Temporary Options

Detailed descriptions of each of these fields appear below in alphabetical order.

See also:

- “Development - Units” on page 5-61
- NightBench Development - Chapter 5

Artificial

Displays whether a unit is defined by the user or is an implementation-defined unit created by the compiler:

real	a unit that the user manually introduced into the environment
artificial	created by the compiler for some generic instantiations

Class

(Ada only)

Description of a library unit's class:

library	a unit that is not syntactically contained within another unit
subunit	proper body of a unit declared with the keyword separate
nested	a unit that is syntactically contained within another unit - nested units will always be artificial

Consistency (Cns.)

The *consistency* of a given unit.

C	consistent	the unit's state has been verified that it and all units on which it depends have been compiled
Inc	inconsistent	either this unit or a unit on which it depends has changed state and should be recompiled
C?	was consistent	this unit and all units on which it depends had been verified as to have been compiled but either this unit or a unit on which it depends may have changed state, possibly requiring a recompilation
Inc?	was inconsistent	either this unit or a unit on which it depends had changed state and needed recompilation
?	unknown	this unit's consistency cannot be determined at this point

Effective Options

The set of *effective options* associated with this unit.

Generic Info*(Ada only)*

Generic information about a unit.

generic	A <i>generic unit</i> acts as a template for the instantiation of other units.
generic-corresponding-to-...	A generic corresponding to another generic is a generic unit that is created using another generic unit as a template.
instance-of-...	An <i>instance</i> of a unit is created using a generic unit as a template.
n/a	Neither a generic unit nor an instance of a unit.

Hidden*(C/C++ only)*

The *hidden* status of a unit. Hidden units are those units that will not be considered for resolving *instantiation* requests.

Home Environment (Home Env.)*(Ada only)*

The environment in which the unit was compiled (including any naturalized copies).

Include(s) Date*(C/C++ only)*

The latest timestamp of the primary source file and any files that it includes.

Item*(Ada only)*

The type of this unit. Units can be one of the following:

- package
- subprogram
- task
- protected

Language

(C/C++ only)

Specifies the language the particular unit is written in thereby indicating which compiler will be used to compile the unit.

C	the unit will be compiled with the C compiler
C++	the unit will be compiled with the C++ compiler

Magnet

(C/C++ only)

Displays the *magnet* status of a unit. Magnets are those units which will be preferred over others to host *artificial units* for *instantiations*.

Main

(Ada only)

Displays whether or not a unit can possibly be a main unit:

yes	meets all criteria for a main unit
no	does not meet all criteria for a main unit
maybe	cannot be determined - needs to be compiled to acquire more information

Name

The name of the unit.

A unit that is marked as *ambiguous* is designated by a different background color across the line for this unit.

Origin Environment (Origin Env.)

The environment into which the unit was originally introduced (ignoring any naturalized copies).

Part

(Ada only)

Units are identified by their name and by their *part*. The part can be designated as either **spec** (*specification*) or **body**.

Specifications can be

- subprogram declarations (including renaming declarations)

- package declarations (including renaming declarations)
- generic declarations (including renaming declarations)
- generic instantiations

Bodies can be

- subprogram bodies
- package bodies
- subunits

Permanent Options

The set of *permanent options* associated with this unit.

Source Date

The date and time of the last modification to the **Source File** for this unit.

Source File

The name of the file from which this unit came.

State

Units in the environment can be in either of the following compilation states:

- uncompiled
- compiled

Temporary Options

The set of *temporary options* associated with this unit.

Unit Date

The date and time the unit last changed compilation state.

Visa

Units have a nationality associated with them which classifies the compiled form of the unit.

Units can be one of the following nationalities:

- native
- naturalized
- fetched

- foreign

Development - Dependencies - (Ada Only)

The Dependencies page allows the user to find out which units are required by specific units, both directly and indirectly, and also which units require a specific unit.

This page is only displayed by NightBench Development for Ada environments.

See also:

- NightBench Development - Chapter 5

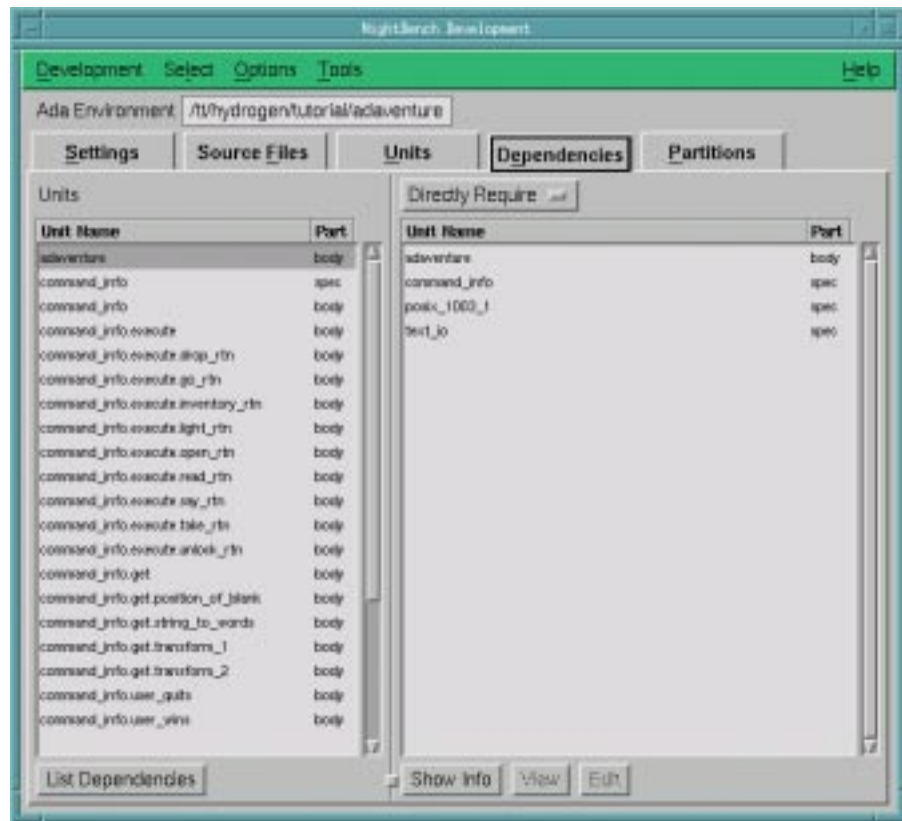


Figure 5-37. NightBench Development - Dependencies page

Units

A listing of all the units in the current environment.

This list is filtered by the following filters on the Units page:

- Filter:Env. - which can filter out foreign or system units
- Filter:Other - which can display or filter out artificial units

Foreign, system, and artificial units will not be included in this list if they are not included by these filters on the Units page.

See also:

- “Development - Units” on page 5-61
- “Accelerated Item Selection” on page 5-16

Unit Name

The name of the unit.

See also:

- “Unit Information” on page 5-87

Part

Units are identified by their name and by their *part*. The part can be designated as either **spec** (*specification*) or **body**.

See also:

- “Unit Information” on page 5-87

Dependencies

A listing of all units that the selected units **Directly Require**, **Require**, or **Are Required By**, depending upon the selection made on this page.

Directly Require	lists the units that the selected unit directly requires
Require	lists the units that the selected unit directly or indirectly requires
Are Required By	lists the units that require the selected unit

For example, in the figure titled “NightBench Development - Dependencies page” on page 5-93, the unit `adventure` has a direct requirement on itself and the specifications for the units `command_info`, `posix_1003_1`, and `text_io`. If the **Require** option was specified, this list would contain all units which `adventure` directly requires, and the units required by those units, and so on, until all units required by transitive closure are satisfied.

See also:

- “Accelerated Item Selection” on page 5-16

Unit Name

The name of the unit.

See also:

- “Unit Information” on page 5-87

Part

Units are identified by their name and by their *part*. The part can be designated as either **spec** (*specification*) or **body**.

See also:

- “Unit Information” on page 5-87

List Dependencies

Lists all units that the selected units **Directly Require**, **Require**, or **Are Required By**, depending upon the selection made on this page.

See also:

- “Dependencies” on page 5-94

Show Info

Brings up a window that displays the complete unit information for a selected unit. This will only show information for one unit at a time.

See also:

- “Unit Information” on page 5-87

View

Loads the units selected from the dependencies list into the configured file viewer.

See also:

- “Preferences - Viewer” on page 1-8

Edit

Loads the units selected from the dependencies list into the configured file editor, allowing changes to be made to the files.

See also:

- “Preferences - Editor” on page 1-9

Development - Partitions - (Ada)

The **Partitions** page allows the user to create, configure and build *partitions* in the NightBench Program Development Environment. A partition is an executable, archive, or shared object that can be invoked outside of NightBench.

This main page and its various subpages display detailed information about the partitions defined in the current environment.

The top portion of the **Partitions** page lists the partitions currently defined in the given environment and has buttons for creating, removing, and building partitions.

The bottom section of the **Partitions** page contains a number of subpages that contain specific information about the partition currently selected in the **Partitions List** at the top of the page. These subpages are:

- **General** - see “Partitions - General” on page 5-110
- **Units** - see “Partitions - Units” on page 5-113
- **Link Control** - see “Partitions - Link Control” on page 5-124
- **Link Options** - see “Partitions - Link Options” on page 5-133
- **Tracing** - see “Partitions - Tracing” on page 5-138
- **Expert** - “Partitions - Expert” on page 5-142

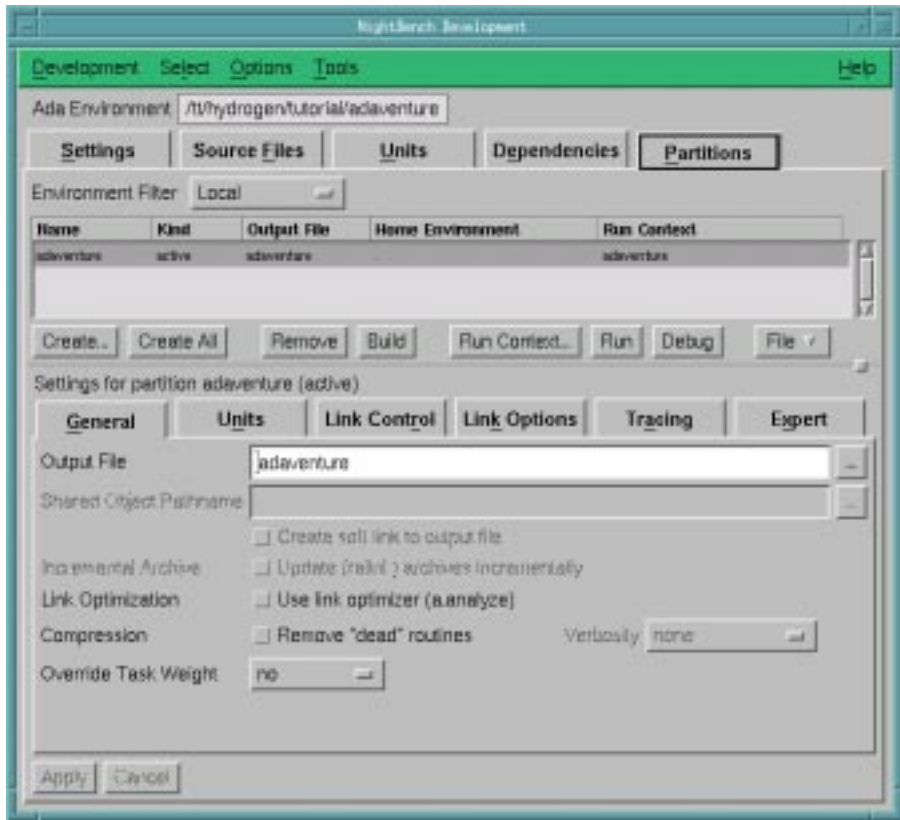


Figure 5-38. NightBench Development - Partitions page

Environment Filter

This filter determines which units and partitions will be displayed in the subpages below.

Local	Filter candidate units to include only units local to the current environment. This includes units introduced directly to the current environment, and those <i>fetched</i> or <i>naturalized</i> into the current environment.
User	Filter candidate units to include only units created by a user. This includes all units, including those obtained via the <i>Environment Search Path</i> , except those local to a predefined environment. The predefined environments are any shipped with MAXAda or related products, such as MAXaxi.
All	Include all units in the environment, including those obtained via the <i>Environment Search Path</i> .

Partitions List

A listing of all the partitions that have been created in the current environment.

See also:

- “Accelerated Item Selection” on page 5-16

Name

The name that has been given to this partition.

Kind

The type of partition. NightBench supports three different kinds of partitions:

- *active*
- *shared object*
- *archive*

See “Create Partition” on page 5-103 for more information on these types of partitions.

Output File

The name of the resultant file created when this partition is built. This name is specified on the **General** subpage of the Partition Settings area.

See also:

- “Partition Settings area” on page 5-102
- “Partitions - General” on page 5-110
- “Output File” on page 5-110

Home Environment

The environment in which the unit was compiled (including any naturalized copies).

Run Context

The name of the *run context* currently associated with this partition. A run context is automatically created for each active partition that is created; by default, the run context is created with the same name as the partition.

The user can associate a different run context for a particular partition by selecting the partition from the **Partitions List**, opening the **Run Context** dialog (using the **Run Context...** button on the **Partitions** page), and either selecting a different run context from the drop-down list associated with the **Run Context** field or creating a new run context (see “New Run Context” on page 6-22).

See also:

- “Run Context” on page 6-19

Create...

Create a new partition within the current environment.

NOTE

A *run context* is automatically created for each active partition that is created; by default, the run context is created with the same name as the partition.

See also:

- “Create Partition” on page 5-103
- “Run Context” on page 6-19

Create All

Creates an active partition for each unit within the current environment that qualifies as a main unit.

This does not create active partitions for units which are already main units for partitions within the current environment.

NOTE

A *run context* is automatically created for each active partition that is created; by default, the run context is created with the same name as the partition.

See also:

- “Run Context” on page 6-19

Remove

Remove the selected partition from the environment.

NightBench verifies that this action is intentional.

NOTE

This action does not remove the *run context* currently associated with that particular partition. Use the **Delete** button in the **Run Context** dialog to delete a specific run context.

See also:

- “Run Context” on page 6-19

Build

Opens the NightBench Builder. The build **Targets** (see “Targets” on page 6-13) are any selected partitions in the partitions list (see “Partitions List” on page 5-98). If no partitions are selected, the default build target is **all partitions**.

The build may automatically start if either the **Automatically Start Builds** option is selected from the **Options** menu of the NightBench Development window or if this is specified in the **Preferences** dialog.

See also:

- NightBench Builder - Chapter 6
- “Builder - Targets” on page 6-36
- “Automatically Start Builds” on page 5-14
- “Preferences - Start” on page 1-14

Run Context...

Opens the **Run Context** dialog to edit the *run context* currently associated with the *selected partition*. See *run context association* on page Glossary-7 for more information.

NOTE

Any changes made to this run context will affect the selected partition. For example, if the user opens the **Run Context** dialog, and deletes the run context (using the **Delete** button), the run context currently associated with the given partition will be the run context that is in the **Run Context** field of the **Run Context** dialog when the dialog is dismissed.

See also:

- “Run Context” on page 6-19

Run

Executes the *run context* currently associated with the selected partition. See *run context association* on page Glossary-7 for more information.

NOTE

The user may configure NightBench to open the Run Context dialog each time the Run button is pressed. For example, the user may want to change the arguments to the program each time it is run. Select the **Popup** this dialog prior to Run or Debug setting on the Preference page of the Run Context dialog for this behavior. (See “Run Context - Preference” on page 6-32 for more information).

See also:

- “Run Context” on page 6-19

Debug

Executes the *run context* currently associated with the selected partition under the NightView debugger. See *run context association* on page Glossary-7 for more information.

NOTE

The user may configure NightBench to open the Run Context dialog each time the Debug button is pressed. For example, the user may want to change the arguments to the program each time it is debugged. Select the **Popup** this dialog prior to Run or Debug setting on the Preference page of the Run Context dialog for this behavior. (See “Run Context - Preference” on page 6-32 for more information).

See also:

- “Run Context” on page 6-19
- *NightView User’s Guide* (0890395)

File

Contains a drop-down menu of utilities for saving the partition definitions from the current environment to a single file and for creating partitions using such a file.

See also:

- “Partition File Commands” on page 5-105

Partition Settings area

The bottom half of the **Partitions** page is composed of the following subpages which contain specific information about the partition currently selected in the **Partitions List**.

NOTE

The options listed in these subpages apply only during the linking phase of the build.

See also:

- **General** - see “Partitions - General” on page 5-110
- **Units** - see “Partitions - Units” on page 5-113
- **Link Control** - see “Partitions - Link Control” on page 5-124
- **Link Options** - see “Partitions - Link Options” on page 5-133
- **Tracing** - see “Partitions - Tracing” on page 5-138
- **Expert** - “Partitions - Expert” on page 5-142

Apply

Saves the changes made within the Settings area for the selected partition.

Cancel

Disregards any changes within the Settings area. All Settings are reset to the values last applied.

Create Partition

The **Create Partition** dialog allows the user to specify the name and type of partition to be created in the current environment.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

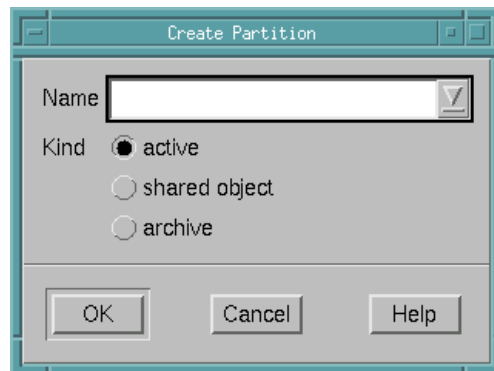


Figure 5-39. Create Partition dialog

Name

The name of the partition to be created.

The drop-down list contains units that may be possible main units.

Kind

The type of partition to be created. NightBench supports three different kinds of partitions:

active	An active partition is the simplest form of partition and it describes how to build an executable program.
shared object	A shared object is a collection of routines and data that can be used by multiple active partitions or foreign language executables without its contents being included in them. An active partition references the shared object during the link phase and associates calls and other references during the execution phase. Shared objects are composed of units built with share mode <code>shared</code> (implying the use of position independent code). (See “Share Mode” on page 7-53 and “Share Mode” on page 7-53 for more information.)
archive	An archive is a collection of routines and data that can be used by active partitions or foreign language executables. An active partition references the archive during the link phase and includes any needed portions of the archive into the output file. An archive is not referenced at execution time. Archives are composed of units built with share mode <code>non-shared</code> (implying the use of static code). (See “Share Mode” on page 7-53 and “Share Mode” on page 7-53 for more information.)

Partition File Commands

The Partition File Commands menu appears on the **Partitions** page of the NightBench Development window. It is labeled on the **Partitions** page as **File** and contains a drop-down list of utilities related to partition definitions.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

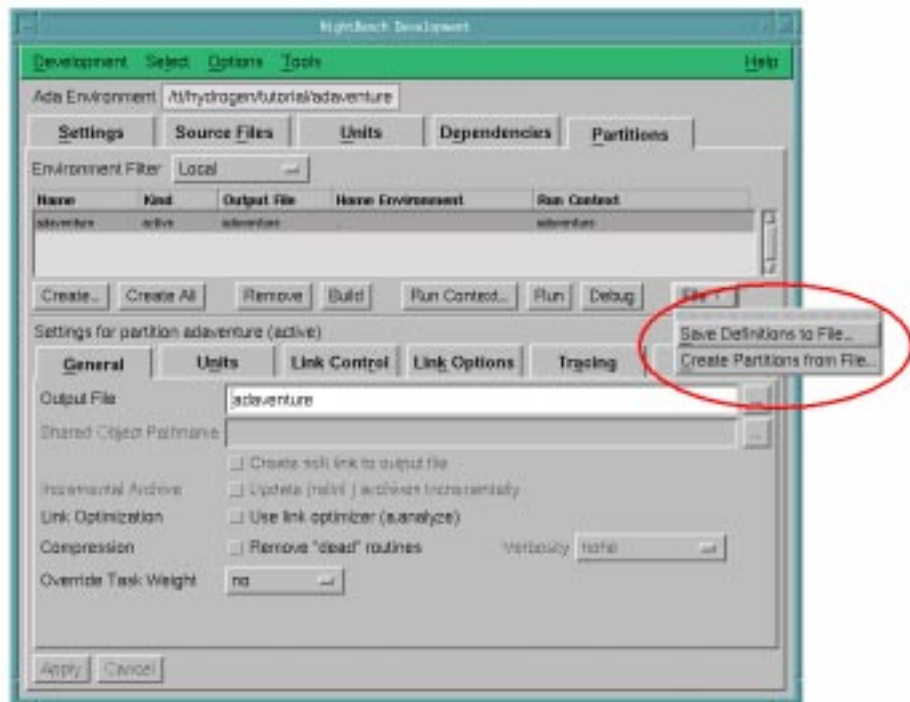


Figure 5-40. Partition File Commands menu

Save Definitions to File...

Use this to save the list of partition definitions contained within the current environment to a file. This file can then be used by the **Create Partitions from File...** option at a later time.

See also:

- “Save Partition Definitions to File” on page 5-106

Create Partitions from File...

Use this to create a number of partitions at once. This requires a file that contains the definitions of each partition to be created. A file of this format is created using the Save Definitions to File... option.

See also:

- “Create Partitions from File” on page 5-107

Save Partition Definitions to File

The current partition definitions may be saved to a file for later use. Partitions may be restored by using the definitions saved in this file with the Create Partitions from File option.

See also:

- “Partition File Commands” on page 5-105
- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

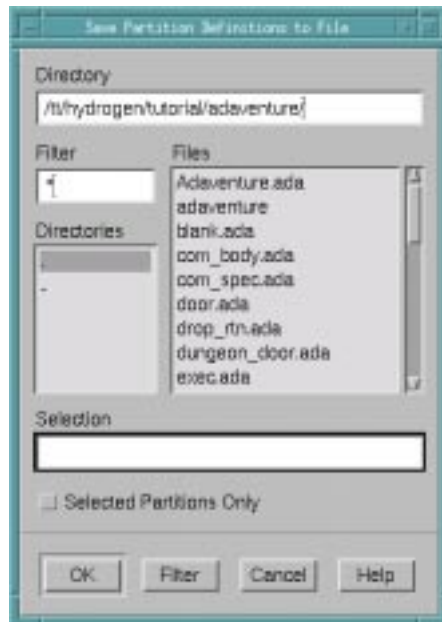


Figure 5-41. Save Partitions to File dialog

Directory

The directory in which to place the file containing the definitions of the partitions within the current environment.

The user may type the name of a directory into this field directly. In this case, the user must press the **Update** button to refresh the items on this dialog.

Filter

Of all the files contained in the current **Directory**, display only those that match the specified filter.

After a filter is entered, the **Update** button should be pressed to update the list of **Files**.

Directories

Contains a list of all the subdirectories within the current directory. Selecting any of these changes the current **Directory** to that subdirectory. Double-clicking on any of these directory names will change to that directory and update the **Files** list accordingly.

Files

Within the current **Directory**, this is a list of the files that match the specified **Filter**. Any of these filenames can be selected. When selected, the filename appears in the **Selection** field.

Selection

The current file in which to save the definitions of the partitions contained within the current environment.

A selection made in the **Files** list will show up here.

The actual filename can be manually entered into this field.

Selected Partitions Only

The default behavior of this dialog is to save the partition definitions for all partitions in the given environment. This option limits the partition definitions to those partitions selected in the partitions list (see “Partitions List” on page 5-98).

Update

Update the list of **Files** based on the information entered in the **Directory** and **Filter** fields.

Create Partitions from File

This option may be used to create a number of partitions that have been previously defined.

See also:

- “Partition File Commands” on page 5-105
- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

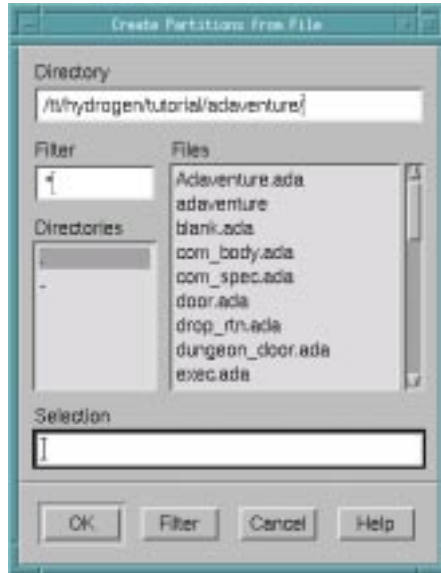


Figure 5-42. Create Partitions from File dialog

Directory

The directory in which to locate the file containing the definitions of the partitions to be created within the current environment.

The user may type the name of a directory into this field directly. In this case, the user must press the **Update** button to refresh the items on this dialog.

Filter

Of all the files contained in the current **Directory**, display only those that match the specified filter.

After a filter is entered, the **Update** button should be pressed to update the list of **Files**.

Directories

Contains a list of all the subdirectories within the current directory. Selecting any of these changes the current **Directory** to that subdirectory. Double-clicking on any of these directory names will change to that directory and update the **Files** list accordingly.

Files

Within the current **Directory**, this is a list of the files that match the specified **Filter**. Any of these filenames can be selected. When selected, the filename appears in the **Selection** field.

Selection

The current file in which to look for the definitions of partitions to be created within the current environment.

A selection made in the **Files** list will show up here.

The actual filename can be manually entered into this field.

Update

Update the list of **Files** based on the information entered in the **Directory** and **Filter** fields.

Partitions - General

General information about the partition is contained on this subpage. The name of the output file is specified here as well as other settings related to such areas as optimization and compression.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

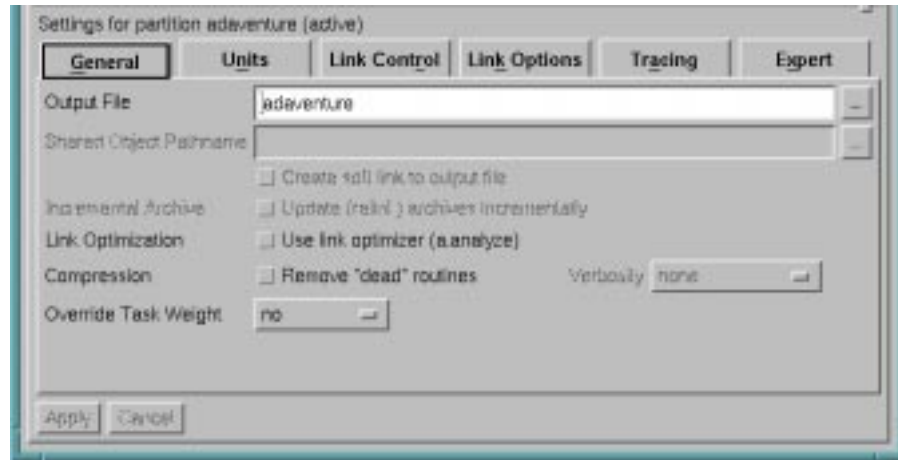


Figure 5-43. Partitions - General settings

Output File

The name of the file generated when the partition is built. This defaults to the name of the partition when it is initially created. An alternate name may be specified in this field.

This field may not be left blank. A name for the output file must be specified.

See also:

- “Create Partition” on page 5-103



Use this to popup a dialog and browse for the file you want. When finished, the relative pathname of the file is added to the Output File field.

Shared Object Pathname

For use with shared object partitions only.

The shared object's pathname on the target system. It does not cause the shared object to be created at the specified pathname; the shared object will still be built at the pathname specified for the **Output File**. However, all user programs created that require units from this shared object will expect it to be in the location specified here. The shared object must be placed at this pathname before any executable using it can be run. A soft link can be created automatically for this purpose by the **Create soft link to output file** checkbox (see below).



Use this to popup a dialog and browse for the pathname you want. When finished, the pathname of the file is added to the **Shared Object Pathname** field.

Create soft link to output file

For use with shared object partitions only.

A soft link is created from the shared object's pathname to the output pathname. Using this option in conjunction with the **Shared Object Pathname** removes the need for the shared object to be explicitly placed at the pathname specified by the **Shared Object Pathname**.

Incremental Archive

Update (relink) archives incrementally

This option is only applicable to archives and is not effective when using MAXAda releases prior to 4.0.

When this option is specified on an archive partition, if the partition contains any units that have been modified, only those units that have been changed will be updated when the partition is relinked. In order to reduce implementation overhead, the partition will be completely rebuilt if units that could have been included in the partition are removed from the environment.

The timestamp of the partition is used to determine which object files need to be replaced within it when the partition is relinked.

WARNING

The user must never change the timestamp of the target file for a partition configured with this option. If the target file's timestamp were changed and then relinked, the target file might contain stale object files.

Link Optimization

Use link optimizer (a.analyze)

A link-time optimization is performed which replaces the two-instruction sequences required for global static memory references with single instructions. This optimization uses the linker registers (r28 through r31) which are reserved for this purpose.

Compression

Remove “dead” routines

Removes those routines from the program which cannot be called by legal Ada means. Only the machine instructions associated with Ada routines are available for removal - data is never removed.

Verbosity

If the Remove “dead” routines option is selected, this allows the user to select how much information is displayed as the “dead” routines are removed.

none	Execute quietly, that is, without statistics.
stats only	Lists how much disk space and how much memory will be saved in subsequent executions after the dead routines are removed.
stats and routines	List the dead routine link names in addition to the above statistics.

Override Task Weight

Sets the task weight for the current partition. This overrides any other specification such as those obtained from pragmas.

no	does not override the task weight
bound	sets the default task weight to <i>bound</i>
multiplexed	sets the default task weight to <i>multiplexed</i>

NOTE

This setting has no effect as only *bound* tasks are supported at this time.

Partitions - Units

Units to be included as part of the current partition's definition are specified on this subpage. In addition, the user may exclude specific units from a partition here as well.

This particular subpage changes slightly depending on whether the current partition is an active partition or a non-active partition (shared object or archive). The main difference between the **Units** settings subpage for a non-active partition and an active partition is that there is no need to designate a main unit for a non-active partition. Hence, the corresponding functionality does not exist on this subpage for a non-active partition.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

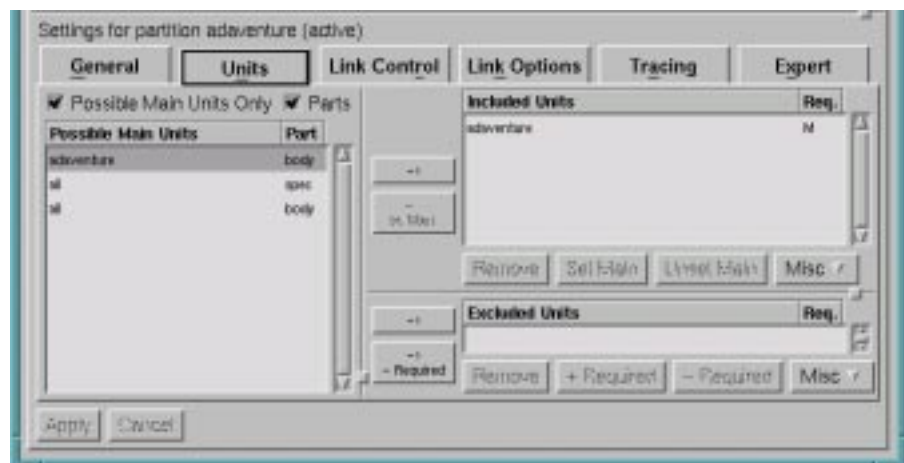


Figure 5-44. Partitions - Units settings

Possible Main Units Only

When this option is checked, only units that can possibly be a main unit are listed in the Units in Environment / Possible Main Units list. (These units are designated by a *yes* or *maybe* under *Main* on the Units page.) Otherwise, all units within the current environment are listed.

Main units are only used by active partitions, so this checkbox has no relevance for non-active partitions.

See also:

- “Development - Units” on page 5-61
- “Unit Information” on page 5-87

Parts

When this option is checked, an entry for each *part* (**spec** and **body**) of every unit is listed in the Units in Environment / Possible Main Units list. Otherwise, only the unit name appears in the list, denoting both the **spec** and the **body**.

Units in Environment / Possible Main Units

This either lists all units in the environment or only possible main units, depending on the state of the Possible Main Units Only checkbox. The heading for this list will be **Units in Environment** or **Possible Main Units**, appropriately. In addition, part information about each unit may be listed, according to the setting of the Parts checkbox.

This list is filtered by the following filters on the Units page:

- Filter:Env. - which can filter out foreign or system units
- Filter:Other - which can display or filter out artificial units

Foreign, system, and artificial units will not be included in this list if they are not included by these filters on the Units page.

See also:

- “Development - Units” on page 5-61

Included Units

The Included Units area changes slightly depending on whether the current partition is active or non-active (shared object or archive). The items that appear in this list are sorted alphabetically. However, new units added to this list are appended to the end until the changes are applied, whereupon the list is once again sorted.

Figure 5-45 shows the Included Units area for an active partition whereas Figure 5-46 shows the same area for a non-active partition. Each of the buttons is described below.

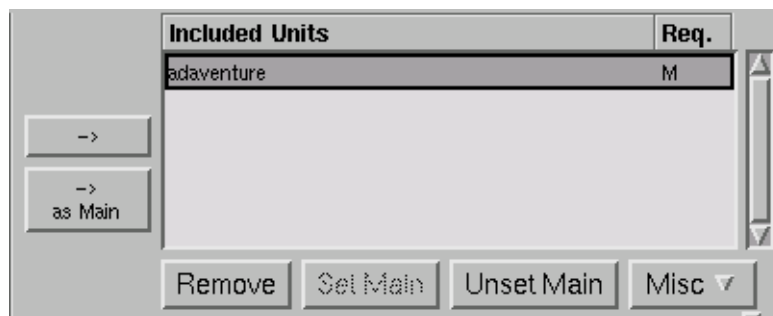


Figure 5-45. Partitions - Included Units - active partition

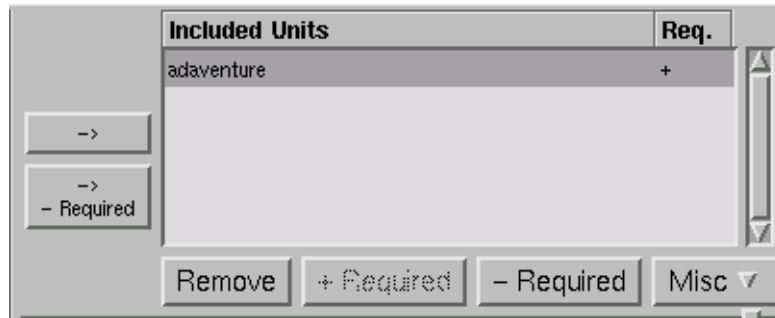
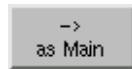


Figure 5-46. Partitions - Included Units - non-active partition



This button adds the selected unit from the Units in Environment / Possible Main Units to the Included Units list for the current partition. In addition, all units that this unit requires are included in the partition. The set of required units is determined as a partition is linked, so new with clauses can be added to units after a partition is defined, and the partition will include the appropriate set of units the next time it is linked.

