# NightBench User's Guide

CONCURRENT COMPUTER CORPORATION™

0890514-050
November 1999

Printed in U. S. A.

| Revision History: | Level: | Effective With: |
|---|---|---|
| Original Release -- November 1997 | 000 | NightBench 1.0 |
| Current Release -- November 1999 | 050 | NightBench 1.5 |

# Preface

## General Information

The *NightBench User's Guide* documents the functions of the graphical user interface for the NightBench$^{TM}$ product. It also shows how to perform common tasks using this interface.

NightBench is a program development environment designed as an interface to MAXAda$^{TM}$, a tool set for the development of Ada programs on Concurrent computers running PowerMAX OS$^{TM}$. MAXAda processes the Ada language as specified by the Reference Manual for the Ada Programming Language, ANSI/ISO/IEC-8652:1995, referred to in this document as the Ada 95 Reference Manual or the RM.

## 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 6 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". It goes step-by-step from creating an environment to building a partition. Many of the common tasks performed in NightBench are discussed here.
- Chapter 3 documents "NightBench Main", 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 4 talks about the "Ada Environment", the window in which environments are created, units and source files are managed, and partitions are defined.
- Chapter 5 discusses the "Options Editors", the various settings contained within them, and their relationship to the units and partitions contained within the environment.

- Chapter 6 details the "NightBench Builder", the mechanism used for compilation and linking of Ada units and partitions.

## Syntax Notation

The following notation is used throughout this guide:

| | |
|---|---|
| *italic* | Books, reference cards, and items that the user must specify appear in *italic* type. Special terms and comments in code may also appear in *italic*. |
| **`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. |
| emphasis | Words or phrases that require extra emphasis use emphasis 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 *is defined as* in Backus-Naur Form (BNF). |

## Referenced Publications

The following publications are referenced in this document:

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

# Contents

**Chapter 1 Introduction to NightBench**

Overview . . . 1-1
NightBench Features . . . 1-1
NightBench Components. . . . 1-2
  Internal Components . . . 1-2
    X Resource Names . . . 1-2
Starting NightBench . . . 1-3
  NightBench Help . . . 1-3
  NightBench Main . . . 1-3
  NightBench Ada Environment . . . 1-3
    Additional Ada Environment Options. . . . 1-4
  NightBench Builder . . . 1-4
    Additional Builder Options . . . 1-4
  Additional NightBench Options . . . 1-5
Configuring NightBench . . . 1-6
  Preferences . . . 1-7
    Using the Emacs Editor - Additional Considerations . . . 1-11
  Resizing columns . . . 1-12
Command Log . . . 1-13
Analyze Errors . . . 1-14
Tools Menu . . . 1-17
Help Menu . . . 1-19

**Chapter 2 Using NightBench**

Hello World - An Example . . . 2-1
  Starting NightBench . . . 2-1
  Creating a new environment. . . . 2-2
  Introducing units into the environment . . . 2-3
  Defining a partition. . . . 2-5
    Including units in the partition . . . 2-5
  Building a partition. . . . 2-6
  Running the program . . . 2-7
  What options do I have? . . . 2-8
    Setting environment-wide compile options. . . . 2-8
    Setting unit compile options . . . 2-9
    Determining the effective options . . . 2-10
    Copying the temporary options to the permanent set . . . 2-12
Hello Galaxy - The Example Continues... . . . 2-13
  Setting up another environment . . . 2-13
  Introducing units into the galaxy . . . 2-14
  Modifying an existing unit . . . 2-16
  Building a partition with references outside the local environment. . . . 2-17
  Adding an environment to the Environment Search Path . . . 2-18
  Making contact!!! . . . 2-21

Who resides on earth? . . . . . 2-22
Hello Again - Ambiguous Units . . . . . 2-24
No more ambiguities!!! . . . . . 2-26
Using NightBench - Conclusion . . . . . 2-28

Chapter 3 NightBench Main

Overview . . . . . 3-1
NightBench Main Menu Bar . . . . . 3-1
NightBench Menu . . . . . 3-2
Options Menu . . . . . 3-3
NightBench Main . . . . . 3-4
New Environment . . . . . 3-6
Add Environment to List . . . . . 3-7

Chapter 4 Ada Environment

Overview . . . . . 4-1
Structure . . . . . 4-1
Ada Environment Menu Bar . . . . . 4-3
Environment Menu . . . . . 4-3
New Environment . . . . . 4-4
Open Environment . . . . . 4-5
Remove Environment . . . . . 4-7
Select Menu . . . . . 4-8
Options Menu . . . . . 4-9
Accelerated Item Selection . . . . . 4-11
Settings . . . . . 4-13
Change Frozen Status . . . . . 4-17
Change Permissions . . . . . 4-19
Add Environment Search Path Element . . . . . 4-20
Replace Environment Search Path Element . . . . . 4-22
Source Files . . . . . 4-23
Source File Commands . . . . . 4-27
Introduce Source Files . . . . . 4-28
Analyze Source Files . . . . . 4-32
Remove Source Files . . . . . 4-34
Save List to File . . . . . 4-35
Introduce from File . . . . . 4-37
Units . . . . . 4-39
Unit Compile Options . . . . . 4-44
Commands . . . . . 4-46
Hidden Units . . . . . 4-49
Resolve Ambiguity . . . . . 4-51
Unit Information . . . . . 4-53
Dependencies . . . . . 4-59
Partitions . . . . . 4-62
Create Partition . . . . . 4-65
Partition File Commands . . . . . 4-67
Save Partition Definitions to File . . . . . 4-68
Create Partitions from File . . . . . 4-70
Settings - General . . . . . 4-72
Settings - Units . . . . . 4-75

Settings - Units - Active Partitions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-75
Settings - Units - Non-Active Partitions . . . . . . . . . . . . . . . . . . . . . . . . . 4-78
Settings - Link Options. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-81
Settings - Link Method. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-83
Settings - Tracing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-87
Settings - Expert . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-91

**Chapter 5 Options Editors**

Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-1
Effective Options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-1
Tri-state Checkboxes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-1
Environment Compile Options Editor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-3
Button Functionality . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-3
General . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-5
Inlining . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-8
Optimization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-9
Expert . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-11
Unit Compile Options Editor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-13
Button Functionality . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-13
General . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-15
Inlining . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-19
Optimization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-21
Expert . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5-23

**Chapter 6 NightBench Builder**

Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-1
Structure. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-1
Builder Menu Bar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-2
Builder Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-2
Open Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-3
Build Transcripts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-5
Options Menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-8
Build . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-9
Targets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-14
Settings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-16
Notification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-18
Expert. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-20

**Screens**

Screen 2-1. Starting NightBench . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2-1
Screen 2-2. Source file **`world.a`** containing `hello` unit . . . . . . . . . . . . . . . . . . 2-3
Screen 2-3. Source file **`planet.a`** containing `alien` unit. . . . . . . . . . . . . . . . . 2-14
Screen 2-4. Reference the `alien` unit within the `hello` unit . . . . . . . . . . . . . . . . 2-16
Screen 2-5. Source file **`newunit.a`** containing ambiguous `hello` unit . . . . . . . 2-24

**Illustrations**

Figure 1-1. Preferences dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1-7
Figure 1-2. Command Log window . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1-13

Figure 1-3. Analyze Errors dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1-14
Figure 1-4. Tools menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1-17
Figure 1-5. Help menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1-19
Figure 2-1. Creating the **`earth`** environment . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2-2
Figure 2-2. Source file, **`world.a`** - newly introduced . . . . . . . . . . . . . . . . . . . . . 2-4
Figure 2-3. Including units in a newly defined partition . . . . . . . . . . . . . . . . . . . . 2-6
Figure 2-4. Builder window - Build page for **`hello`** partition . . . . . . . . . . . . . . . 2-7
Figure 2-5. Unit Compile Options dialog - Effective Options . . . . . . . . . . . . . . . . 2-11
Figure 2-6. Creating the **`galaxy`** environment . . . . . . . . . . . . . . . . . . . . . . . . . 2-13
Figure 2-7. Analyze Errors dialog - `mistake` in unit being introduced . . . . . . . . . 2-15
Figure 2-8. Builder window - Build page with errors for `hello` partition . . . . . . 2-18
Figure 2-9. Environment Search Path for the **`earth`** environment . . . . . . . . . . . . 2-19
Figure 2-10. Updated Environment Search Path for the **`earth`** environment . . . . 2-20
Figure 2-11. Units page with Visa List Format . . . . . . . . . . . . . . . . . . . . . . . . . . 2-22
Figure 2-12. Analyze Errors dialog - ambiguous units . . . . . . . . . . . . . . . . . . . . . 2-25
Figure 2-13. Source file, **`newunit.a`**, newly introduced . . . . . . . . . . . . . . . . . . . 2-26
Figure 3-1. NightBench menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3-2
Figure 3-2. Options menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3-3
Figure 3-3. NightBench Main window . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3-4
Figure 3-4. New Environment dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3-6
Figure 3-5. Add Environment to List dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3-7
Figure 4-1. Environment menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-3
Figure 4-2. New Environment dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-5
Figure 4-3. Open Environment dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-6
Figure 4-4. Remove Environment dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-7
Figure 4-5. Select menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-8
Figure 4-6. Options menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-9
Figure 4-7. Accelerated Item Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-11
Figure 4-8. Ada Environment window - Settings page . . . . . . . . . . . . . . . . . . . . 4-13
Figure 4-9. Change Frozen Status dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-17
Figure 4-10. Change Permissions dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-19
Figure 4-11. Add Environment Search Path Element dialog . . . . . . . . . . . . . . . . 4-20
Figure 4-12. Replace Environment Search Path Element dialog . . . . . . . . . . . . . . 4-22
Figure 4-13. Ada Environment window - Source Files page . . . . . . . . . . . . . . . . . 4-23
Figure 4-14. Source File Commands menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-27
Figure 4-15. Introduce Source Files dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-28
Figure 4-16. Analyze Source Files dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-32
Figure 4-17. Remove Source Files dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-34
Figure 4-18. Save List to File dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-35
Figure 4-19. Introduce from File dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-37
Figure 4-20. Ada Environment window - Units page . . . . . . . . . . . . . . . . . . . . . 4-39
Figure 4-21. Commands menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-46
Figure 4-22. Hidden Units dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-49
Figure 4-23. Resolve Ambiguity dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-51
Figure 4-24. Ada Environment window - Dependencies page . . . . . . . . . . . . . . . 4-59
Figure 4-25. Ada Environment window - Partitions page . . . . . . . . . . . . . . . . . . 4-62
Figure 4-26. Create Partition dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-65
Figure 4-27. Partition File Commands menu . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-67
Figure 4-28. Save Partitions to File dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4-68
Figure 4-29. Create Partitions from File dialog . . . . . . . . . . . . . . . . . . . . . . . . . 4-70
Figure 4-30. Ada Environment - Partitions page - General settings . . . . . . . . . . . 4-72
Figure 4-31. Ada Environment - Partitions page - Units settings - active . . . . . . . 4-75
Figure 4-32. Ada Environment - Partitions page - Units settings - archive . . . . . . 4-78
Figure 4-33. Ada Environment - Partitions page - Link Options settings . . . . . . . 4-81

Figure 4-34. Ada Environment - Partitions page - Link Method settings . . . . . . . . 4-83
Figure 4-35. Ada Environment - Partitions page - Tracing settings . . . . . . . . . . . . 4-87
Figure 4-36. Ada Environment - Partitions page - Expert settings . . . . . . . . . . . . 4-91
Figure 5-1. Environment Compile Options dialog - General Page . . . . . . . . . . . . 5-5
Figure 5-2. Environment Compile Options dialog - Inlining Page . . . . . . . . . . . . 5-8
Figure 5-3. Environment Compile Options dialog - Optimization Page . . . . . . . . . 5-9
Figure 5-4. Environment Compile Options dialog - Expert Page . . . . . . . . . . . . . 5-11
Figure 5-5. Unit Compile Options dialog - General Page . . . . . . . . . . . . . . . . . . . . 5-15
Figure 5-6. Unit Compile Options dialog - Inlining Page . . . . . . . . . . . . . . . . . . . . 5-19
Figure 5-7. Unit Compile Options dialog - Optimization Page . . . . . . . . . . . . . . . . 5-21
Figure 5-8. Unit Compile Options dialog - Expert Page . . . . . . . . . . . . . . . . . . . . . 5-23
Figure 6-1. Builder menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-2
Figure 6-2. Open Environment dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-3
Figure 6-3. Build Transcripts dialog . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-5
Figure 6-4. Options menu . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-8
Figure 6-5. Builder window - Build page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-9
Figure 6-6. Builder window - Targets page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-14
Figure 6-7. Builder window - Settings page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-16
Figure 6-8. Builder window - Notification page . . . . . . . . . . . . . . . . . . . . . . . . . . 6-18
Figure 6-9. Builder window - Expert page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6-20

# 1 Introduction to NightBench

## Overview

The *NightBench User's Guide* documents the NightBench Program Development Environment. It describes the components of the graphical user interface (GUI) 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 at the close of the manual.

## NightBench Features

NightBench is a program development environment providing a graphical user interface to MAXAda$^{TM}$, a tool set for the development of Ada programs on Concurrent computers running under PowerMAX OS$^{TM}$.

In addition to MAXAda, a high-quality, high-performance real-time compiler, NightBench also incorporates NightView$^{TM}$, a fully-symbolic debugger, and NEdit, a fully integrated text editor, bringing together a collection of tools crucial to Ada program 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$^{TM}$ and the X Window System$^{TM}$ standards, providing the user with a work environment in which to develop Ada 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 Main - Chapter 3
- NightBench Ada Environment - Chapter 4
- 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.

| | |
|---|---|
| **`nbench`** | command issued by the user to start NightBench. See "Starting NightBench" on page 1-3 for more information. |
| **`nb.proj`** | handles all the NightBench Main user interactions |
| **`nb.ada`** | handles all of the NightBench Ada Environment 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:

- `NbAda` - Ada Environment
- `NbBuild` -Builder
- `NbProj` - NightBench Main

## 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 Main

**`nbench`**

Brings up the NightBench Main window.

## NightBench Ada Environment

**`nbench -ada`**

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

**`nbench -env`** *env_name*

Brings up the NightBench Ada Environment window, automatically opening the specified environment. If the environment is not valid, an empty Ada Environment window will be opened.

**`nbench -mkenv [-f] [-env`** *env_name*`] [-rel` *rel_name*`]`

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

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 MAXAda release to use when creating the environment.

### Additional Ada Environment Options

**`-log`**

Opens a Command Log window when opening the NightBench Ada Environment 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`** *env_name*

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

### 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, 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, the default to the builder is "All Units and All Partitions" in the environment.

**NOTE**

You may also specify the Automatically Start Builds option on the Options menu of either the NightBench Main or the Ada Environment window for this behavior. See "Automatically Start Builds" on page 3-3 or "Automatically Start Builds" on page 4-9.

## 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.

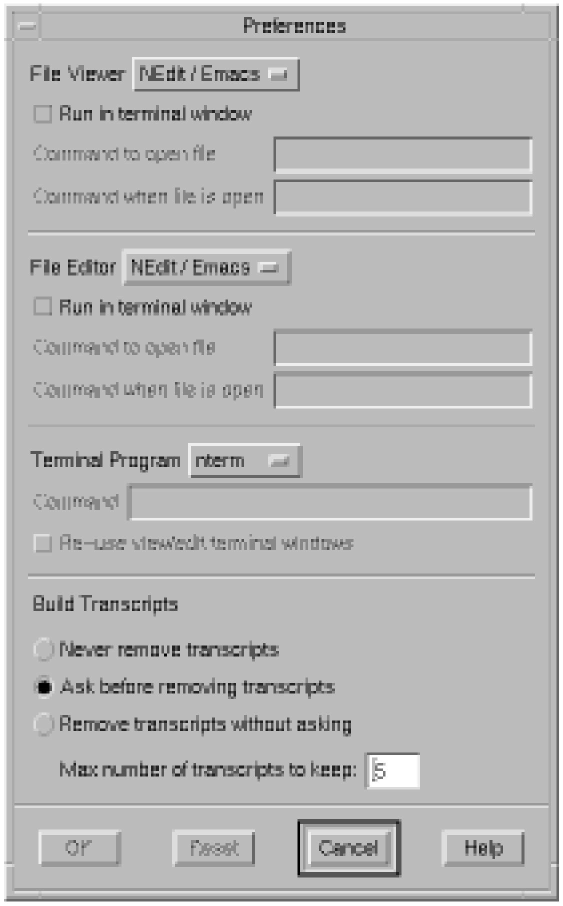
See also:

- “Preferences” on page 1-7
- “Resizing columns” on page 1-12

## 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 the removal of previous build transcripts are presented in this dialog.

After the Preferences have been saved, you must exit the current NightBench session and restart it in order for the new settings to take effect.


Preferences

File Viewer: NEdit / Emacs
☐ Run in terminal window
Command to open file
Command when file is open

File Editor: NEdit / Emacs
☐ Run in terminal window
Command to open file
Command when file is open

Terminal Program: nterm
Command
☐ Re-use view/edit terminal windows

Build Transcripts
○ Never remove transcripts
● Ask before removing transcripts
○ Remove transcripts without asking
Max number of transcripts to keep: 5

**Figure 1-1. Preferences dialog**

### File Viewer

Specifies which program should be used to view files.

| | |
|---|---|
| NEdit / Emacs | 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.<br><br>See "Using the Emacs Editor - Additional Considerations" on page 1-11 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 File Viewer menu.

When the more option is chosen from the File 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 / Emacs option is selected from the File 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.

#### Command when file is open

This field is active only when the Custom menu item is selected.

### File Editor

Specifies which program should be used to edit files.

| | |
|---|---|
| NEdit / Emacs | 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.<br><br>See “Using the Emacs Editor - Additional Considerations” on page 1-11 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 File Editor menu.

When the vi option is chosen from the File 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 / Emacs option is selected from the File 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.

#### Command when file is open

This field is active only when the Custom menu item is selected.

### Terminal Program

Specifies which program should be used as a terminal emulator.

| | |
|---|---|
| nterm | Uses the **`nterm(1)`** terminal emulator. |
| xterm | Uses the **`xterm(1)`** terminal emulator. |
| Custom | 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.

### Build 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 6-5

#### 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.

## 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

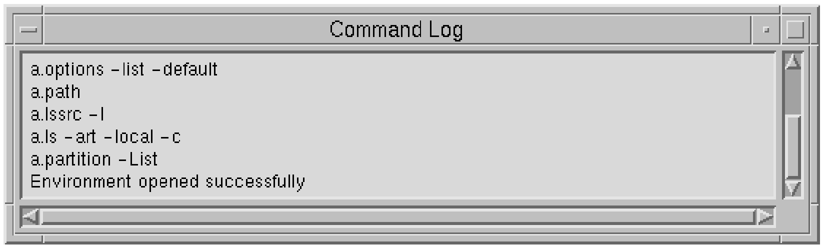
NightBench allows the user to resize columns in the various lists that appear throughout the interface.

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 Ada Environment window, drag the divider that appears between the `Name` and `Part` column titles to the right until the desired width is achieved.