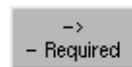


(Active Partitions Only)

For non-active partitions, this button is labeled -> - Required and has a different functionality. See the description below.

This button adds the selected unit from the Units in Environment / Possible Main Units to the Included Units list for the current partition and designates it as the main unit for this partition. In addition, all units that this unit requires are added to the list of included units.

Only one unit can be designated as the main unit for a partition.



(Non-Active Partitions Only)

For active partitions, this button is labeled -> as Main and has a different functionality. See the description above.

Only adds the selected unit as part of this partition definition but does not include the units which that unit requires.

There may be times when the user wishes to add a specific unit to the **Included Units** list but, depending on the setting of the **Environment Filter**, the name of this unit may not appear in the **Units in Environment / Possible Main Units** list. The user can manually enter the name of the unit using the **Add Included Unit** dialog without changing the **Environment Filter** setting.

See also:

- “Add Included Unit” on page 5-121

Included Units List

Included Units

This lists each of the units to be included when linking this partition.

This list is sorted alphabetically. However, units recently added to this list are appended to the bottom until the changes are applied. The advantage of this is that all newly-added units are grouped together at the end of the list where they can be manipulated as a group easily. When the **Apply** button is pressed, the list is re-sorted.

Req.

(Active Partitions Only) An **M** in this column indicates that the specified unit is the current partition's main unit. All units required by this unit must also be included when this partition is linked.

A + in this column indicates that all units required by the given unit are to be included when linking this partition.

Remove

Removes the unit from the current partition definition.

Set Main

(Active Partitions Only)

For non-active partitions, this button is labeled **+ Required** and has a different functionality. See the description below.

Designates the unit selected in the **Included Units** list as the main unit for the current partition. If any other unit was designated as the main unit, its status will be changed so that it is not the main unit, and the selected unit will then be made the main unit.

See Figure 5-45 on page 5-114 for an example of how the **Included Units** list looks for an active partition.

+ Required

(Non-Active Partitions Only)

For active partitions, this button is labeled **Set Main** and has a different functionality. See the description above.

In addition to the unit selected in the **Included Units** list, this button also specifies that any units that this unit requires must be included by the linker for this partition.

See Figure 5-46 on page 5-115 for an example of how the **Included Units** list looks for a non-active partition.

Unset Main

(Active Partitions Only)

For non-active partitions, this button is labeled **- Required** and has a different functionality. See the description below.

Changes the status of the unit selected in the **Included Units** list so that it is not designated to be the main unit for the current partition. After this button is pressed, no unit will be designated as the main unit.

See Figure 5-45 on page 5-114 for an example of how the **Included Units** list looks for an active partition.

- Required

(Non-Active Partitions Only)

For active partitions, this button is labeled **Unset Main** and has a different functionality. See the description above.

This button specifies that only the unit selected in the **Included Units** list is required as part of this partition but not necessarily the units that this unit requires.

See Figure 5-46 on page 5-115 for an example of how the **Included Units** list looks for a non-active partition.

Misc

Add Unlisted Included Unit

This brings up a dialog that may be used to include a unit in a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the **Units in Environment / Possible Main Units** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired unit may not appear in the **Units in Environment / Possible Main Units** list. For instance, if the **Environment Filter** is set to **Local**, units in other environments on the *Environment Search Path* will not be displayed in the **Units in Environment / Possible Main Units** list. This dialog allows the user to add the included unit to the partition's definition regardless of the setting of the **Environment Filter**.

See also:

- “Add Included Unit” on page 5-121

Excluded Units

The items that appear in this list are sorted alphabetically. However, new units added to this list are appended to the end until the changes are applied, whereupon the list is once again sorted.

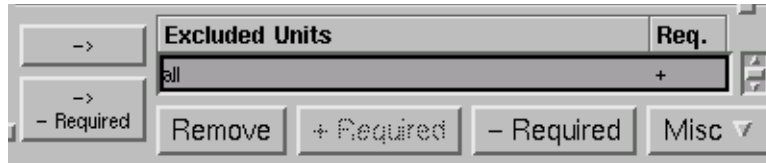
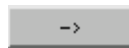
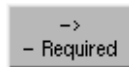


Figure 5-47. Partitions - Excluded Units



This button adds the selected unit from the Units in Environment / Possible Main Units to the Excluded Units list for the current partition. In addition, all units that this unit requires are excluded from this partition definition. The set of required units is determined as a partition is linked, so new *with* clauses can be added to units after a partition is defined, and the partition will exclude the appropriate set of units the next time it is linked.



Only adds the selected unit to the list of excluded units but not those units that it requires.

There may be times when the user wishes to add a specific partition to the Excluded Units list but, depending on the setting of the Environment Filter, the name of this partition may not appear in the Units in Environment / Possible Main Units list. The user can manually enter the name of the partition using the Add Excluded Unit dialog without changing the Environment Filter setting.

See also:

- “Add Excluded Unit” on page 5-122

Excluded Units List

Excluded Units

This lists each of the units that are excluded from the current partition's definition.

This list is sorted alphabetically. However, units recently added to this list are appended to the bottom until the changes are applied. The advantage of this is that all newly-added units are grouped together at the end of the list where they can be manipulated as a group easily. When the Apply button is pressed, the list is re-sorted.

Req.

A + in this column indicates that all units required by this unit are also excluded from this partition's definition. Otherwise, only the unit listed is excluded.

Remove

Removes the unit selected in the Excluded Units list from this list.

+ Required

In addition to the unit selected in the Excluded Units list, this button also specifies that any units that this unit requires should also be excluded.

- Required

This button specifies that only the unit selected in the Excluded Units list should be excluded but not necessarily the units that this unit requires.

Misc

Add Unlisted Excluded Unit

This brings up a dialog that may be used to exclude a unit from a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the Units in Environment / Possible Main Units list.

In addition, depending on the setting of the Environment Filter, the name of the desired unit may not appear in the Units in Environment / Possible Main Units list. For instance, if the Environment Filter is set to Local, units in other environments on the *Environment Search Path* will not be displayed in the Units in Environment / Possible Main Units list. This dialog allows the user to exclude the unit from the partition's definition regardless of the setting of the Environment Filter.

See also:

- “Add Excluded Unit” on page 5-122

Add Included Unit

This dialog allows the user to include a unit in a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the Units in Environment / Possible Main Units list.

In addition, depending on the setting of the Environment Filter, the name of the desired unit may not appear in the Units in Environment / Possible Main Units list. For instance, if the Environment Filter is set to Local, units in other environments on the *Environment Search Path* will not be displayed in the Units in Environment / Possible Main Units list. This dialog allows the user to add the included unit to the partition's definition regardless of the setting of the Environment Filter.

See also:

- “Partitions - Units” on page 5-113
- “Included Units” on page 5-114

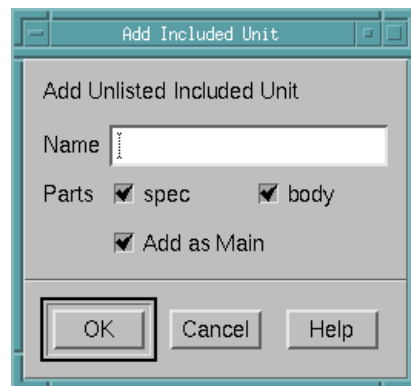


Figure 5-48. Partitions - Add Included Unit dialog

Name

The name of the unit which is to be included in the given partition's definition.

Parts

spec

When this checkbox is checked, the *specification* for the unit will be included in this partition definition.

body

When this checkbox is checked, the *body* for the unit will be included in this partition definition.

Add as Main

(Active Partitions Only)

For non-active partitions, this checkbox is labeled **Required Units Also** (see below).

This checkbox specifies that this unit, to be introduced later, will be designated as the main unit for this partition. In addition, all units that this unit requires are added to the list of included units for this partition.

Only one unit should be designated as the main unit for a partition.

Required Units Also

(Non-Active Partitions Only)

For active partitions, this checkbox is labeled **Add as Main** (see above).

This checkbox specifies that all units that this unit requires are added to the list of included units for this partition.

Add Excluded Unit

This dialog allows the user to exclude a unit from a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the **Units in Environment / Possible Main Units** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired unit may not appear in the **Units in Environment / Possible Main Units** list. For instance, if the **Environment Filter** is set to **Local**, units in other environments on the *Environment Search Path* will not be displayed in the **Units in Environment / Possible Main Units** list. This dialog allows the user to exclude the unit from the partition's definition regardless of the setting of the **Environment Filter**.

See also:

- "Partitions - Units" on page 5-113
- "Excluded Units" on page 5-118



Figure 5-49. Partitions - Add Excluded Unit dialog

Name

The name of the unit which is to be excluded from the given partition's definition.

Parts

spec

When this checkbox is checked, the *specification* for the unit will be excluded from this partition definition.

body

When this checkbox is checked, the *body* for the unit will be excluded from this partition definition.

Required Units Also

This checkbox specifies that all units that this unit requires are added to the list of excluded units for this partition.

Partitions - Link Control

The Link Control subpage allows the user to specify the manner in which required units are included in the given partition during the linking phase. Units that this partition requires may be found in those partitions either listed explicitly in the *dependent partitions* list or included implicitly through the *link rule* for that partition.

Dependent partitions are those specifically from which the linker should first attempt to obtain required units before utilizing the link rule. See “Dependent Partitions” on page 5-124 for details.

The link rule tells the linker which form of partition or system library is preferred when including units in a partition after having searched the dependent partitions. Each of the *link methods* (Object, Archive, and Shared Object) specifies the manner in which a unit or system library is included during the linking process. Note that using the object of the unit directly (the Object method) is the most common method. See “Link Rule” on page 5-126 for more specific information.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

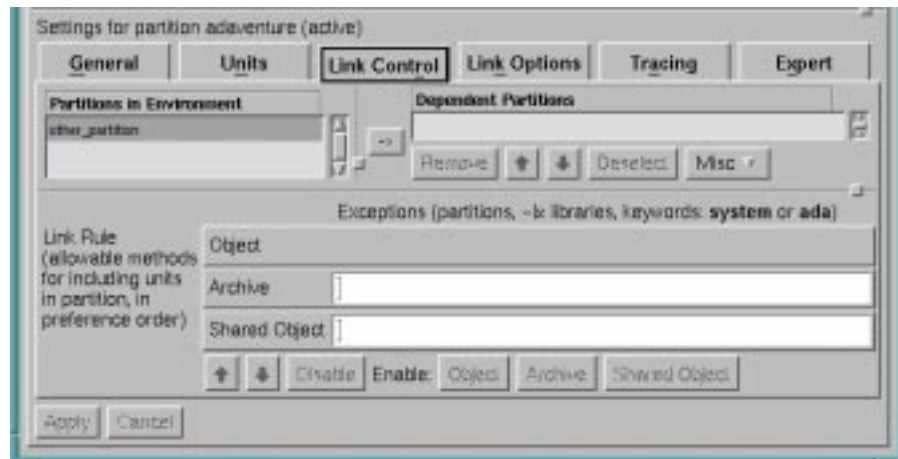


Figure 5-50. Partitions - Link Control settings

Partitions in Environment

This area lists the partitions that are visible to this environment based on the setting of the Environment Filter. See “Environment Filter” on page 5-97 for details.

Dependent Partitions

The *dependent partitions* list allows the user to list specific shared object or archive partitions in the environment (or on the *Environment Search Path*) that should be searched first

and foremost when trying to find the required units for a given partition during the linking phase. Dependent partitions are searched in the order in which they appear in the **Dependent Partitions** list. If a required unit is not found in any of the dependent partitions, the search will continue according to the *link rule* (see “Link Rule” on page 5-126 for more information). Furthermore, partitions that are mentioned in the **Dependent Partitions** list will always be included when linking the partition. This is especially important for active partitions which mention dependent shared object partitions, because the linked active partition will require the dependent shared object partition to exist at run time.

For instance, if two archives have an overlapping set of units, you could specify from which one you would like to get the required unit(s) by adding it to the dependent partitions list.



Adds a partition from the **Partitions in Environment** list to end of the **Dependent Partitions** list. However, if a partition is selected in the **Dependent Partitions** list, any new partitions will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to **Deselect** any selection in that list.

Dependent Partitions

Lists the names of partitions that should be searched first and foremost when trying to find the required units for a given partition during the linking phase.

See also:

- “Dependent Partitions” on page 5-124

Remove

Removes the selected partition from the **Dependent Partitions** list.



Changes the position of the selected partition in the **Dependent Partitions** list to an earlier (more preferred) position. Dependent partitions are searched in the order in which they appear in the **Dependent Partitions** list.



Changes the position of the selected partition in the **Dependent Partitions** list to a later (less preferred) position. Dependent partitions are searched in the order in which they appear in the **Dependent Partitions** list.

Deselect

Clears any selections in the **Dependent Partitions** list. If a partition is selected in the **Dependent Partitions** list, any new partitions will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to deselect any selection in that list.

Misc

Add Unlisted Dependent Partition

This brings up a dialog that may be used to add a partition to the **Dependent Partitions** list by manually entering the name of the partition.

This can be used for partitions which have not yet been defined in the environment (or in an environment on the *Environment Search Path*) and therefore do not appear in the **Partitions in Environment** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired partition may not appear in the **Partitions in Environment** list. For instance, if the **Environment Filter** is set to **Local**, partitions in other environments on the *Environment Search Path* will not be displayed in the **Partitions in Environment** list. This dialog allows the user to add a partition in the **Dependent Partitions** list regardless of the setting of the **Environment Filter**.

See also:

- “Add Dependent Partition” on page 5-131

Link Rule

The ordering of the *link methods* (**Object**, **Archive**, and **Shared Object**) within the link rule tells the linker which form of partition or system library is preferred when including units in a partition after searching the dependent partitions.

The default link rule differs for each type of partition:

Partition	Default Link Rule
active	object archive shared object
archive	object
shared object	object

The link methods may be ordered, however, in any manner the user prefers.

See “Link Rule Examples” on page 5-131 for an example.

Object

A link method that instructs the linker to use the object of a unit directly. Select this method to change its ordering or to **Disable** its functionality in the *link rule*.

Archive

A link method that instructs the linker to utilize the unit contained in an archive. Select this method to change its ordering or to **Disable** its functionality in the *link rule*.

If it is desired that a given partition should use archive partitions and libraries *most* of the time, but there are certain archive partitions and/or libraries that should not be used, the **Archive** link method can be specified, and exceptions can then be mentioned in the text field to its right. Archive partitions and libraries will be preferred over any subsequent link method specified, except for the archive partitions and/or libraries specified as exceptions which will be ignored.

For archive partitions, simply enter the name of the dependent partition.

To specify a system library, you must enter it in this field with the form:

`-lname`

which is standard shorthand notation for:

`libname.a`

NOTE

The libraries listed as exceptions here will only affect libraries that would be included in the link implicitly. See “Implicitly-Included Libraries” on page 5-130 for more information.

To indicate that a class of partitions or libraries is to be excluded for a particular method, one of three keywords should be specified:

- **ada**
- **system**
- **user**

The **ada** keyword indicates all partitions and libraries that are part of MAXAda (those located within `/usr/ada/release_name/lib`).

The **system** keyword indicates all libraries that are part of the PowerMAX OS operating system (those located within `/lib`, `/usr/lib`, or `/usr/ccs/lib`).

The **user** keyword indicates all other libraries.

For instance, to indicate that the **Archive** link method should be used except for:

- all system libraries

- the user's library called **libfoo.a**
- the user's dependent partition call **depbar**,

then the list of exceptions would be:

system -lfoo depbar

Shared Object

A link method that instructs the linker to include the unit found within a shared object. Select this method to change its ordering or to **Disable** its functionality in the *link rule*.

NOTE

Units that are to be included in shared objects must be compiled with their compile option **Share Mode** set to either **shared** or **both**. (See “Share Mode” on page 7-53 for more information.)

If it is desired that a given partition should use shared object partitions and libraries *most* of the time, but there are certain shared object partitions and/or libraries that should not be used, the **Shared Object** link method can be specified, and exceptions can then be mentioned in the text field to its right. Shared object partitions and libraries will be preferred over any subsequent link method specified, except for the shared object partitions and/or libraries specified as exceptions which will be ignored.

For shared object partitions, simply enter the name of the dependent partition.

To specify a system library, you must enter it in this field with the form:

-lname

which is standard shorthand notation for:

libname.so

NOTE

The libraries listed as exceptions here will only affect libraries that would be included in the link implicitly. See “Implicitly-Included Libraries” on page 5-130 for more information.

To indicate that a class of partitions or libraries is to be excluded for a particular method, one of three keywords should be specified:

- **ada**
- **system**
- **user**

The **ada** keyword indicates all partitions and libraries that are part of MAXAda (those located within `/usr/ada/release_name/lib`).

The **system** keyword indicates all libraries that are part of the PowerMAX OS operating system (those located within `/lib`, `/usr/lib`, or `/usr/ccs/lib`).

The **user** keyword indicates all other libraries.

For instance, to indicate that the Shared Object link method should be used except for:

- all system libraries
- the user's library called **libfoo.so**
- the user's dependent partition call **depbar**,

then the list of exceptions would be:

```
system -lfoo depbar
```

The link rule can combine any number of methods in any order by enabling or disabling particular methods and by changing the position of each method using the mechanisms provided here.



Changes the position of the selected link method to an earlier (more preferred) position in the *link rule*.



Changes the position of the selected link method to a later (less preferred) position in the *link rule*.

Disable

Disables the selected link method (Object, Archive, and Shared Object) so that it will not be included in the link rule for the current partition. The linker will not look in partitions of types corresponding to disabled link methods when searching for included units.

If the link methods Archive or Shared Object are disabled in the link method, the linker will not search for units in any partitions of the respective types. If the link method Object is disabled, the linker will never include an object directly in the partition, and will insist that all required units exist in some partition whose type is specified in the link rule.

Enable

Each of the following buttons (Object, Archive, and Shared Object) is activated when its corresponding link method has been disabled. Press the button asso-

ciated with the link method you wish to enable, thereby including it in the link rule for the current partition. You may use the up and down arrow buttons to change its order in the link rule.

Object

Enables the **Object** link method, thereby including it in the link rule for the current partition.

Archive

Enables the **Archive** link method, thereby including it in the link rule for the current partition.

Shared Object

Enables the **Shared Object** link method, thereby including it in the link rule for the current partition.

Implicitly-Included Libraries

The following are libraries which may be included implicitly during the linking phase:

Table 5-1. Implicitly-Included Libraries

-lc	required for all programs
-lgen	required for AXI
-lhF77	required for interfacing to Fortran
-lhI77	required for interfacing to Fortran
-lhU77	required for interfacing to Fortran
-lICE	required for Xt via AXI
-lm	required for interfacing to Fortran
-lMrm	required for Motif 2.1 via AXI
-lnsl	required for AXI
-lntrace	required for programs that use NightTrace bindings or -trace:mechanism=ntraceud
-lPW	required for Motif 2.1 via AXI
-lresolv	required for AXI
-lruntime*	required for all programs
-lsemaf	required for tasking programs
-lsemat	required for tasking programs
-lSM	required for Xt via AXI

Table 5-1. Implicitly-Included Libraries

-lsocket	required for AXI
-lthread	required for interfacing to Fortran or programs that use static libnsl
-lud	required for all programs
-lX11	required for X11R6 via AXI
-lXAda	required for AXI
-lXext	required for X11R6 via AXI
-lXm	required for Motif 2.1 via AXI
-lXmAda	required for AXI
-lXmu	required for X11R6 via AXI
-lXp	required for Motif 2.1 via AXI
-lXt	required for Xt via AXI

Link Rule Examples

A link rule with the following order:

```
shared_object
archive
object
```

directs the linker to search first for each unit within shared object partitions visible in the current environment. It will continue to search for the unit in shared objects on the Environment Search Path until one is found. If the unit is not found within any shared objects along the Environment Search Path, the linker will search in any archive partitions in the current environment or on the Environment Search Path. Finally, if still not found, the linker will attempt to use the actual object for the unit.

In the case of libraries, the linker will attempt to use either the shared object or archive of a library based on the ordering of the link methods in the link rule. The link method **object** has no bearing on the method for inclusion of a library.

Add Dependent Partition

This dialog allows the user to add a partition to the **Dependent Partitions** list by manually entering the name of the partition.

This can be used for partitions which have not yet been defined in the environment (or in an environment on the *Environment Search Path*) and therefore do not appear in the **Partitions in Environment** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired partition may not appear in the **Partitions in Environment** list. For instance, if the

Environment Filter is set to **Local**, partitions in other environments on the *Environment Search Path* will not be displayed in the **Partitions in Environment** list. This dialog allows the user to add a partition in the **Dependent Partitions** list regardless of the setting of the Environment Filter.

See also:

- “Partitions - Link Control” on page 5-124
- “Dependent Partitions” on page 5-124

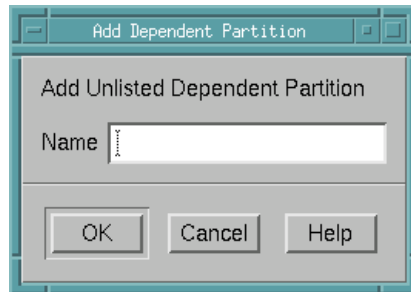


Figure 5-51. Partitions - Add Dependent Partition dialog

Name

The name of the partition to add to the **Dependent Partitions** list.

Partitions - Link Options

Link options for the given partition may be specified on this subpage. The more common link options are presented in a list for easier selection. In addition, other link options for the current partition may be directly entered on this subpage.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

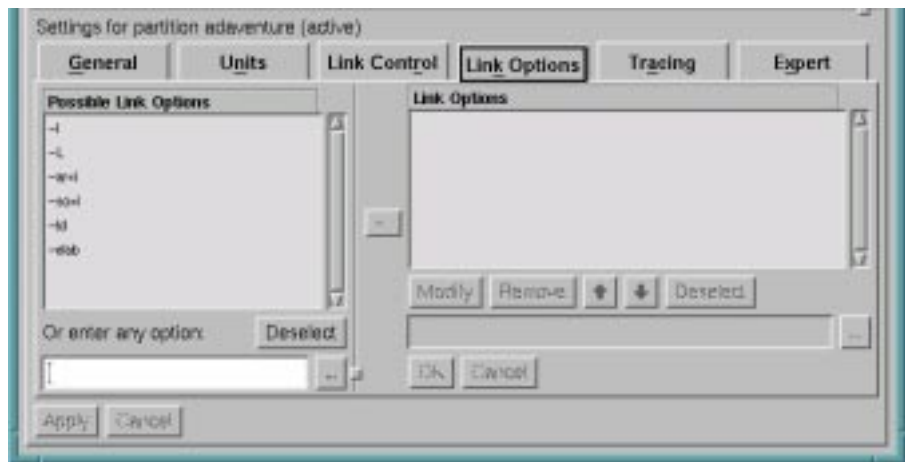


Figure 5-52. Partitions - Link Options settings

Possible Link Options

This area shows a number of the more common link options for the current partition.

When an option is selected, NightBench will enter the option in the text field below the list of Possible Link Options and position the cursor following the option so that the user may specify the any required arguments (e.g. *x*, *path*, *file*, etc.)

-lx	library	When this form is used, the linker will determine the link method (Archive or Shared Object) for the specified library based on the <i>link rule</i> (see "Partitions - Link Control" on page 5-124) and on the presence or absence of archive and shared object versions. If the behavior as specified by the link rule is not desired, either -ar=lx or -so=lx should be used instead.
-L path	library path	This option adds a path to the library search directories. The linker searches for libraries first in any directories specified with -L options (in order), and then in the standard directories. This option is effective only if it precedes a -l option.
-ar=lx	static library	This link option passes a -lx option to the system loader (ld), ensuring the library libx.a will be statically linked.
-so=lx	shared library	This link option passes a -lx option to the system loader (ld), ensuring the library libx.so will be dynamically linked.
-ld file	object file	This link option passes an object file to the system loader (ld). WARNING: It is the user's responsibility to ensure that the object file is correct and meaningful for the partition. For instance, an object with position-independent code (PIC) should not be specified for an active or archive partition, nor should one without position-independent code be specified for a shared object partition.
-elab sym	elaborate	This option inserts the routine specified by <i>sym</i> as the first elaboration routine. This allows the user to specify some routine that they've provided to be called before any Ada elaboration routines are called.

Any link option may be directly entered in the text field below the list of Possible Link Options.

In addition, the text associated with a selected option from the list of Possible Link Options will be entered in this field so the user may modify the option if necessary. For those options which require additional text (e.g. **-ld file**), the label for this text field will read either Option should be completed or Option must be completed, depending on the particular option. The user can then complete the option and add it to the list of Link Options.

For those options that take filenames (or directory names) as arguments, use the button labeled ... to bring up a dialog to find the file/directory you want.

NOTE

Because you may specify multiple options, spaces are reserved as separators. However, if you want a space to be *part* of an argument, you need to quote it with a prepended backslash. And because backslash is reserved for the quote character, if you want a backslash in the argument, you need to use a double backslash. For instance, the following option (quotes are shown for clarity):

```
'-ld' 'argwith spaceand\backslash.o'
```

must be specified in NightBench like:

```
-ld argwith\ spaceand\\backslash.o
```

Deselect

Deselects the selected item in the list of **Possible Link Options** and clears the text entered in the text field below the list of **Possible Link Options**.



For link options that take filenames (or directory names) as arguments, use this to popup a dialog and browse for the file/directory you want. When finished, the file/directory is added as the argument to the link option.



Adds the option that appears in the text field below the list of **Possible Link Options** to the end of the list of **Link Options** for the given partition. However, if a link option is selected in the **Link Options** list, any new link options will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to **Deselect** any selection in that list.

Link Options

Lists the current link options for the given partition in order. The user may **Modify** or **Remove** any of these options by selecting the desired option and pressing the appropriate button. In addition, the user may change the order of these options by using the up and down arrow buttons.

Note that option order is important for a variety of reasons. Libraries and library paths are searched in the order in which they appear in this list. In addition, object files (specified using the **-ld** option) may alleviate the need for a given library if specified before it and may create a multiple symbol error if they follow it. Also, elaboration routines are called in the order in which their **-elab** options are specified.

Modify

Enters the text of the selected option in the **Link Options** list into the text field below this list so that the user may modify the option.

For those options that take filenames (or directory names) as arguments, use the button labeled ... to bring up a dialog to find the file/directory you want.

Remove

Removes the selected option from the **Link Options** list.



Changes the position of the selected link option in the list of **Link Options** to an earlier position.

Note that option order is important for a variety of reasons. Libraries and library paths are searched in the order in which they appear in this list. In addition, object files (specified using the **-ld** option) may alleviate the need for a given library if specified before it and may create a multiple symbol error if they follow it. Also, elaboration routines are called in the order in which their **-elab** options are specified.



Changes the position of the selected link option in the list of **Link Options** to a later position.

Note that option order is important for a variety of reasons. Libraries and library paths are searched in the order in which they appear in this list. In addition, object files (specified using the **-ld** option) may alleviate the need for a given library if specified before it and may create a multiple symbol error if they follow it. Also, elaboration routines are called in the order in which their **-elab** options are specified.

Deselect

Clears any selections in the **Link Options** list. If an option is selected in the **Link Options** list, any new link options will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to deselect any selection in that list.

The text field below the list of **Link Options** and its associated buttons is used when modifying an option. When the **Modify** button is pressed for a given link option, the text of that link option is placed in this field for editing.



For link options that take filenames (or directory names) as arguments, use this to popup a dialog and browse for the file/directory you want. When finished, the file/directory is added as the argument to the text field containing the modified link option.

OK

Apply the changes to the modified link option and incorporate it into the list of Link Options.

Cancel

Disregard any changes to the modified link option.

Partitions - Tracing

The Tracing subpage contains options concerning real-time event tracing. Real-time event tracing is one way to debug and analyze the performance of Ada applications. It allows the user to gather information about important events in an application, such as event occurrences, timings, and data values.

There are two types of trace events: predefined trace events and user-defined trace events. Predefined trace events are generated by tracing versions of the Ada runtime executive and by library elaboration code generated for the ENVIRONMENT task. They typically describe execution in terms of tasking, interrupt handling, exception occurrence and handling, protected actions, and elaboration of library units. User-defined trace events generate information at points of interest within the source code specified by the user.

The information from the predefined trace events and user-defined trace events can subsequently be analyzed using either the NightTrace product or the MAXAda utility, **a.trace**.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96
- *NightTrace Manual* (0890398)
- *MAXAda Reference Manual* (0890516) - Chapter 11

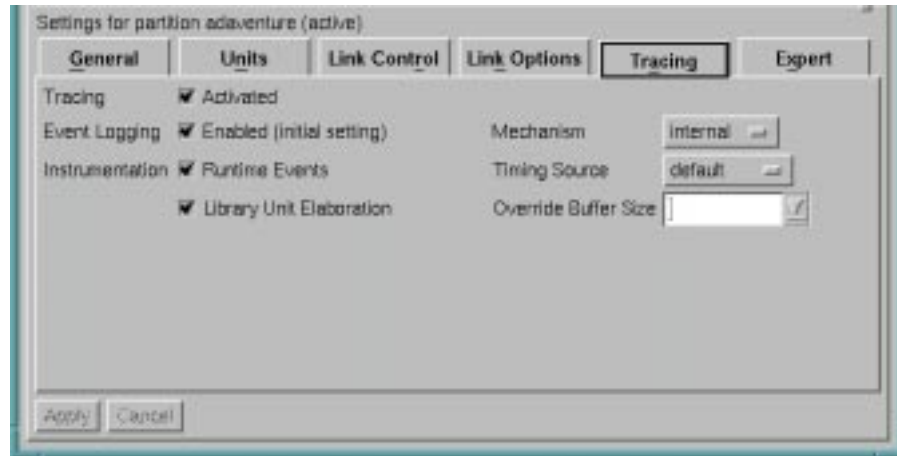


Figure 5-53. Partitions - Tracing settings

Tracing

Activated

Determines whether tracing support will be included in the output file:

When this option is checked, tracing support will be included in the output file. When this option is selected, tracing is automatically enabled. To over-

ride this default, change the setting of **Logging** appropriately (see “Event Logging” on page 5-139).

Tracing may be enabled or disabled at runtime without the need for relinking by calling either:

```
user_trace.set_trace_mode  
or  
user_trace.set_trace_mode_all
```

See the section titled **user_trace package** in Chapter 11 of the *MAXAda Reference Manual* (0890516) for more information.

In addition, tracing may be enabled or disabled using the MAXAda utility, **a.map**. See Chapter 4 of the *MAXAda Reference Manual* (0890516) for more information.

If this checkbox is not checked, tracing support will not be included in the output file. Therefore, if tracing is subsequently desired, the program must be relinked with this checkbox checked.

Event Logging

Enabled

This option allows the user to control initially whether or not trace events are to be logged to a trace buffer. When tracing is first activated (see “Activated” on page 5-138), logging is automatically enabled.

When this option is checked, predefined and user-defined trace events to trace buffer are logged to a trace buffer.

Otherwise, logging of predefined and user-defined trace events is disabled.

Tracing may be enabled or disabled at run-time by calling either:

```
user_trace.set_trace_mode  
or  
user_trace.set_trace_mode_all
```

(See the section titled **user_trace package** in Chapter 11 of the *MAXAda Reference Manual* (0890516) for more information.)

If the partition was linked with tracing activated (see “Activated” on page 5-138), tracing may be enabled or disabled at runtime without the need for relinking by using the MAXAda utility, **a.map**. (See Chapter 4 of the *MAXAda Reference Manual* (0890516) for more information.)

Instrumentation

NightBench provides two different categories of predefined trace events:

Runtime Events

This option causes the tracing version of the Ada runtime executive to generate predefined trace events as the application executes. These trace events describe execution mostly in terms of tasking, interrupt handling, exception occurrence and handling, and protected actions.

Library Unit Elaboration

This option causes the generation of a pair of trace points (entry and exit) for the elaboration of every library unit in the partition.

NOTE

The user may wish to increase the length of the trace buffer used for logging trace events if there are a large number of library units to be traced. (See “Override Buffer Size” on page 5-141 for more information).

Mechanism

Two mechanisms are provided for logging trace events:

internal	This option specifies an Ada runtime executive that logs trace events independent of the NightTrace product. The Ada executive logs trace events to wraparound trace buffers in memory (one buffer per task). When a trace buffer becomes full, the newest trace events overwrite the oldest trace events in that buffer. See “Override Buffer Size” on page 5-141 for details on changing the length of this buffer.
ntraceud	This option specifies an Ada runtime executive that logs trace events via the NightTrace user daemon, ntraceud . This method allows greater flexibility, providing a number of options to tailor the tracing to the needs of the user. See ntraceud(1) and the <i>NightTrace Manual</i> (0890398) for more information about the NightTrace user daemon.

Timing Source

This option allows the user to select the mechanism used to determine timestamps for trace events. The following choices are provided:

default	By default, the interval timer (NightHawk 6000 Series) or the Time Base Register (Power-Hawk/PowerStack) is used to determine timestamps for trace events.
RCIM tick	This option selects the synchronized tick clock on the Real-Time Clock and Interrupt Module (RCIM) as the trace timing source. See the <i>Real-Time Clock and Interrupt Module User's Guide</i> (0891082) for more information about this device.

NOTE

This is not applicable when the tracing **Mechanism** is set to **ntraceud**. (See “Mechanism” on page 5-140.)

The NightTrace user daemon, however, has its own option for selecting a timing source. See the section titled **Option to Select Timestamp Source (-clock)** in Chapter 4 of the *NightTrace Manual* (0890398).

Override Buffer Size

You may specify the size of the trace buffer that the Ada executive uses to log trace events. This length is the maximum number of trace events for each task that can be contained within this buffer. For instance, it may be desirable to specify a fairly large buffer length if there are in excess of 500 library units being traced.

NOTE

This is not applicable when the tracing **Mechanism** is set to **ntraceud**. (See “Mechanism” on page 5-140.)

The NightTrace user daemon, however, has its own option for setting the shared memory buffer size. See the section titled **Option to Define Shared Memory Buffer Size (-memsize)** in Chapter 4 of the *NightTrace Manual* (0890398).