## Command Log

The NightBench Program Development Environment provides a history window from both the Ada Environment window and the NightBench Builder window.  This command log contains a list of all the MAXAda commands executed by NightBench in the carrying out of its tasks.



```
a.options -list -default
a.path
a.lssrc -l
a.ls -art -local -c
a.partition -List
Environment opened successfully
```

**Figure 1-2.  Command Log window**

See also:

- *MAXAda Reference Manual* (0890516)
- Ada Environment - Chapter 4
- NightBench Builder - Chapter 6

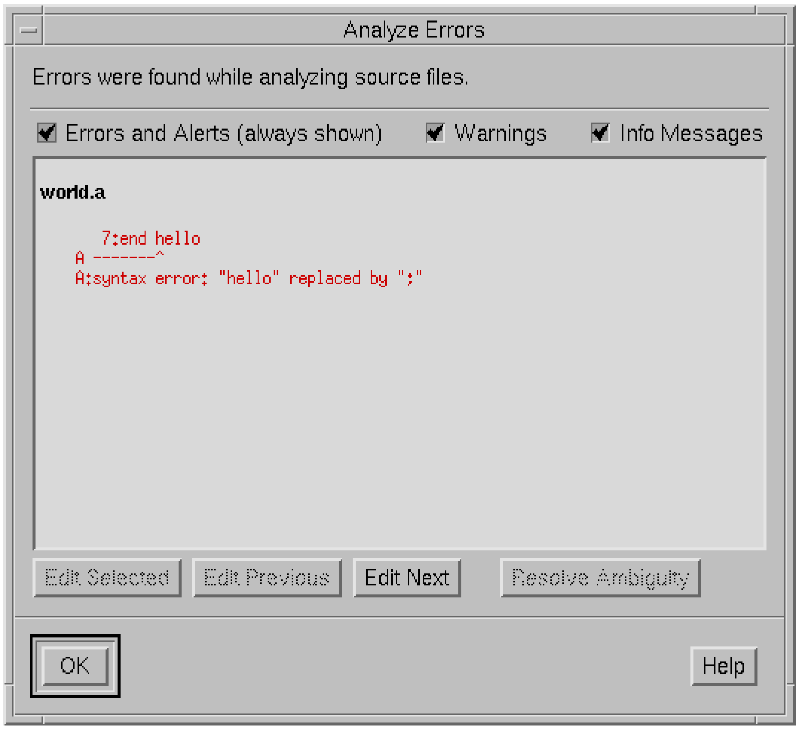
## Analyze Errors

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 subsequenltly 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.



```
world.a

    7:end hello
A -------^
A:syntax error: "hello" replaced by ";"
```

**Figure 1-3. 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" on page 1-7

### 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" on page 1-7

### 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" on page 1-7

### 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 4-51

## 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.

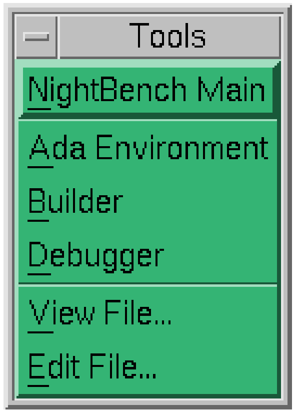


**Figure 1-4. Tools menu**

### NightBench Main

Opens the NightBench Main window.

(This menu item does not appear on the Tools menu of the NightBench Main window.)

See also:

- NightBench Main - Chapter 3

### Ada Environment

Opens the Ada Environment window.

See also:

- Ada Environment - Chapter 4

### Builder

Opens the NightBench Builder window.

See also:

- NightBench Builder - Chapter 6

### 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” on page 1-7

### 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” on page 1-7

## Help Menu

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

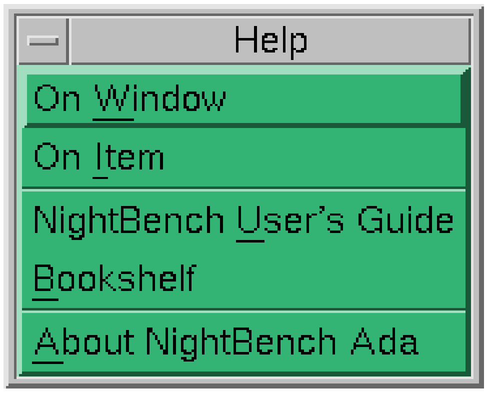


**Figure 1-5. Help menu**

### 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.

# 2 Using NightBench

## Hello World - An Example

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 Main 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 Main 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.

The NightBench Main 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 also:

- "Starting NightBench" on page 1-3

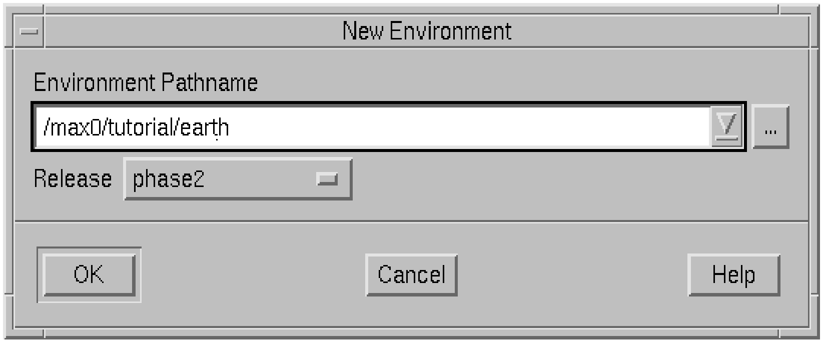
- NightBench Main - Chapter 3

## 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 Main window:

- On the NightBench Main window, press the button marked `New...` This will open the New Environment dialog so we can create our new environment.
  - Type the directory name in the `Environment Pathname` 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 **`/max0/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 sytem, it will appear here.
  - Press `OK`.



Environment Pathname: /max0/tutorial/earth
Release: phase2

**Figure 2-1. Creating the `earth` environment**

This will add the new environment to the list of `Environments` in the NightBench Main window. NightBench will also open the new environment in its own Ada Environment window.

See also:

- "New Environment" on page 3-6
- Ada Environment - Chapter 4

## 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 ;
```

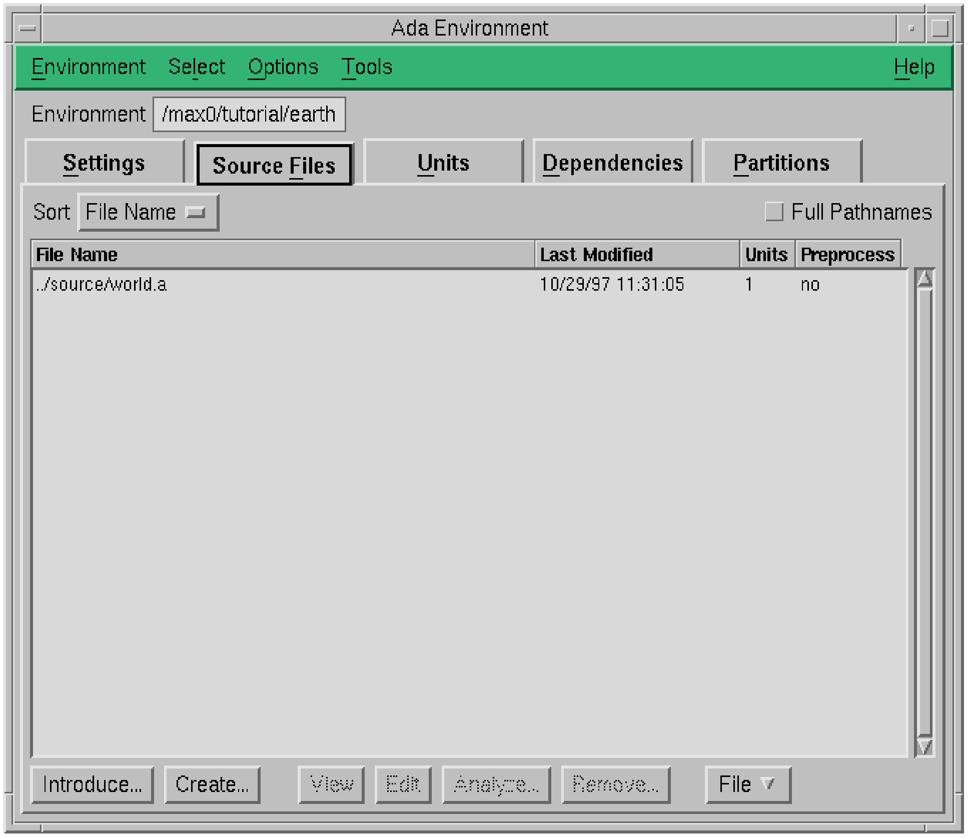
**Screen 2-2. Source file world.a containing hello unit**

**To introduce an existing source file into an environment:**

- Click on the Source Files tab of the Ada Environment window to get to the Source Files page.
- Press the Introduce… 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.
  - Manuever 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 Ada Environment window and the unit contained within now appears on the Units page of the Ada Environment window.



Environment /max0/tutorial/earth

Sort File Name

| File Name | Last Modified | Units | Preprocess |
|---|---|---|---|
| ../source/world.a | 10/29/97 11:31:05 | 1 | no |

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

See also:

- "Source Files" on page 4-23
- "Introduce Source Files" on page 4-28

## Defining a partition

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

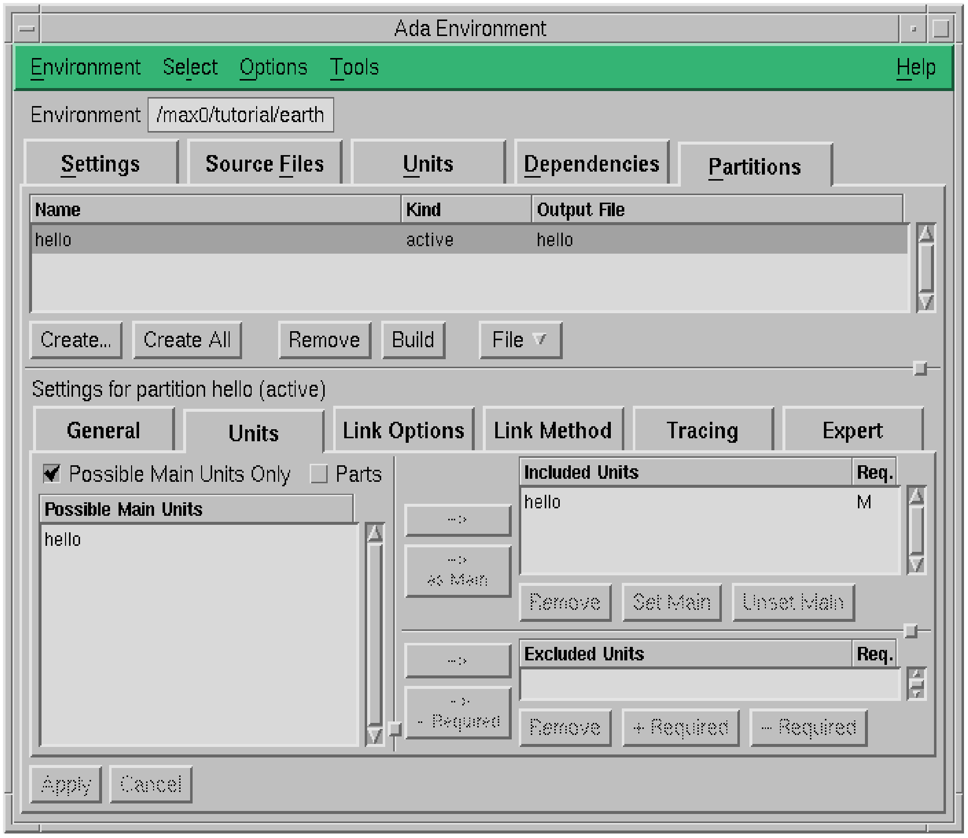
- "Create Partition" on page 4-65

## 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 Settings - Units section 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 2-3:



Environment /max0/tutorial/earth

| Name | Kind | Output File |
|---|---|---|
| hello | active | hello |

Settings for partition hello (active)

[✓] Possible Main Units Only [ ] Parts

| Possible Main Units |
|---|
| hello |

| Included Units | Req. |
|---|---|
| hello | M |

| Excluded Units | Req. |
|---|---|

**Figure 2-3. Including units in a newly defined partition**

As you can see in Figure 2-3, the unit `hello` has been automatically added to the list of `Included Units` for the **`hello`** partition. When the partition has the same name as a library subprogram in the environment, that subprogram is assumed to be the main unit. Otherwise, no main subprogram is assumed and must be manually included.

See also:

- "Settings - Units" on page 4-75

## 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:

- On the Partitions page, press the button marked `Build`. This will open the NightBench Builder window so we can build our new partition.

In Figure 2-4, 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.

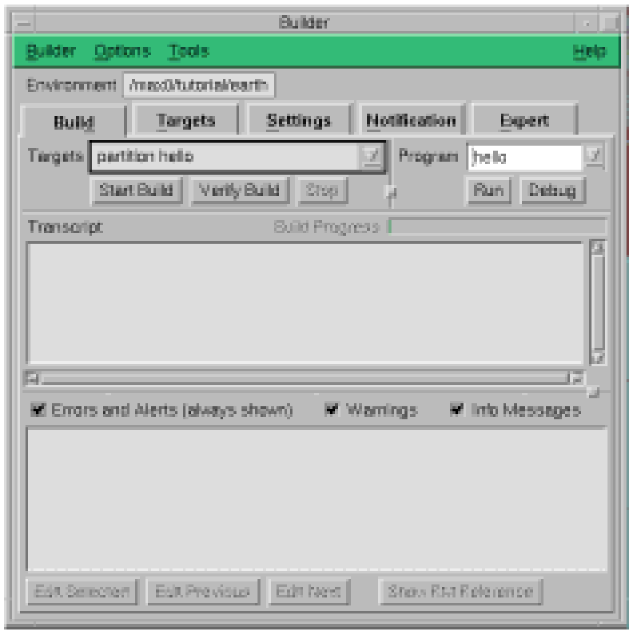

Builder
Builder Options Tools Help
Environment /mac0/tutorial/earth
Build Targets Settings Notification Expert
Targets partition hello
Program hello
Start Build Verify Build Stop
Run Debug
Transcript
Build Progress
Errors and Alerts (always shown)
Warnings
Info Messages
Edit Selected Edit Previous Edit Next Show RM Reference

**Figure 2-4. Builder window - Build page for `hello` partition**

When the build is completed, a “Build Completed” dialog notifies the user. The notification operations can be changed on the Notification page

See also:

- NightBench Builder - Chapter 6
- “Build” on page 6-9
- “Notification” on page 6-18

## 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, press the button marked `Run`.

  This will open a terminal window so our executable can run. (This is configured by bringing up the Preferences dialog from the Options Menu menu.) You may close the window when you are finished viewing the output.

  You should see the following output:

  ```
  [running hello]

  Hello World!!!

  [program exited with status 0]
  ```

See also:

- “Preferences” on page 1-7
- NightBench Builder - Chapter 6
- “Build” on page 6-9

## 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 Ada Environment window to get to the Settings page.

  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.
  - Click on the General tab to get to the General page.
  - Change Value 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 5
- "Environment Compile Options Editor" on page 5-3

## 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 Ada Environment window to get to the Units page.
- Select the unit `hello` from the list.

  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 Unit Compile Options Editor.
  - Click on the General tab to get to the 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 checkmark appears in the box. (See "Tri-state Checkboxes" on page 5-1 for more information.)
  - Press Apply so that the changes take effect but the 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 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.

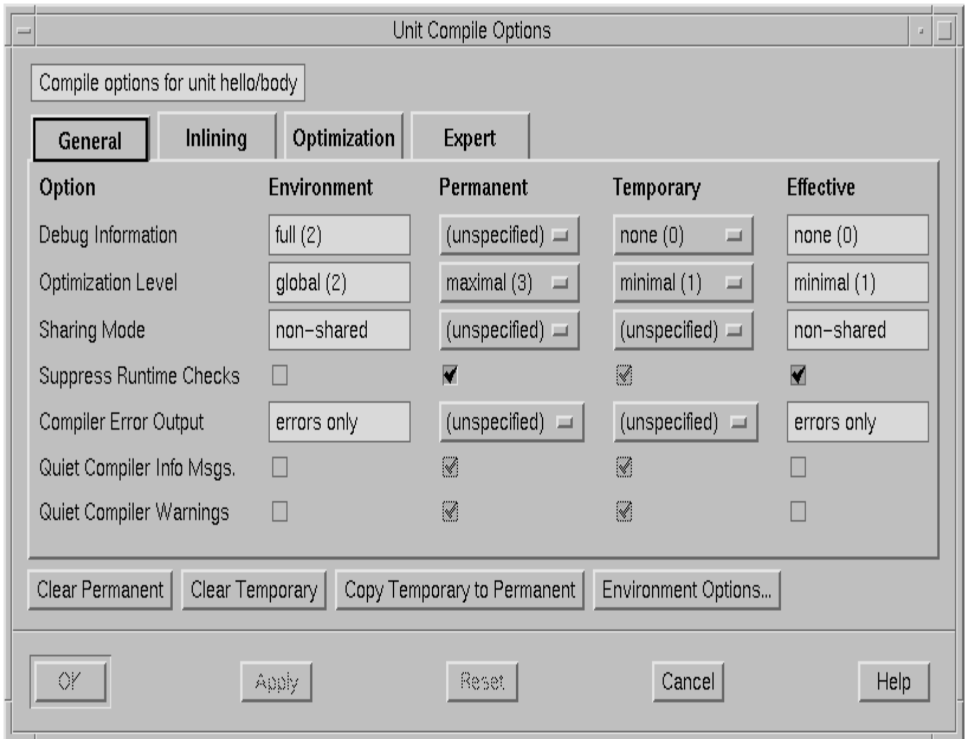
- Change the Temporary value of Optimization Label to minimal (1) (using the drop-down list).
- Press Apply so that the changes take effect but the Unit Compile Options Editor remains open.

See also:

- Options Editors - Chapter 5
- "Unit Compile Options Editor" on page 5-13

## 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.



| Option | Environment | Permanent | Temporary | Effective |
|---|---|---|---|---|
| Debug Information | full (2) | (unspecified) | none (0) | none (0) |
| Optimization Level | global (2) | maximal (3) | minimal (1) | minimal (1) |
| Sharing Mode | non-shared | (unspecified) | (unspecified) | non-shared |
| Suppress Runtime Checks | ☐ | ✓ | ✓ | ✓ |
| Compiler Error Output | errors only | (unspecified) | (unspecified) | errors only |
| Quiet Compiler Info Msgs. | ☐ | ✓ | ✓ | ☐ |
| Quiet Compiler Warnings | ☐ | ✓ | ✓ | ☐ |

**Figure 2-5. Unit Compile Options dialog - Effective Options**

As you can see in Figure 2-5, 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 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 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 5

- "Unit Compile Options Editor" on page 5-13

## 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 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 Unit Compile Options Editor.

See also:

- Options Editors - Chapter 5
- "Unit Compile Options Editor" on page 5-13

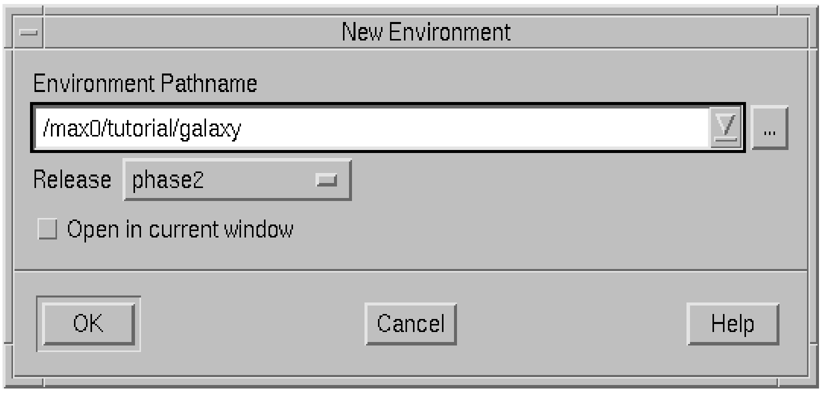
# Hello Galaxy - The Example Continues...

## 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 Main window. This time, we will create it from the Ada Environment window.

**To create a new environment from the Ada Environment window:**

- From the `Environment` menu menu of the Ada Environment window, select the item marked `New...` This will open the New Environment dialog so we can create our new environment.
  - Type the directory name in the `Environment Pathname` 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 **`/max0/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 sytem, it will appear here.
  - This dialog also gives the user the choice to open the new environment in the current Ada Environment window. For our example, we uncheck the `Open in current window` option so that our new environment will be opened in its own Ada Environment window.
  - Press `OK`.


New Environment

Environment Pathname: /max0/tutorial/galaxy

Release: phase2

☐ Open in current window

**Figure 2-6. Creating the `galaxy` environment**

This will add the new environment to the list of Environments in the NightBench Main window. NightBench will also open the new environment in its own Ada Environment window.

See also:

- "New Environment" on page 4-4
- Ada Environment - Chapter 4

## 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 Ada Environment window to get to the Source Files page.
- Press the Create… button. This will open the Create Source File dialog. This is the standard file dialog.
  - Type in the name of the source file in the File to Create field. We will name this source file **planet.a**.
  - Press OK.
- The program configured as the file editor for NightBench (under the Options Menu menu item Preferences…) 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**

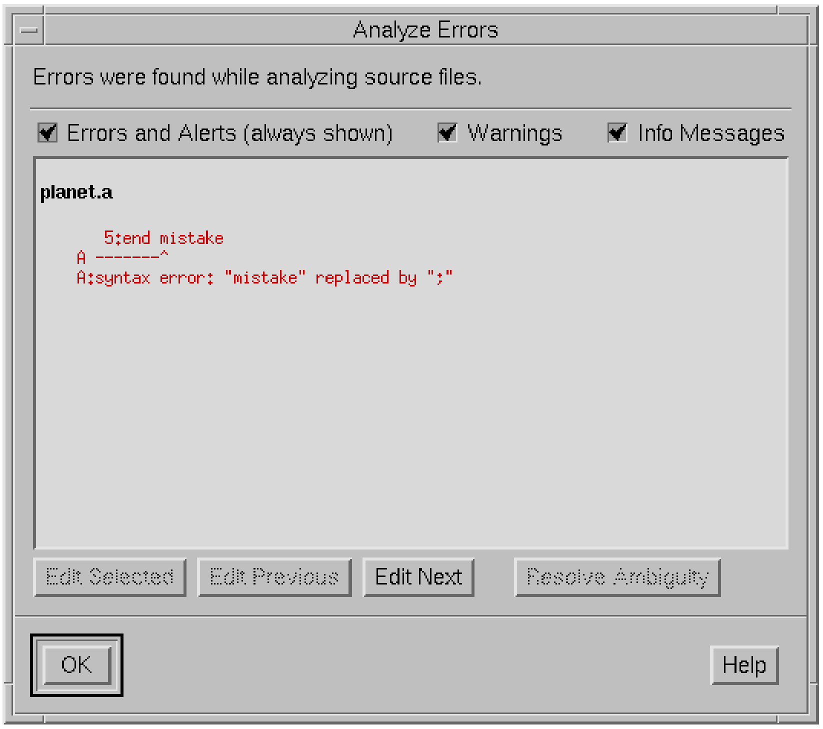
- Close the file in the editor or exit the editor completely.
- The Analyze Errors dialog is presented showing the error on the last line of **planet.a**. NightBench analyzes any file that is introduced into an envi-

ronment 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.