Partitions - Expert

The Expert subpage contains options that will probably not be used by the average user of NightBench. They allow for even further control of the how a particular partition is built.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (Ada)” on page 5-96

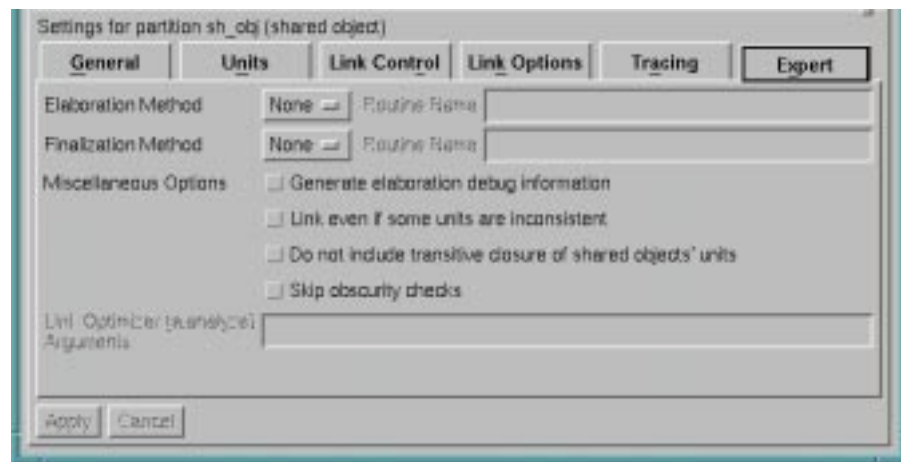


Figure 5-54. Partitions - Expert settings

Elaboration Method

Elaboration is taken care of in active Ada partitions for all archive and shared object partitions included via the *link rule* (see “Partitions - Link Control” on page 5-124) or dependent partitions list (see “Dependent Partitions” on page 5-125). In all other cases, (for example, calling an Ada subprogram from within C++ code, or using a routine that exists in an archive that hasn’t been included in the active partition), this must be done explicitly. Choose the elaboration method from the following:

none	<p>This is the default. Nothing will be done for elaboration. This is generally not recommended for partitions used outside the Ada development environment, but may be useful for partitions containing only pure and preelaborated units.</p>
auto	<p>An elaboration routine is generated at link time and is called before any other user code even runs. The user does not need to be concerned about the routine itself or calling it. Elaboration is handled automatically when this option is specified.</p> <p>This option is not available for archives.</p> <p>NOTE: This option should not be used for partitions that will be included via the <i>link rule</i> (see “Partitions - Link Control” on page 5-124) or dependent partitions list (see “Dependent Partitions” on page 5-125) in active Ada partitions because the automatic elaboration will interfere with the elaboration for the active Ada partition.</p>
user	<p>An elaboration routine named explicitly by the user is generated at link time. The user specifies the actual name in the Routine Name field and makes a call to this routine at some point in the code. The actual call to this Routine Name should be made before any Ada code from this partition is called.</p> <p>WARNING: It is the user's responsibility to ensure that the generated routine is called <u>before</u> any entities in the partition are used.</p> <p>NOTE: If this option is used, Routine Name should not be called for partitions that will be included via the <i>link rule</i> (see “Partitions - Link Control” on page 5-124) or dependent partitions list (see “Dependent Partitions” on page 5-125) in active Ada partitions because the elaboration performed by Routine Name will interfere with the elaboration for the active Ada partition.</p>

Routine Name

When an Elaboration Method of type **user** is chosen, the name for the elaboration routine should be entered here. The user should make the actual call to this **Routine Name** before any Ada code from this partition is called.

Finalization Method

Finalization is taken care of in active Ada partitions for all archive and shared object partitions included via the *link rule* (see “Partitions - Link Control” on page 5-124)

or dependent partitions list (see “Dependent Partitions” on page 5-125). In all other cases, (for example, calling an Ada subprogram from within C++ code, or using a routine that exists in an archive that hasn't been included in the active partition), this must be done explicitly. Choose the finalization method from the following:

none	This is the default. Nothing will be done for finalization. This is generally not recommended for partitions used outside the Ada development environment, but may be useful for partitions that contain no library-level controlled objects with Finalize routines, library-level tasks, or other entities that require finalization.
auto	<p>A finalization routine is generated at link time and is called after all other user code completes. The user does not need to be concerned about the routine itself or calling it. Finalization is handled automatically when this option is specified.</p> <p>This option is not available for archives.</p> <p>NOTE: This option should not be used for partitions that will be included via the <i>link rule</i> (see “Partitions - Link Control” on page 5-124) or dependent partitions list (see “Dependent Partitions” on page 5-125) in active Ada partitions because the automatic finalization will interfere with the finalization for the active Ada partition.</p>
user	<p>A finalization routine named explicitly by the user is generated at link time. The user specifies the actual name in the Routine Name field and makes a call to this routine at some point in the code. The actual call to this Routine Name should be made after all Ada code from this partition is called.</p> <p>WARNING: It is the user's responsibility to ensure that the generated routine is called <u>after</u> all entities in the partition are used.</p> <p>NOTE: If this option is used, Routine Name should not be called for partitions that will be included via the <i>link rule</i> (see “Partitions - Link Control” on page 5-124) or dependent partitions list (see “Dependent Partitions” on page 5-125) in active Ada partitions because the finalization performed by Routine Name will interfere with the finalization for the active Ada partition.</p>

Routine Name

When a Finalization Method of type **user** is chosen, the name for the finalization routine should be entered here. The user should make the actual call to this **Routine Name** after all Ada code from this partition is called.

Miscellaneous Options

Generate elaboration debug information

This option creates a file named “**.ELAB_{main}.a**” representing the elaboration of library units and execution of the main subprogram.

When elaboration debug information is generated for an archive or shared object partition, the file created is named “**.ELAB_{partition_name}.a**”.

Link even if some units are inconsistent

Normally, inconsistent units would cause the linker to generate error messages. This option informs the linker to only issue warning messages when including inconsistently compiled units and excluding uncompiled units.

WARNING

This option must be used with special care. The link could still fail during the execution of the system linker (**ld**) because of unresolved references, or it could contain stale object files.

Do not include transitive closure of shared objects' units

Normally, if the current partition requires any units from a shared object, it implicitly includes all the units from that shared object and any other units and partitions required for those units. If this option is activated, the unrequired units contained within a required shared object are not considered and no units or partition required solely by such units will be included. Extreme caution is recommended when using this option so that attempts to link the partition do not result in undefined symbols.

Skip obscurity checks

Link Optimizer (a.analyze) Arguments

Options to be sent to the MAXAda **a.analyze** tool.

NOTE

Because you may specify multiple arguments, spaces are reserved as separators. However, if you want a space to be *part* of an argument, you need to quote it with a prepended backslash. And because backslash is reserved for the quote character, if you want a backslash in the argument, you need to use a double backslash. For instance, the following option (quotes are shown for clarity):

```
'-g' 'filewith spaceand\backslash'
```

must be specified in NightBench like:

```
-g filewith\ spaceand\backslash
```

See also:

- *MAXAda Reference Manual* (0890516)

Development - Partitions - (C/C++)

The **Partitions** page allows the user to create, configure and build *partitions* in the NightBench Program Development Environment. A partition is an *executable*, *archive*, or *shared object* that can be invoked outside of NightBench.

This main page and its various subpages display detailed information about the partitions defined in the current environment.

The top portion of the **Partitions** page lists the partitions currently defined in the given environment and has buttons for creating, removing, and building partitions.

The bottom section of the **Partitions** page contains a number of subpages that contain specific information about the partition currently selected in the **Partitions List** at the top of the page. These subpages are:

- **General** - see “Partitions - General” on page 5-156
- **Units** - see “Partitions - Units” on page 5-161
- **Partitions** - see “Partitions - Partitions” on page 5-168
- **Libraries** - see “Partitions - Libraries” on page 5-172
- **Link Options** - see “Partitions - Link Options” on page 5-176
- **Expert** - “Partitions - Expert” on page 5-180

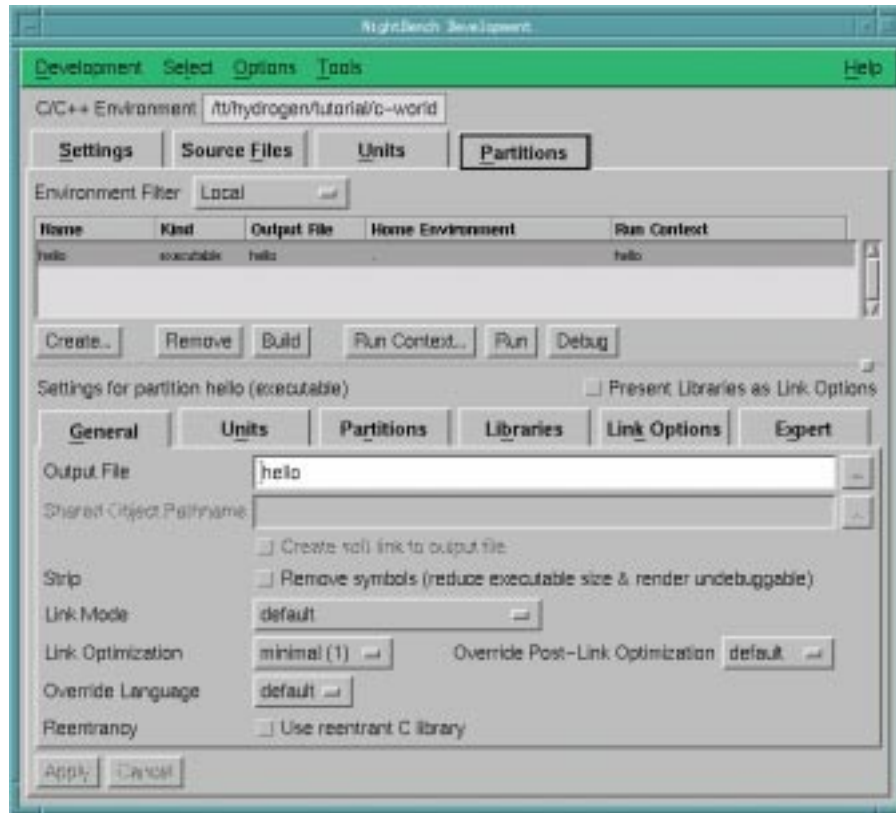


Figure 5-55. NightBench Development - Partitions page

Environment Filter

This filter determines which units and partitions will be displayed in the subpages below.

Local	Filter candidate units to include only units local to the current environment. This includes units introduced directly to the current environment, and those <i>fetch</i> ed or <i>naturalized</i> into the current environment.
All	Include all units in the environment, including those obtained via the <i>Environment Search Path</i> .

Partitions List

A listing of all the partitions that have been created in the current environment.

See also:

- “Accelerated Item Selection” on page 5-16

Name

The name that has been given to this partition.

Kind

The type of partition. NightBench supports four different kinds of partitions:

- *executable*
- *shared object*
- *archive*
- *object*

See “Create Partition” on page 5-154 for more information on these types of partitions.

Output File

The name of the resultant file created when this partition is built. This name is specified on the **General** subpage of the Partition Settings area.

See also:

- “Partition Settings area” on page 5-152
- “Partitions - General” on page 5-156
- “Output File” on page 5-156

Home Environment

The environment in which the unit was compiled.

Run Context

The name of the *run context* currently associated with this partition. A run context is automatically created for each active partition that is created; by default, the run context is created with the same name as the partition.

The user can associate a different run context for a particular partition by selecting the partition from the **Partitions List**, opening the **Run Context** dialog (using the **Run Context...** button on the **Partitions** page), and either selecting a different run context from the drop-down list associated with the **Run Context** field or creating a new run context (see “New Run Context” on page 6-22).

See also:

- “Run Context” on page 6-19

Create...

Create a new partition within the current environment.

NOTE

A *run context* is automatically created for each active partition that is created; by default, the run context is created with the same name as the partition.

See also:

- “Create Partition” on page 5-154
- “Run Context” on page 6-19

Remove

Remove the selected partition from the environment.

NightBench verifies that this action is intentional.

NOTE

This action does not remove the *run context* currently associated with that particular partition. Use the **Delete** button in the Run Context dialog to delete a specific run context.

See also:

- “Run Context” on page 6-19

Build

Opens the NightBench Builder. The build **Targets** (see “Targets” on page 6-13) are any selected partitions in the partitions list (see “Partitions List” on page 5-148). If no partitions are selected, the default build target is **all partitions**.

The build may automatically start if either the **Automatically Start Builds** option (see “Automatically Start Builds” on page 5-14) is selected from the **Options** menu of the NightBench Development window or if this is specified in the **Preferences** dialog.

See also:

- NightBench Builder - Chapter 6
- “Builder - Targets” on page 6-36
- “Preferences - Start” on page 1-14

Run Context...

Opens the Run Context dialog to edit the *run context* currently associated with the *selected partition*. See *run context association* on page Glossary-7 for more information.

NOTE

Any changes made to this run context will affect the selected partition. For example, if the user opens the Run Context dialog, and deletes the run context (using the **Delete** button), the run context currently associated with the given partition will be the run context that is in the Run Context field of the Run Context dialog when the dialog is dismissed.

See also:

- “Run Context” on page 6-19

Run

Executes the *run context* currently associated with the selected partition. See *run context association* on page Glossary-7 for more information.

NOTE

The user may configure NightBench to open the Run Context dialog each time the Run button is pressed. For example, the user may want to change the arguments to the program each time it is run. Select the **Popup this dialog** prior to Run or Debug setting on the **Preference** page of the Run Context dialog for this behavior. (See “Run Context - Preference” on page 6-32 for more information).

See also:

- “Run Context” on page 6-19

Debug

Executes the *run context* currently associated with the selected partition under the NightView debugger. See *run context association* on page Glossary-7 for more information.

NOTE

The user may configure NightBench to open the Run Context dialog each time the **Debug** button is pressed. For example, the user may want to change the arguments to the program each time it is debugged. Select the **Popup** this dialog prior to **Run** or **Debug** setting on the **Preference** page of the Run Context dialog for this behavior. (See “Run Context - Preference” on page 6-32 for more information).

See also:

- “Run Context” on page 6-19
- *NightView User's Guide* (0890395)

Partition Settings area

The bottom half of the **Partitions** page is composed of the following subpages which contain specific information about the partition currently selected in the **Partitions List**.

NOTE

The options listed in these subpages apply only during the linking phase of the build.

See also:

- **General** - see “Partitions - General” on page 5-156
- **Units** - see “Partitions - Units” on page 5-161
- **Partitions** - see “Partitions - Partitions” on page 5-168
- **Libraries** - see “Partitions - Libraries” on page 5-172
- **Link Options** - see “Partitions - Link Options” on page 5-176
- **Expert** - “Partitions - Expert” on page 5-180

Present Libraries as Link Options

The following items are converted by NightBench into link options in the order shown:

- **Units** (see “Partitions - Units” on page 5-161)
- **Dependent Partitions** (see “Partitions - Partitions” on page 5-168)
- **-L** library options (see “Partitions - Libraries” on page 5-172)
- **-l** library options (see “Partitions - Libraries” on page 5-172)

- other link options (see “Partitions - Link Options” on page 5-176)

There may be times when it is necessary for some of the libraries or library paths that appear on the **Libraries** subpage to exist after some other link option. The **Present Libraries as Link Options** option, when checked, takes all of the **-L** and **-l** library options on the **Libraries** subpage, places them at the beginning of the **Link Options** list, and shuts down the **Libraries** subpage. The **-L** and **-l** library options can then be reordered with the other link options that appear in the **Link Options** list.

If **Present Libraries as Link Options** is later unchecked, the **Libraries** subpage will be reactivated and those **-L** and **-l** options that meet the **Libraries** subpage's ordered-first requirement will be moved back there. See “Ordering” on page 5-172 for more information.

Apply

Saves the changes made within the subpages for the selected partition.

Cancel

Disregards any changes within the subpages. All settings are reset to the values last applied.

Create Partition

The Create Partition dialog allows the user to specify the name and type of partition to be created in the current environment.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

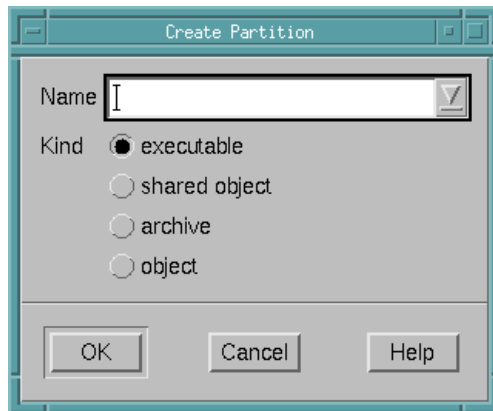


Figure 5-56. Create Partition dialog

Name

The name of the partition to be created.

The drop-down list contains units that may be possible main units.

Kind

The type of partition to be created. NightBench supports four different kinds of partitions:

executable	An executable is an ELF object file that may be executed as a program.
shared object	A shared object is a collection of routines and data that can be used by multiple executable partitions or foreign language executables without its contents being included in them. An executable partition references the shared object during the link phase and associates calls and other references during the execution phase. Shared objects are composed of units built with position independent code (i.e. the compile option Position Independent Code must be applied to those units included in a shared object). (See “Position Independent Code” on page 7-63.)
archive	An archive is a collection of routines and data that can be used by executable partitions or foreign language executables. An executable partition references the archive during the link phase and includes any needed portions of the archive into the output file. An archive is not referenced at execution time. Archives are composed of units built with static code (i.e. the compile option Position Independent Code should not be applied to those units included in an archive). (See “Position Independent Code” on page 7-63.)
object	An object is an ELF object file that may be linked into an executable or shared object.

Partitions - General

General information about the partition is contained on this subpage. The name of the output file is specified here as well as other settings related to such areas as optimization and compression.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

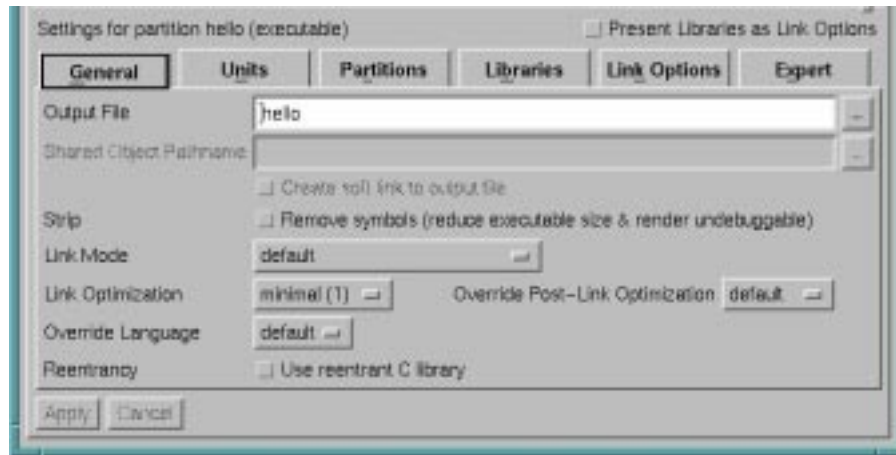


Figure 5-57. Partitions - General settings

Output File

The name of the file generated when the partition is built. This defaults to the name of the partition when it is initially created. An alternate name may be specified in this field.

This field may not be left blank. A name for the output file must be specified.

See also:

- “Create Partition” on page 5-154



Use this to popup a dialog and browse for the file you want. When finished, the pathname of the file is added to the Output File field.

Shared Object Pathname

For use with shared object partitions only.

The shared object's pathname on the target system. It does not cause the shared object to be created at the specified pathname; the shared object will still be built at the pathname specified for the **Output File**. However, all user programs created that require units from this shared object will expect it to be in the location specified here. The shared object must be placed at this pathname before any executable using it can be run. A soft link can be created automatically for this purpose by the **Create soft link to output file** checkbox (see below).



Use this to popup a dialog and browse for the location you would like to specify for the shared object. When finished, the pathname of the file is added to the **Shared Object Pathname** field.

Create soft link to output file

For use with shared object partitions only.

A soft link is created from the shared object's pathname to the output pathname. Using this option in conjunction with the **Shared Object Pathname** removes the need for the shared object to be explicitly placed at the pathname specified by the **Shared Object Pathname**.

Strip

Remove symbols (reduce executable size & render undebuggable)

Strip symbolic information from the output file. The debug and line sections and their associated relocation entries will be removed. Except for relocatable files or shared objects, the symbol table and string table sections will also be removed from the output object file.

Link Mode

Specify whether statically-linked or dynamically-linked object files should be produced.

default	The default is controlled by environment variable <code>STATIC_LINK</code> . If defined, the default is static ; otherwise, the default is dynamic .
static	Produce statically-linked object files (for use in <i>archives</i>)
dynamic	Produce dynamically-linked object files (for use in <i>shared objects</i>)

Link Optimization

The NightBench Builder is capable of performing optimization on linked programs. The same optimization levels that may be specified on compilations may be used to specify link optimization, although not every level is unique:

none (0) minimal (1)	Optimization levels <code>none</code> and <code>minimal</code> perform no link optimizations.
global (2) maximal (3) ultimate (4)	Optimization levels <code>global</code> , <code>maximal</code> , and <code>ultimate</code> optimize generation of memory address computation using four registers as program-wide variables for holding commonly-needed address constants.

NOTE

A future release will additionally support doing interprocedural analysis of register usage to optimize the saving and restoring of registers in subroutine calls using the `ultimate(4)` optimization level.

Override Post-Link Optimization

analyze optimizes programs during the post-linking stage. It uses program-wide, common subexpressions to optimize address and constant computation. (Refer to the **analyze(1)** man page for more information about `analyze`). During the post-linking process, the compiler drivers pass the `-O` option to `analyze`, which invokes the post-linker optimization code in **analyze**. This creates program-wide, common sub-expressions, and insures that the target instruction cache doesn't fail because of instruction misalignment.

default	The default is controlled by the Link Optimization. For levels <code>none(0)</code> and <code>minimal(1)</code> , the default is disabled. For the other levels, the default is enabled.
enabled	Optimize programs during the post-linking stage.
disabled	Do not perform any post-link optimizations.

Override Language

Allows the user to override which link command is used when linking the partition:

default	Automatically selects which link command to use based on the language specified for the units included in the partition. If the prelinker detects any C++ units, ec++ is used as the link command (the -lCruntime , -lm , and -lc link options are implicitly added). If the prelinker does not detect any C++ units, ec is used as the link command (the -lc link option is implicitly added). Note that non-unit object is assumed to be C code.
C	Regardless of the language specified by the units included in this partition, ec is used as the link command (the -lc link option is implicitly added)
C++	Regardless of the language specified by the units included in this partition, ec++ is used as the link command (-lCruntime , -lm , and -lc link options are implicitly added)

The user may wish to override the default for a variety of reasons. For example:

- All the object built by the environment is C code, however, some C++ object is being linked in that wasn't built by the environment. Set the language to C++.
- The user wishes to statically link with **libCruntime.a**, but dynamically link with **libc.so**. To do this, select the C language choice and set the link options

```
--library_linkage=static  
-lCruntime  
-lm  
--library_linkage=dynamic
```

- The user wishes to link with a non-Concurrent-supported math library. To do this, select the C language choice and set the link options:

```
-lCruntime  
-lbobs_math_library
```

See “Partitions - Link Options” on page 5-176 for details on setting link options.

Reentrancy

Use reentrant C library

Disallows the implied use of the non-reentrant C library **libc** at ultimate optimization. The non-reentrant C library **libnc** has better performance compared with **libc** while sacrificing use with threaded or dynamically-linked executables.

See also:

- “Link Optimization” on page 5-158
- “Optimization Level” on page 7-62

Partitions - Units

Units to be included as part of the current partition's definition are specified on this sub-page. In addition, the user may exclude specific units from a partition here as well.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

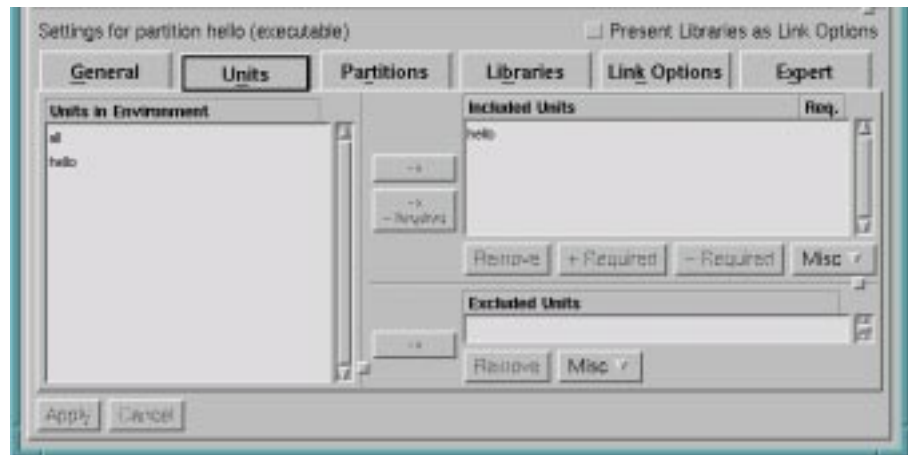


Figure 5-58. Partitions - Units settings

There are three ways a unit may be added to a partition; it may be:

- included (see “Included Units” on page 5-162)
- included transitively (see “Included Units” on page 5-162)
- excluded (see “Excluded Units” on page 5-164)

If it is excluded, then the unit will not be linked/archived in, even if explicitly included, and will not be considered for resolving symbols for transitively included units. It might still get linked in by getting pulled in from a dependent partition, however (there is no way to stop **/bin/ld** from pulling in specific object files from an archive).

Units in Environment

This lists all units in the environment and is filtered by the following filters on the Units page:

- Filter:Env. - which can filter out foreign or system units
- Filter:Other - which can display or filter out artificial units

Foreign, system, and artificial units will not be included in this list if they are not included by these filters on the Units page.

See also:

- “Development - Units” on page 5-61

Included Units

If a unit is included, then its object will be linked/archived into the partition.

If it is included transitively then its object will be linked/archived into the partition unless it is also excluded (see “Excluded Units” on page 5-164). Regardless of whether the object is actually linked/archived, an attempt is made to find a non-excluded unit that defines each undefined symbol referenced in the unit. If one and only one non-excluded local unit defines the symbol, it is transitively included as well. Otherwise, the symbol remains undefined, with the assumption that the symbol will be defined by an explicitly included unit or dependent partition (see “Dependent Partitions” on page 5-168).

The items that appear in this list are sorted alphabetically. However, new units added to this list are appended to the end until the changes are applied, whereupon the list is once again sorted.

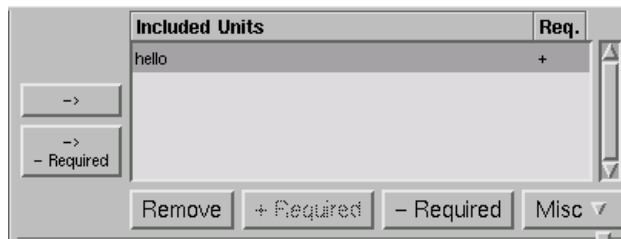
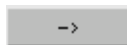
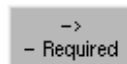


Figure 5-59. Partitions - Included Units



This button adds the selected unit from the Units in Environment to the Included Units list for the current partition. In addition, all units that this unit requires are included in the partition.



Only adds the selected unit as part of this partition definition but does not include the units which that unit requires.

There may be times when the user wishes to add a specific unit to the Included Units list but, depending on the setting of the Environment Filter, the name of this unit may not appear in the Units in Environment list. The user can manually

enter the name of the unit using the Add Included Unit dialog without changing the Environment Filter setting.

See also:

- “Add Included Unit” on page 5-166

Included Units List

Included Units

This lists each of the units to be included when linking this partition.

This list is sorted alphabetically. However, units recently added to this list are appended to the bottom until the changes are applied. The advantage of this is that all newly-added units are grouped together at the end of the list where they can be manipulated as a group easily. When the Apply button is pressed, the list is re-sorted.

Req.

A + in this column indicates that all units required by the given unit are to be included when linking this partition.

Remove

Removes the unit from the current partition definition.

+ Required

In addition to the unit selected in the Included Units list, this button also specifies that any units that this unit requires must be included by the linker for this partition.

- Required

This button specifies that only the unit selected in the Included Units list is required as part of this partition but not necessarily the units that this unit requires.

Misc

Add Unlisted Included Unit

This brings up a dialog that may be used to include a unit in a partition’s definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the Units in Environment list.

In addition, depending on the setting of the Environment Filter, the name of the desired unit may not appear in the Units in Environment list. For instance, if the Environment Filter is set to Local, units in other environments on the *Environment Search Path* will not be displayed in the Units in

Environment list. This dialog allows the user to add the included unit to the partition's definition regardless of the setting of the Environment Filter.

See also:

- “Add Included Unit” on page 5-166

Convert Units to Link Options

The C/C++ interface has two ways of specifying included units in a partition definition. They may either be placed in the list of **Included Units** or they may be specified as link options. This option will change the partition definition by converting the units contained in the list of **Included Units** into **-U** link options.

Changing units into link options can subtly change the definition of the partition by causing the units to be linked into the partition after other dependent partitions, **-L** or **-l** options.

The following items are converted by NightBench into link options in the order shown:

- Units (see “Partitions - Units” on page 5-161)
- Dependent Partitions (see “Partitions - Partitions” on page 5-168)
- **-L** library options (see “Partitions - Libraries” on page 5-172)
- **-l** library options (see “Partitions - Libraries” on page 5-172)
- other link options (see “Partitions - Link Options” on page 5-176)

By converting units into **-U** link options, they are essentially reordered past the dependent partitions, **-L** and **-l** options. The user is warned about doing that (unless there are NO dependent partitions, **-L** or **-l** options).

Excluded Units

The items that appear in this list are sorted alphabetically. However, new units added to this list are appended to the end until the changes are applied, whereupon the list is once again sorted.



Figure 5-60. Partitions - Excluded Units



This button adds the selected unit from the **Units in Environment** to the **Excluded Units** list for the current partition. In addition, all units that this unit requires are excluded from this partition definition.

Excluded Units List

This lists each of the units that are excluded from the current partition's definition.

This list is sorted alphabetically. However, units recently added to this list are appended to the bottom until the changes are applied. The advantage of this is that all newly-added units are grouped together at the end of the list where they can be manipulated as a group easily. When the **Apply** button is pressed, the list is re-sorted.

Remove

Removes the unit selected in the **Excluded Units** list from this list.

Misc

Add Unlisted Excluded Unit

This brings up a dialog that may be used to exclude a unit from a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the **Units in Environment** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired unit may not appear in the **Units in Environment** list. For instance, if the **Environment Filter** is set to **Local**, units in other environments on the *Environment Search Path* will not be displayed in the **Units in Environment** list. This dialog allows the user to exclude the unit from the partition's definition regardless of the setting of the **Environment Filter**.

See also:

- “Add Excluded Unit” on page 5-166

Add Included Unit

This dialog allows the user to include a unit in a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the **Units in Environment** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired unit may not appear in the **Units in Environment** list. For instance, if the **Environment Filter** is set to **Local**, units in other environments on the *Environment Search Path* will not be displayed in the **Units in Environment** list. This dialog allows the user to add the included unit to the partition's definition regardless of the setting of the **Environment Filter**.

See also:

- "Partitions - Units" on page 5-161
- "Included Units" on page 5-162

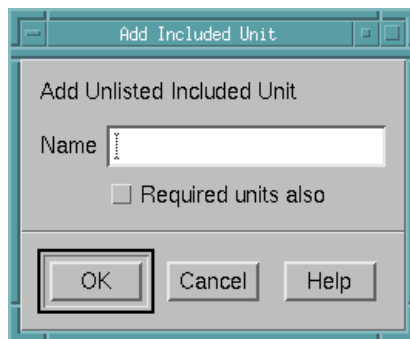


Figure 5-61. Partitions - Add Included Unit dialog

Name

The name of the unit which is to be included in the given partition's definition.

Required Units Also

This checkbox specifies that all units that this unit requires are added to the list of included units for this partition.

Add Excluded Unit

This dialog allows the user to exclude a unit from a partition's definition by manually entering the name of the unit.

This can be used for units which have not yet been introduced into the environment and therefore do not appear in the **Units in Environment** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired unit may not appear in the **Units in Environment** list. For instance, if the **Environment Filter** is set to **Local**, units in other environments on the *Environment Search Path* will not be displayed in the **Units in Environment** list. This dialog allows the user to exclude the unit from the partition's definition regardless of the setting of the **Environment Filter**.

See also:

- “Partitions - Units” on page 5-161
- “Excluded Units” on page 5-164

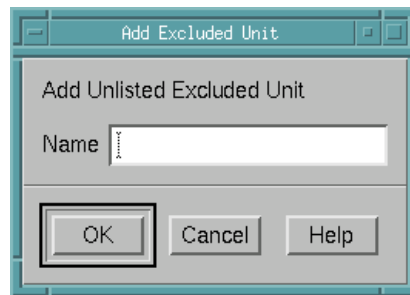


Figure 5-62. Partitions - Add Excluded Unit dialog

Name

The name of the unit which is to be excluded from the given partition's definition.

Partitions - Partitions

The **Partitions** subpage allows users to list the names of dependent partitions. Dependent partitions are those partitions in the *environment* (or on the *Environment Search Path*) that should be searched first and foremost when trying to find the required units for a partition during the linking phase.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

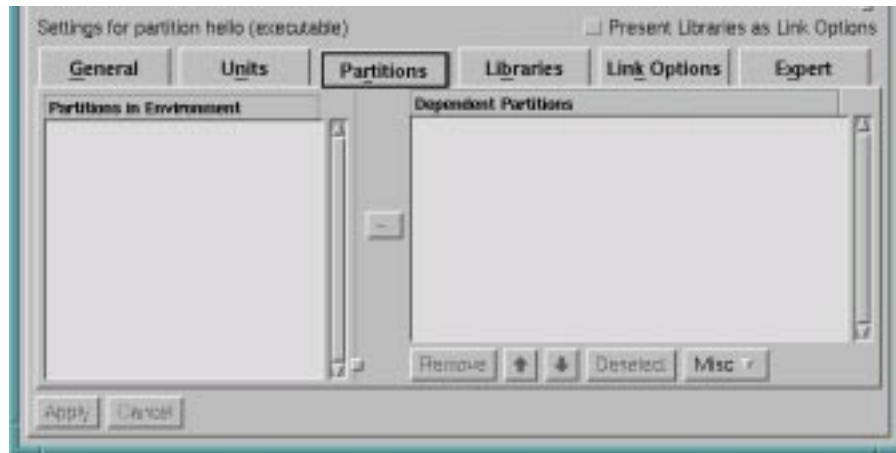


Figure 5-63. Partitions - Partitions settings

Partitions in Environment

This area lists the *shared object*, *archive*, or *object* partitions that are visible to this environment based on the setting of the **Environment Filter**. See “**Environment Filter**” on page 5-148 for details.