```
planet.a

      5:end mistake
    A -------^
    A:syntax error: "mistake" replaced by ";"
```

**Figure 2-7. 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.
- 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`** is now a part of the environment **`galaxy`**.

The source file now appears in the list of files on the Source Files page of the Ada Environment window and the unit contained within now appears on the Units page of the Ada Environment 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 1-7
- "Analyze Errors" on page 1-14

## Modifying an existing unit

Let's go back to the Ada Environment 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 Ada Environment window that contains the **`earth`** environment to get to the Units page.
- 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.
- Make the following changes to the `hello` unit.

```
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:

- “Units” on page 4-39
- “Preferences” on page 1-7

## 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:**

- On the Partitions page of the **`earth`** environment, 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.

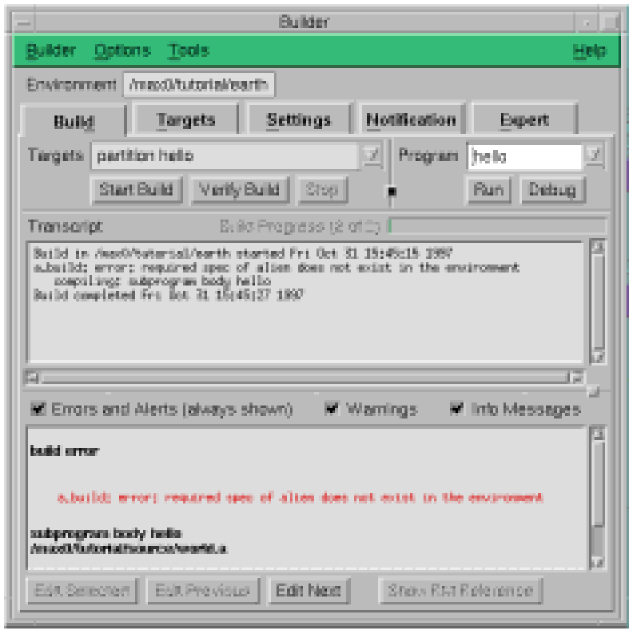
- Press `Start Build`.

  When the build is completed, a “Build Completed” dialog notifies the user. The notification operations can be changed on the Notification page.

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

  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
```


Builder
Builder Options Tools Help
Environment /mxo0/tutorial/earth
Build Targets Settings Notification Expert
Targets partition hello
Program hello
Start Build Verify Build Stop
Run Debug
Transcript
Build Progress (2 of 2)
Errors and Alerts (always shown) Warnings Info Messages
Edit Selection Edit Previous Edit Next Show Rst Reference

```
Build in /mxo0/tutorial/earth started Fri Oct 31 15:45:15 1997
a.build: error: required spec of alien does not exist in the environment
   compiling: subprogram body hello
Build completed Fri Oct 31 15:45:27 1997
```

```
build error

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

subprogram body hello
/mxo0/tutorial/source/earth.a
```

**Figure 2-8. Builder window - Build page with errors for `hello` partition**

See also:

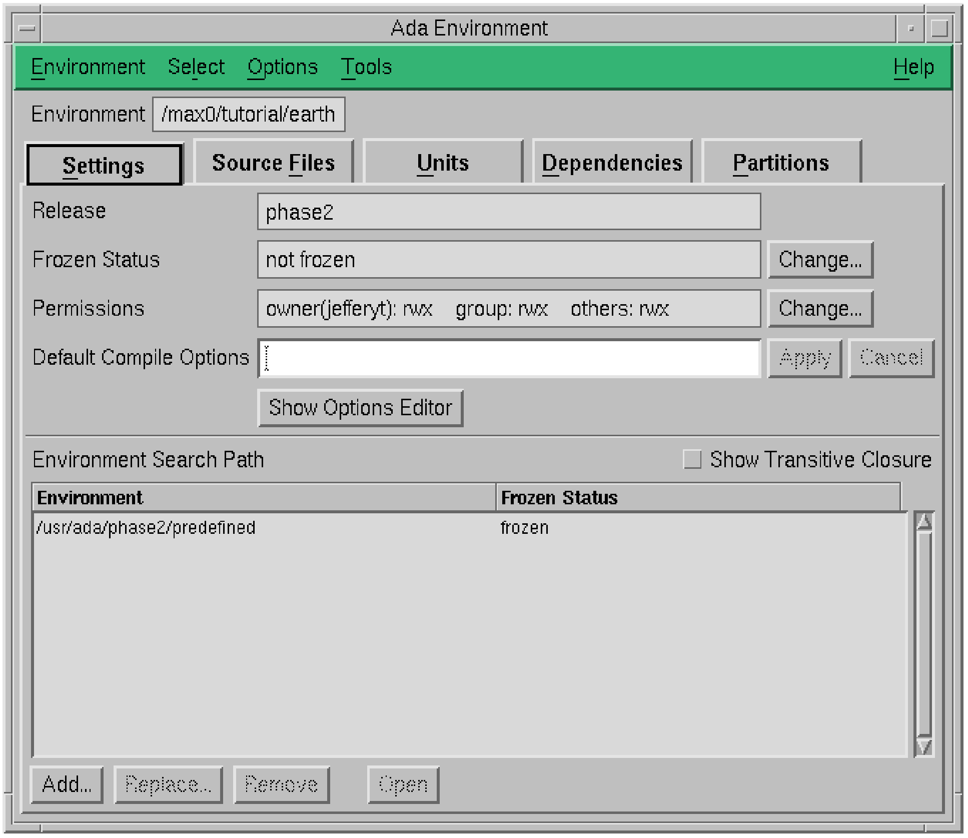
- "Partitions" on page 4-62
- NightBench Builder - Chapter 6
- "Build" on page 6-9
- "Notification" on page 6-18

## Adding an environment to the Environment Search Path

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

**To add an environment to the Environment Search Path:**

- Click on the `Settings` tab of the Ada Environment window that contains the **`earth`** environment to get to the Settings page.


Ada Environment
Environment Select Options Tools Help

Environment /max0/tutorial/earth

Settings Source Files Units Dependencies Partitions

| | |
|---|---|
| Release | phase2 |
| Frozen Status | not frozen |
| Permissions | owner(jefferyt): rwx group: rwx others: rwx |
| Default Compile Options | |

Change...
Change...
Apply Cancel
Show Options Editor
Environment Search Path
Show Transitive Closure

| Environment | Frozen Status |
|---|---|
| /usr/ada/phase2/predefined | frozen |

Add... Replace... Remove Open

**Figure 2-9. 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 your 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!

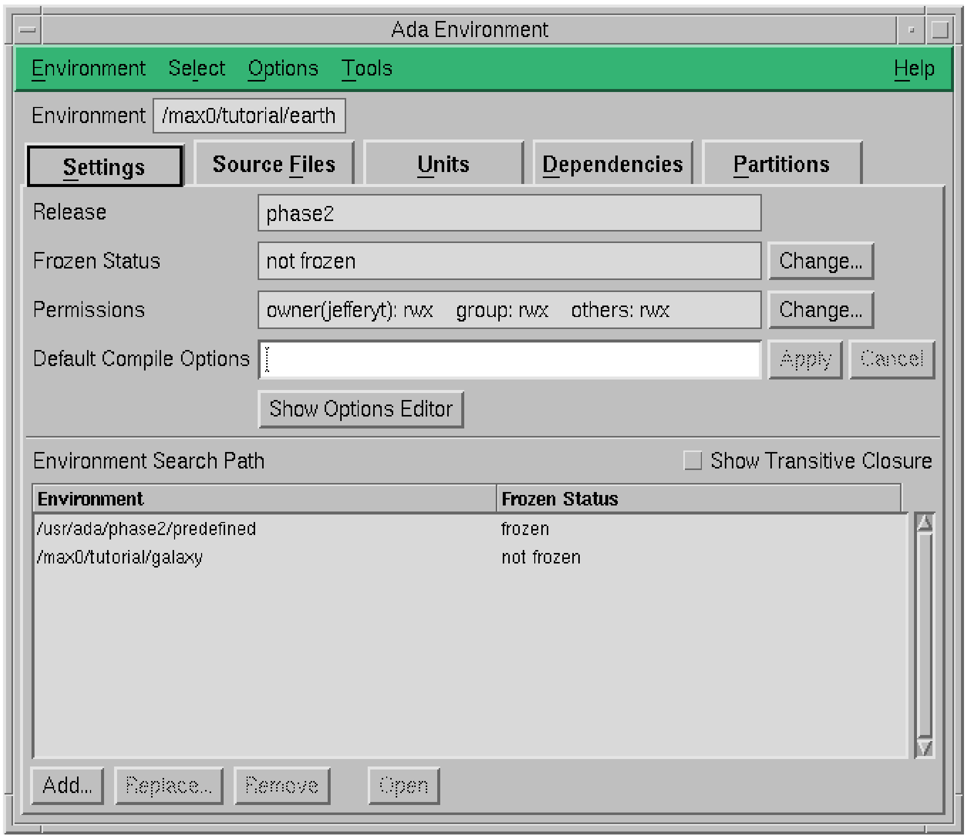
And that is precisely how we will make the `alien` unit visible to our `hello` unit.

- Press the `Add…` button. This brings up the Add Environment Search Path Element dialog.

- 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 **`/max0/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.



Environment: /max0/tutorial/earth

Release: phase2

Frozen Status: not frozen

Permissions: owner(jefferyt): rwx group: rwx others: rwx

Default Compile Options:

Environment Search Path

| Environment | Frozen Status |
|---|---|
| /usr/ada/phase2/predefined | frozen |
| /max0/tutorial/galaxy | not frozen |

**Figure 2-10. Updated Environment Search Path for the `earth` environment**

See also:

- “Add Environment Search Path Element” on page 4-20
- “Settings” on page 4-13
- *Ada 95 Reference Manual*

## Making contact!!!

Let’s try to build it again.

**Building a partition with references on the Environment Search Path:**

- On the Partitions page of the **`earth`** environment, 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`.

  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. The notification operations can be changed on the Notification page.

**To run the program:**

- On the Build page of the NightBench Builder window, press the button marked `Run`.

  This will open a terminal window so our executable can run. (This is configured by bringing up the Preferences dialog from the Options Menu menu.) You may close the window when you are finished viewing the output.

  You should see the following output:

  ```
  [running hello]

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

  [program exited with status 0]
  ```

See also:

- “Partitions” on page 4-62
- NightBench Builder - Chapter 6
- “Build” on page 6-9
- “Notification” on page 6-18
- “Preferences” on page 1-7

## Who resides on earth?

If we look at the Units page of the Ada Environment window, we can see that there are two units in the environment, `alien` and `hello`.

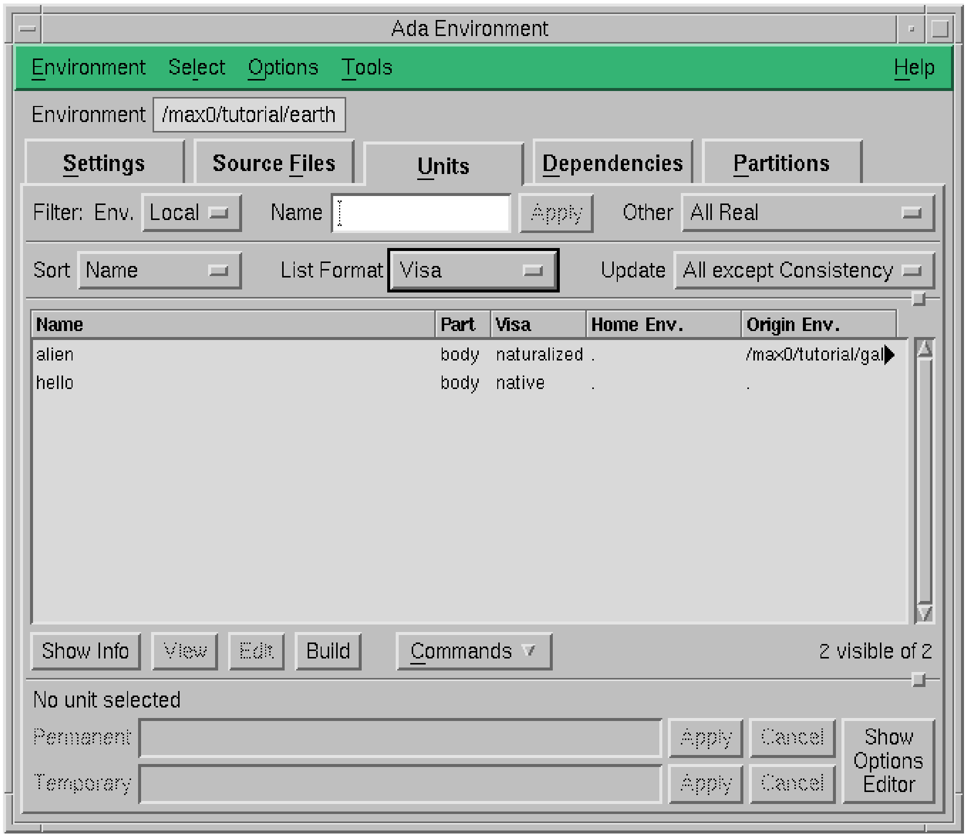
The Units page offers different formats to view these units. The format of the list of units can be changed by choosing a different format from the List Format list.

Let's look at where these units came from:

### Listing the visa for each unit in our environment:

- Click on the Units tab of the Ada Environment window that contains the **`earth`** environment to get to the Units page.
- Choose Visa from the List Format drop-down list.

  You will see the following:



Environment /max0/tutorial/earth

Filter: Env. Local | Name | Other All Real

Sort Name | List Format Visa | Update All except Consistency

| Name | Part | Visa | Home Env. | Origin Env. |
|---|---|---|---|---|
| alien | body | naturalized | . | /max0/tutorial/gal |
| hello | body | native | . | . |

2 visible of 2

No unit selected

Permanent

Temporary

**Figure 2-11. 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, 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:

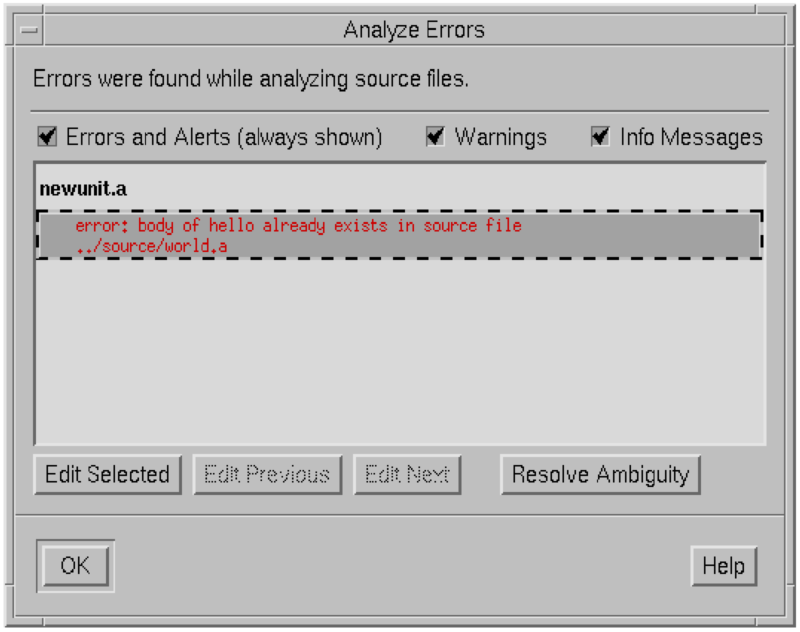
- Click on the `Source Files` tab of the Ada Environment window for the **`earth`** environment to get to the Source Files page.
- Press the `Create…` button. This will open the Create Source File dialog. This is the standard file dialog.
  - Type in the name of the source file in the `File to Create` field. We will name this source file **`newunit.a`**.
  - Press `OK`.
- The program configured as the file editor for NightBench (under the Options Menu menu item `Preferences…`) 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. NightBench analyzes any file that is introduced into an 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.



newunit.a

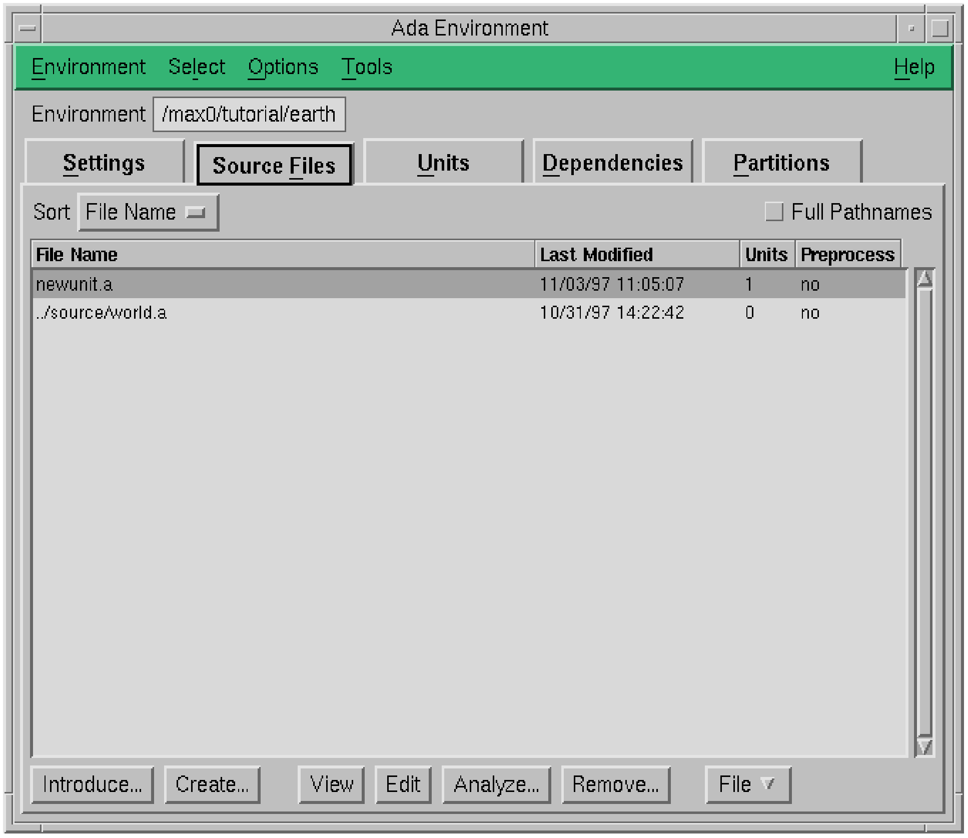
```
error: body of hello already exists in source file
../source/world.a
```

**Figure 2-12. 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.

- 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.
  - Click on 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.


Ada Environment
Environment Select Options Tools Help
Environment /max0/tutorial/earth
Settings Source Files Units Dependencies Partitions
Sort File Name
Full Pathnames
Introduce... Create... View Edit Analyze... Remove... File

| File Name | Last Modified | Units | Preprocess |
|---|---|---|---|
| newunit.a | 11/03/97 11:05:07 | 1 | no |
| ../source/world.a | 10/31/97 14:22:42 | 0 | no |

**Figure 2-13. 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:

- "Source Files" on page 4-23
- "Analyze Errors" on page 1-14
- "Resolve Ambiguity" on page 4-51
- "Hidden Units" on page 4-49

## 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, 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. The notification operations can be changed on the Notification page.

**To run the program:**

- On the Build page of the NightBench Builder window, press the button marked `Run`.

  This will open a terminal window so our executable can run. (This is configured by bringing up the Preferences dialog from the Options Menu menu.) You may close the window when you are finished viewing the output.

  You should see the following output:

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

**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.

See also:

- “Partitions” on page 4-62
- NightBench Builder - Chapter 6
- “Build” on page 6-9
- “Notification” on page 6-18
- “Preferences” on page 1-7

## Using NightBench - Conclusion

This concludes the 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. 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.

See also:

- “Help Menu” on page 1-19

# 3
# NightBench Main

## Overview

NightBench Main 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 Ada Environment window or the NightBench Builder.

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

See also:

- Ada Environment - Chapter 4
- NightBench Builder - Chapter 6

## NightBench Main Menu Bar

The NightBench Main menu bar consists of the following items:

- "NightBench Menu" on page 3-2
- "Options Menu" on page 3-3
- "Tools Menu" on page 1-17
- "Help Menu" on page 1-19

## NightBench Menu

The NightBench menu appears on the NightBench Main menu bar.

See also:

- “NightBench Main Menu Bar” on page 3-1

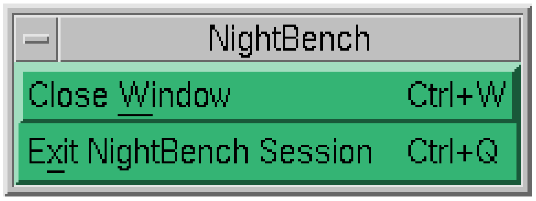


**Figure 3-1. NightBench menu**

### Close Window

Closes the NightBench Main 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 utilites.

## Options Menu

The Options menu appears on the NightBench Main menu bar.

See also:

- “NightBench Main Menu Bar” on page 3-1

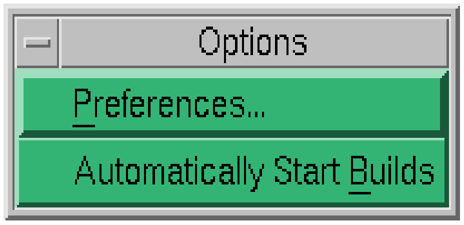


**Figure 3-2. Options menu**

### 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 1-7

### 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.

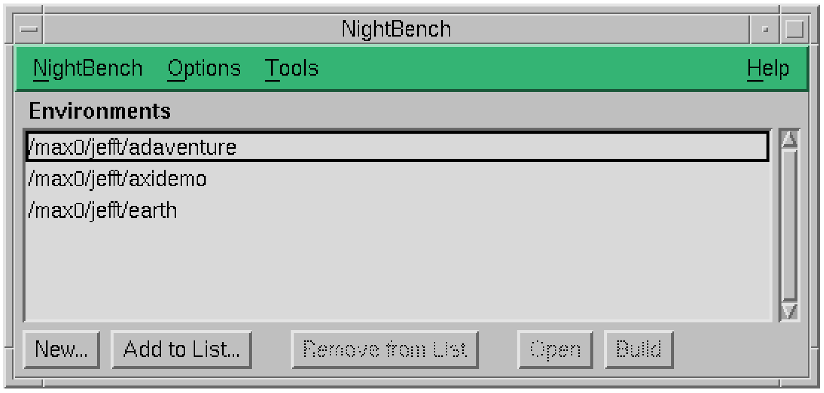
See also:

- “Units - Build” on 4-43
- “Partitions - Build” on 4-63
- NightBench Builder - Chapter 6

## NightBench Main

NightBench Main 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 Ada Environment window or the NightBench Builder.

In addition, other functionality can be found on the NightBench Main menu bar (see page 3-1).



/max0/jefft/adaventure
/max0/jefft/axidemo
/max0/jefft/earth

**Figure 3-3. NightBench Main window**

### Environments

Lists the environments previously used by NightBench.

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

See also:

- Ada Environment - Chapter 4

### New...

Creates a new NightBench environment, adds the environment name to the list in the NightBench Main window, and opens the new environment in an Ada Environment 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 3-6
- Ada Environment - Chapter 4

### Add to List...

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

See also:

- "Add Environment to List" on page 3-7

### Remove from List

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

### Open

Open the selected NightBench environment in an Ada Environment window.

See also:

- Ada Environment - Chapter 4

### Build

Open the NightBench Builder window for the selected NightBench environment.

See also:

- NightBench Builder - Chapter 6

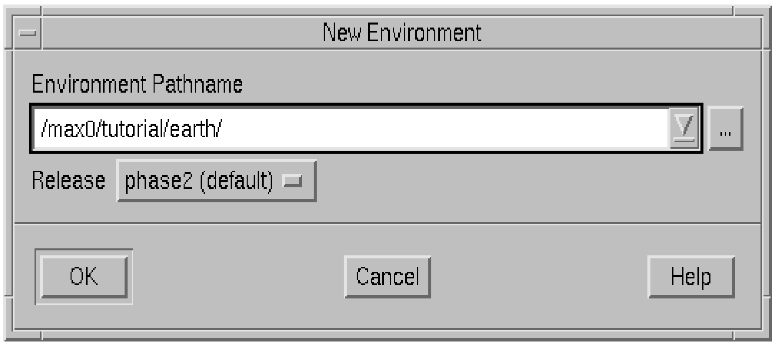
## New Environment

Creates a new NightBench environment, adds the environment name to the list in the NightBench Main window, and opens the new environment in an Ada Environment 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:

- Ada Environment - Chapter 4
- "NightBench Main" on page 3-4



Environment Pathname

/max0/tutorial/earth/

Release phase2 (default)

**Figure 3-4. New Environment dialog**

### Environment Pathname

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

### Release

The MAXAda release associated with the current environment. If more than one release of MAXAda is installed on the system, their names will appear in the popup list for this field.

See also:

- *MAXAda Reference Manual* (0890516)

## Add Environment to List

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

See also:

- Ada Environment - Chapter 4
- "NightBench Main" on page 3-4

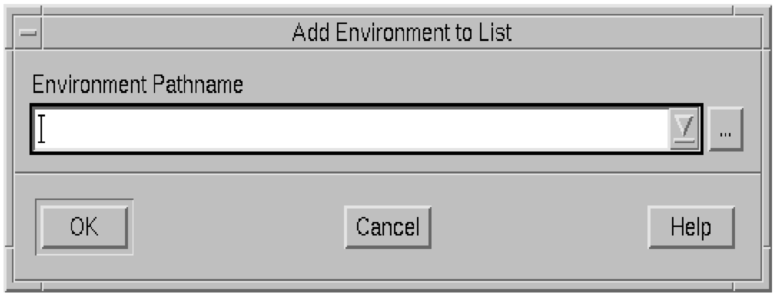


**Figure 3-5.  Add Environment to List dialog**

### Environment Pathname

The environment which is to be added to the list of environements in the Night-Bench Main window is specified in this field, either by manually entering the path-name or by using the file selection dialog (…) to navigate to it.

# 4
# Ada Environment

## Overview

The Ada Environment 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 Ada Environment window is composed of the following five pages:

- Settings (see page 4-13)
- Source Files (see page 4-23)
- Units (see page 4-39)
- Dependencies (see page 4-59)
- Partitions (see page 4-62)

At the top of the Ada Environment window is listed:

#### Environment

The name of the environment currently being operated upon in the Ada Environment window.

In addition, other functionality can be found on the Ada Environment menu bar (see page 4-3).

## Ada Environment Menu Bar

The Ada Environment menu bar consists of the following items:

- "Environment Menu" on page 4-3
- "Select Menu" on page 4-8
- "Options Menu" on page 4-9
- "Tools Menu" on page 1-17
- "Help Menu" on page 1-19

## Environment Menu

The Environment menu appears on the Ada Environment menu bar.

See also:

- "Ada Environment Menu Bar" on page 4-3

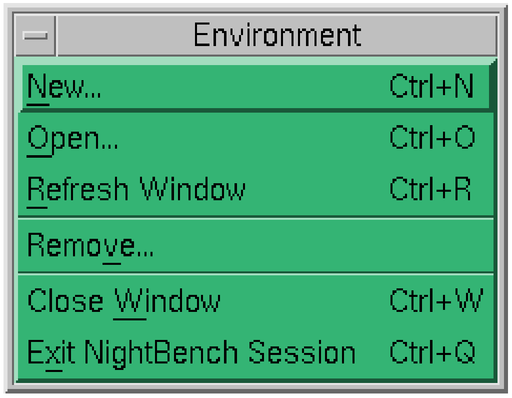


**Figure 4-1. Environment menu**

### New...

Creates a new NightBench environment.

See also:

- "New Environment" on page 4-4

### Open...

Opens an existing NightBench environment.

See also:

- "Open Environment" on page 4-5

### Refresh Window

Updates all the information in the Ada Environment 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).

### Remove...

Removes an existing NightBench environment.

See also:

- "Remove Environment" on page 4-7

### Close Window

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

### Exit NightBench Session

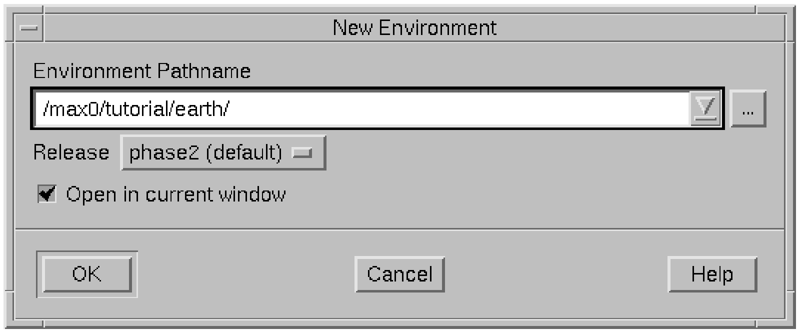
Closes all of the NightBench windows and shuts down all NightBench servers and internal utilites.

## New Environment

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

See also:

- "Overview" on page 4-1
- "Environment Menu" on page 4-3



Environment Pathname

/max0/tutorial/earth/

Release phase2 (default)

✓ Open in current window

**Figure 4-2. New Environment dialog**

### Environment Pathname

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

### 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.

### Open in current window

If this option is checked, the new environment will be displayed in the current Ada Environment window. If this option is not checked, a new Ada Environment 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 Ada Environment Window or a new Ada Environment window.

See also:

- “Overview” on page 4-1
- “Environment Menu” on page 4-3

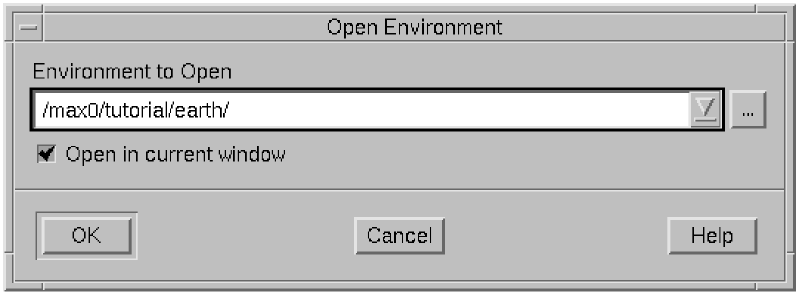


**Figure 4-3. Open Environment dialog**

### Environment to Open

The environment which is 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.

### Open in current window

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

## 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:

- “Overview” on page 4-1
- “Environment Menu” on page 4-3

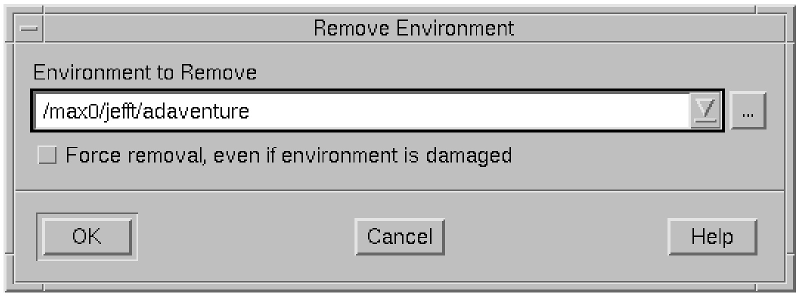


**Figure 4-4. Remove Environment dialog**

### Environment to Remove

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.

### Force removal, even if environment is damaged

This option is used to force an environment’s destruction, even if some portion of it does not exist. For example, if the creation of the environment was interrupted (due to not enough disk space, power failure, etc.), the environment may not have been successfully created. Trying to remove an environment like this may result in an error. This option forces the environment to be removed, overriding any such error situations.

## Select Menu

The Select menu appears on the Ada Environment menu bar.

See also:

- "Ada Environment Menu Bar" on page 4-3

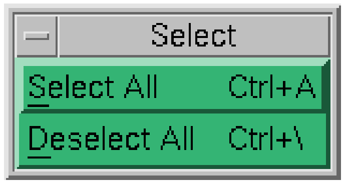


**Figure 4-5. Select menu**

### Select All

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

## Options Menu

The Options menu appears on the Ada Environment menu bar.

See also:

- "Ada Environment Menu Bar" on page 4-3

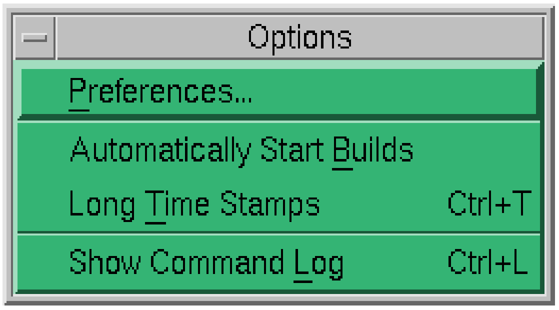


**Figure 4-6. Options menu**

### 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 1-7

### 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.

See also:

- "Units - Build" on 4-43
- "Partitions - Build" on 4-63
- NightBench Builder - Chapter 6

### 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 Ada Environment window was opened.

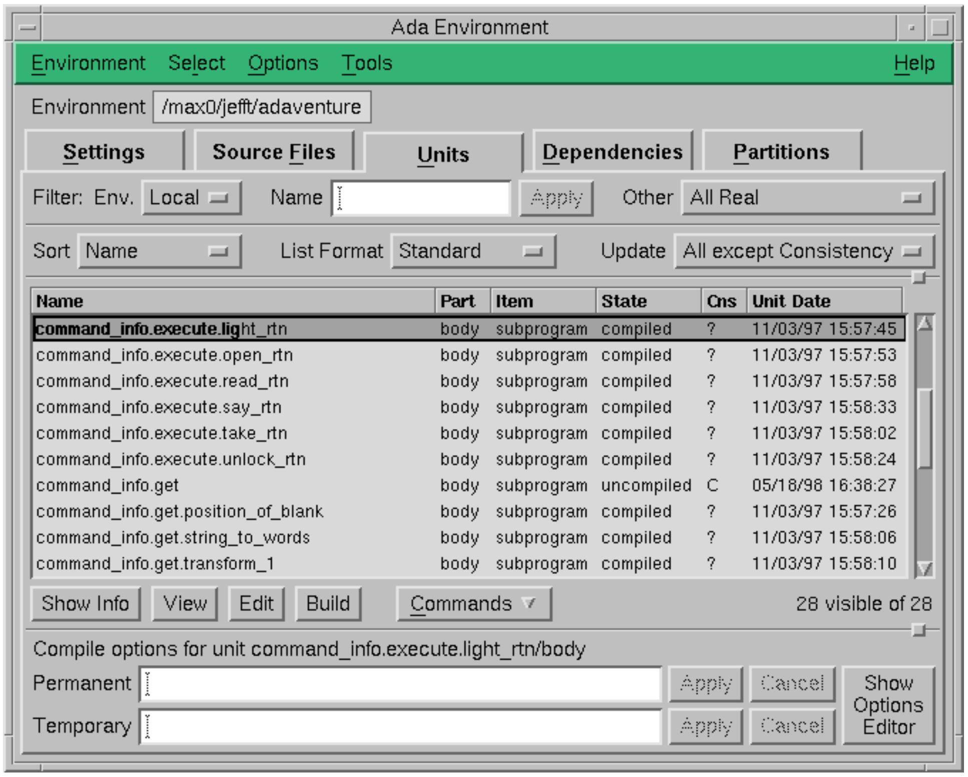
See also:

- *MAXAda Reference Manual* (0890516)
- “Command Log” on page 1-13

## Accelerated Item Selection

On each page of the Ada Environment 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 4-7. 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.



Environment /max0/jefft/adaventure

Filter: Env. Local Name Other All Real

Sort Name List Format Standard Update All except Consistency

| Name | Part | Item | State | Cns | Unit Date |
|---|---|---|---|---|---|
| **command_info.execute.lig**ht_rtn | body | subprogram | compiled | ? | 11/03/97 15:57:45 |
| command_info.execute.open_rtn | body | subprogram | compiled | ? | 11/03/97 15:57:53 |
| command_info.execute.read_rtn | body | subprogram | compiled | ? | 11/03/97 15:57:58 |
| command_info.execute.say_rtn | body | subprogram | compiled | ? | 11/03/97 15:58:33 |
| command_info.execute.take_rtn | body | subprogram | compiled | ? | 11/03/97 15:58:02 |
| command_info.execute.unlock_rtn | body | subprogram | compiled | ? | 11/03/97 15:58:24 |
| command_info.get | body | subprogram | uncompiled | C | 05/18/98 16:38:27 |
| command_info.get.position_of_blank | body | subprogram | compiled | ? | 11/03/97 15:57:26 |
| command_info.get.string_to_words | body | subprogram | compiled | ? | 11/03/97 15:58:06 |
| command_info.get.transform_1 | body | subprogram | compiled | ? | 11/03/97 15:58:10 |

28 visible of 28

Compile options for unit command_info.execute.light_rtn/body

Permanent

Temporary

**Figure 4-7. 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.

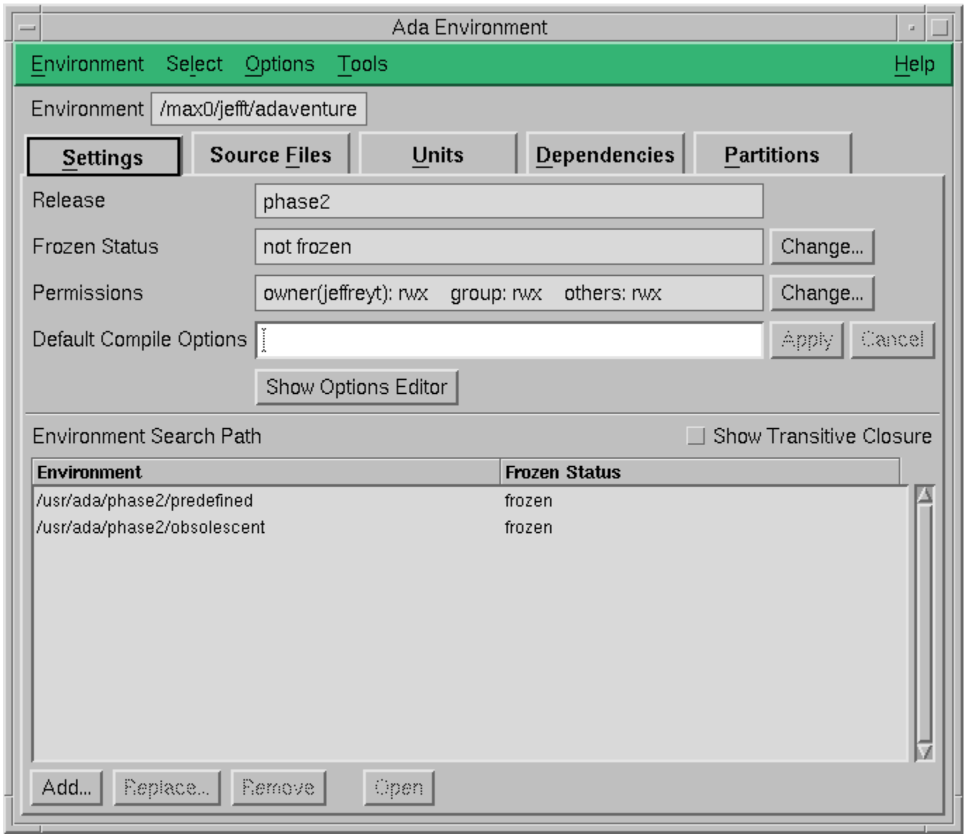