Dependent Partitions

The *dependent partitions* list allows the user to list specific *shared object*, *archive*, or *object* partitions in the environment (or on the *Environment Search Path*) that should be searched first and foremost when trying to find the required units for a given partition during the linking phase. Dependent partitions are searched in the order in which they appear in the **Dependent Partitions** list. Furthermore, partitions that are mentioned in the **Dependent Partitions** list will always be included when linking the partition. This is especially important for *executable partitions* which mention dependent shared object partitions, because the linked executable will require the dependent shared object partition to exist at run time.

For instance, if two archives have an overlapping set of units, you could specify from which one you would like to get the required unit(s) by adding it to the dependent partitions list.



Adds a partition from the **Partitions in Environment** list to end of the **Dependent Partitions** list. However, if a partition is selected in the **Dependent Partitions** list, any new partitions will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to **Deselect** any selection in that list.

Dependent Partitions

Lists the names of partitions that should be searched first and foremost when trying to find the required units for a given partition during the linking phase.

See also:

- “Dependent Partitions” on page 5-168

Remove

Removes the selected partition from the **Dependent Partitions** list.



Changes the position of the selected partition in the **Dependent Partitions** list to an earlier (more preferred) position. Dependent partitions are searched in the order in which they appear in the **Dependent Partitions** list.



Changes the position of the selected partition in the **Dependent Partitions** list to a later (less preferred) position. Dependent partitions are searched in the order in which they appear in the **Dependent Partitions** list.

Deselect

Clears any selections in the **Dependent Partitions** list. If a partition is selected in the **Dependent Partitions** list, any new partitions will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to deselect any selection in that list.

Misc

Add Unlisted Dependent Partition

This brings up a dialog that may be used to add a partition to the **Dependent Partitions** list by manually entering the name of the partition.

This can be used for partitions which have not yet been defined in the environment (or in an environment on the *Environment Search Path*) and therefore do not appear in the **Partitions in Environment** list.

In addition, depending on the setting of the **Environment Filter**, the name of the desired partition may not appear in the **Partitions in Environment** list. For instance, if the **Environment Filter** is set to **Local**, partitions in other environments on the *Environment Search Path* will not be displayed in the **Partitions in Environment** list. This dialog allows the user to add a partition in the **Dependent Partitions** list regardless of the setting of the **Environment Filter**.

See also:

- “Add Dependent Partition” on page 5-171

Convert Dependent Partitions to Link Options

The C/C++ interface has two ways of specifying dependent partitions in a partition definition. They may either be placed in the list of **Dependent Partitions** or they may be specified as link options. This option will change the partition definition by converting the dependent partitions contained in the list of **Dependent Partitions** into **-P** link options.

Changing dependent partitions into link options can subtly change the definition of the partition by causing the dependent partitions to be linked into the partition after other **-L** or **-l** options.

The following items are converted by NightBench into link options in the order shown:

- Units (see “Partitions - Units” on page 5-161)
- Dependent Partitions (see “Partitions - Partitions” on page 5-168)
- **-L** library options (see “Partitions - Libraries” on page 5-172)
- **-l** library options (see “Partitions - Libraries” on page 5-172)
- other link options (see “Partitions - Link Options” on page 5-176)

By converting dependent partitions into **-P** link options, they are essentially reordered past other **-L** and **-l** options. The user is warned about doing that (unless there are **NO -L** or **-l** options).

Add Dependent Partition

This dialog allows the user to add a partition to the Dependent Partitions list by manually entering the name of the partition.

This can be used for partitions which have not yet been defined in the environment (or in an environment on the *Environment Search Path*) and therefore do not appear in the Partitions in Environment list.

In addition, depending on the setting of the Environment Filter, the name of the desired partition may not appear in the Partitions in Environment list. For instance, if the Environment Filter is set to Local, partitions in other environments on the *Environment Search Path* will not be displayed in the Partitions in Environment list. This dialog allows the user to add a partition in the Dependent Partitions list regardless of the setting of the Environment Filter.

See also:

- “Partitions - Partitions” on page 5-168
- “Dependent Partitions” on page 5-168

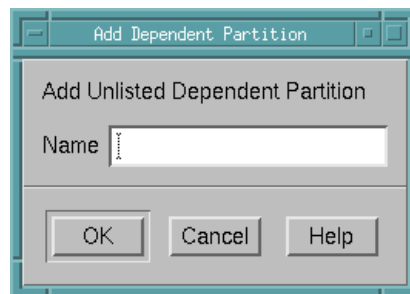


Figure 5-64. Partitions - Add Dependent Partition dialog

Name

The name of the dependent partition which is to be included in the given partition's definition.

Partitions - Libraries

The **Libraries** subpage maintains a list of libraries to link with the partition as well as a list of library paths to search when including those libraries.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

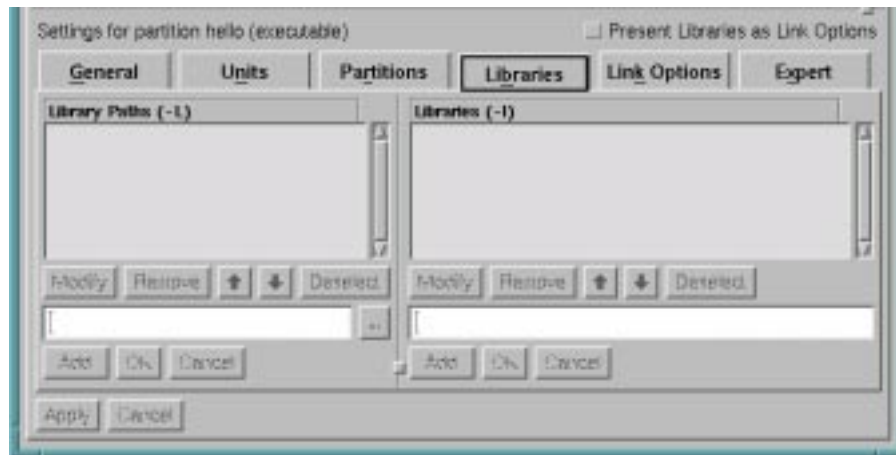


Figure 5-65. Partitions - Libraries settings

Ordering

Normally, **-l** and **-L** link options (as well as their long counterparts, **--library** and **--library_directory**) are presented on the **Libraries** page, since they indicate libraries and library directories (see “Libraries (-l)” on page 5-174 and “Library Paths (-L)” on page 5-173). When a partition is defined or its definition is modified, NightBench orders the **-L** options on the **Libraries** page first, then the **-l** options on the **Libraries** page, and then the link options on the **Link Options** page.

Only **-L** link options that appear before:

- other **-l** options, and
- other options that would appear on the **Link Options** page

are presented on the **Libraries** page.

Likewise, only **-l** options that appear before other options that would appear on the **Link Options** page appear on the **Libraries** page.

Any other **-L** and **-l** options are always presented as **Link Options**, because it is concluded from their position that their ordering relative to the other **Link Options** is important.

There may be times when it is necessary for some of the libraries or library paths that appear on the **Libraries** subpage to exist after some other link option. See “Present Libraries as Link Options” on page 5-152 for more information on how to do this.

See also:

- “Partitions - Link Options” on page 5-176

Library Paths (-L)

The paths specified here are added to the library search path. The linker first searches for libraries within the directories listed here in the order in which they appear and then the standard directories. See “Libraries (-l)” on page 5-174 for more information about specifying libraries.

Modify

Enters the text of the selected library path from the **Library Paths** list into the text field below this list so that the user may modify the library path.

Use the button labeled ... to bring up a dialog to navigate to the directory you would like to add to the library search directories.

The user may either press the **OK** button when the modification is complete or press **Cancel** to disregard any changes to the library path.

Remove

Removes the selected library path from the **Library Paths** list.



Changes the position of the selected library path in the **Library Paths** list to an earlier (more preferred) position. Library paths are searched in the order in which they appear in the **Library Paths** list.



Changes the position of the selected library path in the **Library Paths** list to a later (less preferred) position. Library paths are searched in the order in which they appear in the **Library Paths** list.

Deselect

Clears any selections in the **Library Paths** list. If a library path is selected in the **Library Paths** list, any new library paths will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to deselect any selection in that list.

Any library path may be directly entered in the text field below the list of Library Paths.



Use this to popup a dialog and browse for the directory you would like to add to the library search directories. When finished, the pathname is added as the argument to the **-L** option.

Add / Insert

If a library path is selected in the Library Paths list, any new library paths entered in the text field will be inserted *before* the selected one. If you want your additions to be appended to the end of the Library Paths list, you need to **Deselect** any selection in that list.

OK

Apply the changes to the modified library path and incorporate it into the list of Library Paths.

Cancel

Disregard any changes to the modified library path.

Libraries (-l)

Search a library **libx**. The linker searches each directory specified in the library search path (see “Library Paths (-L)” on page 5-173) for a file **libx.so** or **libx.a**, the conventional names for shared object and archive libraries, respectively. The directory search stops at the first directory containing either. The linker chooses the file ending in **.so** if **libx** expands to two files whose names are of the form **libx.so** and **libx.a**. If no **libx.so** is found, then the linker accepts **libx.a**.

Modify

Enters the text of the selected library path from the Libraries list into the text field below this list so that the user may modify the library name.

Use the button labeled ... to bring up a dialog to navigate to the library you would like to add to the list of libraries that are linked when building the current partition.

Remove

Removes the selected library from the Libraries list.



Changes the position of the selected library in the **Libraries** list to an earlier (more preferred) position. Libraries are searched in the order in which they appear in the **Libraries** list.



Changes the position of the selected library in the **Libraries** list to a later (less preferred) position. Libraries are searched in the order in which they appear in the **Libraries** list.

Deselect

Clears any selections in the **Libraries** list. If a library is selected in the **Libraries** list, any new libraries will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to deselect any selection in that list.

Any library may be directly entered in the text field below the list of **Libraries**.



Use this to popup a dialog and browse for the library you would like to be linked with this partition. When finished, the pathname to this library is added as the argument to the **-l** option.

Add / Insert

If a library is selected in the **Libraries** list, any new libraries will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to **Deselect** any selection in that list.

OK

Apply the changes to the modified library and incorporate it into the list of **Libraries**.

Cancel

Disregard any changes to the modified library.

Partitions - Link Options

Link options for the given partition may be specified on this subpage. The more common link options are presented in a list for easier selection. In addition, other link options for the current partition may be directly entered on this subpage.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

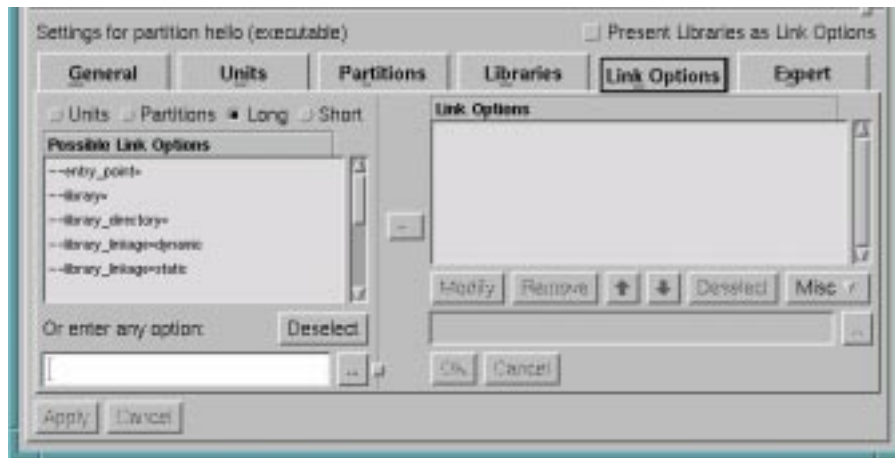


Figure 5-66. Partitions - Link Options settings

Possible Link Options

This area shows a number of the more common link options for the current partition. This list varies depending on which radiobutton above this list is selected.

When an item is selected from this list, NightBench may enter the option in the text field below the list of **Possible Link Options** and position the cursor following the option so that the user may specify any required arguments. For those options that do not require any arguments (e.g. **--library_linkage=dynamic** and **--library_linkage=static**), the link option can be added directly to the list of **Link Options** either by double-clicking the item in the list of **Possible Link Options** or by selecting the item and pressing the -> button.

Any link option may be directly entered in the text field below the list of **Possible Link Options**.

In addition, the text associated with a selected option from the list of **Possible Link Options** will be entered in this field so the user may modify the option if necessary. For those options which require additional text (e.g. **--library=**), the label for this text field will read either **Option should be completed** or **Option must be completed**, depending on the particular option. The user can then complete the option and add it to the list of **Link Options**.

For those options that take filenames (or directory names) as arguments, use the button labeled ... to bring up a dialog to find the file/directory you want.

NOTE

Because you may specify multiple options, spaces are reserved as separators. However, if you want a space to be *part* of an argument, you need to quote it with a prepended backslash. And because backslash is reserved for the quote character, if you want a backslash in the argument, you need to use a double backslash. For instance, the following option (quotes are shown for clarity):

```
'--library_directory=\dirwith spaceand\backslash'
```

must be specified in NightBench like:

```
--library_directory=\\dirwith\ spaceand\\backslash
```

Deselect

Deselects the selected item in the list of **Possible Link Options** and clears the text entered in the text field below the list of **Possible Link Options**.



For link options that take filenames (or directory names) as arguments, use this to popup a dialog and browse for the file/directory you want. When finished, the file/directory is added as the argument to the link option.



Adds the option that appears in the text field below the list of **Possible Link Options** to the end of the list of **Link Options** for the given partition. However, if a link option is selected in the **Link Options** list, any new link options will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to **Deselect** any selection in that list.

Link Options

Lists the current link options for the given partition in order. The user may **Modify** or **Remove** any of these options by selecting the desired option and pressing the appropriate button. In addition, the user may change the order of these options by using the up and down arrow buttons.

Note that option order is important for a variety of reasons. Libraries and library paths are searched in the order in which they appear in this list. In addition, object files may alleviate the need for a given library if specified before it and may create a multiple symbol error if they follow it.

Modify

Enters the text of the selected option in the **Link Options** list into the text field below this list so that the user may modify the option.

For those options that take filenames (or directory names) as arguments, use the button labeled ... to bring up a dialog to find the file/directory you want.

Remove

Removes the selected option from the **Link Options** list.



Changes the position of the selected link option in the list of **Link Options** to an earlier position.

Note that option order is important for a variety of reasons. Libraries and library paths are searched in the order in which they appear in this list. In addition, object files may alleviate the need for a given library if specified before it and may create a multiple symbol error if they follow it.



Changes the position of the selected link option in the list of **Link Options** to a later position.

Note that option order is important for a variety of reasons. Libraries and library paths are searched in the order in which they appear in this list. In addition, object files may alleviate the need for a given library if specified before it and may create a multiple symbol error if they follow it.

Deselect

Clears any selections in the **Link Options** list. If an option is selected in the **Link Options** list, any new link options will be inserted *before* the selected one. If you want your additions to be appended to the end, you need to deselect any selection in that list.

Misc

Convert Link Options to Units and Dependent Partitions

The C/C++ interface has two ways of specifying included units and dependent partitions in a partition definition. They may either be placed in the **Included Units** list or **Dependent Partitions** list or they may be specified as link options. This option will change the partition definition by converting the **-U** and **-P** link options into corresponding entries in the **Included Units** list and **Dependent Partitions** list, appropriately.

Changing **-U** and **-P** link options into entries in the **Included Units** list or **Dependent Partitions** list, respectively, can subtly change the definition of the partition by causing the units to be linked into the partition after other dependent partitions, **-L** or **-l** options.

By converting units into **-U** link options, they are essentially reordered past the dependent partitions, **-L** and **-l** options. The user is warned about doing that (unless there are **NO** dependent partitions, **-L** or **-l** options).

See also:

- “Included Units” on page 5-162
- “Dependent Partitions” on page 5-168

The text field below the list of **Link Options** and its associated buttons is used when modifying an option. When the **Modify** button is pressed for a given link option, the text of that link option is placed in this field for editing.



For link options that take filenames (or directory names) as arguments, use this to popup a dialog and browse for the file/directory you want. When finished, the file/directory is added as the argument to the text field containing the modified link option.

OK

Apply the changes to the modified link option and incorporate it into the list of **Link Options**.

Cancel

Disregard any changes to the modified link option.

Partitions - Expert

The Expert subpage contains options that will probably not be used by the average user of NightBench. They allow for even further control of the how a particular partition is built.

See also:

- NightBench Development - Chapter 5
- “Development - Partitions - (C/C++)” on page 5-147

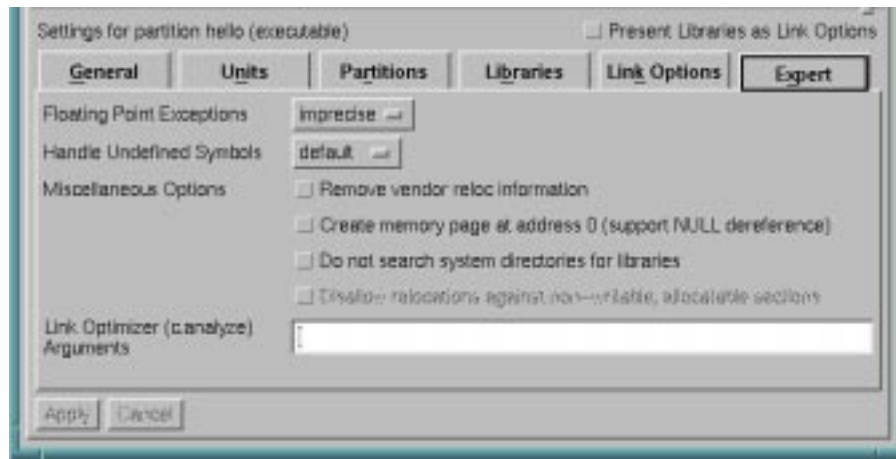


Figure 5-67. Partitions - Expert settings

Floating Point Exceptions

Control hardware floating-point exception handling.

Handle Undefined Symbols

Miscellaneous Options

Remove vendor reloc information

Do not generate relocation in the vendor section. This disables the ability of **analyze(1)** to optimize a program.

See also:

- “Link Optimization” on page 5-158
- “Override Post-Link Optimization” on page 5-158

Create memory page at address 0 (support NULL dereference)

Support dereferencing of null pointers. By default, the link editor excludes addresses 0 through 0xffff (inclusive) from the address space of the program. This option directs the link editor to create a segment at addresses 0 through 0xffff (inclusive), consisting entirely of read-only zeroes.

This option corresponds to the `-zlowzeroes` (or `-zlowzeros`) link option. See `ld(1)` for more information.

Do not search system directories for libraries

Do not look in unspecified search paths for include files or compilation processors. An error message will be generated if the files cannot be found in the specified search paths.

Disallow relocations against non-writable, allocatable sections

In dynamic mode only, force a fatal error if any relocations against non-writable, allocatable sections remain. (See “Link Mode” on page 5-157).

This option corresponds to the `-ztext` link option. See `ld(1)` for more information.

Link Optimizer (c.analyze) Arguments

Options to be sent to the `c.analyze` tool.

NOTE

Because you may specify multiple arguments, spaces are reserved as separators. However, if you want a space to be *part* of an argument, you need to quote it with a prepended backslash. And because backslash is reserved for the quote character, if you want a backslash in the argument, you need to use a double backslash. For instance, the following option (quotes are shown for clarity):

```
'-g' 'filewith spaceand\backslash'
```

must be specified in NightBench like:

```
-g filewith\ spaceand\\backslash
```

See also:

- *C/C++ Reference Manual* (0890497)

The NightBench Builder is the mechanism used for compilation and linking of Ada units and partitions. It can be used in conjunction with the NightBench Development window or stand-alone on existing environments.

The Builder allows the user to specify any combination of units and partitions within the current environment as its targets.

Structure

The NightBench Builder window contains the following five pages:

- **Build** (see page 6-13)
- **Targets** (see page 6-36)
- **Settings** (see page 6-38)
- **Notification** (see page 6-41)
- **Expert** (see page 6-44)

At the top of every page of the NightBench Builder window is listed:

Ada Environment / C/C++ Environment

The name of the environment currently being operated upon by the NightBench Builder.

In addition, other functionality can be found on the Builder menu bar (see page 6-2).

Builder Menu Bar

The NightBench Builder menu bar consists of the following items:

- “Builder Menu” on page 6-2
- “Options Menu” on page 6-12
- “Tools Menu” on page 1-19
- “Help Menu” on page 1-21

Builder Menu

The Builder menu appears on the Builder menu bar.

See also:

- “Builder Menu Bar” on page 6-2

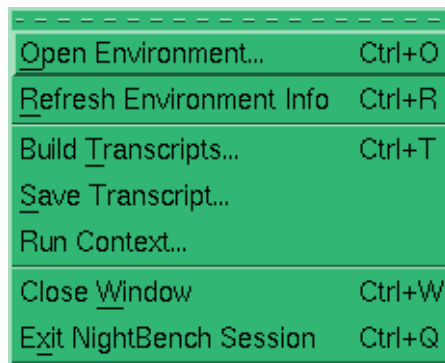


Figure 6-1. Builder menu

Open Environment...

Opens a new environment in the Builder window.

See also:

- “Open Environment” on page 6-3

Refresh Environment Info

Updates all the information in the NightBench Builder window for the current environment.

This is useful for informing NightBench when an environment changes due to an external source (for instance, the command-line interface or another NightBench session).

Build Transcripts...

Opens the Build Transcripts dialog for the management of build transcripts.

See also:

- “Build Transcripts” on page 6-6

Save Transcript...

Saves the current build transcript to a file.

See also:

- “Save Transcript” on page 6-9

Run Context...

Opens a dialog to edit *run contexts*.

See also:

- “Run Context” on page 6-19

Close Window

Closes the NightBench Builder window, leaving any other NightBench windows or internal utilities running.

Exit NightBench Session

Closes all of the NightBench windows and shuts down all NightBench servers and internal utilities.

Open Environment

The Open Environment dialog allows the user to open a preexisting environment either in the current Builder window or a new Builder window.

See also:

- NightBench Builder - Chapter 6
- “Builder Menu” on page 6-2
- “Builder - Build” on page 6-13

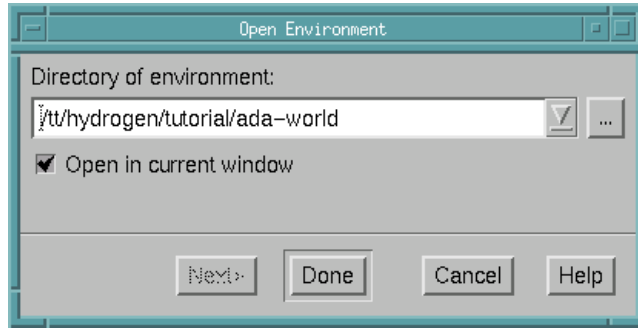


Figure 6-2. Open Environment dialog

Directory of environment

The environment to be opened in the Builder window is specified in this field, either by manually entering the environment name or by using the file selection dialog (...) to navigate to it.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the Directory of environment field.

Open in current window

If this option is checked, the selected environment will be displayed in the current Builder window. If this option is not checked, a new Builder window will be opened for the specified environment.

If more than one environment exists in the directory specified, the **Next** button will be activated to allow you to specify the language of the environment to open:

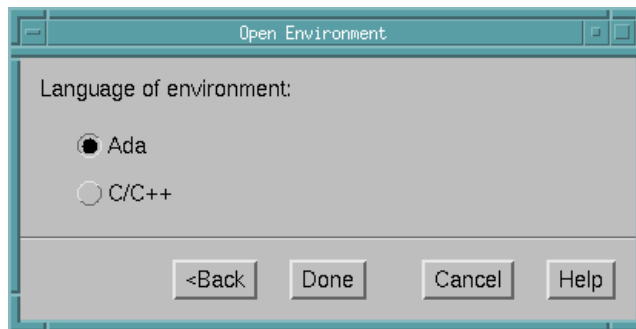


Figure 6-3. Open Environment - Language Selection

NOTE

This dialog is bypassed if a default language has been selected in the **Languages** section of the **Preferences** dialog. (See “Preferences - Language” on page 1-13.)

Language of environment

This dialog allows the user to specify the programming language of the environment you wish to open.

Ada

Open the environment used for the development of Ada programs.

C/C++

Open the environment used for the development of C or C++ programs.

Build Transcripts

The **Build Transcripts** dialog lists the transcripts of previous builds that have occurred in the current environment. This dialog allows the user to view details regarding a particular transcript and also permits the user to delete selected transcripts.

When any of these transcripts are opened, they are loaded into the Builder as though the build had just completed. This allows the user to peruse the results of the build, using the **Edit** buttons to edit source files associated with errors. In addition, for Ada builds, the user may look up RM and MAXAda references for specific errors. Note that the source files associated with any error messages in the transcript may have changed since the build was completed; NightBench does not attempt to preserve the source files at the time of each build.

Removal of transcripts can also be performed by NightBench whenever the Builder exits from an environment (such as when the Builder exits or when a new environment is opened in the Builder window). The settings related to this automatic transcript removal are located in the **Preferences** dialog under the **Build Transcripts** section. The user can select the number of transcripts, if any, that are to remain on the system and also whether verification at the time of deletion is required.

See also:

- “Builder Menu” on page 6-2
- “Preferences - Transcripts” on page 1-12
- “Transcript” on page 6-15
- “Save Transcript” on page 6-9

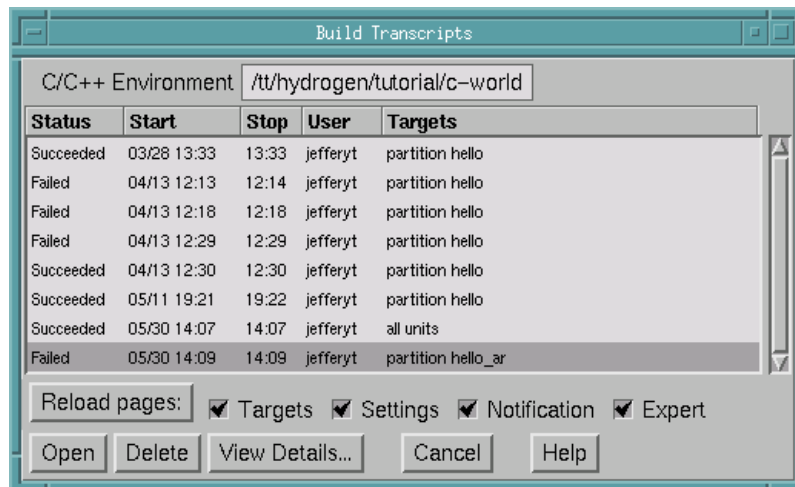


Figure 6-4. Build Transcripts dialog

Ada Environment / C/C++ Environment

The name of the environment associated with this list of transcripts.

Build Transcript List

A listing of all the saved build transcripts in the current environment.

Status

The status of the build. This can be one of the following:

- Running

The current build is executing. No stop time is indicated.

- Succeeded

The build completed with no errors. Warnings and informational messages may have been issued, however.

- Failed

The build terminated with errors. Warnings and informational messages may have been issued, however.

- Stopped

The build was manually stopped by the user.

- Died Abnormally

The build was terminated by an unexpected event such as a system crash or a program error.

Start

The date and time the build was started.

Stop

The time the build ended. For a currently executing build, this field is left blank.

User

The user who executed the build.

Targets

The partitions and/or units specified for this particular build.

Reload pages

When this button is pressed, the settings for the pages in the Builder corresponding to the selected checkboxes (Targets, Settings, Notification, Expert) will be set to the values as saved by the selected transcript.

See also:

- “Builder - Targets” on page 6-36
- “Builder - Settings” on page 6-38
- “Builder - Notification” on page 6-41
- “Builder - Expert” on page 6-44

Open

Loads the selected transcript in the Builder window on the **Build** page as if the build had just completed. Any errors, warnings, or informational messages that have been issued appear in the **Errors Window**.

Use the **Reload pages** button and its associated checkboxes on this page to also load the settings in the **Targets**, **Settings**, **Notification**, and **Expert** pages to the values as saved in the selected transcript.

See also:

- “Builder - Build” on page 6-13
- “Builder - Targets” on page 6-36
- “Builder - Settings” on page 6-38
- “Builder - Notification” on page 6-41
- “Builder - Expert” on page 6-44

Delete

Removes the currently selected transcript.

View Details...

Brings up a dialog which lists build-specific information about the selected transcript such as the list of targets, executable program name, and whether the program is to be run in a terminal window.

Cancel

Exits the Build Transcripts dialog without making any changes.

Help

Opens the online help to a discussion of this dialog.

Save Transcript

This dialog allows the user to save the current build transcript to a file. The user may save the Transcript text, the text in the Errors Window or both. This dialog also gives the user the option to view or edit the file after it has been saved.

See also:

- “Builder Menu” on page 6-2
- “Build Transcripts” on page 6-6
- “Transcript” on page 6-15
- “Errors Window” on page 6-17

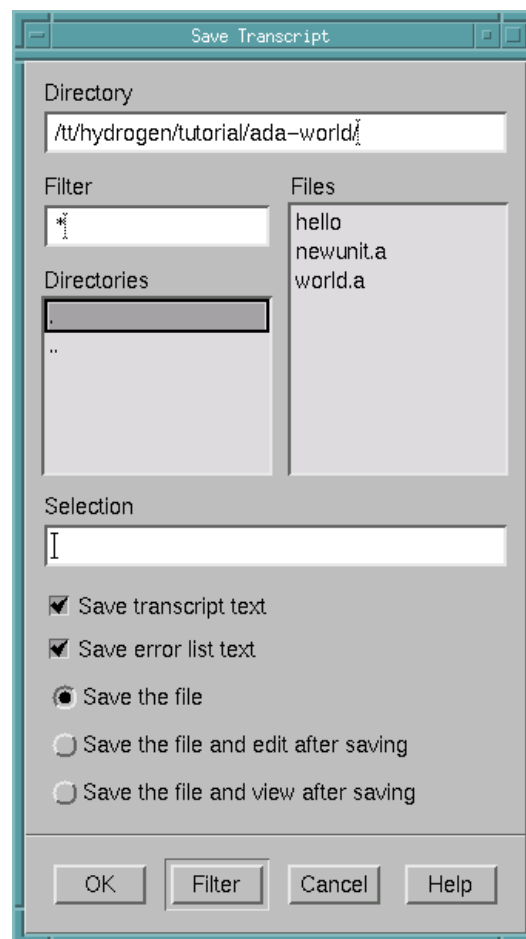


Figure 6-5. Save Transcript dialog

Directory

The current directory in which to save the build transcript.

The user may type the name of a directory into this field directly. Press the **Filter** button to refresh the contents of the **Directories** and **Files** lists after any changes to this field.

Filter

Of all the files contained in the current **Directory**, display only those that match the specified filter. Press the **Filter** button to refresh the contents of the **Directories** and **Files** lists after any changes to this field.

Directories

Contains a list of all the subdirectories within the current directory. Selecting any of these changes the current **Directory** to that subdirectory. Double-clicking on any of these directory names will change to that directory and update the **Files** list accordingly.

Files

Within the current **Directory**, this lists all files that match the specified **Filter**. Any of these files can be selected. When selected, the filename appears in the **Selection** field.

Selection

The name of the file to save the build transcript in the specified **Directory**. The user either may type the name for that file in this field or may select a file from the **Files** list whereupon it will be entered in this field.

Save transcript text

When this option is checked, the text that appears in the **Transcript** window of the NightBench Builder for the current transcript will be saved to the file specified in the **Selection** field in the given **Directory**.

See also:

- “Transcript” on page 6-15
- “Build Transcripts” on page 6-6

Save error list text

When this option is checked, the text that appears in the **Errors Window** of the NightBench Builder for the current transcript will be saved to the file specified in the **Selection** field in the given **Directory**.

See also:

- “Errors Window” on page 6-17
- “Build Transcripts” on page 6-6

Save the file

Saves the current transcript to the file specified in the **Selection** field in the given **Directory**. Other options available are **Save the file and edit after saving** or **Save the file and view after saving**.

Save the file and edit after saving

Saves the current transcript to the file specified in the **Selection** field in the given **Directory** and opens the saved file in the configured file editor. Other options available are **Save the file** or **Save the file and view after saving**.

See also:

- “Preferences - Editor” on page 1-9

Save the file and view after saving

Saves the current transcript to the file specified in the **Selection** field in the given **Directory** and opens the saved file in the configured file viewer. Other options available are **Save the file** or **Save the file and edit after saving**.

See also:

- “Preferences - Viewer” on page 1-8

Filter

This button refreshes the contents of the **Directories** and **Files** lists based on any changes made to the **Directory** or **Filter** fields.

Options Menu

The Options menu appears on the Builder menu bar.

See also:

- “Builder Menu Bar” on page 6-2

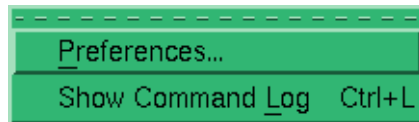


Figure 6-6. Options menu

Preferences...

Opens the Preferences dialog so that the user may configure which file viewer, file editor, and terminal program NightBench should use.

See also:

- “Preferences” on page 1-8

Show Command Log

This opens a window which displays the specific commands that are being performed for each action specified and contains a history of these commands since this Builder window was opened.

There is a setting in the Preferences dialog which controls the value of this when a new Builder window is opened.

See also:

- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)
- “Command Log” on page 1-18
- “Preferences - Start” on page 1-14

Builder - Build

The Build page contains the functionality for the compilation of units and linking of partitions within the current environment. Programs can be built, verified, run and debugged from this page. A transcript of the current build is displayed here. Also, errors, warnings and other informational messages appear on this page and the source files where these errors occurred can be opened to those locations using features on this page.

See also:

- NightBench Builder - Chapter 6

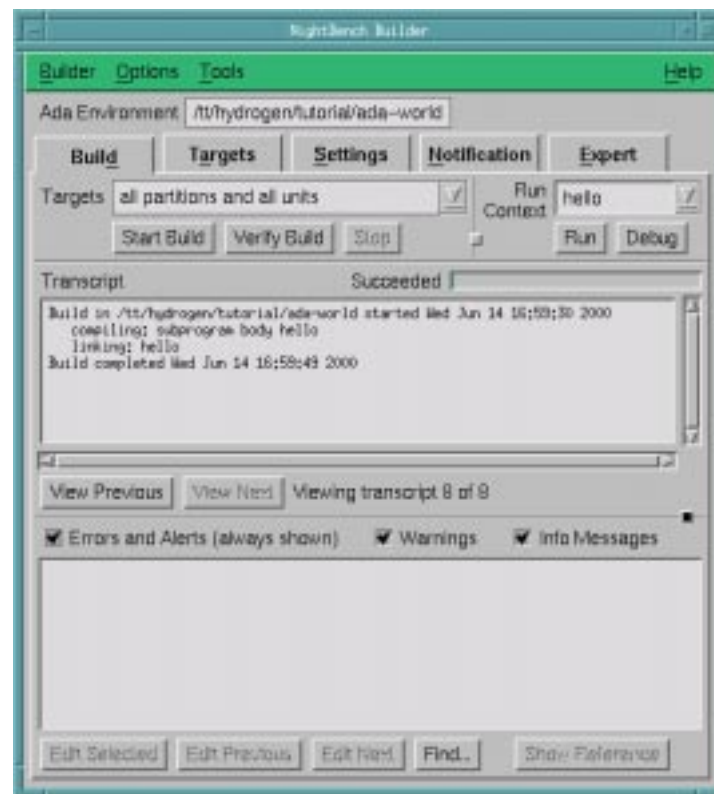


Figure 6-7. NightBench Builder - Build page

Targets

Specifies which partitions and/or units should be built.