If the user then extends the selection by typing a "`g`" and then an "`r`", the selection will change to the line containing the item "`mongrel`" and "**`mongr`**" will be highlighted. If the user then presses the space bar, the set of matching letters will be extended to "**`mon-grel`**" because it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

## 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:

- “Overview” on page 4-1



Environment /max0/jefft/adaventure

Release: phase2

Frozen Status: not frozen

Permissions: owner(jeffreyt): rwx group: rwx others: rwx

Default Compile Options:

Environment Search Path

| Environment | Frozen Status |
|---|---|
| /usr/ada/phase2/predefined | frozen |
| /usr/ada/phase2/obsolescent | frozen |

**Figure 4-8. Ada Environment window - Settings page**

### 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)

## 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 4-17

## 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 of 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 4-19

## Default Compile Options

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.

See also:

- "Environment Compile Options Editor" on page 5-3
- *MAXAda Reference Manual* (0890516)

#### Apply

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

See also:

- "Default Compile Options" on page 4-14

#### Cancel

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

### 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 5-3

### 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 4-20

- “Replace Environment Search Path Element” on page 4-22
- “Accelerated Item Selection” on page 4-11

### 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 4-20

### Replace...

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

See also:

- “Replace Environment Search Path Element” on page 4-22

### Remove

Remove the selected environment from the Environment Search Path.

### Open

Open the selected environment in a new Ada Environment window.

## Change Frozen Status

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

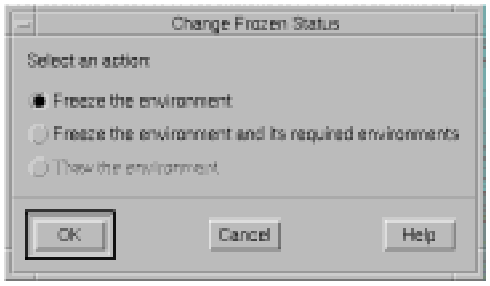
An environment can be *frozen*, making it unalterable unless it is subsequently thawed. The advantage of freezing an environment is that the consistency of the contents of the environment is ascertained and subsequently stored. Access to the contents of a frozen environment from other, usually unfrozen, 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 requried environments are frozen again, whereupon the invalidly frozen environment must be re-frozen.

See also:

- "Overview" on page 4-1
- "Settings" on page 4-13



Change Frozen Status

Select an action:

[✓ Freeze the environment]
Freeze the environment and its required environments
Thaw the environment

**Figure 4-9. 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 4-14

### 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 4-14
- “Environment Search Path” on page 4-15

#### 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 4-14

## Change Permissions

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

See also:

- “Overview” on page 4-1
- “Settings” on page 4-13

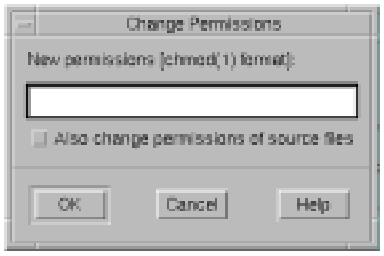

Change Permissions
New permissions [chmod(1) format]:
Also change permissions of source files
OK
Cancel
Help

**Figure 4-10. 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

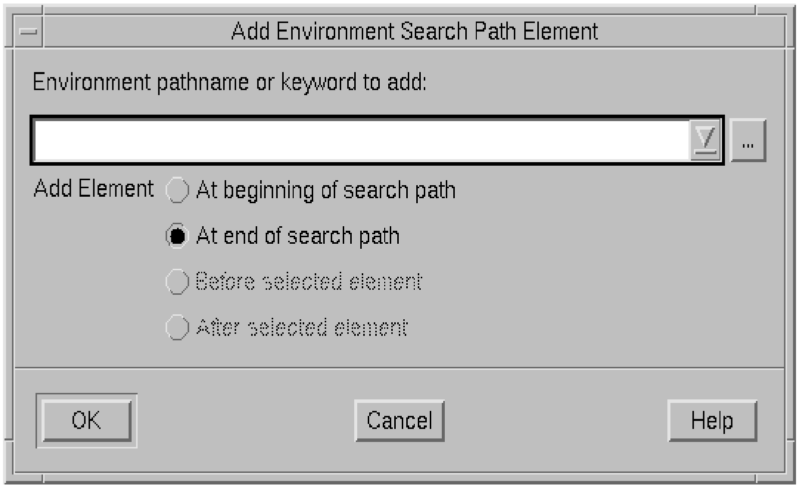
In addition to the internal NightBench files for this environment, the specified permissions will be applied to source files and partition targets as well.

## 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:

- “Overview” on page 4-1
- “Settings” on page 4-13


Add Environment Search Path Element
Environment pathname or keyword to add:
...
Add Element
At beginning of search path
At end of search path
Before selected element
After selected element
OK
Cancel
Help

Add Element: At beginning of search path [✓ At end of search path] Before selected element After selected element

**Figure 4-11. 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:

- "Overview" on page 4-1
- "Settings" on page 4-13

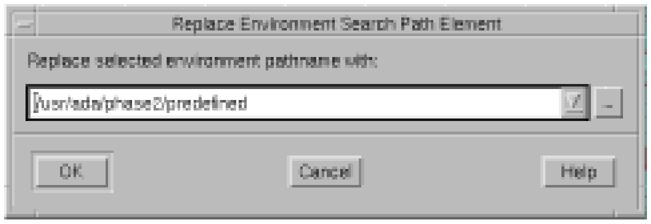

Replace Environment Search Path Element
Replace selected environment pathname with:
/usr/ada/phase2/predefined
...
OK
Cancel
Help

**Figure 4-12. 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.

#### ...

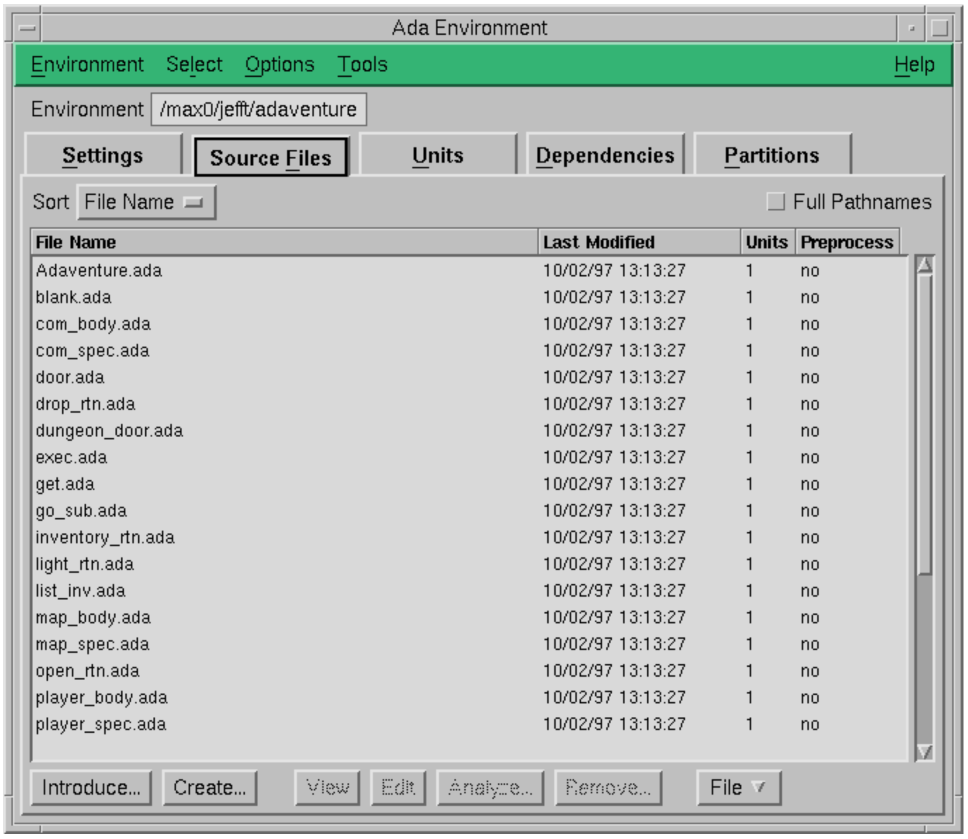
Use this to select the environment pathname. A standard file dialog is displayed in order to navigate to the directory of the desired environment.

## 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 prepreocessed. 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:

- "Overview" on page 4-1



| File Name | Last Modified | Units | Preprocess |
|---|---|---|---|
| Adaventure.ada | 10/02/97 13:13:27 | 1 | no |
| blank.ada | 10/02/97 13:13:27 | 1 | no |
| com_body.ada | 10/02/97 13:13:27 | 1 | no |
| com_spec.ada | 10/02/97 13:13:27 | 1 | no |
| door.ada | 10/02/97 13:13:27 | 1 | no |
| drop_rtn.ada | 10/02/97 13:13:27 | 1 | no |
| dungeon_door.ada | 10/02/97 13:13:27 | 1 | no |
| exec.ada | 10/02/97 13:13:27 | 1 | no |
| get.ada | 10/02/97 13:13:27 | 1 | no |
| go_sub.ada | 10/02/97 13:13:27 | 1 | no |
| inventory_rtn.ada | 10/02/97 13:13:27 | 1 | no |
| light_rtn.ada | 10/02/97 13:13:27 | 1 | no |
| list_inv.ada | 10/02/97 13:13:27 | 1 | no |
| map_body.ada | 10/02/97 13:13:27 | 1 | no |
| map_spec.ada | 10/02/97 13:13:27 | 1 | no |
| open_rtn.ada | 10/02/97 13:13:27 | 1 | no |
| player_body.ada | 10/02/97 13:13:27 | 1 | no |
| player_spec.ada | 10/02/97 13:13:27 | 1 | no |

**Figure 4-13. Ada Environment window - 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 4-11

### 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

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

## Introduce...

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

The source file will be analyzed for errors before it can be 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 4-28
- "Analyze Errors" on page 1-14

### Create...

Brings up the standard file dialog to choose the directory and name of the new source file to create. After the file name and path is determined, the newly created file is loaded into the editor.

The source file will be analyzed for errors before it can be introduced into the environment. If errors are encountered, the Analyze Errors dialog will be presented, showing the errors found.

See also:

- "Preferences" on page 1-7
- "Analyze Errors" on page 1-14

### View

Loads the selected files into the configured file viewer.

See also:

- "Preferences" on page 1-7

### 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" on page 1-7
- "Analyze Errors" on page 1-14

### Analyze...

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" on page 4-32

### 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 4-34

### File

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

See also:

- “Source File Commands” on page 4-27

## Source File Commands

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

See also:

- "Overview" on page 4-1
- "Source Files" on page 4-23

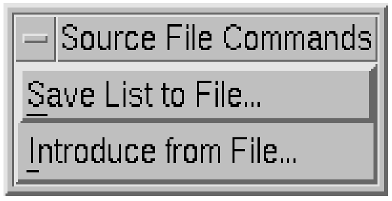


**Figure 4-14. Source File Commands menu**

### 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 4-35

### 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 4-37

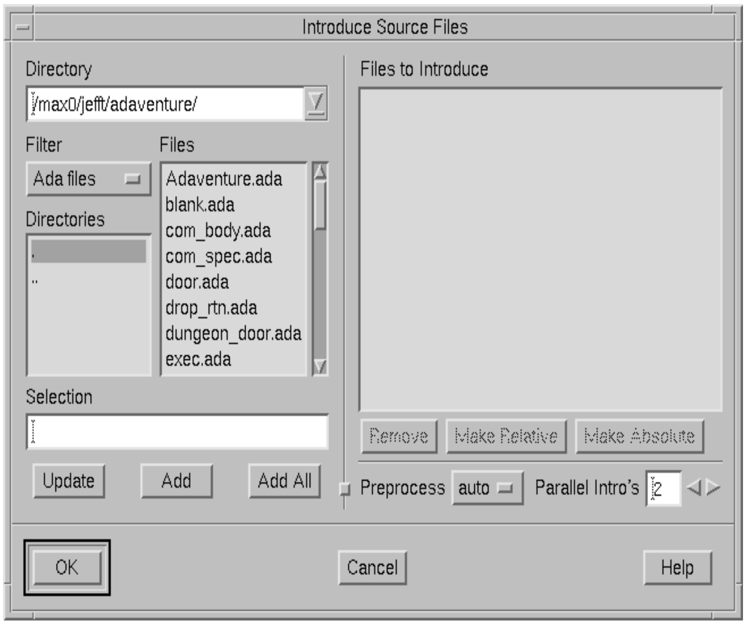
## Introduce Source Files

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 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.

See also:

- "Overview" on page 4-1
- "Source Files" on page 4-23



Directory: /max0/jefft/adaventure/

Filter: Ada files

Files:
Adaventure.ada
blank.ada
com_body.ada
com_spec.ada
door.ada
drop_rtn.ada
dungeon_door.ada
exec.ada

Selection:

Preprocess: auto

Parallel Intro's: 2

**Figure 4-15. 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.

### 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, Ada Files, includes all files with the following suffixes:

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

### 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.

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

Multiple files can be selected by dragging the mouse pointer over multiple 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.

### Update

Allows the list of Files to be updated with respect to the current Directory and Filter.

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

### 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.

### Files to Introduce

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

### Remove

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

### Make Relative

Makes the pathnames for the selected items within the list of Files to Introduce relative to the directory containing the current environment.

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.

### Make Absolute

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 “Make Relative” above). Also, when using absolute pathanames, 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.

### Preprocess

Allows preprocessing of source files before introduction into NightBench.

| | |
|---|---|
| Auto | specifies that only source files with a **.pp** suffix are preprocessed |
| 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 |

## Parallel Introductions

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**

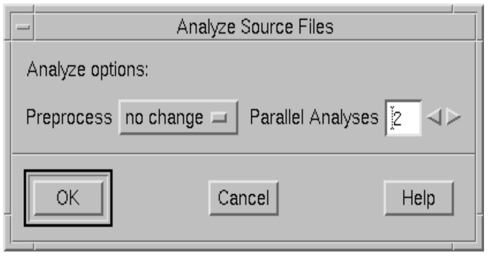
In practice, specifying a number larger than twice the number of CPUs on the system is probably counterproductive.

## Analyze Source Files

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 1-7
- "Analyze Errors" on page 1-14
- "Overview" on page 4-1
- "Source Files" on page 4-23


Analyze options:

Preprocess: no change

Parallel Analyses: 2

**Figure 4-16. Analyze Source Files dialog**

### Preprocess

Allows preprocessing during analysis of source files.

| | |
|---|---|
| `no change` | preproccess 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**

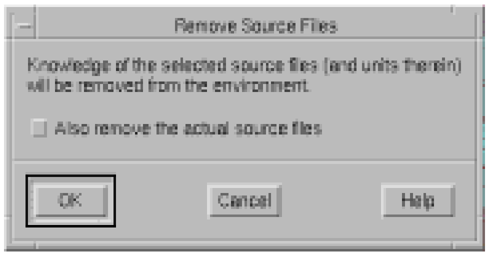
In practice, specifying a number larger than twice the number of CPUs on the system is probably counterproductive.

## 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:

- "Overview" on page 4-1
- "Source Files" on page 4-23



Knowledge of the selected source files (and units therein) will be removed from the environment.

☐ Also remove the actual source files

Figure 4-17. Remove Source Files dialog

### Also remove the actual source files

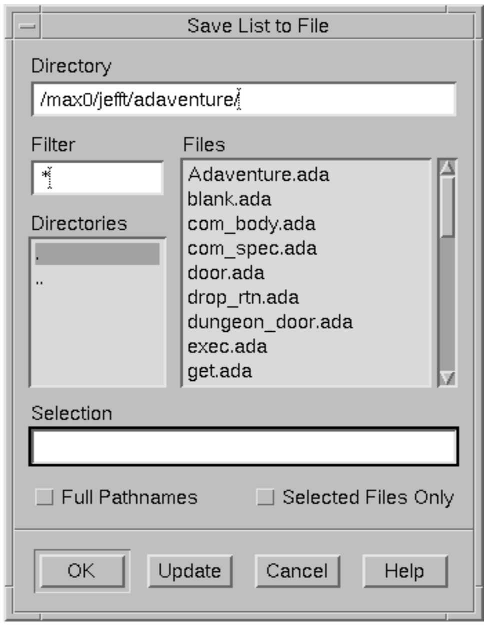
Delete the physical source files in addition to any information used internally by NightBench about the selected source files (and the units contained therein).

## 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 4-37
- “Overview” on page 4-1
- “Source Files” on page 4-23


Save List to File

Directory: /max0/jefft/adaventure/

Filter: *

Directories:
.
..

Files:
Adaventure.ada
blank.ada
com_body.ada
com_spec.ada
door.ada
drop_rtn.ada
dungeon_door.ada
exec.ada
get.ada

Selection:

☐ Full Pathnames ☐ Selected Files Only

OK | Update | Cancel | Help

**Figure 4-18. 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:

- "Source Files" on page 4-23

### Update

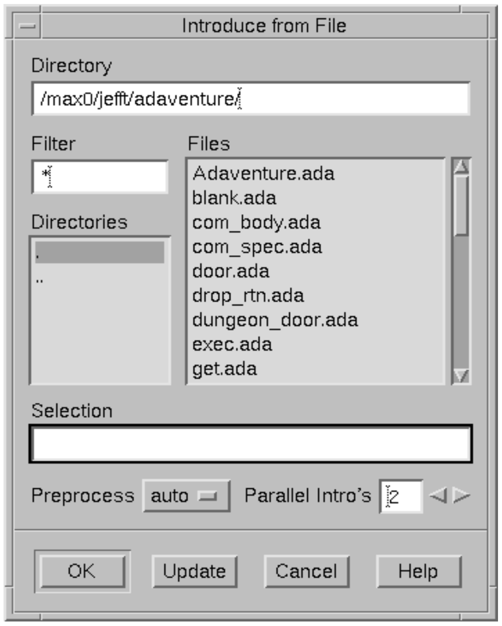
Update the list of Files based on the information entered in the Directory and Filter fields.

## 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 4-35
- "Overview" on page 4-1
- "Source Files" on page 4-23



Directory: /max0/jefft/adaventure/

Filter: *

Directories:
.
..

Files:
Adaventure.ada
blank.ada
com_body.ada
com_spec.ada
door.ada
drop_rtn.ada
dungeon_door.ada
exec.ada
get.ada

Preprocess: auto

Parallel Intro's: 2

**Figure 4-19. 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 preprocessed |
| Yes | specifies that every source file in the list should be preprocessed |
| 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

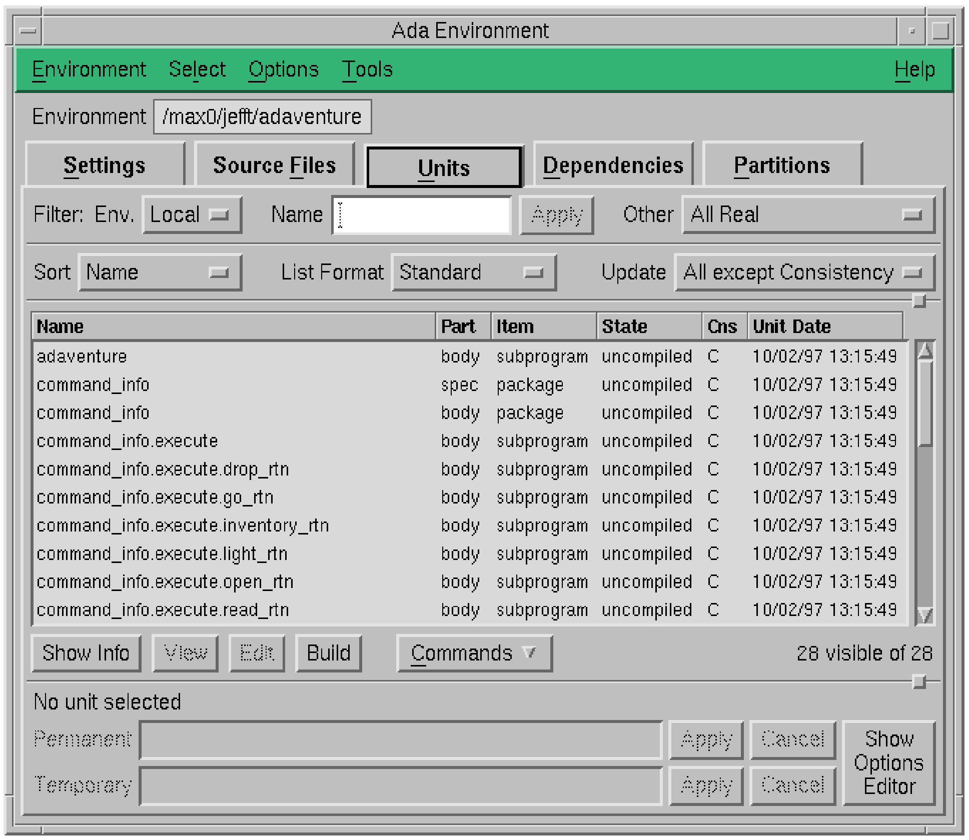
Update the list of Files based on the information entered in the Directory and Filter fields.

## 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:

- "Overview" on page 4-1



Environment /max0/jefft/adaventure

Filter: Env. Local Name Other All Real

Sort Name List Format Standard Update All except Consistency

| Name | Part | Item | State | Cns | Unit Date |
|---|---|---|---|---|---|
| adaventure | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info | spec | package | uncompiled | C | 10/02/97 13:15:49 |
| command_info | body | package | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute.drop_rtn | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute.go_rtn | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute.inventory_rtn | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute.light_rtn | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute.open_rtn | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |
| command_info.execute.read_rtn | body | subprogram | uncompiled | C | 10/02/97 13:15:49 |

28 visible of 28

No unit selected

Permanent

Temporary

**Figure 4-20. Ada Environment window - Units page**

#### Env.

Display only those units in 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 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 Units list, displaying only those units that begin with the characters in that field.

#### Other

Filter the list 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 | display only units which are specifications |
| Body | display only units which are bodies |
| All (including Artificial) | display all units in the environment including those created by the environment related to instances of generics |

**Sort**

Sort the list 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 |
| Source File | sort the units alphabetically based on the names of their respective source files |
| State | sort the units by compilation state |
| Part | sort the units into the categories of **spec** and **body** |
| Item | sort the units into the categories subprogram, package, task, protected |