Commonly used targets can be selected from the popup list; more detailed control is available on the Targets page.

See also:

- “Builder - Targets” on page 6-36

Start Build

Builds the selected targets through the pre-build, compilation, linking, and post-build phases.

Verify Build

Displays the activities (compilations and links) that would occur during the build, but does not actually perform them.

Stop

Stops the build during program generation. The **Stop** button is applicable only when an actual build is in progress. It has no effect during pre-build or post-build.

Run Context

The name of the *run context* to Run or Debug. Changes made to this field will also be reflected in the Run Context field on the Targets page.

See also:

- “Run Context” on page 6-19
- “Builder - Targets” on page 6-36
- *NightView User's Guide* (0890395)

Run

Executes the specified Run Context.

NOTE

The user may configure NightBench to open the Run Context dialog each time the Run button is pressed. For example, the user may want to change the arguments to the program each time it is run. Select the **Popup** this dialog prior to Run or Debug setting on the Preference page of the Run Context dialog for this behavior. (See “Run Context - Preference” on page 6-32 for more information).

See also:

- “Run Context” on page 6-19
- “Preferences - Terminal” on page 1-11

Debug

Executes the specified Run Context under the NightView debugger.

NOTE

The user may configure NightBench to open the Run Context dialog each time the Debug button is pressed. For example, the user may want to change the arguments to the program each time it is debugged. Select the Popup this dialog prior to Run or Debug setting on the Preference page of the Run Context dialog for this behavior. (See “Run Context - Preference” on page 6-32 for more information).

See also:

- “Run Context” on page 6-19
- *NightView User’s Guide* (0890395)

Build Progress

When a build is in progress, this area is labeled **Build Progress** and the status bar gives an approximation of the number of actions (compilations and links) left to perform in the current build.

When a build is completed, this field may be labeled one of the following, depending on the conditions of the current build’s completion:

- Succeeded

The build completed with no errors. Warnings and informational messages may have been issued, however.

- Failed

The build terminated with errors. Warnings and informational messages may have been issued, however.

- Stopped

The build was manually stopped by the user.

- Died Abnormally

The build was terminated by an unexpected event such as a system crash or a program error.

Transcript

The Transcript window logs all build messages, including pre-build and post-build messages. The last transcript that was viewed in this window is automatically loaded.

The build transcripts may be managed using the **Build Transcripts** dialog.

See also:

- “Build Transcripts” on page 6-6
- “Save Transcript” on page 6-9

View Previous

This button allows the user to view the previous build transcript of those that are saved.

See also:

- “Build Transcripts” on page 6-6
- “Save Transcript” on page 6-9

View Next

This button allows users to view the build transcript that is next in the sequence of saved transcripts.

See also:

- “Build Transcripts” on page 6-6
- “Save Transcript” on page 6-9

Viewing transcript ... of ...

This area specifies which transcript in the sequence of saved transcripts is currently displayed.

See also:

- “Build Transcripts” on page 6-6
- “Save Transcript” on page 6-9

Errors and Alerts (always shown)

Errors and alerts occurring during the build are always shown in the **Errors Window**.

See also:

- “Save Transcript” on page 6-9

Warnings

Filters warning messages both during the actual build and when viewing a completed transcript.

If checked, warning messages will be displayed. Actual generation of warning messages can be suppressed using either the Environment Compile Options as the default for all units in the environment or the Unit Compile Options for specific units.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26

Info Messages

Filters informational messages both during the actual build and when viewing a completed transcript.

If checked, informational messages will be displayed. Actual generation of informational messages can be suppressed using either the Environment Compile Options as the default for all units in the environment or the Unit Compile Options for specific units.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26

Errors Window

The Errors Window shows errors from the Builder only. Specific error messages from the compiler or linker are sent to the Transcript window.

See also:

- “Save Transcript” on page 6-9

Edit Selected

Opens the configured file editor to the location where the selected error in the Errors Window occurred.

See also:

- “Preferences - Editor” on page 1-9

Edit Previous

Allows the user to step backward through the errors in the Errors Window.

Opens the source file in the configured file editor to the location where the error previous to the currently selected error occurred.

See also:

- “Preferences - Editor” on page 1-9

Edit Next

Allows the user to step forward through the errors in the **Errors Window**.

Opens the source file in the configured file editor to the location where the error following the currently selected error occurred.

See also:

- “Preferences - Editor” on page 1-9

Find

Opens a dialog that allows the user to locate regular expressions, plain text, or file-names that occur in the currently viewed transcript or errors list.

See also:

- “Search Transcript” on page 6-34

Show Reference

(Ada only)

Opens the HyperHelp online *Ada 95 Reference Manual* or *MAXAda Reference Manual* to the location specified by the selected error, warning or informational message. If the selected error, warning, or informational message lists more than one reference, pressing the **Show Reference** button repeatedly cycles through the references.

Run Context

Run contexts are used when running or debugging programs, allowing the user to specify items such as program arguments, the directory in which to execute, the system (local or remote) on which to run, as well as other attributes for a particular program. The user also may use run contexts to specify debugger commands to be issued prior to debugging a program.

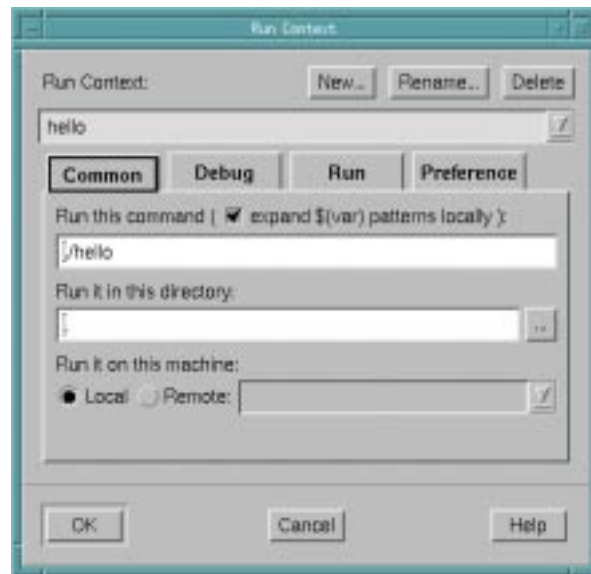


Figure 6-8. Run Context dialog

A run context is automatically created for each active partition that is created; by default, the run context is created with the same name as the partition. When you press the **Run** or **Debug** button on either the **Partitions** page of NightBench Development or the **Build** page of the NightBench Builder, NightBench uses the information in the run context to run or debug the program.

The **Run Context** dialog may be accessed from either the **Development** menu of NightBench Development or the **Builder** menu of the NightBench Builder. In addition the **Run Context...** button on the **Partitions** page allows the user to edit the run context currently associated with the partition selected on that page.

See also:

- “Development Menu” on page 5-3
- “Development - Partitions - (Ada)” on page 5-96
- “Development - Partitions - (C/C++)” on page 5-147
- “Builder Menu” on page 6-2
- “Builder - Build” on page 6-13

New...

Creates a new *run context*.

See also:

- “New Run Context” on page 6-22

Rename...

Renames the *current run context*.

See also:

- “Rename Run Context” on page 6-23

Delete

Deletes the *current run context*.

NOTE

If the run context currently associated with an active partition is deleted, NightBench will automatically assign to that partition the run context with the same name. If that run context does not exist, it will be automatically generated (using the current values in the *Environment Default Run Context*).

Run Context

The name of the *run context* currently being edited. You may edit a different run context by selecting one from the drop-down list.

In addition, you may edit the *Environment Default Run Context* by selecting **Environment Default Run Context** from this list. The values specified in the *Environment Default Run Context* are used when creating new run contexts (unless otherwise specified).

NOTE

Modifications to the *Environment Default Run Context* do not affect run contexts that have already been created. To propagate the information in the *Environment Default Run Context* into an existing run context, open the **Run Context** dialog (from either the **Development** menu of NightBench Development or the **Builder** menu of the NightBench Builder), select the desired run context from the drop-down list associated with the **Run Con-**

text field, and press the **Delete** button. If it is still associated with an active partition, a new run context will be automatically generated (using the current values in the *Environment Default Run Context*). See *run context association* on page Glossary-7 for more information.

The bottom section of the **Run Context** dialog contains a number of subpages that contain specific information about the current **Run Context**. These subpages are:

- **Common** - see “Run Context - Common” on page 6-24
- **Debug** - see “Run Context - Debug” on page 6-27
- **Run** - see “Run Context - Run” on page 6-29
- **Preference** - see “Run Context - Preference” on page 6-32

New Run Context

The New Run Context dialog allows the user to create a new *run context* using the values of the *current run context* or those values specified in the *Environment Default Run Context*.

See also:

- “Run Context” on page 6-19

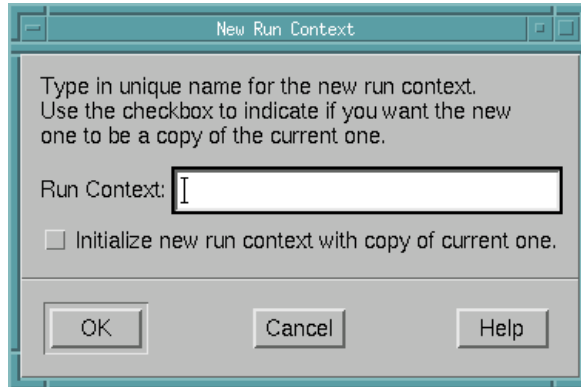


Figure 6-9. New Run Context dialog

Run Context

The name of the *run context* being created.

See also:

- “Run Context” on page 6-19

Initialize new run context with copy of current one

When this checkbox is checked, the *current run context* is used as a basis for the new *run context* being created. Otherwise, the values in the *Environment Default Run Context* will be used.

- “Development - Partitions - (Ada)” on page 5-96
- “Development - Partitions - (C/C++)” on page 5-147
- “Builder - Build” on page 6-13

Rename Run Context

The Rename Run Context dialog allows the user to rename the *current run context*.

See also:

- “Run Context” on page 6-19



Figure 6-10. Rename Run Context dialog

Run Context

The new name for the *current run context*.

See also:

- “Run Context” on page 6-19

Run Context - Common

The Common page contains information about the command to execute when this *run context* is run, whether or not to substitute environment variables passed to the command, the directory in which to run the command, and the machine on which to run the command (local or remote).

See also:

- “Run Context” on page 6-19

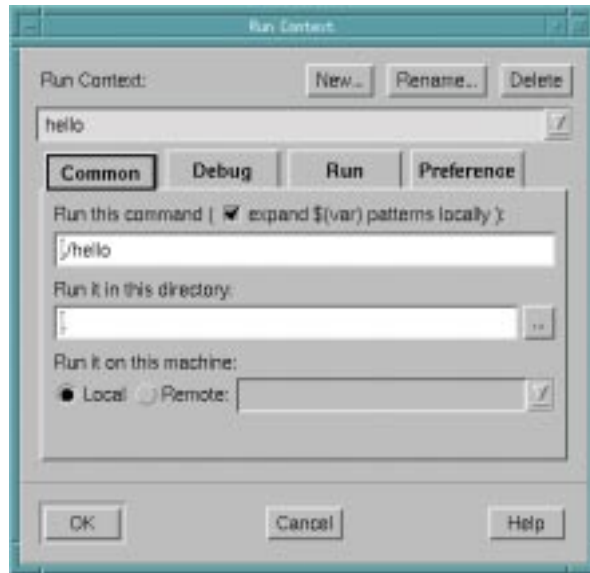


Figure 6-11. Run Context - Common page

Run this command

The command to execute when this *run context* is run (when the Run button on either the Partitions page of NightBench Development or the Build page of the NightBench Builder is pressed for this run context).

In addition, you may modify this field to specify arguments to the program. Environment variable substitution will follow the manner specified by the `expand $(var) patterns locally` checkbox (see “`expand $(var) patterns locally`” on page 6-25).

Typically, the command is the name of the active partition associated with this run context. However, the command could be a user-specified shell script as well. For example, if the user wants to log trace events from a particular program, `a.out`, using the NightTrace performance analysis tool, the run context currently associated with that program could specify a shell script which starts the NightTrace user dae-

mon, **ntraceud**, before the program executes and shuts down the NightTrace user daemon after the program exits. An example of such a shell script is shown below:

```
#!/bin/ksh
ntraceud trace_file
./a.out
ntraceud -q trace_file
```

See also:

- “Development - Partitions - (Ada)” on page 5-96
- “Development - Partitions - (C/C++)” on page 5-147
- “Builder - Build” on page 6-13

expand \$(var) patterns locally

When this checkbox is checked, arguments of the form:

\$(var)

are expanded. The value of the specified environment variable *var* is substituted in place of \$(var) before invoking the command on the target machine (see “Run it on this machine” on page 6-26). The evaluation of the environment variable is done on the local system. If the environment variable *var* does not exist, NightBench expands it to a null string.

This behavior may be overridden by preceding the argument with a \$. For example,

\$\$ (var)

will be passed to the program as \$(var).

NOTE

Arguments that have the form \$var or \${var} will not be expanded, regardless of this setting.

Run it in this directory

Specify the directory in which the program is to execute.

NOTE

It is important when running remotely that the remote machine knows the path to the program. An easy method of specifying the absolute pathname is to bring up the directory browser associated with this field by pressing the ... button and *unchecking* the Use relative directory name checkbox.



Use this to popup a dialog and browse for the directory you want. When finished, the directory is entered in the Run it in this directory field.

Run it on this machine

Local

When this radiobutton is selected, the program specified in the Run this command field will be executed on the same machine where the current NightBench session is running.

If the program needs a terminal window in which to run, the Run in terminal window checkbox on the Run page should be checked.

See also:

- “Run Context - Run” on page 6-29

Remote

When this radiobutton is selected, the program specified in the Run this command field will be executed on the machine specified here. The drop-down list associated with this field contains all remote machines previously specified.

The Run page of this dialog contains more information about running programs remotely.

See also:

- “Run Context - Run” on page 6-29

Run Context - Debug

The **Debug** page contains information related to the NightView debugger, including which commands to issue to NightView before the program associated with this *run context* is executed, as well as the name of the NightView dialogue with which to communicate.

See also:

- “Run Context” on page 6-19
- *NightView User’s Guide* (0890395)

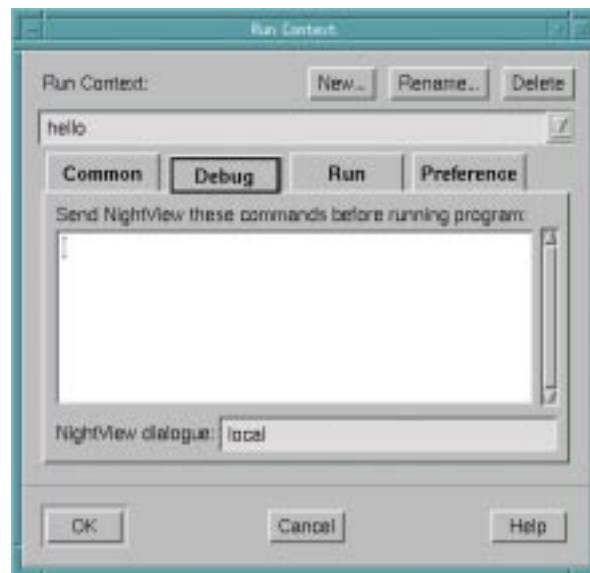


Figure 6-12. Run Context - Debug page

Send NightView these commands before running program

Before running the program specified in the **Run this command** field on the **Common** page using the NightView debugger, issue the NightView commands specified here.

This may be useful when running programs remotely. For instance, it may be necessary to issue the NightView command, **translate-object-file** (which can be abbreviated as **x1**), before the debug session.

For example, a local system, **fred**, may have a directory:

```
/usr
```

A remote system, **barney**, may mount this directory as:

```
/fredusr
```

From a NightBench session running on **fred**, the following command:

```
x1 /usr/ /fredusr/
```

should be added to the run context if the user wanted to remotely debug the program on **barney**. This command would translate paths that begin with **/usr/** on **fred** with **/fredusr/** on the remote system **barney** for any programs being debugged in the remote debug session.

See also:

- *NightView User's Guide* (0890395)

NightView dialogue

The name of the NightView dialogue with which to communicate. This dialogue is essentially just an ordinary shell which provides the opportunity to debug any or all of the programs executed in this dialogue shell.

At startup, NightView creates an initial dialogue named `local` by default. For programs being debugged locally, the dialogue name in this field will be `local`.

For programs being debugged remotely, the dialogue for the remote session initially will be set to the name of the remote system (the name of the machine specified in the **Remote** field on the **Common** page). The user may specify a different name if desired.

In addition, if a NightView remote dialogue is already running on the remote machine, the user may specify the name of that remote dialogue here.

See also:

- “Run Context - Common” on page 6-24
- *NightView User's Guide* (0890395)

Run Context - Run

The RUN page contains information related to running the *run context* locally or remotely.

See also:

- “Run Context” on page 6-19

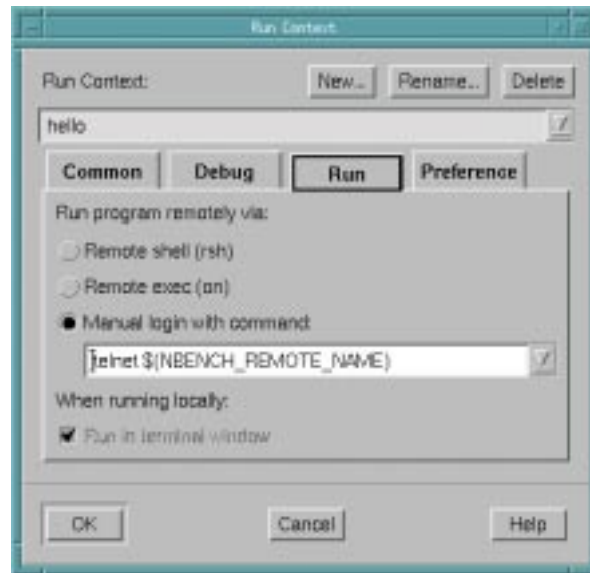


Figure 6-13. Run Context - Run page

Run program remotely via

Remote shell (rsh)

Use the **rsh(1)** command to execute the program associated with this *run context* on the machine specified in the **Remote** field on the **Common** page.

You may specify a command other than **rsh** to execute when this radiobutton is selected by selecting the **Use this program instead of “rsh”** option on the **Preference** page of the Run Context dialog.

See also:

- “Run Context - Common” on page 6-24
- “Run Context - Preference” on page 6-32

Remote exec (on)

Use the **on(1C)** command to execute the program associated with this *run context* on the machine specified in the **Remote** field on the **Common** page.

You may specify a command other than **on** to execute when this radiobutton is selected by selecting the **Use this program instead of "on"** option on the **Preference** page of the **Run Context** dialog.

See also:

- “Run Context - Common” on page 6-24
- “Run Context - Preference” on page 6-32

Manual login with command

The user may select this option in order to specify the command to use to log in to the remote machine before running the program associated with this *run context*.

The drop-down list associated with this radiobutton is pre-populated with the following commands:

```
telnet $(NBENCH_REMOTE_NAME)
rlogin $(NBENCH_REMOTE_NAME)
```

which use either **telnet(1)** or **rlogin(1)**, respectively, to log in to the remote machine.

NOTE

`$(NBENCH_REMOTE_NAME)` is the name of the machine specified in the **Remote** field on the **Common** page of the **Run Context** dialog.

The user may enter any command in the field provided in order to log in to the remote machine. The command will then be added to the drop-down list for future use.

When this run context is run, the command specified in this field is executed in a terminal window which then allows the user to interact with the window to complete the login. NightBench will also popup a dialog with a button to dismiss it when you have finished logging in to the remote system. At that time, NightBench will use that terminal window for running commands on that machine (for any run context that specifies that machine) until NightBench exits or that terminal window is closed.

NOTE

This could be particularly useful for programs that require special privileges to run (e.g. Ada programs). The user could set up the **.profile** or **.login** file on the remote machine to automatically issue the **tfadmin(1M)** command and **telnet** into that machine using this option before running the program. Alternatively, the user could use this option to remotely log in to a machine, manually execute any necessary commands (such as **tfadmin**), and then dismiss the dialog which informs NightBench that the login is complete.

See the section titled “Privileges” in Chapter 1 of the *MAXAda Reference Manual* (0890516) for more detailed information about using the **tfadmin** command to set privileges for Ada programs.

See also:

- “Run Context - Common” on page 6-24
- “Preferences - Terminal” on page 1-11

Run in terminal window

The user should check this if the program associated with this *run context* needs a terminal window in which to run. This only applies when the program is to be run on the local machine (i.e. when the **Local** option is selected on the **Common** page).

See also:

- “Run Context - Common” on page 6-24
- “Preferences - Terminal” on page 1-11

Run Context - Preference

The Preference page allows the user to change the default behavior of *run contexts* such as allowing alternate programs to be used to run programs remotely. There is also a setting which allows the Run Context dialog to be presented every time the Run or Debug button is pressed.

See also:

- “Run Context” on page 6-19

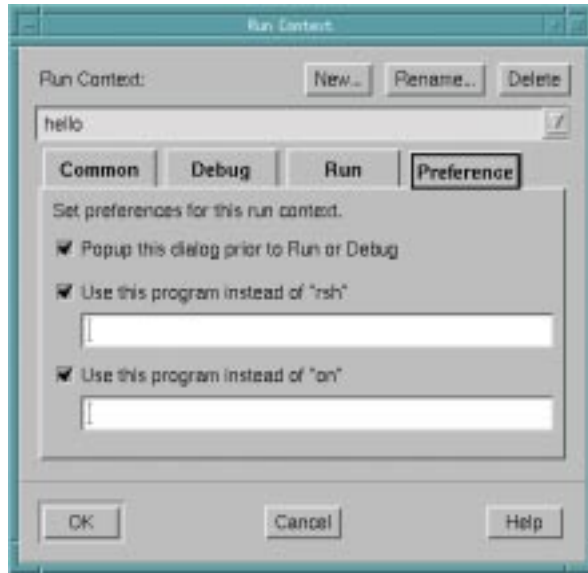


Figure 6-14. Run Context - Preference page

Popup this dialog prior to Run or Debug

Select this option to display the Run Context dialog every time this *run context* is run or debugged.

See also:

- “Run Context” on page 6-19

Use this program instead of “rsh”

When the Remote shell (rsh) radiobutton is selected on the Run page, use the command specified here instead of `rsh(1)` to execute the program associated with this *run context*.

See also:

- “Run Context - Common” on page 6-24
- “Run Context - Run” on page 6-29

Use this program instead of “on”

When the Remote `exec (on)` radiobutton is selected on the Run page, use the command specified here instead of `on (1C)` to execute the program associated with this *run context*.

See also:

- “Run Context - Common” on page 6-24
- “Run Context - Run” on page 6-29

Search Transcript

The **Search Transcript** dialog allows the user to locate regular expressions, plain text, or filenames that occur in the transcript currently being viewed in the Builder. In addition, the user may confine the search to either the transcript window or the error list.

See also:

- “Builder - Build” on page 6-13
- “Transcript” on page 6-15
- “Preferences - Transcripts” on page 1-12

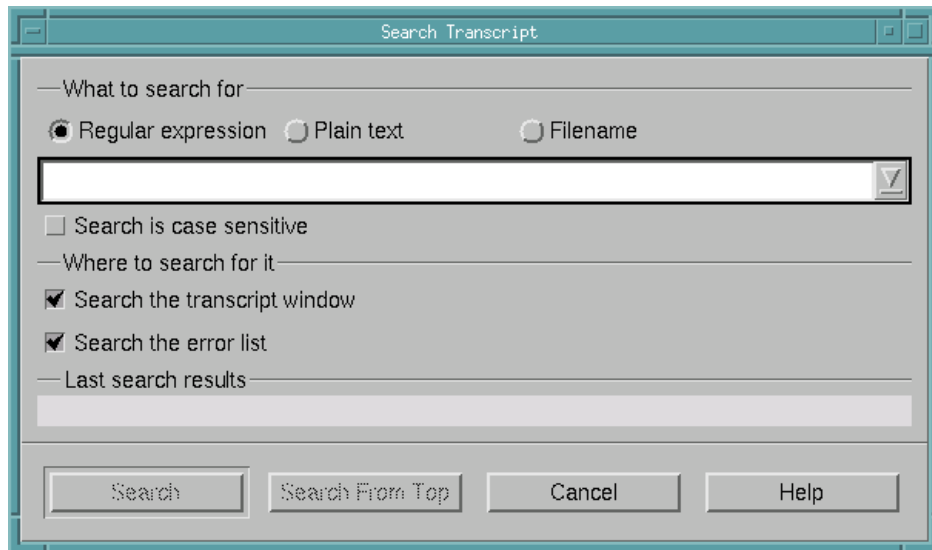


Figure 6-15. Search Transcript dialog

Regular expression

Perform regular expression matching on the text given in the text field on either the contents of the Transcript window, the Errors list or both. (See **Search the transcript window** and **Search the error list**.)

Plain text

Attempt to directly match the text given in the text field to either the contents of the Transcript window, the Errors list or both. (See **Search the transcript window** and **Search the error list**.)

Filename

Searches the error list only (automatically deactivates the **Search the transcript window** checkbox when selected) for errors that occur in the file specified in the text field. Wildcard expressions are allowed when specifying the **Filename**.

Text to search using the method selected (Regular expression, Plain text, or Filename) may be entered in the text field.

Search is case-sensitive

The text in the text field must also match in case when this checkbox is selected.

Search the transcript window

If this item is selected, the search will attempt to match the text specified in the text field with the contents of the Transcript window.

This item is automatically deactivated when searching for a Filename.

Search the error list

If this item is selected, the search will attempt to match the text specified in the text field with the contents of the Errors list.

Last search results

Gives the status of the last search.

Search

Attempts to search for the text given in the text field (as a Regular expression, Plain text, or a Filename, as indicated) with the contents of the Transcript window, the Errors list or both. (See Search the transcript window and Search the error list.)

The search begins at the top of either the Transcript window or the Errors list (depending on the settings of Search the transcript window and Search the error list). Subsequent searches pick up from the position of the last successful match.

Search from top

Attempts to search for the text given in the text field (as a Regular expression, Plain text, or a Filename, as indicated) with the contents of the Transcript window, the Errors list or both. (See Search the transcript window and Search the error list.)

The search begins at the top of either the Transcript window or the Errors list (depending on the settings of Search the transcript window and Search the error list).

Builder - Targets

The Targets page details which partitions and/or units should be built by the Builder. Also, the name of an executable program to run may be specified on this page.

See also:

- NightBench Builder - Chapter 6

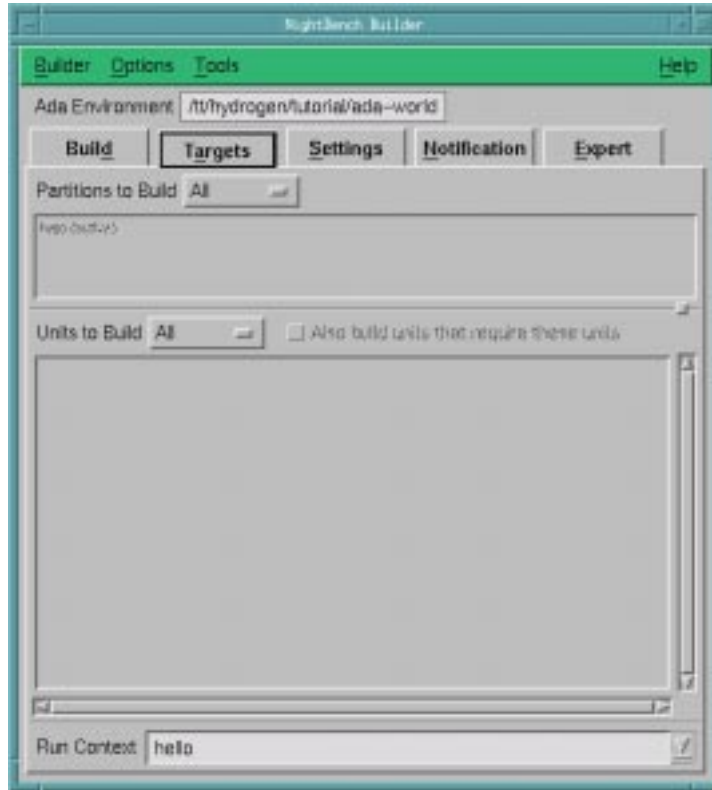


Figure 6-16. NightBench Builder - Targets page

Partitions to Build

Lists each of the partitions to be built the next time a build is started.

All	build all partitions within the current environment
None	do not build any partitions within the current environment
Selected	build only those partitions selected in the Partitions to Build list.

The default setting is All.

Units to Build

Lists each of the units to be built the next time a build is started.

All	build all units within the current environment
None	do not build any units within the current environment
Selected	build only those units selected from the Units to Build list

The default setting is All.

Also build units that require these units

In addition to building the selected units and all units on which they directly or indirectly depend, also build units which directly or indirectly depend upon the selected units.

Run Context

The name of the *run context* to be used by the Run and Debug buttons on the Build page. Changes made to this field will also be reflected in the Run Context field on the Build page.

See also:

- “Builder - Build” on page 6-13

Builder - Settings

See also:

- NightBench Builder - Chapter 6



Figure 6-17. NightBench Builder - Settings page

Interoptimization Level

NightBench provides a method of optimization that controls the compilation order such that all language-dependence rules are obeyed.

There are currently two levels of interoptimization available.

0 - minimal	no effort to attain interoptimization (default)
1 - inline aware	better ordering of compilation of units such that inlined subprogram calls will be performed whenever possible

Nice Increment

If nicing is desired, uncheck the **Do not nice** checkbox and enter the nice increment in this field. The **nice(1)** command is issued using this nice increment before the build is performed.

Do not nice

No nicing is performed. This is the default.

Parallel Compilations

Units can be compiled in parallel, thereby better utilizing the available CPUs. The user may enter the number of compilations to be performed in parallel in this field.

Use default (# CPUs)

The number of parallel compilations defaults to the number of CPUs on the system.

Parallel Dependency Analyses

Units can be analyzed for dependencies in parallel, thereby better utilizing the available CPUs. The user may enter the number of analyses to be performed in parallel in this field.

Use default (Par. Comp's * 2)

The number of parallel dependency analyses defaults to twice the number of CPUs on the system.

Stop build when an error is encountered

Normally, the Builder will try to build any units which are not dependent on erroneous units. This option halts the Builder when the first error is encountered.

The default for this checkbox is unchecked.

Automatically import out-of-date non-local units

Foreign units which are not consistently compiled but which are required to be for the build to succeed are *naturalized* and compiled. A naturalized unit is the compiled form of a foreign unit created by the compilation system in the local environment. See the Glossary for more details.

The default for this checkbox is checked.

Attempt compiles and links that will fail

If disabled, the Builder will skip compilation of units when it is known that the compilation will fail. An example of this is a source file that has had previous syntax

errors but which has not been modified since. The Builder knows this will fail again so it will not attempt the compilation of this unit its source file has been modified.

The default for this checkbox is unchecked.

Builder - Notification

The selected notification operations will be performed when a build completes. However, the operations will not be performed when the user manually cancels the build using the Stop button on the Build page.

See also:

- NightBench Builder - Chapter 6
- “Builder - Build” on page 6-13

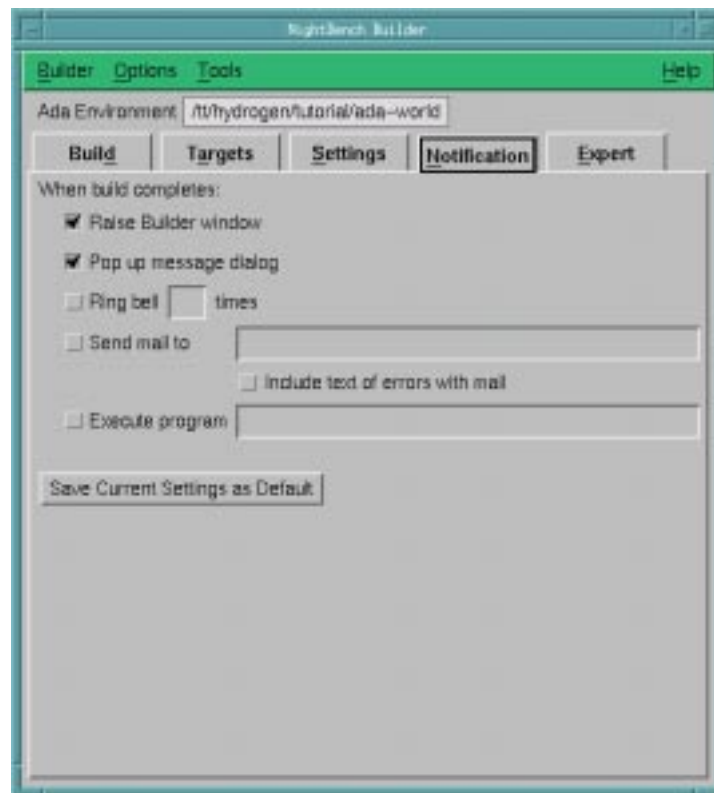


Figure 6-18. NightBench Builder - Notification page

Raise Builder window

The Builder window is brought to the foreground upon completion of the build when this option is selected.

Pop up message dialog

A dialog containing a “Build completed” message will pop up upon completion of the build when this option is selected.

Ring bell ___ times

An audible alert will sound the specified number of times when this option is selected.

Send mail to

If this option is selected, `/usr/bin/mail` is executed to send a “build completion” message to the specified recipient(s).

Include text of errors with mail

The text of the errors generated by the most recently completed build will be sent as part of the notification mail to the recipients specified in the **Send mail to** field.

Execute program

The program specified will run at the completion of the build.