| Class | sort the units into groupings of library, subunit, and nested |
| Generic Info | sort the units into groupings of generic, instance-of-…, generic-corresponding-to…, and n/a |
| Visa | sort the units into categories of native, naturalized, fetched, and foreign |
| Home Env. | 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 4-53 for information about these fields

### Update

This menu controls the extent to which NightBench keeps the information in the Units list up-to-date.

| | |
|---|---|
| All except Consistency | 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 List Format specification.

See also:

- "List Format" on page 4-41
- "Unit Information" on page 4-53
- "Accelerated Item Selection" on page 4-11

### 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 4-53

### View

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

See also:

- "Preferences" on page 1-7

### Edit

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

See also:

- "Preferences" on page 1-7

### Build

Opens the NightBench Builder with the selected units as the targets. The build may automatically start if the Automatically Start Builds option is selected from the Options menu of the Ada Environment window.

See also:

- NightBench Builder - Chapter 6
- "Automatically Start Builds" on page 4-9

### Commands

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

See also:

- "Commands" on page 4-46

## Unit Compile Options

### 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 4-45
- "Unit Compile Options Editor" on page 5-13

#### Apply

Changes the permanent unit options set to reflect any modifications made to the Permanent field.

#### Cancel

Resets the Permanent options to the set of permanent options previously applied to the selected units, disregarding any changes made since.

### 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 4-45
- "Unit Compile Options Editor" on page 5-13

#### Apply

Changes the permanent unit options set to reflect any modifications made to the Temporary field.

#### Cancel

Resets the Temporary options to the set of temporary options previously applied to the selected units, disregarding any changes made since.

### Show Options Editor

Opens the NightBench Unit Compile Options Editor.

See also:

- “Unit Compile Options Editor” on page 5-13

## 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:

- "Overview" on page 4-1
- "Units" on page 4-39

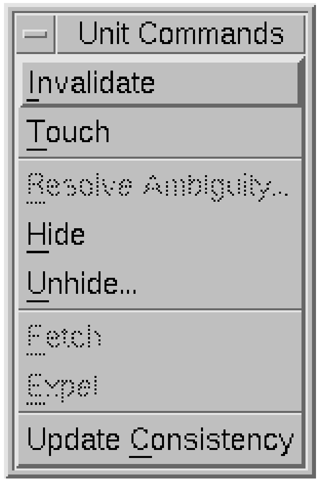


**Figure 4-21. Commands menu**

### 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 4-46 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 4-46 for the opposite functionality

### Resolve Ambiguity...

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 Ada Environment 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 4-51

### Hide

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 4-47 for the opposite functionality

### Unhide...

Bring up the Hidden Units dialog if there are hidden units in the current environment.

See also:

- “Hidden Units” on page 4-49
- “Hide” on page 4-47 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 visiblity 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 4-48 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 4-48 for the opposite functionality

### Update Consistency

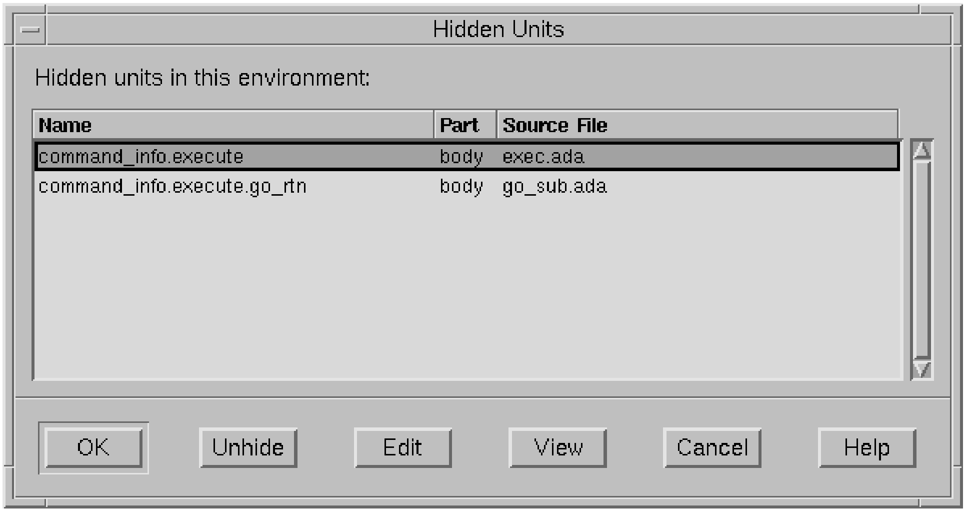
Determines the consistency of the selected units or of all units if none are selected.

## 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 4-47
- "Resolve Ambiguity..." on page 4-47
- "Overview" on page 4-1
- "Units" on page 4-39



Hidden units in this environment:

| Name | Part | Source File |
|---|---|---|
| command_info.execute | body | exec.ada |
| command_info.execute.go_rtn | body | go_sub.ada |

**Figure 4-22. 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 4-53

### Source File

The name of the file from which this unit came.

Double-clicking a list item will open the corresponding source file in the user’s configured file editor.

See also:

- “Preferences” on page 1-7

### OK

The dialog window 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” on page 1-7

### View

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

See also:

- “Preferences” on page 1-7

## 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:

- “Overview” on page 4-1
- “Units” on page 4-39

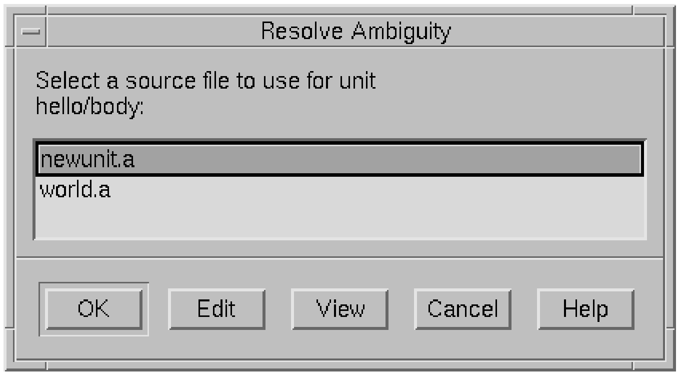


**Figure 4-23. Resolve Ambiguity dialog**

### 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” on page 1-7

### 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” on page 1-7

### View

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

See also:

- "Preferences" on page 1-7

## 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. The fields below appear in the order corresponding to the list format `Everything` (see "List Format" on page 4-41). Other list formats contain a subset of the fields below.

In addition, this information can also be seen using the `Show Info` button on the Units page. See "Show Info" on page 4-43 for more information.

See also:

- "Overview" on page 4-1
- "Units" on page 4-39

### 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.

### Part

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

### Item

The type of this unit. Units can be one of the following:

- `package`
- `subprogram`
- `task`

- `protected`

### Class

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 |

### 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 |

### Main

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 |

### State

Units in the environment can be in either of the following compilation states:

- `uncompiled`
- `compiled`

### Consistency (Cns.)

The *consistency* of a given unit.

| | | |
|---|---|---|
| C | consistent | the units'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 |

### Unit Date

The date and time the unit last changed compilation state.

### Source File

The name of the file from which this unit came.

### Source Date

The date and time of the last modification to the Source File for this unit.

### Generic Info

Generic information about a unit.

| | |
|---|---|
| generic | A *generic unit* 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 *instance* of a unit is created using a *generic unit* as a template. |
| n/a | Neither a *generic unit* nor an *instance* of a unit. |

### 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

### Home Environment (Home Env.)

The environment in which the unit was compiled (including any naturalized copies).

### Origin Environment (Origin Env.)

The environment into which the unit was originally introduced (ignoring any naturalized copies).

### Permanent Options

The set of permanent options associated with this unit.

### Temporary Options

The set of temporary options associated with this unit.

### Effective Options

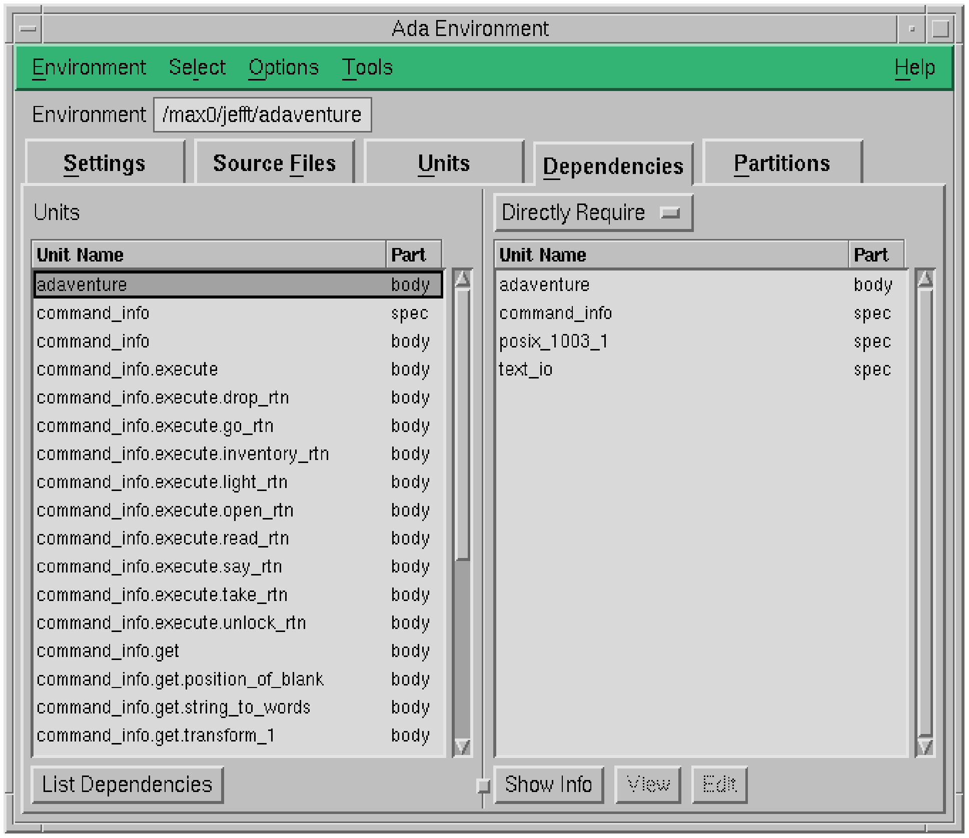
The set of effective options associated with this unit.

## Dependencies

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.

See also:

- “Overview” on page 4-1



Environment /max0/jefft/adaventure

| Unit Name | Part |
|---|---|
| adaventure | body |
| command_info | spec |
| command_info | body |
| command_info.execute | body |
| command_info.execute.drop_rtn | body |
| command_info.execute.go_rtn | body |
| command_info.execute.inventory_rtn | body |
| command_info.execute.light_rtn | body |
| command_info.execute.open_rtn | body |
| command_info.execute.read_rtn | body |
| command_info.execute.say_rtn | body |
| command_info.execute.take_rtn | body |
| command_info.execute.unlock_rtn | body |
| command_info.get | body |
| command_info.get.position_of_blank | body |
| command_info.get.string_to_words | body |
| command_info.get.transform_1 | body |

| Unit Name | Part |
|---|---|
| adaventure | body |
| command_info | spec |
| posix_1003_1 | spec |
| text_io | spec |

**Figure 4-24. Ada Environment window - Dependencies page**

### Units

A listing of all the units in the current environment.

See also:

- “Accelerated Item Selection” on page 4-11

#### Unit Name

The name of the unit.

See also:

- "Unit Information" on page 4-53

#### 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 4-53

### 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 "Ada Environment window - Dependencies page" on page 4-59, the unit `adaventure` 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 `adaventure` 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 4-11

#### Unit Name

The name of the unit.

See also:

- "Unit Information" on page 4-53

#### 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 4-53

### 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 4-60

### 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 4-53

### View

Loads the units selected from the dependencies list into the configured file viewer.

See also:

- “Preferences” on page 1-7

### Edit

Loads the units selected from the dependencies list into the configured file editor, allowing changes to be made to the files.

See also:

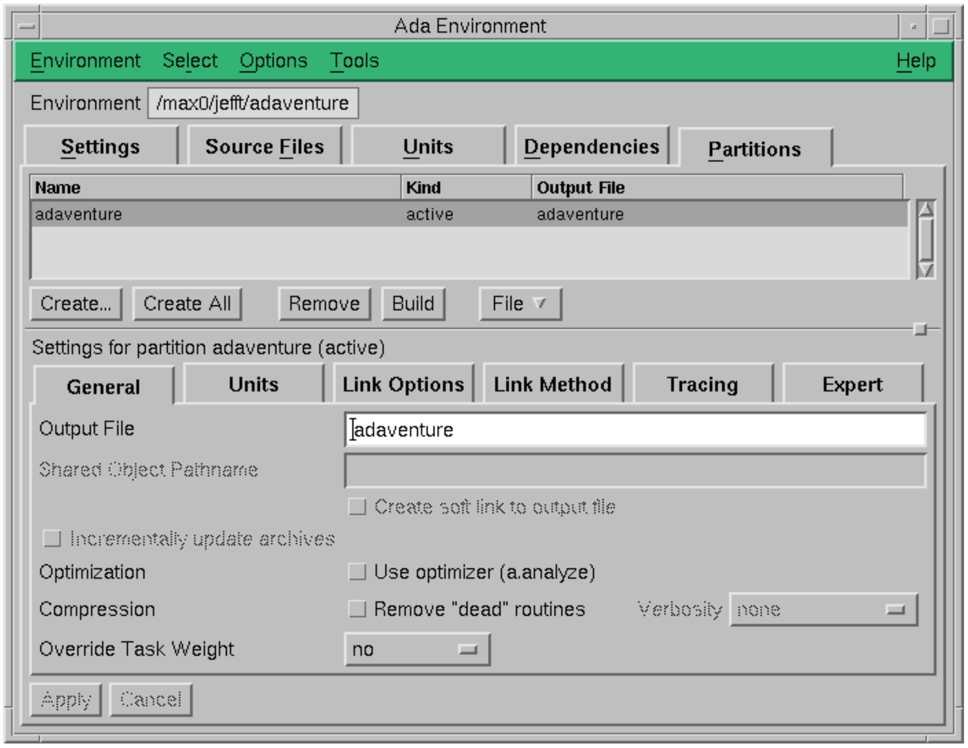
- “Preferences” on page 1-7

## Partitions

The Partitions page allows the user to create, configure and build partitions in the NightBench Program Development Environment. This main page and its various subpages display detailed information about the partitions defined in the current environment.

See also:

- “Overview” on page 4-1


Ada Environment
Environment Select Options Tools Help
Settings Source Files Units Dependencies Partitions
Create... Create All Remove Build File
General Units Link Options Link Method Tracing Expert
Apply Cancel

Environment /max0/jefft/adaventure

| Name | Kind | Output File |
|---|---|---|
| adaventure | active | adaventure |

Settings for partition adaventure (active)

Output File adaventure

Shared Object Pathname

☐ Create soft link to output file

☐ Incrementally update archives

Optimization ☐ Use optimizer (a.analyze)

Compression ☐ Remove "dead" routines Verbosity none

Override Task Weight no

**Figure 4-25. Ada Environment window - Partitions page**

### Partitions List

A listing of all the partitions that have been created in the current environment.

See also:

- “Accelerated Item Selection” on page 4-11

#### Name

The name of the partition.

#### Kind

The type of partition. NightBench supports three different kinds of partitions:

- active
- shared object
- archive

See the Glossary for definitions of these categories.

#### Output File

The name of the resultant file created when this partition is built. This name is specified on the General page of the Partition Settings area.

See also:

- “Partition Settings area” on page 4-64
- “Settings - General” on page 4-72
- “Output File” on page 4-72

### Create...

Create a new partition within the current environment.

See also:

- “Create Partition” on page 4-65

### 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.

### Remove

Remove the selected partition from the environment.

NightBench verifies that this action is intentional.

### Build

Opens the NightBench Builder window. The build Targets (see “Targets” on page 6-9) are any selected partitions in the partitions list (see “Partitions List” on page 4-62). If no partitions are selected, the default build target is all partitions. The build may automatically start if the Automatically Start Builds option is selected from the Options menu of the Ada Environment window.

See also:

- NightBench Builder - Chapter 6
- "Targets" on page 6-14
- "Automatically Start Builds" on page 4-9

### 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 4-67

### Partition Settings area

The bottom half of the Partitions page is composed of the following five subpages which contain information about the currently selected partition. The options listed in these subpages apply only during the linking phase of the build.

See also:

- "Settings - General" on page 4-72
- "Settings - Units" on page 4-75
- "Settings - Link Options" on page 4-81
- "Settings - Link Method" on page 4-83
- "Settings - Tracing" on page 4-87
- "Settings - Expert" on page 4-91

#### 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:

- “Overview” on page 4-1
- “Partitions” on page 4-62

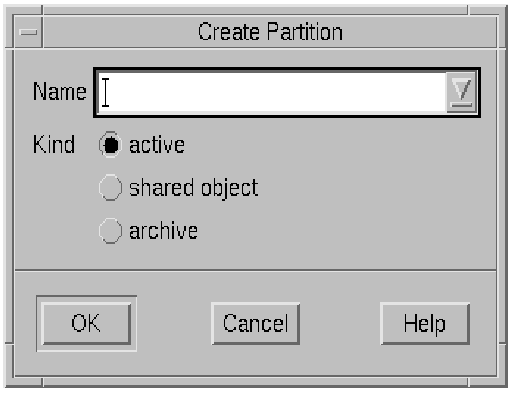


Figure 4-26. Create Partition dialog

### 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 sharing mode shared (implying the use of position independent code). (See “Sharing Mode” on page 5-6 and “Sharing Mode” on page 5-16 for more information.) |
| archive | An archive is a collection of routines and data that can be used 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 sharing mode non-shared (implying the use of static code). (See “Sharing Mode” on page 5-6 and “Sharing Mode” on page 5-16 for more information.) |

## Partition File Commands

The Partition File Commands menu appears on the Partitions page of the Ada Environment window. It is labeled on the Partitions page as File and contains a drop-down list of utilities related to partition definitions.

See also:

- “Overview” on page 4-1
- “Partitions” on page 4-62

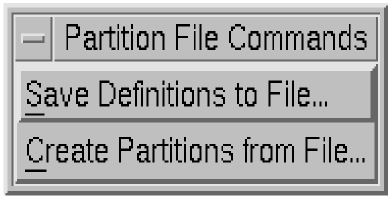


**Figure 4-27. Partition File Commands menu**

### 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 4-68

### 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:

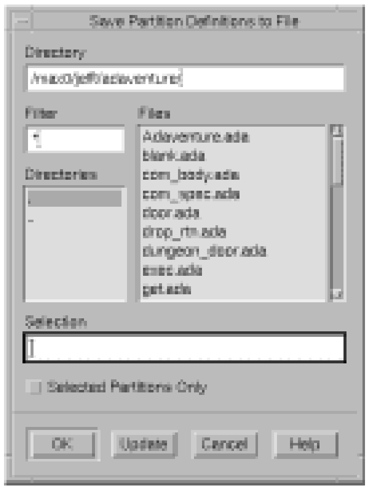
- “Create Partitions from File” on page 4-70

## 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 4-67
- “Overview” on page 4-1
- “Partitions” on page 4-62



/mscd/jeff/adaventure/

*

.
..

Adaventure.ada
blank.ada
com_body.ada
com_spec.ada
door.ada
drop_rtn.ada
dungeon_door.ada
exec.ada
get.ada

Figure 4-28. 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 4-62).

### 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 4-67
- "Overview" on page 4-1
- "Partitions" on page 4-62

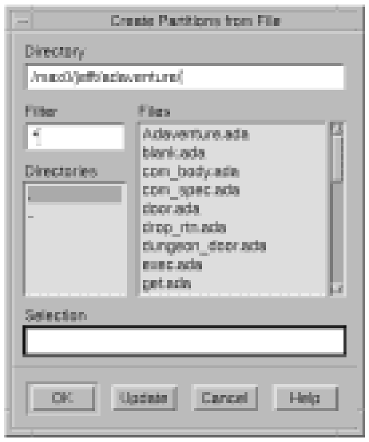