The following environment variables will be set when the program specified is invoked:

`NBENCH_ERROR_FILE_NAME`

The name of the file holding the plain text copy of errors

`NBENCH_TRANSCRIPT_FILE_NAME`

The name of the file holding the plain text copy of the transcript

`NBENCH_LOG_FILE_NAME`

The name of the file holding the log messages

`NBENCH_ERROR_COUNT`

`NBENCH_WARNING_COUNT`

`NBENCH_INFO_COUNT`

`NBENCH_ALERT_COUNT`

The count of errors, warnings, info messages, and alerts

`NBENCH_BUILD_STATUS`

The final build status string

`NBENCH_ENVIRONMENT_PATH`

The path to the environment (should also be current directory when the program starts)

`NBENCH_ENVIRONMENT_LANGUAGE`

The language of the environment

`NBENCH_BUILD_TARGET`

The description of the targets

Save Current Settings as Default

The current settings may be made the default notification operations by using this button.

Builder - Expert

See also:

- NightBench Builder - Chapter 6

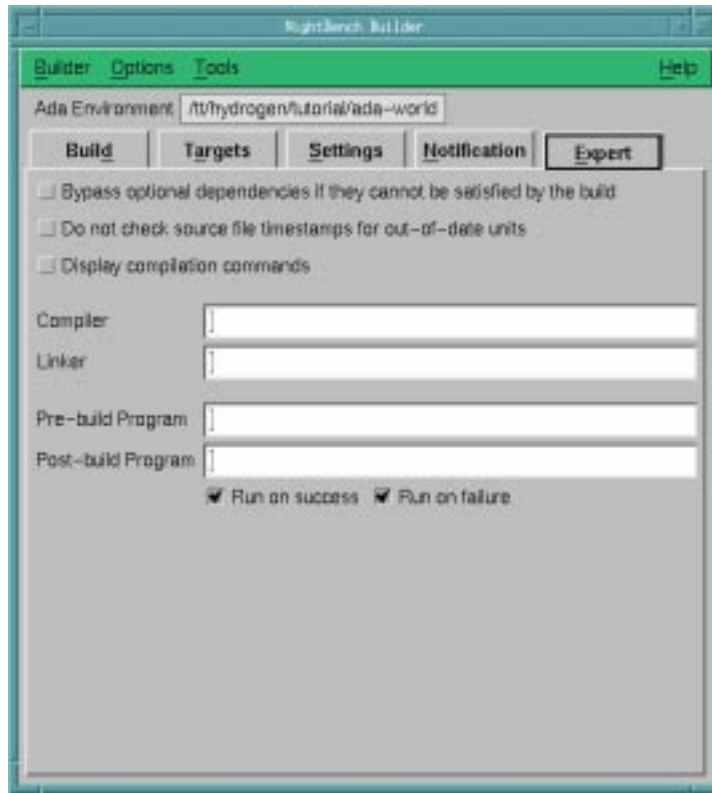


Figure 6-19. NightBench Builder - Expert page

Bypass optional dependencies if they cannot be satisfied by the build

Normally, in the presence of any interoptimization other than minimal, the Builder will refuse to compile units which require a body for inlining purposes if that body cannot be compiled (e.g. it has syntax errors, or is both foreign and inconsistent and it has been specified that such units are not to be naturalized). If this option is enabled, then these units will be compiled but inline optimizations will not occur.

See also:

- “Builder - Settings” on page 6-38

Do not check source file timestamps for out-of-date units

The Builder normally checks the timestamp of every source file for out-of-date units. However, NightBench performs an analysis of every source file after it has been edited so all dependency information is kept up-to-date. If the source files

have only been edited using NightBench, there is no need for the Builder to check these timestamps since these analyses have been automatically performed.

Display compilation commands

Displays the actual commands used for compilation and linking in the Transcript window of the Builder.

See also:

- “Builder - Build” on page 6-13
- *MAXAda Reference Manual* (0890516)
- *C/C++ Reference Manual* (0890497)

Compiler

The pathname of the compiler to be used by the Builder (if different from the standard NightBench compiler).

Linker

The pathname of the linker to be used by the Builder (if different from the standard NightBench linker).

Pre-build Program

The pathname of a program to be executed before the build is attempted.

Post-build Program

The pathname of a program to be executed after the build completes. The execution of this program is contingent upon the **Run on success** and **Run on failure** options on this page.

If a post-build program is specified and is executed, it is executed immediately prior to any of the notification operations.

Run on success

Execute the Post-build Program only if there are no errors in the Errors Window.

See also:

- “Errors Window” on page 6-17

Run on failure

Execute the Post-build Program only if there are errors in the Errors Window.

See also:

- “Errors Window” on page 6-17

Options Editors

Unlike most compilation systems, NightBench uses the concept of *persistent options*. These options do not need to be specified for each compilation. Rather, they are stored as part of the environment or as part of an individual unit's information. These options are "remembered" when the NightBench Builder tool is used.

There are three "types" of compilation options:

- Environment-wide compile options
- Permanent unit compile options
- Temporary unit compile options

These three options sets have a hierarchical relationship that is represented by the *effective options* (see "Effective Options" below).

The environment-wide options are manipulated using the Environment Compile Options Editor. See "Environment Compile Options Editor" on page 7-3 for more details.

Both the permanent and temporary options can be changed using the Unit Compile Options Editor. See "Unit Compile Options Editor" on page 7-24 for more details.

Effective Options

The *environment-wide compile options* set, *permanent unit compile options* set and *temporary unit compile options* set have a hierarchical relationship to one another. Environment-wide options can be overridden by permanent unit options which can be overridden by temporary unit options. The set of *effective options* for a unit are that unit's sum total of these three option sets, with respect to this hierarchical relationship.

See "Determining the effective options" on page 2-14 for an example of this relationship in an Ada environment.

Tri-state Checkboxes

NightBench uses tri-state checkboxes on some pages of the Environment Compile Options Editor and Unit Compile Options Editor.

For those unfamiliar with tri-state checkboxes, each of the states is shown below with a description of its meaning:



When a checkbox is in this state, the option associated with this checkbox will NOT be applied to the unit(s) with which it is associated. For instance, if an option was specified as such for an option in the permanent unit options set, it would take precedence over the same option in the environment-wide option set and force that particular option NOT to be applied.



When a checkbox is in this state, the option associated with this checkbox WILL be applied to the unit(s) with which it is associated. For instance, if an option was specified as such for an option in the permanent unit options set, it would take precedence over the same option in the environment-wide option set and force that particular option to be applied.



When a checkbox is in this state, it is considered *unspecified*, meaning that it is neither set nor unset. No action will be taken for the option associated with this checkbox. Furthermore, a checkbox in this state will not affect the *effective options* in any way since its value is considered unspecified. For instance, if an option was unspecified as such for an option in the permanent unit options set, the effective setting of the option would be determined by any setting in the environment-wide options or by a default value if unspecified in the environment-wide option set.

See also:

- “Environment Compile Options Editor” on page 7-3
- “Unit Compile Options Editor” on page 7-24

Environment Compile Options Editor

There are two Environment Compile Options Editors implemented by NightBench for setting *environment-wide compile options*, depending on the language used for development in the current environment:

- “Ada Environment Compile Options Editor” on page 7-4
- “C/C++ Environment Compile Options Editor” on page 7-9

In addition, these environment-wide compile options may be overridden by setting compile options for a specific unit.

See also:

- “Unit Compile Options Editor” on page 7-24

Button Functionality

The following buttons appear at the bottom of the Environment Compile Options Editor and have the specified meaning:

OK

This button applies changes made to any of the options and closes the Environment Compile Options Editor.

Apply

This button applies changes made to any of the options but leaves the Environment Compile Options Editor open.

Reset

This button restores any options to the values last applied to those options and leaves the Environment Compile Options Editor open.

Clear

This button resets all of the options in the Environment Compile Options Editor to their default values.

Cancel

This button restores any options to the values last applied to those options and closes the Environment Compile Options Editor.

Help

This button brings up the help topic for this page.

In addition, “Tri-state Checkboxes” on page 7-1 discusses the various states in which the checkboxes in this dialog may exist.

See also:

- “Environment Compile Options Editor” on page 7-3

Ada Environment Compile Options Editor

The Ada Environment Compile Options Editor contains the following pages:

- General (see page 7-4)
- Inlining (see page 7-6)
- Optimization (see page 7-7)
- Expert (see page 7-8)

Ada Environment Compile Options Editor - General

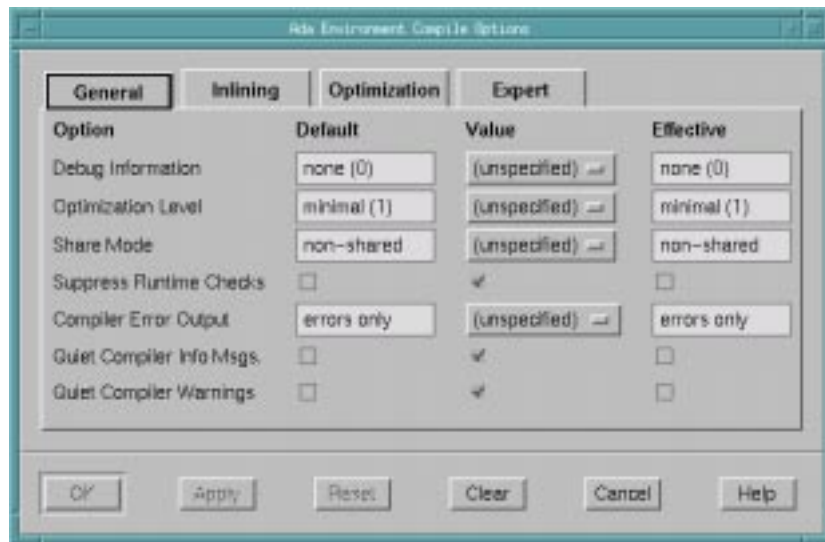


Figure 7-1. Ada Environment Compile Options dialog - General Page

Debug Information

See “Debug Information” on page 7-48.

Optimization Level

See “Optimization Level” on page 7-51.

Share Mode

See “Share Mode” on page 7-53.

Suppress Runtime Checks

See “Suppress Runtime Checks” on page 7-54.

Compiler Error Output

See “Compiler Error Output” on page 7-47.

Quiet Compiler Info Msgs.

See “Quiet Compiler Info Msgs.” on page 7-52.

Quiet Compiler Warnings

See “Quiet Compiler Warnings” on page 7-53.

Ada Environment Compile Options Editor - Inlining



Figure 7-2. Ada Environment Compile Options dialog - Inlining Page

Inline Line Count

See "Inline Line Count" on page 7-49.

Inline Nesting Depth

See "Inline Nesting Depth" on page 7-49.

Inlines Per Compilation

See "Inlines Per Compilation" on page 7-50.

Inline Statement Limit

See "Inline Statement Limit" on page 7-49.

Ada Environment Compile Options Editor - Optimization



Figure 7-3. Ada Environment Compile Options dialog - Optimization Page

Optimization Class

See “Optimization Class” on page 7-51.

Optimize for Space

See “Optimize for Space” on page 7-52.

Optimization Size Limit

See “Optimization Size Limit” on page 7-52.

Variables to Optimize

See “Variables to Optimize” on page 7-55.

Loops to Optimize

See “Loops to Optimize” on page 7-50.

Unroll Limit

See “Unroll Limit” on page 7-54.

Growth Limit

See “Growth Limit” on page 7-49.

Ada Environment Compile Options Editor - Expert

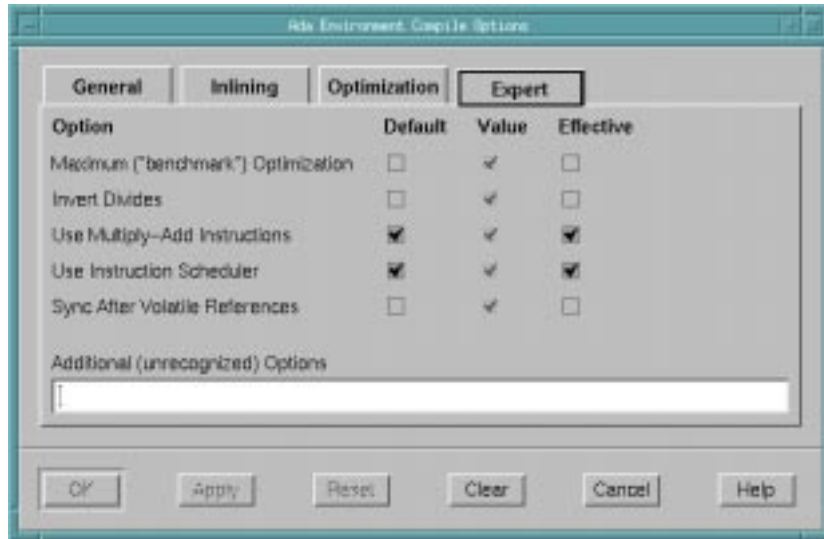


Figure 7-4. Ada Environment Compile Options dialog - Expert Page

Maximum ("benchmark") Optimization

See "Maximum ("benchmark") Optimization" on page 7-51.

Invert Divides

See "Invert Divides" on page 7-50.

Use Multiply-Add Instructions

See "Use Multiply-Add Instructions" on page 7-55.

Use Instruction Scheduler

See "Use Instruction Scheduler" on page 7-54.

Sync After Volatile References

See "Sync After Volatile References" on page 7-54.

Additional (unrecognized) Options

Any other options not explicitly referenced by the Ada Environment Compile Options Editor may be specified here.

C/C++ Environment Compile Options Editor

The C/C++ Environment Compile Options Editor contains the following pages:

- General (see page 7-9)
- Macros (see page 7-11)
- Includes (see page 7-13)
- Optimizer (see page 7-16)
- Language (see page 7-18)
- Dialects (see page 7-20)
- Expert (see page 7-22)

C/C++ Environment Compile Options Editor - General

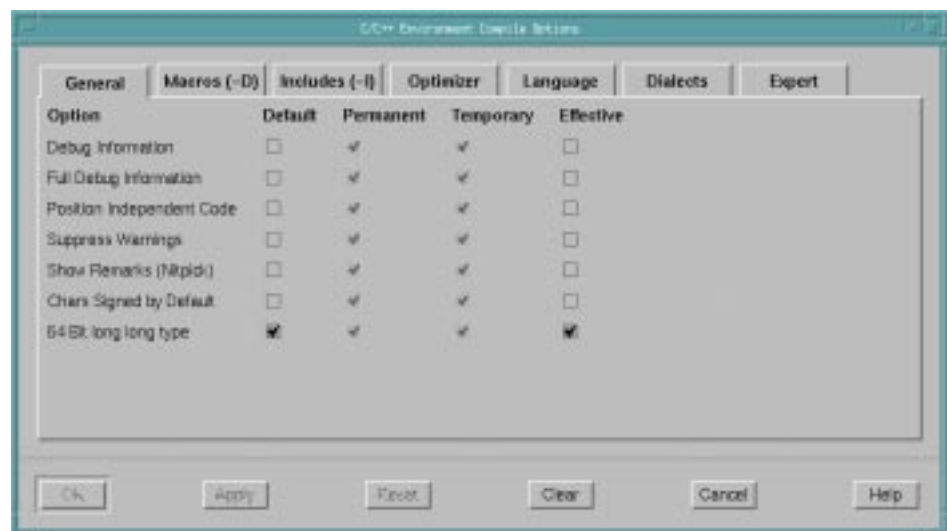


Figure 7-5. C/C++ Environment Compile Options dialog - General Page

Debug Information

See “Debug Information” on page 7-58.

Full Debug Information

See “Full Debug Information” on page 7-60.

Position Independent Code

See “Position Independent Code” on page 7-63.

Suppress Warnings

See “Suppress Warnings” on page 7-64.

Show Remarks (Nitpick)

See “Show Remarks (Nitpick)” on page 7-63.

Chars Signed by Default

See “Chars Signed by Default” on page 7-58.

64 Bit long long type

See “64 Bit long long type” on page 7-55.

C/C++ Environment Compile Options Editor - Macros

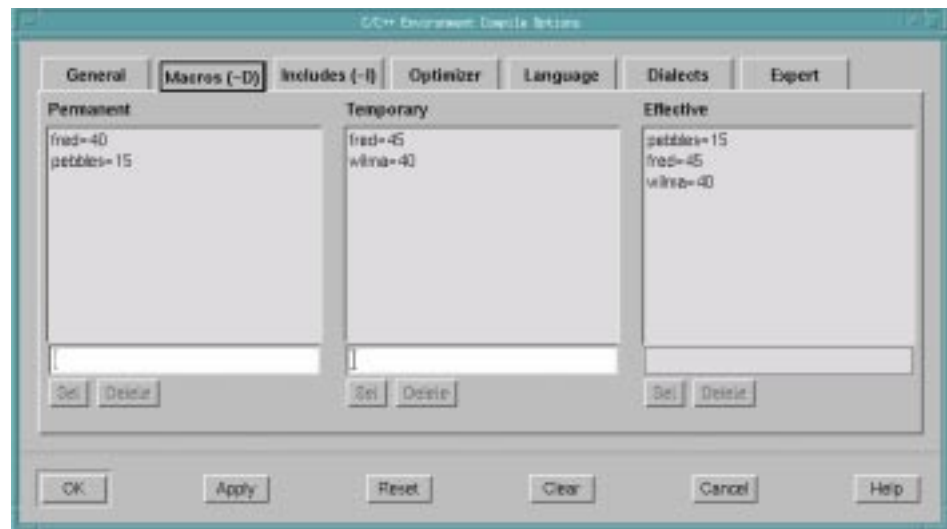


Figure 7-6. C/C++ Environment Compile Options dialog - Macros Page

Permanent

A macro in the list of **Permanent** Environment Compile Options will apply to all units in the environment unless overridden by the same macro defined as either a **Temporary** Environment Compile Option or as a **Permanent** or **Temporary** Unit Compile Option.

See also:

- “C/C++ Unit Compile Options Editor - Macros” on page 7-33

Any macro may be directly entered in the text field below the list of **Permanent** Environment Compile Options and subsequently **Set** for inclusion in that list. In addition, a macro may be selected from the list of **Permanent** Environment Compile Options, edited in this field, and **Set** when editing is completed.

Set

Adds the macro defined in the text field below the list of **Permanent** Environment Compile Options to that list.

Delete

Removes the macro selected in the list of **Permanent** Environment Compile Options from that list.

Temporary

A macro in the list of **Temporary Environment Compile Options** overrides the same macro defined as a **Permanent Environment Compile Option** and applies to all units in the environment. **Temporary Environment Compile Options** may be overridden by the same macro defined as a **Permanent** or **Temporary Unit Compile Option**.

See also:

- “C/C++ Unit Compile Options Editor - Macros” on page 7-33

Any macro may be directly entered in the text field below the list of **Temporary Environment Compile Options** and subsequently **Set** for inclusion in that list. In addition, a macro may be selected from the list of **Temporary Environment Compile Options**, edited in this field, and **Set** when editing is completed.

Set

Adds the macro defined in the text field below the list of **Temporary Environment Compile Options** to that list.

Delete

Removes the macro selected in the list of **Temporary Environment Compile Options** from that list.

Effective

Macros in the list of **Permanent Environment Compile Options** may be overridden by those macros redefined in the list of **Temporary Environment Compile Options**. The **Effective** set of macros is the sum total of the **Permanent** and **Temporary** environment-wide macro definitions, with respect to this hierarchical relationship.

NOTE

The buttons under the list of **Effective** options have no effect since the **Effective** options are a resultant set of the **Permanent** and **Temporary** options.

C/C++ Environment Compile Options Editor - Includes

Allows the user to add directories to the include search path.

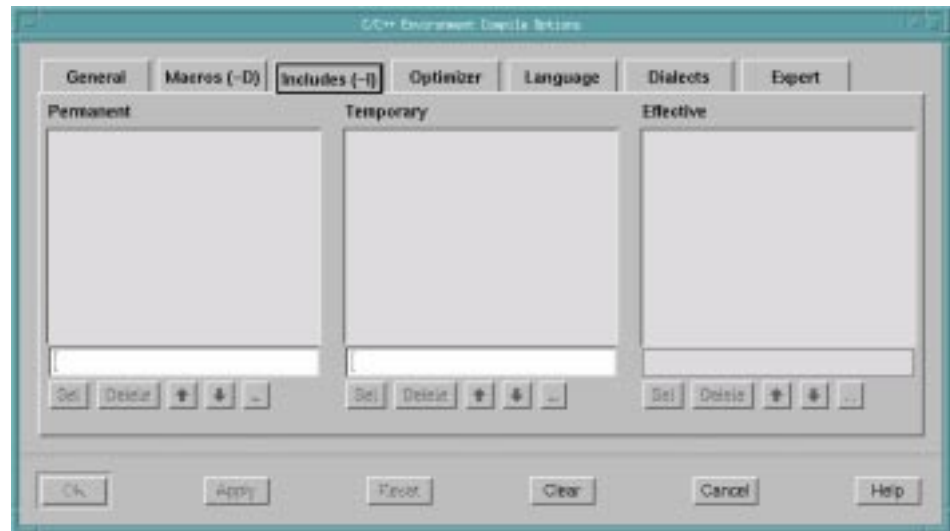


Figure 7-7. C/C++ Environment Compile Options dialog - Includes Page

Permanent

A path in the list of **Permanent** Environment Compile Options will apply to all units in the environment unless the same path is preceded by a ! in either the list of **Temporary** Environment Compile Options or in the lists of the **Permanent** or **Temporary** Unit Compile Options.

Paths are searched in the order in which they appear in the **Permanent** Environment Compile Options list.

See also:

- “C/C++ Unit Compile Options Editor - Includes” on page 7-36

Any path may be directly entered in the text field below the list of **Permanent** Environment Compile Options and subsequently **Set** for inclusion in that list. In addition, a path may be selected from the list of **Permanent** Environment Compile Options, edited in this field, and **Set** when editing is completed.

Set

Adds the path specified in the text field below the list of **Permanent** Environment Compile Options to that list.

Delete

Removes the path selected in the list of **Permanent Environment Compile Options** from that list.



Changes the position of the selected path in the list of **Permanent Environment Compile Options** to an earlier (more preferred) position. Paths are searched in the order in which they appear in the **Permanent Environment Compile Options** list.



Changes the position of the selected path in the list of **Permanent Environment Compile Options** to a later (less preferred) position. Paths are searched in the order in which they appear in the **Permanent Environment Compile Options** list.



Use this to popup a dialog and browse for the path you wish to add. When finished, the pathname appears in the text field.

Temporary

A path in the list of **Temporary Environment Compile Options** will apply to all units in the environment unless the same path is preceded by a ! in either the **Permanent** or **Temporary Unit Compile Options** lists.

Paths are searched in the order in which they appear in the **Temporary Environment Compile Options** list.

See also:

- “C/C++ Unit Compile Options Editor - Macros” on page 7-33

Any path may be directly entered in the text field below the list of **Temporary Environment Compile Options** and subsequently **Set** for inclusion in that list. In addition, a macro may be selected from the list of **Temporary Environment Compile Options**, edited in this field, and **Set** when editing is completed.

Set

Adds the path specified in the text field below the list of **Temporary Environment Compile Options** to that list.

Delete

Removes the path selected in the list of **Temporary Environment Compile Options** from that list.



Changes the position of the selected path in the list of **Temporary Environment Compile Options** to an earlier (more preferred) position. Paths are searched in the order in which they appear in the **Temporary Environment Compile Options** list.



Changes the position of the selected path in the list of **Temporary Environment Compile Options** to a later (less preferred) position. Paths are searched in the order in which they appear in the **Temporary Environment Compile Options** list.



Use this to popup a dialog and browse for the path you wish to add. When finished, the pathname appears in the text field.

Effective

Paths in the list of **Permanent Environment Compile Options** may be overridden by those same paths preceded by a ! in the list of **Temporary Environment Compile Options**. The **Effective** set of paths is the sum total of the paths in the **Permanent** and **Temporary** lists.

NOTE

The buttons under the list of **Effective** options have no effect since the **Effective** options are a resultant set of the **Permanent** and **Temporary** options.

C/C++ Environment Compile Options Editor - Optimizer



Figure 7-8. C/C++ Environment Compile Options dialog - Optimizer Page

Optimization Level

See “Optimization Level” on page 7-62.

Optimization Class

See “Optimization Class” on page 7-62.

Optimize for Space

See “Optimize for Space” on page 7-63.

Total Alias Object Limit

See “Total Alias Object Limit” on page 7-64.

Variables to Optimize

See “Variables to Optimize” on page 7-66.

Loops to Optimize

See “Loops to Optimize” on page 7-61.

Constant Loop Unroll Limit

See “Constant Loop Unroll Limit” on page 7-58.

Variable Loop Unroll Limit

See “Variable Loop Unroll Limit” on page 7-66.

Growth Limit

See “Growth Limit” on page 7-61.

C/C++ Environment Compile Options Editor - Language

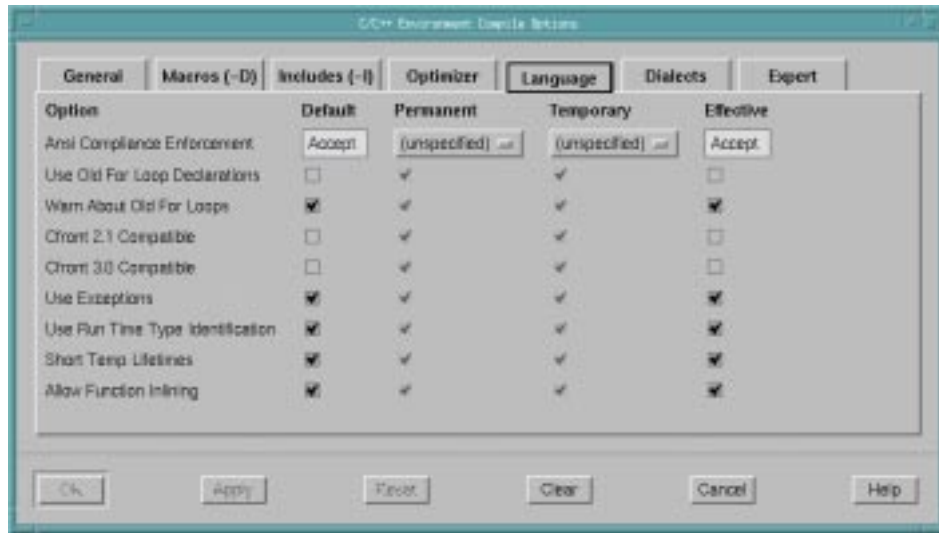


Figure 7-9. C/C++ Environment Compile Options dialog - Language Page

Ansi Compliance Enforcement

See “Ansi Compliance Enforcement” on page 7-57.

Use Old For Loop Declarations

See “Use Old For Loop Declarations” on page 7-65.

Warn About Old For Loops

See “Warn About Old For Loops” on page 7-66.

Cfront 2.1 Compatible

See “Cfront 2.1 Compatible” on page 7-57.

Cfront 3.0 Compatible

See “Cfront 3.0 Compatible” on page 7-57.

Use Exceptions

See “Use Exceptions” on page 7-64.

Use Run Time Type Identification

See “Use Run Time Type Identification” on page 7-65.

Short Temp Lifetimes

See “Short Temp Lifetimes” on page 7-63.

Allow Function Inlining

See “Allow Function Inlining” on page 7-57.

C/C++ Environment Compile Options Editor - Dialects

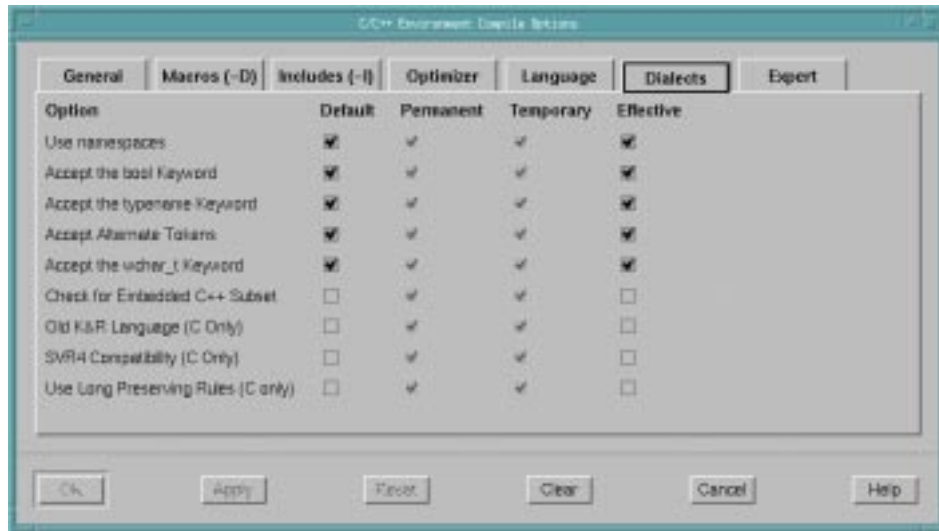


Figure 7-10. C/C++ Environment Compile Options dialog - Dialects Page

Use namespaces

See “Use namespaces” on page 7-65.

Accept the bool Keyword

See “Accept the bool Keyword” on page 7-56.

Accept the typename Keyword

See “Accept the typename Keyword” on page 7-56.

Accept Alternate Tokens

See “Accept Alternate Tokens” on page 7-56.

Accept the wchar_t Keyword

See “Accept the wchar_t Keyword” on page 7-56.

Check for Embedded C++ Subset

See “Check for Embedded C++ Subset” on page 7-58.

Old K&R Language (C Only)

See “Old K&R Language (C Only)” on page 7-61.

SVR4 Compatibility (C Only)

See “SVR4 Compatibility (C Only)” on page 7-64.

Use Long Preserving Rules (C Only)

See “Use Long Preserving Rules (C Only)” on page 7-65.

C/C++ Environment Compile Options Editor - Expert

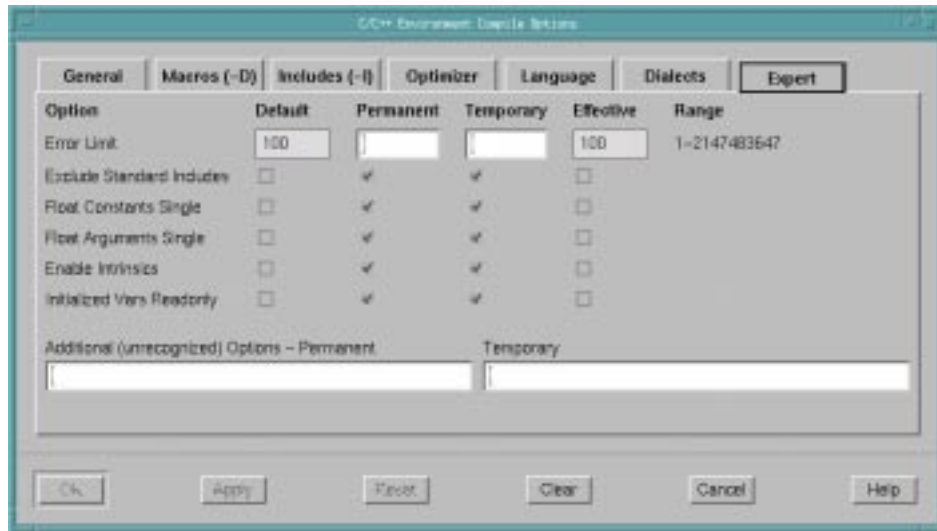


Figure 7-11. C/C++ Environment Compile Options dialog - Expert Page

Error Limit

See “Error Limit” on page 7-59.

Exclude Standard Includes

See “Exclude Standard Includes” on page 7-59.

Float Constants Single

See “Float Constants Single” on page 7-60.

Float Arguments Single

See “Float Arguments Single” on page 7-59.

Enable Intrinsic

See “Enable Intrinsic” on page 7-59.

Initialized Vars Readonly

See “Initialized Vars Readonly” on page 7-61.

Additional (unrecognized) Options

Permanent

Any other permanent options not explicitly referenced by the Environment Compile Options Editor may be specified here.

Temporary

Any other temporary options not explicitly referenced by the Environment Compile Options Editor may be specified here.

Unit Compile Options Editor

There are two Unit Compile Options Editors implemented by NightBench for setting *permanent unit compile options* and *temporary unit compile options*, depending on the language used for development in the current environment:

- “Ada Unit Compile Options Editor” on page 7-24
- “C/C++ Unit Compile Options Editor” on page 7-31

In addition, *environment-wide compile options* may be set for all units in the current environment.

See also:

- “Environment Compile Options Editor” on page 7-3

Ada Unit Compile Options Editor

The Ada Unit Compile Options Editor contains the following pages:

- General (see page 7-26)
- Inlining (see page 7-27)
- Optimization (see page 7-28)
- Expert (see page 7-29)

Button Functionality

The following buttons appear at the bottom of the Unit Compile Options Editor and have the specified meaning:

OK

Applies changes made to any of the options and closes the Unit Compile Options Editor.

Apply

Applies changes made to any of the options but leaves the Unit Compile Options Editor open.

Reset

Restores any options to the values last applied to those options and leaves the Unit Compile Options Editor open.

Cancel

Restores any options to the values last applied to those options and closes the Unit Compile Options Editor.

Help

This button brings up the help topic for this page.

Clear Permanent

Clears all of the options from the permanent unit compile options set.

Clear Temporary

Clears all of the options from the temporary unit compile options set.

Copy Temporary to Permanent

Propagates the set of temporary unit compile options into the permanent unit compile options and clears the temporary set.

Environment Options...

Opens the Environment Compile Options Editor.

See also:

- “Environment Compile Options Editor” on page 7-3

In addition, “Tri-state Checkboxes” on page 7-1 discusses the various states in which the checkboxes in this dialog may exist.

See also:

- “Unit Compile Options Editor” on page 7-24

Ada Unit Compile Options Editor - General

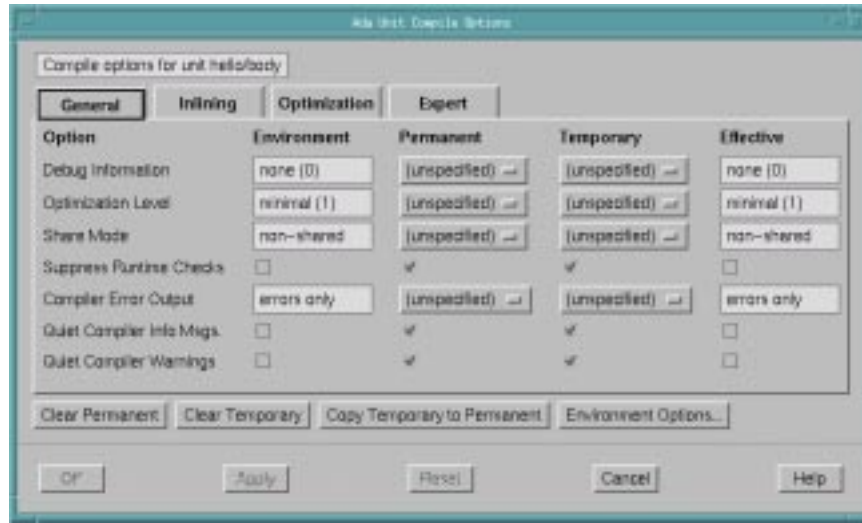


Figure 7-12. Ada Unit Compile Options dialog - General Page

Debug Information

See “Debug Information” on page 7-48.

Optimization Level

See “Optimization Level” on page 7-51.

Share Mode

See “Share Mode” on page 7-53.

Suppress Runtime Checks

See “Suppress Runtime Checks” on page 7-54.

Compiler Error Output

See “Compiler Error Output” on page 7-47.

Quiet Compiler Info Msgs.

See “Quiet Compiler Info Msgs.” on page 7-52.

Quiet Compiler Warnings

See “Quiet Compiler Warnings” on page 7-53.

Ada Unit Compile Options Editor - Inlining

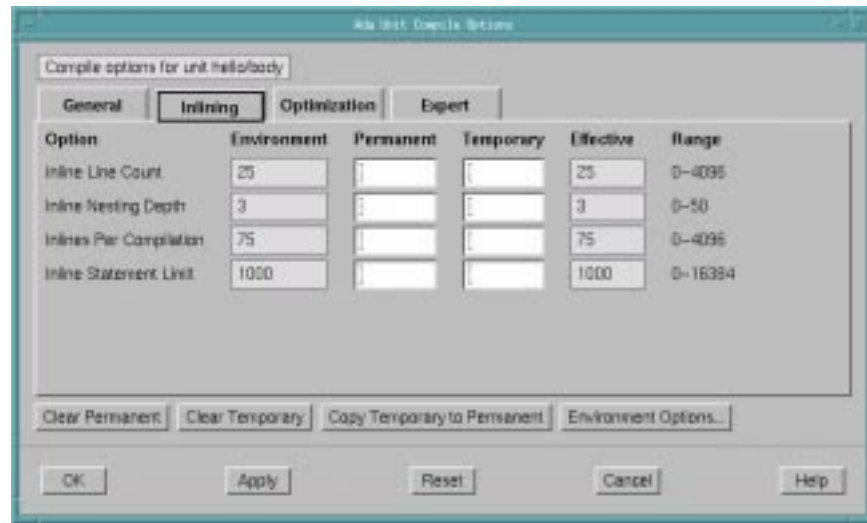


Figure 7-13. Ada Unit Compile Options dialog - Inlining Page

Inline Line Count

See “Inline Line Count” on page 7-49.

Inline Nesting Depth

See “Inline Nesting Depth” on page 7-49.

Inlines Per Compilation

See “Inlines Per Compilation” on page 7-50.

Inline Statement Limit

See “Inline Statement Limit” on page 7-49.

Ada Unit Compile Options Editor - Optimization

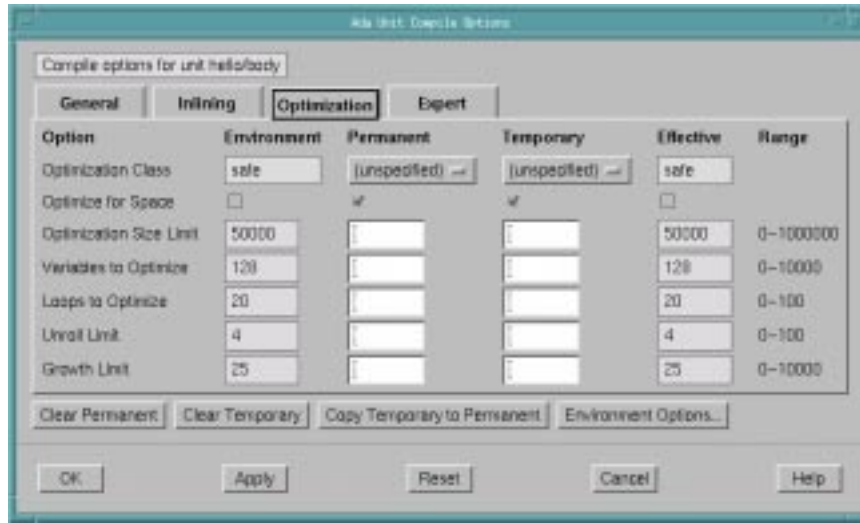


Figure 7-14. Ada Unit Compile Options dialog - Optimization Page

Optimization Class

See “Optimization Class” on page 7-51.

Optimize for Space

See “Optimize for Space” on page 7-52.

Optimization Size Limit

See “Optimization Size Limit” on page 7-52.

Variables to Optimize

See “Variables to Optimize” on page 7-55.

Loops to Optimize

See “Loops to Optimize” on page 7-50.

Unroll Limit

See “Unroll Limit” on page 7-54.

Growth Limit

See “Growth Limit” on page 7-49.

Ada Unit Compile Options Editor - Expert

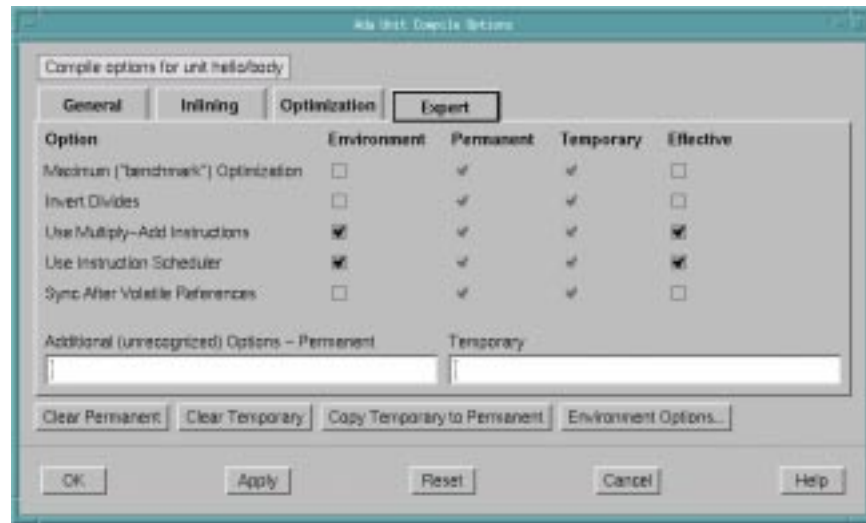


Figure 7-15. Ada Unit Compile Options dialog - Expert Page

Maximum ("benchmark") Optimization

See "Maximum ("benchmark") Optimization" on page 7-51.

Invert Divides

See "Invert Divides" on page 7-50.

Use Multiply-Add Instructions

See "Use Multiply-Add Instructions" on page 7-55.

Use Instruction Scheduler

See "Use Instruction Scheduler" on page 7-54.

Sync After Volatile References

See "Sync After Volatile References" on page 7-54.

Additional (unrecognized) Options

Permanent

Any other permanent options not explicitly referenced by the Unit Compile Options Editor may be specified here.

Temporary

Any other temporary options not explicitly referenced by the Unit Compile Options Editor may be specified here.

C/C++ Unit Compile Options Editor

The C/C++ Unit Compile Options Editor contains the following pages:

- General (see page 7-31)
- Macros (see page 7-33)
- Includes (see page 7-36)
- Optimizer (see page 7-40)
- Language (see page 7-42)
- Dialects (see page 7-44)
- Expert (see page 7-46)

C/C++ Unit Compile Options Editor - General

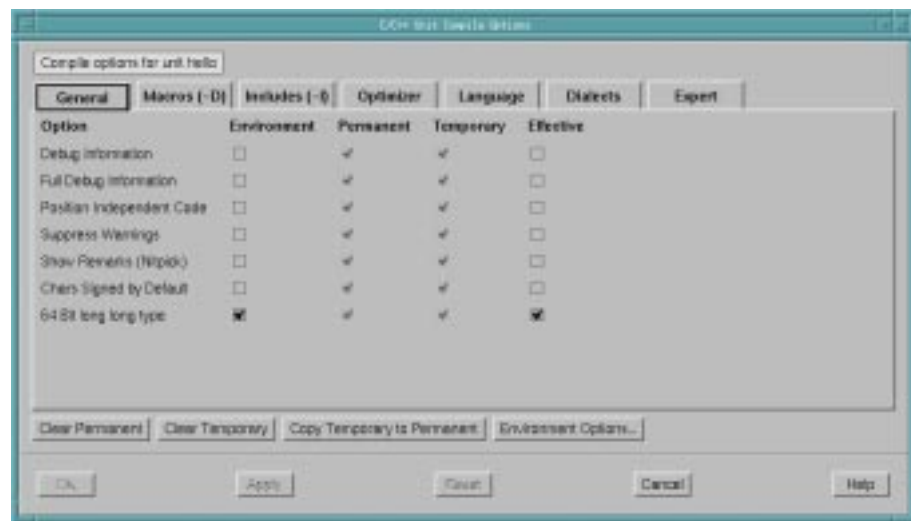


Figure 7-16. C/C++ Unit Compile Options dialog - General Page

Debug Information

See “Debug Information” on page 7-58.

Full Debug Information

See “Full Debug Information” on page 7-60.

Position Independent Code

See “Position Independent Code” on page 7-63.

Suppress Warnings

See “Suppress Warnings” on page 7-64.

Show Remarks (Nitpick)

See “Show Remarks (Nitpick)” on page 7-63.

Chars Signed by Default

See “Chars Signed by Default” on page 7-58.

64 Bit long long type

See “64 Bit long long type” on page 7-55.

C/C++ Unit Compile Options Editor - Macros



Figure 7-17. C/C++ Unit Compile Options dialog - Macros Page

Environment

Lists the Effective Environment Compile Options.

NOTE

The buttons under the list of Environment options have no effect since the Environment options are a resultant set of the Permanent and Temporary Environment Compile Option sets.

See also:

- “C/C++ Environment Compile Options Editor - Macros” on page 7-11

Permanent

A macro in the list of Permanent Unit Compile Options overrides any Environment Compile Options for the unit to which it is applied and may be overridden by the same macro defined as a Temporary Unit Compile Option.

See also:

- “C/C++ Environment Compile Options Editor - Macros” on page 7-11

Any macro may be directly entered in the text field below the list of **Permanent Unit Compile Options** and subsequently **Set** for inclusion in that list. In addition, a macro may be selected from the list of **Permanent Unit Compile Options**, edited in this field, and **Set** when editing is completed.

Set

Adds the macro defined in the text field below the list of **Permanent Unit Compile Options** to that list.

Delete

Removes the macro selected in the list of **Permanent Unit Compile Options** from that list.

Temporary

A macro in the list of **Temporary Unit Compile Options** overrides the same macro defined in either the list of **Permanent Unit Compile Options** or **Environment Compile Options**

See also:

- “C/C++ Environment Compile Options Editor - Macros” on page 7-11

Any macro may be directly entered in the text field below the list of **Temporary Unit Compile Options** and subsequently **Set** for inclusion in that list. In addition, a macro may be selected from the list of **Temporary Unit Compile Options**, edited in this field, and **Set** when editing is completed.

Set

Adds the macro defined in the text field below the list of **Temporary Unit Compile Options** to that list.

Delete

Removes the macro selected in the list of **Temporary Unit Compile Options** from that list.

Effective

Macros in the list of **Environment Compile Options** may be overridden by those macros redefined in the list of **Permanent Unit Compile Options** which may be

further overridden by those macros redefined in the list of **Temporary Unit Compile Options**. The **Effective** set of macros is the sum total of the **Environment**, **Permanent** and **Temporary** macro definitions, with respect to this hierarchical relationship.

See also:

- “C/C++ Environment Compile Options Editor - Macros” on page 7-11

NOTE

The buttons under the list of **Effective** options have no effect since the **Effective** options are a resultant set of the **Environment**, **Permanent** and **Temporary** options.

C/C++ Unit Compile Options Editor - Includes

Allows the user to add directories to the include search path.



Figure 7-18. C/C++ Unit Compile Options dialog - Includes Page

Environment

Lists the Effective Environment Compile Options.

NOTE

The buttons under the list of Environment options have no effect since the Environment options are a resultant set of the Permanent and Temporary Environment Compile Option sets.

See also:

- “C/C++ Environment Compile Options Editor - Includes” on page 7-13

Permanent

A path preceded by a ! in the list of Permanent Unit Compile Options overrides the same path in the Environment Compile Options list for the unit to which it is applied. A path in the list of Permanent Unit Compile Options may be overridden if the same path is preceded by a ! in the list of Temporary Unit Compile Options.

Paths are searched in the order in which they appear in the **Permanent Unit Compile Options** list.

See also:

- “C/C++ Environment Compile Options Editor - Includes” on page 7-13

Any path may be directly entered in the text field below the list of **Permanent Unit Compile Options** and subsequently **Set** for inclusion in that list. In addition, a path may be selected from the list of **Permanent Unit Compile Options**, edited in this field, and **Set** when editing is completed.

Set

Adds the path specified in the text field below the list of **Permanent Unit Compile Options** to that.

Delete

Removes the path selected in the list of **Permanent Unit Compile Options** from that list.



Changes the position of the selected path in the list of **Permanent Unit Compile Options** to an earlier (more preferred) position. Paths are searched in the order in which they appear in the **Permanent Unit Compile Options** list.



Changes the position of the selected path in the list of **Permanent Unit Compile Options** to a later (less preferred) position. Paths are searched in the order in which they appear in the **Permanent Unit Compile Options** list.



Use this to popup a dialog and browse for the path you wish to add. When finished, the pathname appears in the text field.

Temporary

A path in the list of **Temporary Unit Compile Options** overrides the same macro defined in either the list of **Permanent Unit Compile Options** or **Environment Compile Options**

See also:

- “C/C++ Environment Compile Options Editor - Macros” on page 7-11

Any path may be directly entered in the text field below the list of **Temporary Unit Compile Options** and subsequently **Set** for inclusion in that list. In addition, a macro may be selected from the list of **Temporary Unit Compile Options**, edited in this field, and **Set** when editing is completed.

Set

Adds the path specified in the text field below the list of **Temporary Unit Compile Options** to the list of **Temporary Unit Compile Options**.

Delete

Removes the path selected in the list of **Temporary Unit Compile Options** from the list of **Temporary Unit Compile Options**.



Changes the position of the selected path in the list of **Temporary Unit Compile Options** to an earlier (more preferred) position. Paths are searched in the order in which they appear in the **Temporary Unit Compile Options** list.



Changes the position of the selected path in the list of **Temporary Unit Compile Options** to a later (less preferred) position. Paths are searched in the order in which they appear in the **Temporary Unit Compile Options** list.



Use this to popup a dialog and browse for the path you wish to add. When finished, the pathname appears in the text field.

Effective

Paths in the list of **Environment Compile Options** may be overridden by those paths redefined in the list of **Permanent Unit Compile Options** which may be further overridden by those paths redefined in the list of **Temporary Unit Compile Options**. The **Effective** set of paths is the sum total of the paths in the **Environment**, **Permanent** and **Temporary** lists, with respect to this hierarchical relationship.

NOTE

The buttons under the list of **Effective** options have no effect since the **Effective** options are a resultant set of the **Environment**, **Permanent** and **Temporary** options.

C/C++ Unit Compile Options Editor - Optimizer



Figure 7-19. C/C++ Unit Compile Options dialog - Optimizer Page

Optimization Level

See “Optimization Level” on page 7-62.

Optimization Class

See “Optimization Class” on page 7-62.

Optimize for Space

See “Optimize for Space” on page 7-63.

Total Alias Object Limit

See “Total Alias Object Limit” on page 7-64.

Variables to Optimize

See “Variables to Optimize” on page 7-66.

Loops to Optimize

See “Loops to Optimize” on page 7-61.

Constant Loop Unroll Limit

See “Constant Loop Unroll Limit” on page 7-58.

Variable Loop Unroll Limit

See “Variable Loop Unroll Limit” on page 7-66.

Growth Limit

See “Growth Limit” on page 7-61.

C/C++ Unit Compile Options Editor - Language



Figure 7-20. C/C++ Unit Compile Options dialog - Language Page

Ansi Compliance Enforcement

See “Ansi Compliance Enforcement” on page 7-57.

Use Old For Loop Declarations

See “Use Old For Loop Declarations” on page 7-65.

Warn About Old For Loops

See “Warn About Old For Loops” on page 7-66.

Cfront 2.1 Compatible

See “Cfront 2.1 Compatible” on page 7-57.

Cfront 3.0 Compatible

See “Cfront 3.0 Compatible” on page 7-57.

Use Exceptions

See “Use Exceptions” on page 7-64.

Use Run Time Type Identification

See “Use Run Time Type Identification” on page 7-65.

Short Temp Lifetimes

See “Short Temp Lifetimes” on page 7-63.

Allow Function Inlining

See “Allow Function Inlining” on page 7-57.

C/C++ Unit Compile Options Editor - Dialects

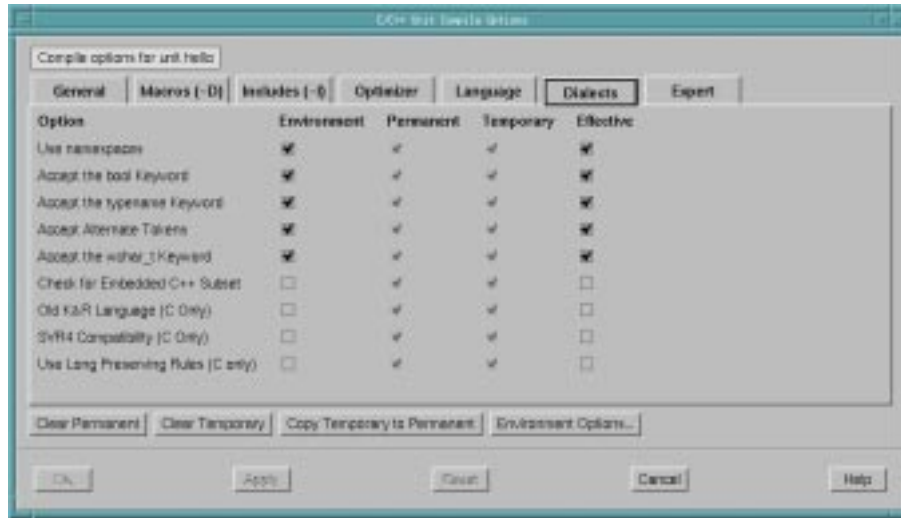


Figure 7-21. C/C++ Unit Compile Options dialog - Dialects Page

Use namespaces

See “Use namespaces” on page 7-65.

Accept the bool Keyword

See “Accept the bool Keyword” on page 7-56.

Accept the typename Keyword

See “Accept the typename Keyword” on page 7-56.

Accept Alternate Tokens

See “Accept Alternate Tokens” on page 7-56.

Accept the wchar_t Keyword

See “Accept the wchar_t Keyword” on page 7-56.

Check for Embedded C++ Subset

See “Check for Embedded C++ Subset” on page 7-58.

Old K&R Language (C Only)

See “Old K&R Language (C Only)” on page 7-61.

SVR4 Compatibility (C Only)

See “SVR4 Compatibility (C Only)” on page 7-64.

Use Long Preserving Rules (C Only)

See “Use Long Preserving Rules (C Only)” on page 7-65.

C/C++ Unit Compile Options Editor - Expert

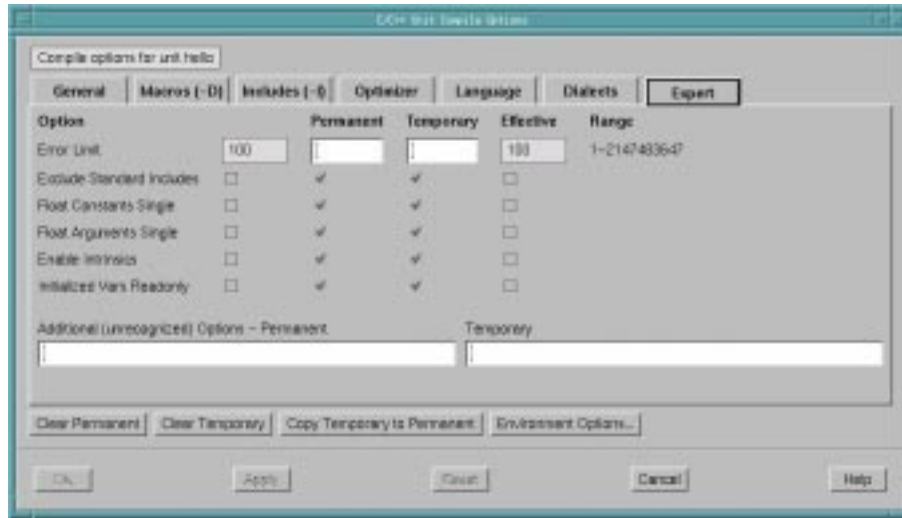


Figure 7-22. C/C++ Unit Compile Options dialog - Expert Page

Error Limit

See "Error Limit" on page 7-59.

Exclude Standard Includes

See "Exclude Standard Includes" on page 7-59.

Float Constants Single

See "Float Constants Single" on page 7-60.

Float Arguments Single

See "Float Arguments Single" on page 7-59.

Enable Intrinsic

See "Enable Intrinsic" on page 7-59.

Initialized Vars Readonly

See "Initialized Vars Readonly" on page 7-61.

Additional (unrecognized) Options

Permanent

Any other permanent options not explicitly referenced by the Unit Compile Options Editor may be specified here.

Temporary

Any other temporary options not explicitly referenced by the Unit Compile Options Editor may be specified here.

Compile Options

There are two sets of compile options, depending on the language used for development:

- “Ada Options” on page 7-47
- “C/C++ Options” on page 7-55

Ada Options

Compiler Error Output

This setting only applies when using the MAXAda **a.build** command-line tool. The NightBench Builder ignores this setting.

errors only	Lists errors encountered.
errors & lines	Lists the erroneous lines with an explanation for each error.
list in source	Displays the error messages in the file at the positions in which they occur. Also, lists the line number for each line in the source file and displays a banner with the source file’s name at the top of the listing. Each error is marked where it is found in the file and an explanation is given.

<code>list in source!</code>	Displays the error messages in the file in the same manner as the <code>list in source</code> option. However, if no errors are present, the source is listed nonetheless.
<code>edit in source</code>	Embeds the error messages in the file at the positions in which they occur. The file is subsequently opened in the editor specified by the <code>\$EDITOR</code> environment variable.
<code>vi in source</code>	Embeds the error messages in the file at the positions in which they occur. The file is subsequently opened in the <code>vi (1)</code> editor.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26
- “Ada Options” on page 7-47
- *MAXAda Reference Manual* (0890516)

Debug Information

This option controls the level of debug information generated for units in a given environment.

<code>none (0)</code>	No debug information. Debugging tools requiring line numbers or symbolic information will not fully function on modules compiled at this debug level.
<code>lines (1)</code>	Minimal level of debug information which provides line number information only. Debugging tools requiring symbolic information will not fully function on modules compiled at this debug level.
<code>full (2)</code>	Full level of debug information. This is required for most debugging tools and packages to fully function.

When new environments are created, the value of this parameter is set at the default value, `none (0)`.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26
- “Ada Options” on page 7-47

Growth Limit

The growth limit parameter is a raw percentage that specifies the percentage increase allowed in a program's size due to the optimization performed on the program. By default, the combined effect of all optimizations which trade space for time cannot increase the size of a program by more than 25 percent. The raw percentage argument is an integer value that represents the percentage in size above 100 percent that the program may grow to be.

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

Inline Line Count

The maximum number of statements allowed within an inline expanded subprogram. The default value is 25 lines. In other words, if the default is used, then only those subprograms which contain 25 or fewer lines will be expanded inline.

See also:

- “Ada Environment Compile Options Editor - Inlining” on page 7-6
- “Ada Unit Compile Options Editor - Inlining” on page 7-27
- “Ada Options” on page 7-47

Inline Nesting Depth

The maximum depth level of inline expanded subprograms. For example, if this value is 3, the compiler will perform nested inlines up to and including three levels deep. Any nested inline calls greater than three levels deep will not be expanded inline. The default value for this parameter is 3.

See also:

- “Ada Environment Compile Options Editor - Inlining” on page 7-6
- “Ada Unit Compile Options Editor - Inlining” on page 7-27
- “Ada Options” on page 7-47

Inline Statement Limit

The maximum number of Ada statements that will be inlined. When the running total of statements included within inline-expanded subprograms exceeds this limit, then all subsequent inline expansions will not be performed. The default value for this parameter is 1,000.

See also:

- “Ada Environment Compile Options Editor - Inlining” on page 7-6
- “Ada Unit Compile Options Editor - Inlining” on page 7-27
- “Ada Options” on page 7-47

Inlines Per Compilation

The maximum number of inline expansions that will be performed in a single compilation. Once this number of inline expansions has been performed for a given compilation, no other inline expansions will be performed by the compiler. The default value for this parameter is 75.

See also:

- “Ada Environment Compile Options Editor - Inlining” on page 7-6
- “Ada Unit Compile Options Editor - Inlining” on page 7-27
- “Ada Options” on page 7-47

Invert Divides

This option transforms divides by constants into multiplies by the reciprocals. This improves performance at the possible expense of a loss in accuracy.

See also:

- “Ada Environment Compile Options Editor - Expert” on page 7-8
- “Ada Unit Compile Options Editor - Expert” on page 7-29
- “Ada Options” on page 7-47

Loops to Optimize

The maximum number of loops (per routine) that will be considered for optimization. Loop optimizations that occur at the higher levels of optimization are loop unrolling, test replacement, strength reduction, and code motion. By default, only the 20 most deeply nested loops in a given routine will be optimized.

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

Maximum (“benchmark”) Optimization

By specifying this option, all safety limits are removed on optimizations and all time and space limits are reset from their default values to extremely high values. This option will also set the level of optimization to `MAXIMAL` regardless of any other option or pragma that may indicate otherwise.

See also:

- “Ada Environment Compile Options Editor - Expert” on page 7-8
- “Ada Unit Compile Options Editor - Expert” on page 7-29
- “Ada Options” on page 7-47

Optimization Class

Acceptable values for this parameter are `safe`, `unsafe`, and `standard`. Currently, `safe` and `standard` have the same effect. `safe` is the default value. If set to `unsafe`, additional optimizations will be performed that do not ensure that a program will perform correctly. (For instance, if set to `unsafe`, a loop test replacement may cause a program to loop infinitely).

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

Optimization Level

The NightBench Builder is capable of performing various levels of program object code optimization. There are three levels of optimization available: `minimal` (1), `global` (2), and `maximal` (3). Each higher level of optimization is a superset of the level of optimization which precedes it.

The quality of code produced by the compiler is representative of the level of optimization at which it was compiled.

- Optimization level `minimal` produces less efficient code, but allows for faster compilation times and easier debugging.
- Level `global` produces highly optimized code at the expense of greater compilation times.
- `maximal` is an extension of `global` that is capable of producing even better code, but may change the behavior of the program in some cases. `maximal` attempts strength reduction operations that may raise `OVERFLOW_ERROR` exceptions when dealing with values that approach the limits of the architecture of the machine.

The default for the optimization level is `minimal`.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26
- “Ada Options” on page 7-47

Optimization Size Limit

The maximum number of “expressions” that will be processed at the GLOBAL or MAXIMAL level of optimization. If this number of expressions is reached, the compiler performs all remaining optimization at level MINIMAL. The default value for this parameter is set at a relatively high number because the number of “expressions” processed during a compilation are not easily identified by inspection of the Ada source code. This parameter is best used as a ceiling to prevent the compiler from growing dangerously large (resulting in excessive swapping or perhaps the exhaustion of available system memory). The default value for this parameter is 50,000.

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

Optimize for Space

A boolean value that determines whether all routines in a compilation will be optimized for space regardless of the values of other compiler directives. By default, this parameter is `false`.

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

Quiet Compiler Info Msgs.

Generation of info messages is suppressed by the compiler if this option is checked.

This is usually selected for individual units which generate informational messages that the user desires to ignore.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26

- “Ada Options” on page 7-47

Quiet Compiler Warnings

Generation of warning messages is suppressed by the compiler if this option is checked. Generation of info messages by the compiler is suppressed as well.

This is usually selected for individual units which generate warnings that the user desires to ignore.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26
- “Ada Options” on page 7-47

Share Mode

You control whether units are compiled for ordinary static linking (for use directly or in an archive) or as position independent code (for inclusion in a shared object). There are three possible values of share mode:

non-shared	Compilations generate code that will be statically linked. Units with this share mode cannot be included in a shared object partition, but may be included in an archive or used directly in an active partition.
shared	Compilations generate position independent code. Units with this share mode may be included only in a shared object partition. They cannot be statically linked either directly in an active partition or in an archive.
both	Compilations generate both code that is position independent and code that can be statically linked (in two distinct object files). Units with this share mode may be included in a shared object or archive partition, or used directly in an active partition.

The default value of share mode is **non-shared**.

A unit that is compiled with the share mode option set to **shared** must be placed in a shared object to be used. Units with share mode **both** may be put in a shared object, but only if its shared version is to be used.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26

- “Ada Options” on page 7-47

Suppress Runtime Checks

This option gives permission to the implementation to suppress all run-time checks.

Its effect is equivalent to `pragma SUPPRESS_ALL`, `pragma SUPPRESS(ALL_CHECKS)`, or a list of `SUPPRESS` pragmas, naming every possible check.

See also:

- “Ada Environment Compile Options Editor - General” on page 7-4
- “Ada Unit Compile Options Editor - General” on page 7-26
- “Ada Options” on page 7-47

Sync After Volatile References

Before any reference to a volatile object, an `ei_eio` instruction is inserted to enforce in-order execution of I/O.

See also:

- “Ada Environment Compile Options Editor - Expert” on page 7-8
- “Ada Unit Compile Options Editor - Expert” on page 7-29
- “Ada Options” on page 7-47

Unroll Limit

The maximum number of times that a single loop will be unrolled. By default, all loops are unrolled up to four times. Several factors are involved when determining the number of times a loop should be unrolled; therefore, a user may have a hard time determining a suitable number for this parameter. The default value for this parameter is 4.

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

Use Instruction Scheduler

The Instruction Scheduler attempts to fill the instruction pipeline by reordering instructions.

See also:

- “Ada Environment Compile Options Editor - Expert” on page 7-8

- “Ada Unit Compile Options Editor - Expert” on page 7-29
- “Ada Options” on page 7-47

Use Multiply-Add Instructions

Combines multiplies and adds in a single instruction.

If unchecked, the compiler does not use floating-point multiply-add instructions. These instructions carry higher than double-precision accuracy between the multiply and the add. Some users may wish to force rounding to single- or double-precision on the intermediate result.

See also:

- “Ada Environment Compile Options Editor - Expert” on page 7-8
- “Ada Unit Compile Options Editor - Expert” on page 7-29
- “Ada Options” on page 7-47

Variables to Optimize

The maximum number of objects (per routine) that will be optimized. An *object* is any scalar program variable or compiler-generated temporary variable that is a unique object in the eyes of the compiler. For example, if this number is set to 100, then only the 100 most-used objects in a given routine will be considered as “real” objects by the compiler. *Real objects* are the only objects taken into consideration by the optimizer when it comes time to perform optimizations such as copy propagation and dead-code elimination. By default, only the 128 most often used objects will be considered for optimizations.

See also:

- “Ada Environment Compile Options Editor - Optimization” on page 7-7
- “Ada Unit Compile Options Editor - Optimization” on page 7-28
- “Ada Options” on page 7-47

C/C++ Options

64 Bit long long type

Enable or disable the extension `long long`, a 64-bit integer. Also defines the predefined macro `__LONG_LONG`

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31

- “C/C++ Options” on page 7-55

Accept Alternate Tokens

Enable or disable recognition of alternative tokens. This controls recognition of the digraph tokens in C and C++, and controls recognition of the operator keywords (e.g., `and`, `bitand`, etc.) in C++.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Accept the bool Keyword

Enable or disable recognition of `bool`. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Accept the typename Keyword

Enable or disable recognition of `typename`. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Accept the wchar_t Keyword

Enable or disable recognition of `wchar_t` as a keyword. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Allow Function Inlining

Enable or disable function inlining. If disabled, calls to inline functions will call out-of-line instances.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Ansi Compliance Enforcement

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Cfront 2.1 Compatible

Enable compilation of C++ with compatibility with cfront version 2.1. This causes the compiler to accept language constructs that, while not part of the C++ language definition, are accepted by the AT&T C++ Language System (cfront) release 2.1. This option also enables acceptance of anachronisms.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Cfront 3.0 Compatible

Enable compilation of C++ with compatibility with cfront version 3.0. This causes the compiler to accept language constructs that, while not part of the C++ language definition, are accepted by the AT&T C++ Language System (cfront) release 3.0. This option also enables acceptance of anachronisms.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Chars Signed by Default

Make plain `char` signed. The default “signedness” for `char` is unsigned, as this is more efficient on the PowerPC architecture. When plain `char` is signed, the macro `__SIGNED_CHARS__` is defined by the front end.

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31
- “C/C++ Options” on page 7-55

Check for Embedded C++ Subset

Enable the diagnosis of noncompliance with the “Embedded C++” subset (from which templates, exceptions, namespaces, new-style casts, RTTI, multiple inheritance, virtual base classes, and `mutable` are excluded).

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Constant Loop Unroll Limit

Limit the number of times a loop with a number of iterations known at compile time may be unrolled. For more information, see the “Program Optimization” chapter of the *Compilation Systems Volume 2 (Concepts)*. The default is 10.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Debug Information

Produce additional symbolic debugging information for use with NightView.

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31
- “C/C++ Options” on page 7-55

Enable Intrinsic

Turn on intrinsic functions. The compiler will then generate in-line code for accessing special machine instructions. Use of this option also defines the preprocessor macro `_FAST_MATH_INTRINSICS`.

See also:

- “C/C++ Environment Compile Options Editor - Expert” on page 7-22
- “C/C++ Unit Compile Options Editor - Expert” on page 7-46
- “C/C++ Options” on page 7-55

Error Limit

Set the error limit to the number specified. The front end will abandon compilation after this number of errors (remarks and warnings are not counted toward the limit). By default, the limit is 100.

See also:

- “C/C++ Environment Compile Options Editor - Expert” on page 7-22
- “C/C++ Unit Compile Options Editor - Expert” on page 7-46
- “C/C++ Options” on page 7-55

Exclude Standard Includes

Do not look in unspecified search paths for include files or compilation processors. An error message will be generated if the files cannot be found in the specified search paths.

See also:

- “C/C++ Environment Compile Options Editor - Expert” on page 7-22
- “C/C++ Unit Compile Options Editor - Expert” on page 7-46
- “C/C++ Options” on page 7-55

Float Arguments Single

Like the Float Constants Single option (see below), but also disable the automatic type promotion of floating-point expressions to type `double` when passed as parameters to functions. With this option, it is possible to write and use true single-precision functions, but it becomes the user’s responsibility to provide double-precision arguments to functions that expect them (such as standard library routines like `printf`).

NOTE

Note that function prototypes may be used to declare routines that accept float arguments, regardless of default type-promotion rules. This can eliminate the need to use this option. In ANSI mode, prototype declarations for the single-precision math library routines are available in `<math.h>`. See **trig(3M)**.

See also:

- “C/C++ Environment Compile Options Editor - Expert” on page 7-22
- “C/C++ Unit Compile Options Editor - Expert” on page 7-46
- “C/C++ Options” on page 7-55

Float Constants Single

Cause all floating-point constants to have type `float` instead of the default type `double`. This can be used to prevent type promotion of floating-point expressions involving constants to double precision.

NOTE

It is possible to force individual floating point constants to have type `float` by adding an `f` or `F` suffix. For example, `3.13f` has type `float` while `3.14` has type `double` (by default). This eliminates the need to use the `-fsingle` option and allows greater flexibility in controlling the types of floating-point literals.

See also:

- “C/C++ Environment Compile Options Editor - Expert” on page 7-22
- “C/C++ Unit Compile Options Editor - Expert” on page 7-46
- “C/C++ Options” on page 7-55

Full Debug Information

Generate debugging information for every entity declared in a compilation unit. Normally debugging information is created only for types that are actually used in the compilation unit.

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31
- “C/C++ Options” on page 7-55

Growth Limit

Limit the amount of intermediate code the optimizer is allowed to duplicate when performing optimizations such as loop unrolling and repairing irreducible flow graphs. The represents the percentage increase in code size permitted. The default value is 25.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Initialized Vars Readonly

Make initialized variables shared and read-only.

See also:

- “C/C++ Environment Compile Options Editor - Expert” on page 7-22
- “C/C++ Unit Compile Options Editor - Expert” on page 7-46
- “C/C++ Options” on page 7-55

Loops to Optimize

Set the number of loops for which the compiler will perform the copy-variable optimization. The default value is 20.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Old K&R Language (C Only)

Enable K&R/pcc mode, which approximates the behavior of the standard UNIX `pcc`. ANSI C features that do not conflict with K&R/pcc features are still supported in this mode.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Optimization Class

Acceptable values for this parameter are `safe`, `unsafe`, and `standard`. Currently, `safe` and `standard` have the same effect. `safe` is the default value. If set to `unsafe`, additional optimizations will be performed that do not ensure that a program will perform correctly. (For instance, if set to `unsafe`, a loop test replacement may cause a program to loop infinitely).

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Optimization Level

The NightBench Builder is capable of performing various levels of program object code optimization. There are five levels of optimization available: `none` (0), `minimal` (1), `global` (2), `maximal` (3) and `ultimate` (4). Each higher level of optimization is a superset of the level of optimization which precedes it.

The quality of code produced by the compiler is representative of the level of optimization at which it was compiled.

- Optimization level `none` places strict controls on minimal optimization. This optimization level is usually used only on extremely large machine-generated source files.
- Optimization level `minimal` produces less efficient code, but allows for faster compilation times and easier debugging.
- Level `global` produces highly optimized code at the expense of greater compilation times.
- `maximal` is an extension of `global` that is capable of producing even better code, but may change the behavior of the program in some cases.
- `ultimate` performs maximal optimizations, uses fastest transformations and links with the fastest libraries. The compiler takes a substantial amount of time to produce code at this level of optimization.

The default for the optimization level is `minimal`.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Optimize for Space

A boolean value that determines whether all routines in a compilation will be optimized for space regardless of the values of other compiler directives. By default, this parameter is `false`.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Position Independent Code

Cause the compiler to produce position-independent code. This compile option must be applied to units included in *shared objects*. It should not be applied, however, to units included in *executables* or in *objects*.

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31
- “C/C++ Options” on page 7-55

Short Temp Lifetimes

Select the lifetime for temporaries: “short” means to end of full expression; “long” means to the earliest of end of scope, end of switch clause, or the next label. “short” is standard C++, and “long” is what cfront uses (the cfront compatibility modes select “long” by default).

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Show Remarks (Nitpick)

Issue remarks, which are diagnostic messages even milder than warnings.

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31
- “C/C++ Options” on page 7-55

Suppress Warnings

Suppress warnings. Errors are still issued.

See also:

- “C/C++ Environment Compile Options Editor - General” on page 7-9
- “C/C++ Unit Compile Options Editor - General” on page 7-31
- “C/C++ Options” on page 7-55

SVR4 Compatibility (C Only)

Enable or disable recognition of SVR4 C compatibility features. This option also specifies that the source language being compiled is ANSI C.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Total Alias Object Limit

Limit the number of objects used in alias analysis, which is a measure of the preciseness of the analysis. A value of zero means no limits will be applied. Default value is 10000.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Use Exceptions

Enable or disable support for exception handling. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Use Long Preserving Rules (C Only)

Enable or disable the K&R usual arithmetic conversion rules with respect to `long`. This means the rules of K&R I, Appendix A, 6.6, not the rules used by the `pcc` compiler. The significant difference is in the handling of “`long op unsigned int`” when `int` and `long` are the same size. The ANSI/ISO/`pcc` rules say the result is `unsigned long`, but K&R I says the result is `long` (`unsigned long` did not exist in K&R I).

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Use namespaces

Enable or disable support for namespaces. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Dialects” on page 7-20
- “C/C++ Unit Compile Options Editor - Dialects” on page 7-44
- “C/C++ Options” on page 7-55

Use Old For Loop Declarations

Control the scope of a declaration in a `for-init-statement`. The old (cfront-compatible) scoping rules mean the declaration is in the scope to which the `for` statement itself belongs; the new (standard-conforming) rules in effect wrap the entire `for` statement in its own implicitly generated scope. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Use Run Time Type Identification

Enable or disable support for RTTI (runtime type information) features: `dynamic_cast`, `typeid`. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18

- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Variable Loop Unroll Limit

Limit the number of times a loop with an unknown number of iterations may be unrolled. See also “Constant Loop Unroll Limit” on page 7-58.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Variables to Optimize

Set the maximum number of variables that the compiler will optimize when global or maximal optimization is specified. (Limits optimizations such as dead code elimination, copy propagation and copy variables). The default is 128.

See also:

- “C/C++ Environment Compile Options Editor - Optimizer” on page 7-16
- “C/C++ Unit Compile Options Editor - Optimizer” on page 7-40
- “C/C++ Options” on page 7-55

Warn About Old For Loops

Enable or disable a warning that is issued when programs compiled under the new for-init scoping rules would have had different behavior under the old rules. The diagnostic is only put out when the new rules are used. This option is valid only in C++ mode.

See also:

- “C/C++ Environment Compile Options Editor - Language” on page 7-18
- “C/C++ Unit Compile Options Editor - Language” on page 7-42
- “C/C++ Options” on page 7-55

Keyboard Shortcuts and Accelerators

This appendix lists all the keyboard shortcuts and accelerators found in NightBench.

NightBench Project

Accelerators

NightBench Menu	Alt+n
Options Menu	Alt+o
Tools Menu	Alt+t
Help Menu	Alt+h

Shortcuts

Close Window	Ctrl+w
Exit NightBench Session	Ctrl+q

NightBench Development

Accelerators

Settings Page	Alt+s
Source Files Page	Alt+f
Units Page	Alt+u
Dependencies Page	Alt+e
Partitions Page	Alt+p
Partition Settings - General	Alt+g
Partition Settings - Units	Alt+n
Partition Settings - Link Control	Alt+r

Partition Settings - Link Options	Alt+k
Partition Settings - Tracing	Alt+a
Partition Settings - Expert	Alt+x
Development Menu	Alt+d
Select Menu	Alt+l
Options Menu	Alt+o
Tools Menu	Alt+t
Help Menu	Alt+h

Shortcuts

New Environment	Ctrl+n
Open Environment	Ctrl+o
Refresh Environment Info	Ctrl+r
Close Window	Ctrl+w
Exit NightBench Session	Ctrl+q
Select All	Ctrl+a
Deselect All	Ctrl+\
Long Time Stamps	Ctrl+t
Show Command Log	Ctrl+l

NightBench Builder

Accelerators

Build Page	Alt+d
Targets Page	Alt+a
Settings Page	Alt+s
Notification Page	Alt+n
Expert Page	Alt+e

Builder Menu	Alt+b
Options Menu	Alt+o
Tools Menu	Alt+t
Help Menu	Alt+h

Shortcuts

Open Environment	Ctrl+o
Refresh Environment Info	Ctrl+r
Build Transcripts	Ctrl+t
Close Window	Ctrl+w
Exit NightBench Session	Ctrl+q
Show Command Log	Ctrl+l

Glossary

This glossary defines terms used in NightBench. Terms in *italics* are defined here.

active partition

An active partition is the simplest form of partition and it describes how to build an executable program.

ambiguous

Upon introducing a unit having the same name as a previously introduced unit, NightBench labels both units as ambiguous. It will then refuse to perform any operations on either of the two versions, or on any units dependent upon the ambiguous unit. The user will be forced to choose which of the two units should actually exist in the environment.

archive

An archive is a collection of routines and data that can be used by *active partitions (Ada)*, *executable partitions (C/C++)* or foreign language executables. Active partitions and executables reference the archive during the link phase and include any needed portions of the archive into the output file. An archive is not referenced at execution time. Archives are composed of units build with static code. In an Ada environment, the compile option **Share Mode** for units in an archive should be set to non-shared (or both). In C/C++ environments, the compile option **Position Independent Code** should not be applied to those units included in an archive.

archive link method

A link method which instructs the linker to search all archive partitions for included units.

artificial unit

Those units created by the implementation to support *generic instantiations (Ada)* or *template* and extern inline *instantiations (C++)*.

automatic instantiation

The process which occurs during the prelinking phase of a C++ program that assigns instantiations of instantiatable entities (such as template functions or extern inlines) to specific compilation units in the program.

body

The part of an Ada unit which contains the details of its implementation.

bound task

A task that is served by an anonymous *server group* containing exactly one server. This server group exists only to execute the single task for which it was created, dedicated for its exclusive use. See *multiplexed task*.

configuration pragma

Configuration pragmas are syntactical entities that are not part of a unit. Configuration pragmas can appear either at the beginning of a source file containing library units or independently in a source file with no units. See also *pragma*.

compiled

A compilation state in which the semantic meaning has been determined, all implementation details have been determined, and object files have been generated, if appropriate.

consistency

The compilation of a unit is consistent if its source file has not been modified since it has been compiled and all of the units on which it depends are still consistently compiled. In addition, the unit can only remain consistent if it and the units on which it depends have not become *ambiguous* or *obscured*. In addition, a unit can only remain consistent if the compile options for that unit and the units on which it depends have not changed.

current run context

The *current run context* is the *run context* in the Run Context field of the Run Context dialog.

If the Run Context dialog was opened by selecting the Run Context... menu item from the Development menu of NightBench Development, the current run context is initially set to the run context associated with the partition selected on the Partitions page.

If the Run Context dialog was opened by selecting the Run Context... menu item from the Builder menu of the NightBench Builder, the current run context is initially set to the run context that appears in the Run Context field on the Build page.

dependent partitions

Dependent partitions are those partitions in the *environment* (or on the *Environment Search Path*) that should be searched first and foremost when trying to find the required units for a partition during the linking phase. In Ada environments, units that a partition requires may be found in those partitions either listed explicitly in the dependent partitions list or included implicitly through the *link rule* for that partition.

drafted

A compilation state in which all semantic information has been determined, all implementation details have been determined, but no actual object files have been created.

effective options

The resultant set of compile options based on the hierarchical relationship between the environment-wide options, permanent unit options, and temporary unit options.

environment

The context in which *partitions* and *units* are built and maintained by NightBench. See 10.1.4(1) of the for a complete definition. An environment contains all the units introduced to it, fetched or naturalized into it, and all the units found on its *Environment Search Path*.

Environment Default Run Context

A *run context* whose values may be used when creating new run contexts (unless otherwise specified). The Environment Default Run Context may be changed by selecting it from the drop-down list associated with the **Run Context** field in the **Run Context** dialog and editing it accordingly. (Note that run contexts that had been previously created using the Environment Default Run Context do not inherit any future changes made to the Environment Default Run Context.)

Environment Search Path

NightBench uses the concept of an Environment Search Path to allow users to specify that units from environments other than the current environment should be made available to the current environment. This Environment Search Path relates only to each particular environment and each environment has its own Environment Search Path.

environment-wide compile options

Environment-wide compile options apply to all units within that environment. All compilations of units native to the environment observe these environment-wide options unless overridden.

executable

An executable is an ELF object file that may be executed as a program.

fetched unit

A fetched unit is the compiled form of a unit which has been manually placed from another environment into the local environment. A fetched unit retains the permanent and temporary unit compile options from the original unit but these options may be changed in the local environment. However, it does not retain the environment-wide compile options of its original environment. It uses those of the current environment instead.

foreign unit

A foreign unit is a unit which exists in another environment on the Environment Search Path of the current environment. Foreign units are automatically available to the current environment once their environment is added to the Environment Search Path of the current environment.

frozen environment

An environment can be frozen, making it unalterable unless it is subsequently thawed. The advantage of freezing an environment is that the consistency of the contents of the environment is ascertained and subsequently stored. Access to the contents of a frozen environment from other, usually unfrozen, environments is substantially faster since the consistency information does not have to be determined. An environment may only be frozen if all of its required environments (those on its Environment Search Path) are also frozen. See also *invalidly frozen environment*.

generic unit

A generic unit is a program unit that is either a generic subprogram or a generic package (or, for C/C++, a generic function or a generic class). A generic unit is a template, which can be parameterized, and from which corresponding (nongeneric) subprograms or packages can be obtained.

hidden unit

In a C/C++ environment, hidden units are those units that will not be considered for resolving *instantiation* requests.

instance

A subprogram or package obtained as the result of instantiating a generic unit with appropriate generic actual parameters for the generic formal parameters.

instantiation

The act of creating a specific version of a *generic (Ada)* or *template (C/C++)* function for the particular argument type specified.

invalidly frozen environment

An environment may only be frozen if all of its required environments (those on its Environment Search Path) are also frozen. If any of those required environments changes its state to *not frozen*, the state of the currently frozen environment will change to *invalidly frozen*. If an environment is invalidly frozen, it will behave as if it is not frozen until all its required environments are frozen again, whereupon the invalidly frozen environment must be re-frozen. If an environment is invalidly frozen, no speed improvements can be derived from it until it is thawed and frozen again. An invalidly frozen environment is still unalterable, though. See also *frozen environment*.

keyword

Keywords are simply shorthand for *environments*. These environments contain various packages that can be used for program development. NightBench defines a number of keywords that correspond to environments provided with the NightBench product.

See Chapter 9 of the *MAXAda Reference Manual* (0890516) for details on these environments and the packages contained within them.

link method

The link method specifies the manner in which a unit or system library is included in the linking process. It can instruct the linker to use the object of a unit directly (**Object link method**), utilize the unit found in an archive partition (**Archive link method**), or include the unit contained within a shared object partition (**Shared Object link method**).

Similarly, when including system libraries, the ordering of link methods tells the linker whether the shared object or archive of the system library should be used.

These methods are used in conjunction with the link rule. See *link rule*.

link rule

The order of *link methods* which the linker follows to include the units or system libraries for a given partition.

magnet

In a C/C++ environment, magnets are units which will be preferred over others to host *artificial units* for *instantiations*.

main subprogram

A non-generic library subprogram without parameters that is either a procedure or a function returning an Ada `STANDARD_INTEGER` (the predefined type).

multiplexed task

A task that shares the resources of a single pool and is served by a named *server group*, which may contain one or more servers. See *bound task*.

native unit

A native unit is a unit which has been manually introduced directly into the current environment or which has been fetched into the current environment.

naturalized unit

A naturalized unit is the compiled form of a foreign unit created by the compilation system in the local environment. A naturalized unit retains the options from its original environment. These options can only be altered by changing them in the original environment. It is created to allow units and partitions in the local environment

to be built even though they require uncompiled or inconsistent units in a foreign environment.

object

An object is an ELF object file that may be linked into an *executable* or *shared object*.

object link method

A link method which instructs the linker to use the object of a unit directly when linking.

obscurities

Obscurities occur when the natural behavior of NightBench and the *Environment Search Path* mechanism prevent an intended file from being used for a particular compilation.

package

From the *Ada 95 Reference Manual*:

Packages are program units that allow the specification of groups of logically related entities. Typically, a package contains the declaration of a type (often a private type or private extension) along with the declarations of primitive subprograms of the type, which can be called from outside the package, while their inner workings remain hidden from outside users.

parsed

A compilation state in which semantic information has been determined, but no implementation details have been determined, nor have any object files been generated.

part

A designated section of an Ada package. A part can either be a *specification* (**spec**) or a *body* (**body**).

Also known as the *portion* of the unit.

partition

A partition is an executable, archive, or shared object that can be invoked outside of NightBench. The user can explicitly assign *units* to partitions. The units included in a partition are those of the explicitly assigned units and, optionally, other units needed by those explicitly assigned units. NightBench manages these units and their dependencies, as well as link options and configuration information for each partition within the context of an *environment*. See 10.2(2) of the *Ada 95 Reference Manual* for a complete definition.

permanent unit compile options

This set of compile options is associated with a unit and override its *environment-wide compile options*. Each unit has its own set of permanent unit compile options.

pragma

A pragma is a compiler directive. There are language-defined pragmas that give instructions for optimization, listing control, etc. An implementation may support additional (implementation-dependent) pragmas. See also *configuration pragmas*.

protected object

From the *Ada 95 Reference Manual*:

A protected object provides coordinated access to shared data, through calls on its visible protected operations, which can be protected subprograms or protected entries.

run context

Run contexts are used when running or debugging programs, allowing the user to specify items such as program arguments, the directory in which to execute, the system (local or remote) on which to run, as well as other attributes for a particular program. The user also may use run contexts to specify debugger commands to be issued prior to debugging a program.

run context association

The run context currently associated with each partition is listed under the **Run Context** column in the **Partitions List** on the **Partitions** page. The user can associate a different run context for a particular partition by selecting the partition from the **Partitions List**, opening the **Run Context** dialog (using the **Run Context...** button on the **Partitions** page), and either selecting a different run context from the drop-down list associated with the **Run Context** field or creating a new run context.

shared object

A shared object is a collection of routines and data that can be used by multiple *active partitions (Ada)*, *executable partitions (C/C++)* or foreign language executables without its contents being included in them. Active partitions and executables reference the shared object during the link phase and associate calls and other references during the execution phase. Shared objects are composed of units built with position independent code. In an Ada environment, the compile option **Share Mode** for units in a shared object should be set to **shared** (or **both**). In C/C++ environments, the compile option **Position Independent Code** must be applied to those units included in a shared object.

shared object link method

A link method which instructs the linker to look in shared object partitions for included units.

source file

A file which may contain *units* or *configuration pragmas*.

specification

The visible part of an Ada unit, containing the interface to the outside world.

subprogram

From the *Ada 95 Reference Manual*:

A subprogram is a program unit or intrinsic operation whose execution is invoked by a subprogram call. There are two forms of subprogram: procedures and functions. A procedure call is a statement; a function call is an expression and returns a value. The definition of a subprogram can be given in two parts: a subprogram declaration defining its interface, and a subprogram_body defining its execution.

task

From the *Ada 95 Reference Manual*:

The execution of an Ada program consists of the execution of one or more tasks. Each task represents a separate thread of control that proceeds independently and concurrently between the points where it interacts with other tasks.

template

A description of a class or function that is a model for a family of related classes or functions. For example, one can write a template for a `Stack` class, and then use a stack of integers, a stack of floats, and a stack of some user-defined type. In the source, these might be written `Stack<int>`, `Stack<float>`, and `Stack<X>`. From a single source description of the template for a stack, the compiler can create *instantiations* of the template for each of the types specified.

temporary unit compile options

This set of compile options is temporarily associated with a unit and override its *permanent unit compile options*. Temporary options allow users to “try out” some options or change particular options for a specific compilation but only “temporarily”. The temporary unit compile options are persistent in the same way that permanent options are persistent, however, they may be cleared easily without altering the permanent options. In addition, the temporary unit compile options, if desired, can be added to the set of permanent unit compile options.

uncompiled

The compilation state of a newly-introduced unit, or one that has been invalidated. The environment is aware of the unit and some basic dependency information but very little else.

unit

Shorthand for *compilation units* as defined in the *Ada 95 Reference Manual*. Units are the basic building blocks of the NightBench environments. It is through units that NightBench performs most all its library management and compilation activities.

Symbols

\$(var) 6-25
\$(NBENCH_REMOTE_NAME) 6-30
\$(var) 6-24, 6-25
\${var} 6-25
\$NBENCH_ALERT_COUNT 6-42
\$NBENCH_BUILD_STATUS 6-42
\$NBENCH_BUILD_TARGET 6-42
\$NBENCH_ENVIRONMENT_LANGUAGE 6-42
\$NBENCH_ENVIRONMENT_PATH 6-42
\$NBENCH_ERROR_COUNT 6-42
\$NBENCH_ERROR_FILE_NAME 6-42
\$NBENCH_INFO_COUNT 6-42
\$NBENCH_LOG_FILE_NAME 6-42
\$NBENCH_TRANSCRIPT_FILE_NAME 6-42
\$NBENCH_WARNING_COUNT 6-42
\$var 6-25
.a 5-38
.ada 5-38
.adb 5-38
.ads 5-38
.C 5-38
.c 5-38
.cc 5-38
.CPP 5-38
.cpp 5-38
.CXX 5-38
.cxx 5-38
.pp 5-38, 5-45, 5-60
__SIGNED_CHARS__ 7-58
_FAST_MATH_INTRINSICS 7-59

A

accelerated item selection 5-16
Active partitions 2-6, 2-10, 2-25, 2-31, 3-11, 5-99,
5-150, 6-19, 6-24
Ada 6-31
Ambiguities
 resolving 5-55
ambiguities

 resolving 5-69
ambiguous 5-69
Analyze Errors dialog 5-32, 5-33
Analyze Source Files dialog 5-51
Archive partitions 2-6, 3-7

B

body 5-62, 5-63, 5-72
bool 7-56
Build transcripts 1-12
Builder 1-2, 1-4, 1-19, 2-27, 5-14, 5-65
 and Environment Search Path 5-28
 Notification page 2-9, 2-21, 2-25, 2-31, 3-10
builds
 automatically start 5-14, 5-65

C

cfront 7-57
changing permissions 5-27
chmod 5-27
Command Log 1-4, 1-5, 1-18, 5-15
Common page (Run Context) 6-21, 6-24
Compilation states 5-63
 compiled 5-91
 uncompiled 5-91
Compile options
 effective 2-14, 3-14, 7-1, 7-2, Glossary-3
 environment-wide Glossary-3
 unit
 permanent Glossary-3, Glossary-7
 temporary Glossary-3, Glossary-8
Concurrent C/C++
 release 3-3
Configuration pragmas Glossary-2
consistency 5-64, 5-68, 5-69, 5-70
Create Partition dialog 2-6, 3-7

D

Dead-code elimination 7-55
Debug page (Run Context) 6-21, 6-27
Debugging 5-101, 5-151, 6-14, 6-15, 7-58, Glossary-7
dependencies 5-51, 5-70
Dialogs
 Create Partition 2-6, 3-7
 Introduce Source Files 2-4, 2-17, 2-28, 3-4
 New Environment 2-2, 2-16, 3-2

E

editing
 source files 5-33, 5-65
Editor
 custom 1-10
 Emacs 1-10
 NEdit 1-10
 vi 1-10
Effective compile options 2-14, 3-14, 7-1, 7-2,
 Glossary-3
effective options 5-64
Emacs 1-9, 1-10
 additional considerations 1-16
Embedded C++ 7-58
Environment Default Run Context 6-20, 6-21, 6-22,
 Glossary-3
 inheritance 6-22, Glossary-3
Environment Search Path 5-1, 5-70, 5-131
 add new environment to 5-23, 5-28
 removing environments 5-24
 replace environment to 5-24, 5-30
Environment variable substitution 6-24, 6-25
Environment variables 6-42
 \$NBENCH_ALERT_COUNT 6-42
 \$NBENCH_BUILD_STATUS 6-42
 \$NBENCH_BUILD_TARGET 6-42
 \$NBENCH_ENVIRONMENT_LANGUAGE 6-42
 \$NBENCH_ENVIRONMENT_PATH 6-42
 \$NBENCH_ERROR_COUNT 6-42
 \$NBENCH_ERROR_FILE_NAME 6-42
 \$NBENCH_INFO_COUNT 6-42
 \$NBENCH_LOG_FILE_NAME 6-42
 \$NBENCH_TRANSCRIPT_FILE_NAME 6-42
 \$NBENCH_WARNING_COUNT 6-42
Environments 2-2, 3-2, 5-1
 adding to Environment Search Path 5-23, 5-28
 creating 2-2, 3-2
 freezing 5-25
 home 5-63

local 5-62
new 5-3, 5-5
open 5-3, 5-8, 5-24
origin 5-63
remove 5-5, 5-11
removing from Environment Search Path 5-24
replacing environments on Environment Search
 Path 5-24, 5-30
thawing 5-26
transitive closure 5-23
Environment-wide compile options Glossary-3
Error messages 1-2
Errors
 Analyze Errors dialog 5-53
Executable partitions 3-7
expelling units 5-70

F

fetched units 5-63, 5-70
File editor 5-33, 5-65
 preferences 1-9
File viewer 5-33, 5-65
 preferences 1-8
foreign units 5-63
freezing environments 5-25
frozen status 5-25

G

generics 5-62, 5-63, 5-64

H

Help 1-3
 Bookshelf 1-21
 context-sensitive 1-21
 menu 1-21
 on item 1-21
 on window 1-21
hidden units 5-69, 5-72

I

inconsistent 5-68
initialized variable 7-61

inline 7-57
 Introduce Source Files dialog 2-4, 2-17, 2-28, 3-4
 invalidating units 5-68

K

keywords 5-28
 ktserv 1-2

L

Language
 preferences 1-13

M

Main subprogram 2-6, 2-7
 MAXAda iii, 1-1
 commands 5-15
 release 2-3, 5-7
 MAXAda release 5-19
 more 1-9

N

Native units Glossary-5
 native units 5-63
 Naturalization 2-27
 Naturalized units Glossary-5
 naturalized units 5-63
 nb.build 1-2
 nb.dev 1-2
 nb.error 1-2
 nb.proj 1-2
 nb.serv 1-2
 nb.term 1-2
 nbench 1-2, 2-1, 3-1
 NEdit 1-1, 1-9, 1-10
 New Environment dialog 2-2, 2-16, 3-2
 NightBench
 Builder 1-2, 1-4, 1-16, 1-19, 2-27
 Development 1-1, 1-16
 exiting 5-5
 help 1-3
 NightBench Development 1-2, 1-19
 preferences 5-14

Project 1-1, 1-2, 1-3, 1-19, 2-1, 2-2, 2-16, 3-1, 3-2
 release 2-3, 3-3
 starting 1-3, 2-1, 3-1

NightBench Builder 1-16

NightBench Development 1-1, 1-2, 1-16, 1-19, 5-1
 accelerated item selection 5-16
 Partitions page 2-6, 2-8, 2-31, 3-7, 3-8, 3-9
 Settings page 5-18
 Source Files page 2-4, 2-5, 3-4, 3-5, 5-31
 Units page 5-61

NightTrace 6-24

NightView 1-1, 1-19, 5-101, 5-151, 6-15, 6-27, 7-58
 dialogue 6-28

Notification page

 Builder 2-9, 2-21, 2-25, 2-31, 3-10

nterm 1-11

ntraceud 6-25

O

Object partitions 3-7
 Obscurities Glossary-6
 on 6-29, 6-33
 Operating system quantum Glossary-6
 Options
 effective (compile) 2-14, 3-14, 7-1, 7-2, Glossary-3
 hierarchical relationship 7-1, Glossary-3
 permanent unit (compile) Glossary-3, Glossary-7
 persistent 7-1
 temporary unit (compile) Glossary-3, Glossary-8
 options
 effective 5-64
 permanent 5-64, 5-66
 temporary 5-64, 5-66
 Options Editor 5-61, 5-67
 OSF/Motif 1-1

P

parallel analyses 5-51
 Partitions 2-6, 3-7
 active 2-6
 archive 2-6, 3-7
 defining 2-6, 3-7
 executable 3-7
 including units in 2-6, 3-8
 object 3-7
 shared object 2-6, 3-7
 Partitions page
 NightBench Development 2-6, 2-8, 2-31, 3-7, 3-8,

- 3-9
- pathnames 5-32
 - absolute 5-44
 - relative 5-44
- pcc 7-65
- permanent options 5-64, 5-66
- Permanent unit compile options Glossary-3, Glossary-7
- permissions
 - changing 5-27
- position-independent code 7-63
- PowerMAX OS iii, 1-1
- Pragmas Glossary-7
- Predefined macro
 - `__SIGNED_CHARS__` 7-58
 - `__FAST_MATH_INTRINSICS` 7-59
- Preference page (Run Context) 6-21
- Preference settings (Run Context) 6-32
- Preferences 1-8
 - file editor 1-9
 - file viewer 1-8
 - language 1-13
 - start 1-14
 - terminal program 1-11
 - transcripts 1-12
- preferences 5-14
- preprocessing 5-51, 5-60
- preprocssing 5-45
- Privileges
 - setting 6-31

Q

- Quantum
 - operating system Glossary-6

R

- read-only 7-61
- Release 2-3, 3-3
- release 5-7, 5-19
- Remote exec 6-29, 6-33
- Remote shell 6-29, 6-32
- Remote systems 6-28
- resolving ambiguities 5-69
- rlogin 6-30
- rsh 6-29, 6-32
- Run contexts 5-98, 6-14, 6-19, Glossary-7
 - automatically generated 2-10, 2-25, 2-31, 3-11, 5-98, 5-99, 5-149, 5-150, 6-19, 6-20
 - automatically open dialog 6-14, 6-15, 6-32

- Common settings 6-21, 6-24
 - creating 6-20, 6-22
- current run context Glossary-2
- Debug settings 6-21, 6-27
- debugging 5-101, 5-151, 6-14, 6-15, 6-19, Glossary-7
 - deleting 5-99, 5-100, 5-150, 5-151, 6-20
 - editing 5-5, 5-100, 5-151, 6-3, 6-19, 6-20
- Environment Default Run Context 6-20, 6-21, 6-22, Glossary-3
 - environment variable substitution 6-24, 6-25
- inheritance 6-22
- manual login 6-30
- new 6-20, 6-22
- Preference settings 6-21
 - preference settings 6-32
- renaming 6-20, 6-23
- Run settings 6-21
 - run settings 6-29
- running 2-10, 2-25, 2-31, 3-11, 5-101, 5-151, 6-14, 6-19, Glossary-7
 - running locally Glossary-7
 - running remotely 6-19, 6-24, 6-25, 6-26, 6-29, 6-32, Glossary-7
 - running remotely (manual login) 6-30
 - running remotely (using on) 6-29, 6-33
 - running remotely (using rsh) 6-29, 6-32
- shell scripts 6-24
 - using 2-10, 2-25, 2-31, 3-11
- Run page (Run Context) 6-21, 6-29

S

- Settings page (NightBench Development) 5-18
- Share mode 7-53
- Shared object partitions 2-6, 3-7
- Shared objects
 - share mode 7-53
- Shell scripts 6-24
- sorting
 - units 5-63
- Source files
 - introducing 2-4, 3-4
- source files
 - absolute pathnames 5-44
 - analyzing 5-32, 5-33, 5-51
 - changing permissions of 5-27
 - editing 5-33, 5-65
 - introducing 5-32, 5-37
 - introducing (parallel) 5-60
 - introducing from file 5-36, 5-59
 - parallel analyses 5-51

- preprocessing 5-45, 5-60
- relative pathnames 5-44
- removing 5-33, 5-56
- saving list to file 5-57
- sorting 5-31
- viewing 5-33, 5-65
- Source Files page
 - NightBench Development 2-4, 2-5, 3-4, 3-5
- Source Files page (NightBench Development) 5-31
- spec 5-62, 5-63, 5-72
- Start
 - preferences 1-14
 - state 5-63
- SVR4 C 7-64

T

- telnet 6-30
- temporary options 5-64, 5-66
- Temporary unit compile options Glossary-3, Glossary-8
- Terminal emulator
 - nterm 1-11
- Terminal emulator
 - custom 1-11
 - xterm 1-11
- Terminal program
 - preferences 1-11
- Terminal window 6-30, 6-31
- terminal windows 1-2
- tfadmin 6-31
- thawing environments 5-26
- timestamps 5-15, 5-69
- Transcripts
 - preferences 1-12
- transitive closure 5-23
- translate-object-file 6-27
- Translations 6-27
- typename 7-56

U

- unhiding units 5-70, 5-73
- Unit Compile Options Editor 5-67
- Units 2-3, 3-3
 - configuration pragmas Glossary-2
 - including in partition 2-6, 3-8
 - introducing 2-3, 3-3
 - native Glossary-5
 - naturalized Glossary-5
- units

- artificial 5-62
- bodies 5-62
- consistency 5-64, 5-70
- dependencies 5-51
- expelling 5-70
- fetches 5-63
- fetching 5-70
- foreign 5-63
- formatting list 5-64
- hidden 5-69, 5-72
- hiding 5-69
- instances of generics 5-62
- invalidate 5-68
- native 5-63
- naturalized 5-63
- options 5-61, 5-64, 5-66
- part 5-72
- sorting 5-63
- specifications 5-62
- touch 5-69
- unhiding 5-70, 5-73
- visa 5-63
- Units page (NightBench Development) 5-61

V

- vi 1-10
- view 1-9
- Viewer
 - custom 1-9
 - Emacs 1-9
 - more 1-9
 - NEdit 1-9
 - view 1-9
- viewing
 - source files 5-33, 5-65
- visa 5-63

W

- wchar_t 7-56

X

- X Toolkit 1-6
- X Window System 1-1
- xterm 1-11

Spine for 1.0" Binder

**Product Name: 0.5" from
top of spine, Helvetica,
36 pt, Bold**

**Volume Number (if any):
Helvetica, 24 pt, Bold**

**Volume Name (if any):
Helvetica, 18 pt, Bold**

**Manual Title(s):
Helvetica, 10 pt, Bold,
centered vertically
within space above bar,
double space between
each title**

**Bar: 1" x 1/8" beginning
1/4" in from either side**

**Part Number: Helvetica,
6 pt, centered, 1/8" up**

NightBench

**User's
Guide**

0890514