**Figure 4-29. Create Partitions from File dialog**

### 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

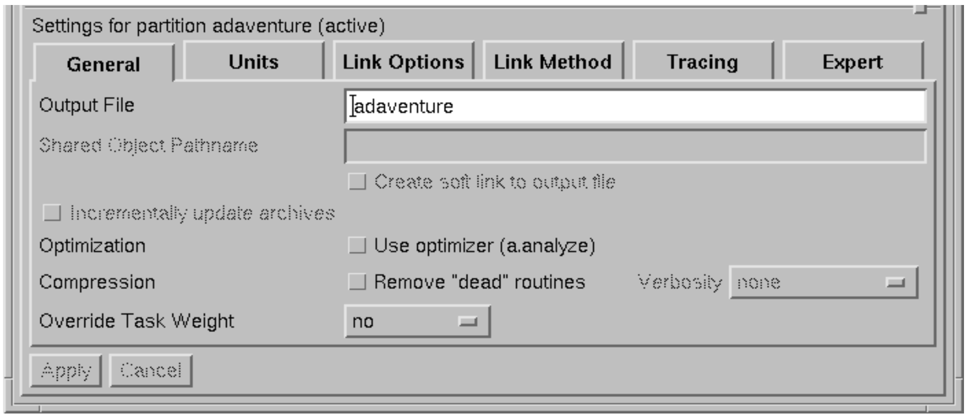
Update the list of Files based on the information entered in the Directory and Filter fields.

## Settings - General

General information about the partition is contained on this page. The output file is specifed here and other settings such as optimization and compression can be configured here. These options apply only during the linking phase of the build.

See also:

- "Overview" on page 4-1
- "Partitions" on page 4-62


Settings for partition adaventure (active)

General | Units | Link Options | Link Method | Tracing | Expert

Output File: adaventure
Shared Object Pathname:
☐ Create soft link to output file
☐ Incrementally update archives
Optimization: ☐ Use optimizer (a.analyze)
Compression: ☐ Remove "dead" routines    Verbosity: none
Override Task Weight: no

**Figure 4-30. Ada Environment - Partitions page - 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 4-65

### 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).

### 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 conjuction with the `Shared Object Pathname` removes the need for the shared object to be explicitly placed at the pathname specifed by the `Shared Object Pathname`.

### Incrementally update archives

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.

### Use optimizer

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.

### 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 *bound* |
| multiplexed | sets the default task weight to *multiplexed* |

**NOTE**

This setting has no effect as only *bound* tasks are supported at this time.

## Settings - Units

Units to be included as part of the current partition's definition are specified on this page. In addition, the user may exclude specific units from a partition on this page as well. These options apply only during the linking phase of the build.

This particular page changes slightly depending on whether the current partition is an active partition or a non-active partition (shared object or archive).

See also:

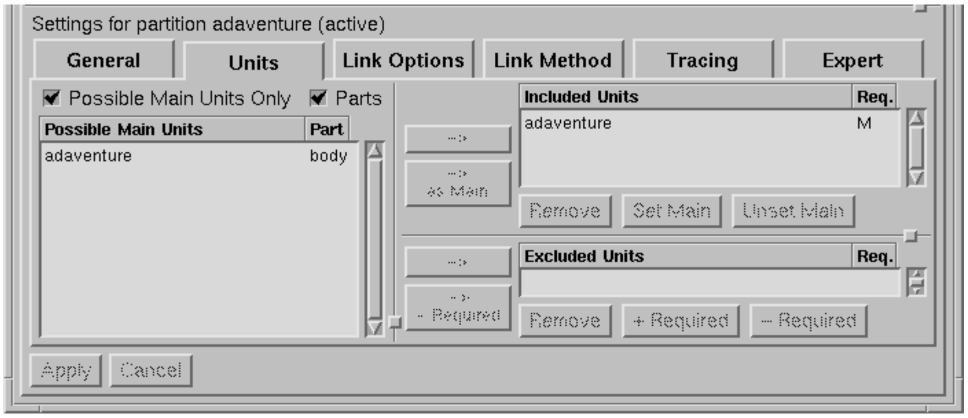
- "Settings - Units - Active Partitions" on page 4-75
- "Settings - Units - Non-Active Partitions" on page 4-78
- "Overview" on page 4-1
- "Partitions" on page 4-62

## Settings - Units - Active Partitions

The Units settings page for an active partition allows the user to specify which unit should be used as the main subprogram for the active partition.

See also:

- "Overview" on page 4-1
- "Partitions" on page 4-62


Settings for partition adaventure (active)

☑ Possible Main Units Only ☑ Parts

| Possible Main Units | Part |
|---|---|
| adaventure | body |

| Included Units | Req. |
|---|---|
| adaventure | M |

| Excluded Units | Req. |
|---|---|

**Figure 4-31. Ada Environment - Partitions page - Units settings - active**

### Possible Main Units Only

When this option is checked, only units that can possibly be a main unit are listed here. (These units are designated by a `yes` or `maybe` under `Main` on the Units page.)

Otherwise, all units within the current environment are listed.

See also:

- "Units" on page 4-39
- "Unit Information" on page 4-53

### Parts

When this option is checked, an entry for each *part* (**spec** and **body**) of every unit is listed. Otherwise, only the unit name appears in the list.

### Units List

This either lists all units in the environment or only possible main units, depending on the state of Possible Main Units Only. In addition, part information about each unit may be listed, according to the setting of the Parts checkbox.

### -->

This button adds the selected unit from the Units List to the Included Units List for the current partition. In addition, all units that this unit requires are added to the list of included units.

### --> As Main

This button adds the selected unit from the Units List 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 should be designated as the main unit for a partition.

### Included Units List

This lists each of the units that are included in the definition of this partition.

#### Req.

An M in this column indicates that the specified unit is the current partition's main unit. All units required by this unit are also included in the current partition.

A + in this column indicates that all units required by the specified unit are also included in the current partition's definition, even though the specified unit is not the main unit.

#### Remove

Removes the unit selected in the Included Units List from this list.

#### Set Main

Designates the unit selected in the Included Units List as the main unit for the current partition.

#### Unset Main

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.

### -->

This button adds the selected unit from the Units List to the Excluded Units List for the current partition. In addition, all units that this unit requires are added to the list of excluded units.

### -->- Required

Only adds the selected unit to the list of excluded units but not those units that it requires.

### Excluded Units List

This lists each of the units that are excluded from the current partition's definition.

#### 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.

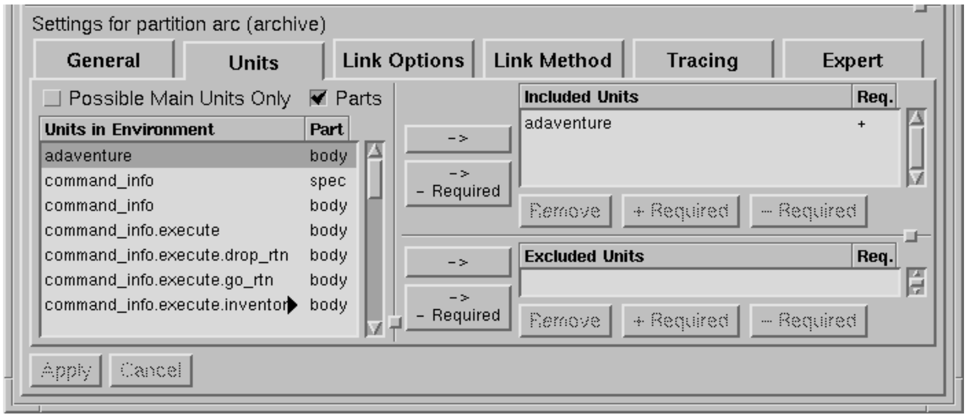
## Settings - Units - Non-Active Partitions

Some of the buttons on the Units settings page for a non-active partition are labeled differently and have different functionality from those buttons on the Units settings page for an active partition. They are listed here to avoid any confusion.

The main difference between the Units settings page 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 functionality for this does not exist on the Units settings page for a non-active partition.

See also:

- "Overview" on page 4-1
- "Partitions" on page 4-62



| Units in Environment | Part |
|---|---|
| adaventure | body |
| command_info | spec |
| command_info | body |
| command_info.execute | body |
| command_info.execute.drop_rtn | body |
| command_info.execute.go_rtn | body |
| command_info.execute.inventor | body |

| Included Units | Req. |
|---|---|
| adaventure | + |

| Excluded Units | Req. |
|---|---|

Figure 4-32. Ada Environment - Partitions page - Units settings - archive

### Possible Main Units Only

For non-active partitions, this option should not be checked so that all units within the current environment are listed.

When this option is checked, only units that can possibly be a main unit are listed here. (These units are designated by a `yes` or `maybe` under `Main` on the Units page.) However, possible main units are usually of interest to active partitions only.

See also:

- "Units" on page 4-39
- "Unit Information" on page 4-53

### Parts

When this option is checked, an entry for each *part* (**spec** and **body**) of every unit is listed. Otherwise, only the unit name appears in the list.

## Units List

This either lists all units in the environment or only possible main units, depending on the state of Possible Main Units Only. In addition, part information about each unit may be listed, according to the setting of the Parts checkbox.

## -->

This button adds the selected unit from the Units List to the Included Units List for the current partition. In addition, all units that this unit requires are added to the list of included units.

## --> - Required

This button adds only the selected unit from the Units List to the Included Units List for the current partition. It does not add any units that this unit requires to the list of included units.

## Included Units List

This lists each of the units that are included in the definition of this partition.

### Req.

All units required by this partition are listed here (a + in this column indicates that all units that each particular unit requires are also included in this partition's definition).

### Remove

Removes the unit selected in the Included Units List from this list.

### + Required

In addition to the unit selected in the Included Units List, this button also specifies that any units that this unit requires should also be included.

### - Required

This button specifies that only the unit selected in the Included Units List should be included but not necessarily the units that this unit requires.

## -->

This button adds the selected unit from the Units List to the Excluded Units List for the current partition. In addition, all units that this unit requires are added to the list of excluded units.

**-->- Required**

Only adds the selected unit to the list of excluded units but not those units that it requires.

**Excluded Units List**

This lists each of the units that are excluded from the current partition's definition.

**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.

## Settings - Link Options

Options may be specified to the linker (**`a.link`**) on this page. The more common link options have their own buttons which, when pressed, enter their text into the Additional Link Options area. Any other options to be sent to the linker for the current partition may be manually entered into this area. These options apply only during the linking phase of the build.

See also:

- "Overview" on page 4-1
- "Partitions" on page 4-62

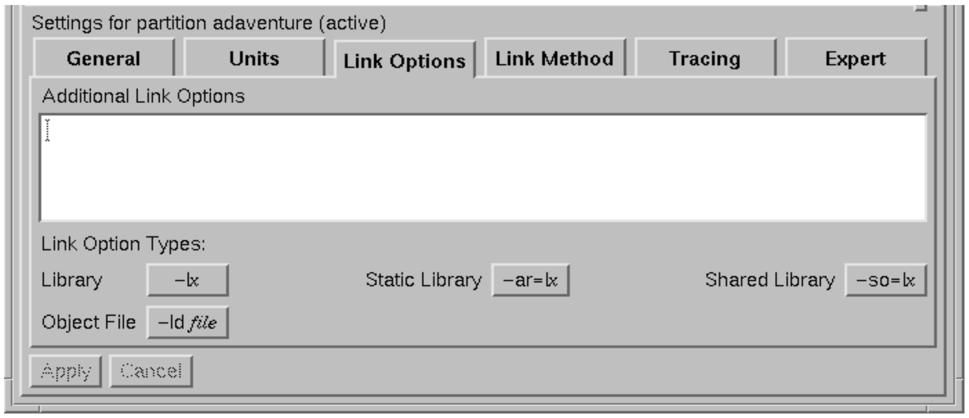


**Figure 4-33. Ada Environment - Partitions page - Link Options settings**

### Additional Link Options

This area shows the additional link options for the current partition. The buttons below this area assist the user in specifying some of the more common link options. Additional link options may be manually entered here as well.

### Library (-l*x*)

When this form is used, the linker will determine the link method (Archive or Shared Object) based on the *link rule* (see "Settings - Link Method" on page 4-83) and on the presence or absence of archive and shared object versions of the specified library. If the behavior as specified by the link rule is not desired, either -ar=l*x* or -so=l*x* should be used instead. See Static Library (-ar=lx) and Shared Library (-so=lx) below.

When this button is pressed, NightBench will add the option to the Additional Link Options area and position the cursor following the added text so that the user may specify the *x* portion of the library name.

### Static Library (-ar=l*x*)

This link option passes a **`-l`***x* option to the system loader (**`ld`**), ensuring the library **`lib`***x***`.a`** will be statically linked.

When this button is pressed, NightBench will add the option to the Additional Link Options area and position the cursor following the added text so that the user may specify the *x* portion of the library name.

### Shared Library (-so=l*x*)

This link option passes a **`-l`***x* option to the system loader (**`ld`**), ensuring the library **`lib`***x***`.so`** will be dynamically linked.

When this button is pressed, NightBench will add the option to the Additional Link Options area and position the cursor following the added text so that the user may specify the *x* portion of the library name.

### Object File (-ld *file*)

This link option passes an object *file* to the system loader (**`ld`**).

When this button is pressed, NightBench will add the option to the Additional Link Options area and position the cursor following the added text so that the user may specify the *file* name.

**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-indepedent 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.

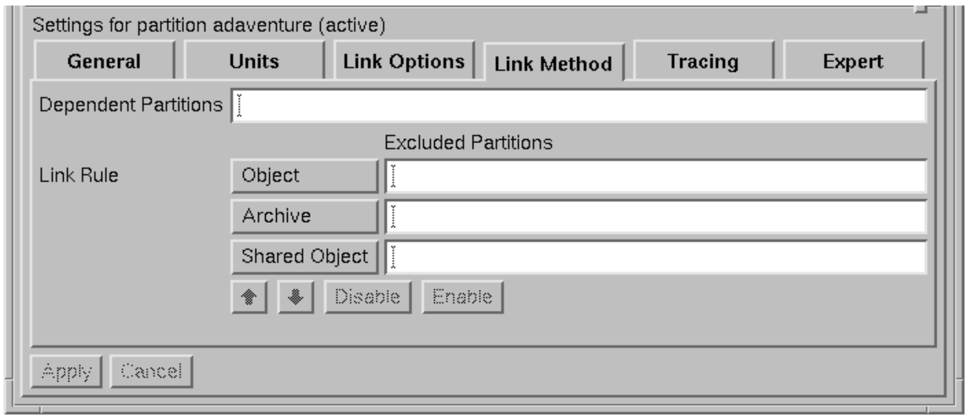
## Settings - Link Method

The Link Method page allows the user to specify the dependent partitions and the *link rule* for a given partition. The link rule is an ordering of the link methods (Object, Archive, and Shared Object) which specifies the manner in which a unit or system library is included during the linking process. Note that using the object directly (the Object method) is the most common method of utilizing units.

These options apply only during the linking phase of the build.

See also:

- "Overview" on page 4-1
- "Partitions" on page 4-62



Settings for partition adaventure (active)

General | Units | Link Options | Link Method | Tracing | Expert

Dependent Partitions:

Link Rule — Excluded Partitions:
- Object:
- Archive:
- Shared Object:

**Figure 4-34. Ada Environment - Partitions page - Link Method settings**

The default link rule differs for each type of partition:

| Partition | Default Link Rule |
|---|---|
| active | **`object`**<br>**`archive`**<br>**`shared_object`** |
| archive | **`object`** |
| shared_object | **`object`** |

The ordering of the link methods within the link rule tells the linker which form of partition or system library is preferred. For example, 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 system libraries, the linker will attempt to use either the shared object or archive of a system library based on the ordering of the link methods in the link rule.

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 tools provided on this screen.

See also:

- *MAXAda Reference Manual* (0890516)

### Dependent Partitions

Enter the names of any partitions that the current partition is dependent upon.

### Object

A link method that instructs the linker to use the object of a unit directly. Select this method to change its ordering or enable/disable its functionality in the *link rule*.

#### Excluded Partitions

**NOTE**

Exclusions should not be specified for the Object link method.

### Archive

A link method that instructs the linker to utilize the unit contained in an archive.

#### Excluded Partitions

Enter the names of any archive partitions or system libraries that should be passed over by the linker.

To specify a system library, you must enter it in this field with the form:

**-l***name*

which is standard shorthand notation for:

**lib***name***.a**

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 MAX-Ada (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.

### Shared Object

A link method that instructs the linker to include the unit found within a shared object.

#### Excluded Partitions

Enter the names of any shared object partitions or system libraries that should be passed over by the linker.

To specify a system library, you must enter it in this field with the form:

**-l***name*

which is standard shorthand notation for:

**lib***name***.so**

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 MAX-Ada (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.

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

Disable the selected link method 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.

### Enable

Enable the selected link method so that it will be included in the link rule for the current partition.

## Settings - Tracing

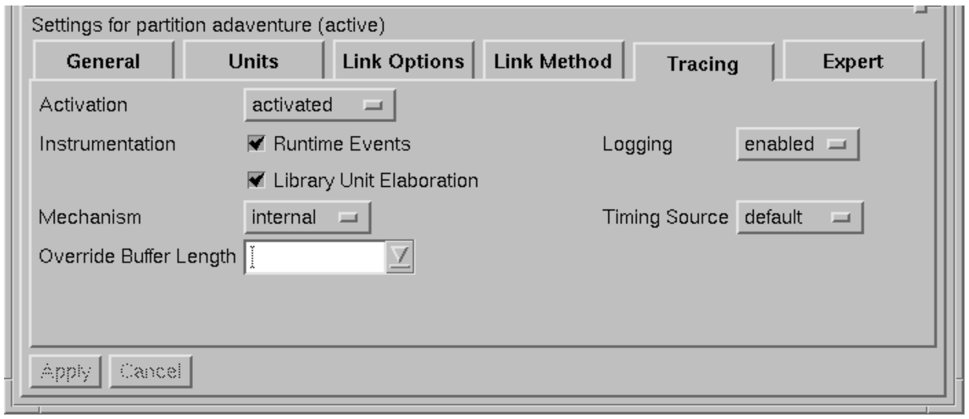
The Tracing page contains options concerning real-time event tracing. Real-time event tracing is one way to debug and analyze the performance of Ada applications. It allows the user to gather information about important events in an application, such as event occurrences, timings, and data values.

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 subseqently be analyzed using either the NightTrace product or the MAXAda utility, **`a.trace`**.

See also:

- "Overview" on page 4-1
- "Partitions" on page 4-62
- *NightTrace Manual* (0890398)
- *MAXAda Reference Manual* (0890516) - Chapter 11


Settings for partition adaventure (active)

Activation: activated

Instrumentation: ✓ Runtime Events; ✓ Library Unit Elaboration

Logging: enabled

Mechanism: internal

Timing Source: default

Override Buffer Length:

**Figure 4-35. Ada Environment - Partitions page - Tracing settings**

### Activation

Determines whether tracing support will be included in the output file:

| | |
|---|---|
| `activated` | Tracing support will be included in the output file. When this option is selected, tracing is automatically enabled. To override this default, change the setting of `Logging` appropriately (see "Logging" on page 4-89).<br><br>Tracing may be enabled or disabled at runtime without the need for relinking by calling `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.<br><br>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. |
| `deactivated` | Tracing support will not be included in the output file. Therefore, if tracing is subsequently desired, the program must be relinked with this option set to `activated`. |

### 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 Length" on page 4-90 for more information).

### Logging

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 “Activation” on page 4-88), logging is automatically enabled.

| | |
|---|---|
| `enabled` | Enables logging of predefined and user-defined trace events to trace buffer. |
| `disabled` | Disables logging of predefined and user-defined trace events. |

Tracing may be enabled or disabled at run-time by calling `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 “Activation” on page 4-88), 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.)

### 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 Length” on page 4-90 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 *NightTrace Manual* (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 *Real-Time Clock and Interrupt Module User's Guide* (0891082) for more information about this device. |

**NOTE**

This is not applicable when the tracing `Mechanism` is set to `ntraceud`. (See "Mechanism" on page 4-89.)

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 Length

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 4-89.)

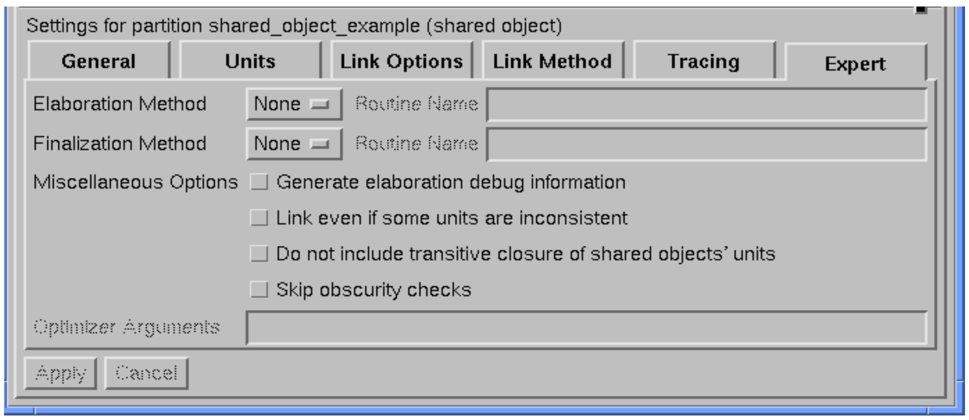
The NightTrace user daemon, however, has its own option for setting the shared memory buffer size. See the section titled **Option to Define Shared Memory Buffer Size (-memsize)** in Chapter 4 of the *NightTrace Manual* (0890398).

## Settings - Expert

The Expert page 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:

- "Overview" on page 4-1
- "Partitions" on page 4-62


Settings for partition shared_object_example (shared object)

Elaboration Method: None, Routine Name:
Finalization Method: None, Routine Name:
Miscellaneous Options:
- ☐ Generate elaboration debug information
- ☐ Link even if some units are inconsistent
- ☐ Do not include transitive closure of shared objects' units
- ☐ Skip obscurity checks

Optimizer Arguments:

**Figure 4-36. Ada Environment - Partitions page - 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 "Settings - Link Method" on page 4-83) or dependent partitions list (see "Dependent Partitions" on page 4-84). 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 | 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. |
| auto | 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.<br><br>This option is not available for archives.<br><br>**NOTE:** This option should not be used for partitions that will be included via the *link rule* (see "Settings - Link Method" on page 4-83) or dependent partitions list (see "Dependent Partitions" on page 4-84) in active Ada partitions because the automatic elaboration will interfere with the elaboration for the active Ada partition. |
| user | 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.<br><br>**WARNING:** It is the user's responsibility to ensure that the generated routine is called before any entities in the partition are used.<br><br>**NOTE:** If this option is used, Routine Name should not be called for partitions that will be included via the *link rule* (see "Settings - Link Method" on page 4-83) or dependent partitions list (see "Dependent Partitions" on page 4-84) in active Ada partitions because the elaboration performed by Routine Name will interfere with the elaboration for the active Ada partition. |

#### 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 "Settings - Link Method" on page 4-83) or dependent partitions list (see "Dependent Partitions" on page 4-84). 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 | 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.<br><br>This option is not available for archives.<br><br>**NOTE:** This option should not be used for partitions that will be included via the *link rule* (see “Settings - Link Method” on page 4-83) or dependent partitions list (see “Dependent Partitions” on page 4-84) in active Ada partitions because the automatic finalization will interfere with the finalization for the active Ada partition. |
| user | 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.<br><br>**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.<br><br>**NOTE:** If this option is used, Routine Name should not be called for partitions that will be included via the *link rule* (see “Settings - Link Method” on page 4-83) or dependent partitions list (see “Dependent Partitions” on page 4-84) in active Ada partitions because the finalization performed by Routine Name will interfere with the finalization for the active Ada partition. |

### 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.

### 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

### Optimizer Arguments

Options to be sent to the MAXAda **a.analyze** tool.

See also:

- *MAXAda Reference Manual* (0890516)

# 5
# Options Editors

## Overview

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 5-3 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 5-13 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 2-10 for an example of this relationship.

## 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 5-3
- “Unit Compile Options Editor” on page 5-13

## Environment Compile Options Editor

The Environment Compile Options Editor contains the following four pages:

- General (see page 5-5)
- Inlining (see page 5-8)
- Optimization (see page 5-9)
- Expert (see page 5-11)

### 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

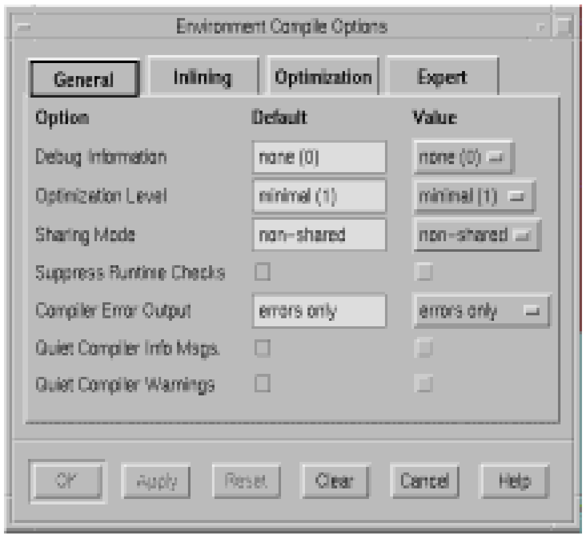
This button brings up the help topic for this page.

In addition, “Tri-state Checkboxes” on page 5-1 discusses the various states in which the checkboxes in this dialog may exist.

See also:

- "Environment Compile Options Editor" on page 5-3

## General



| Option | Default | Value |
|---|---|---|
| Debug Information | none (0) | none (0) |
| Optimization Level | minimal (1) | minimal (1) |
| Sharing Mode | non-shared | non-shared |
| Suppress Runtime Checks | ☐ | ☐ |
| Compiler Error Output | errors only | errors only |
| Quiet Compiler Info Msgs. | ☐ | ☐ |
| Quiet Compiler Warnings | ☐ | ☐ |

**Figure 5-1. Environment Compile Options dialog - General Page**

### Debug Information

This option controls the level of debug information generated for units in a given environment.

| | |
|---|---|
| `none (0)` | No debug information. Debugging tools requiring line numbers or symbolic information will not fully function on modules compiled at this debug level. |
| `lines (1)` | 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. |
| `full (2)` | 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)`.

### 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`.

### Sharing 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 sharing 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` (or `both`) is considered a `shared` unit. All shared units must be placed in a shared object to be used.

### Suppress Runtime Checks

This option gives permission to the implementation to suppress all run-time checks.

Its effect is equivalent to pragma SUPPRESS_ALL or pragma SUPPRESS(ALL_CHECKS).

### 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. |
| list in source! | Displays the error messages in the file in the same manner as the list in source option. However, if no errors are present, the source is listed nonetheless. |
| edit in source | 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 `$EDITOR` environment variable. |
| vi in source | Embeds the error messages in the file at the positions in which they occur. The file is subsequently opened in the **`vi(1)`** editor. |

See also:

- *MAXAda Reference Manual* (0890516)

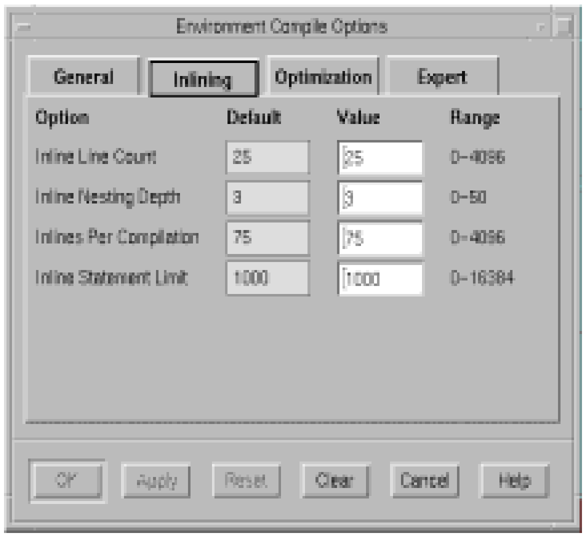
### Quiet Compiler Info Msgs.

Generation of info messages is suppressed by the compiler if this option is checked. When checked, this applies to all units in the current environment.

### 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. When checked, this applies to all units in the current environment.

## Inlining



| Option | Default | Value | Range |
|---|---|---|---|
| Inline Line Count | 25 | 25 | 0–4096 |
| Inline Nesting Depth | 3 | 3 | 0–50 |
| Inlines Per Compilation | 75 | 75 | 0–4096 |
| Inline Statement Limit | 1000 | 1000 | 0–16384 |

**Figure 5-2. Environment Compile Options dialog - Inlining Page**

### 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.

### 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.

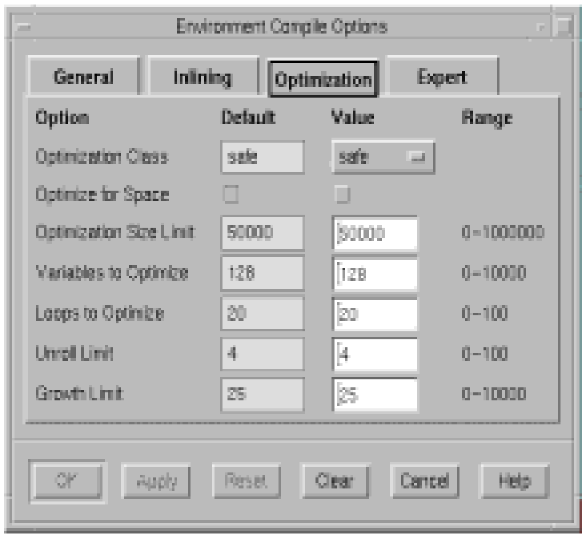
### Inlines Per Compilation

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.

### 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.

## Optimization



| Option | Default | Value | Range |
|---|---|---|---|
| Optimization Class | safe | safe | |
| Optimize for Space | ☐ | ☐ | |
| Optimization Size Limit | 50000 | 50000 | 0-1000000 |
| Variables to Optimize | 128 | 128 | 0-10000 |
| Loops to Optimize | 20 | 20 | 0-100 |
| Unroll Limit | 4 | 4 | 0-100 |
| Growth Limit | 25 | 25 | 0-10000 |

**Figure 5-3. Environment Compile Options dialog - Optimization Page**

### 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).

### 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`.

### 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.

### 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.

### 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.

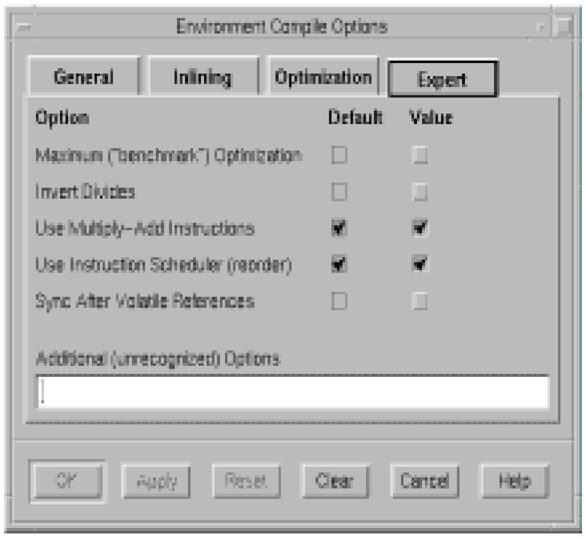
### Unroll Limit

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.

### 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.

## Expert



| Option | Default | Value |
|---|---|---|
| Maximum ("benchmark") Optimization | ☐ | ☐ |
| Invert Divides | ☐ | ☐ |
| Use Multiply–Add Instructions | ✓ | ✓ |
| Use Instruction Scheduler (reorder) | ✓ | ✓ |
| Sync After Volatile References | ☐ | ☐ |

Additional (unrecognized) Options

**Figure 5-4. Environment Compile Options dialog - Expert Page**

### 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.

### 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.

### 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.

### Use Instruction Scheduler (reorder)

The Instruction Scheduler attempts to fill the instruction pipeline by reordering instructions.

### Sync After Volatile References

Before any reference to a volatile object, an `eieio` instruction is inserted to enforce in-order execution of I/O.

### Additional (unrecognized) Options

Any other options not explicitly referenced by the Environment Compile Options Editor may be specified here.

## Unit Compile Options Editor

The Unit Compile Options Editor contains the following four pages:

- General (see page 5-15)
- Inlining (see page 5-19)
- Optimization (see page 5-21)
- Expert (see page 5-23)

### 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:

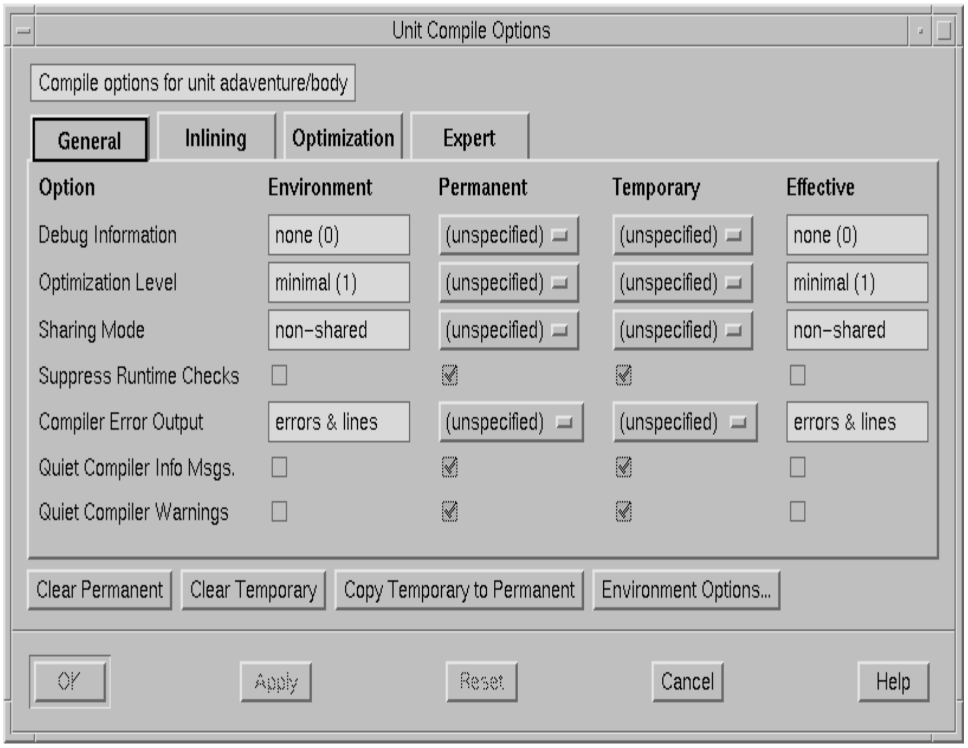
- “Environment Compile Options Editor” on page 5-3

In addition, “Tri-state Checkboxes” on page 5-1 discusses the various states in which the checkboxes in this dialog may exist.

See also:

- “Unit Compile Options Editor” on page 5-13

## General


Unit Compile Options

Compile options for unit adaventure/body

| Option | Environment | Permanent | Temporary | Effective |
|---|---|---|---|---|
| Debug Information | none (0) | (unspecified) | (unspecified) | none (0) |
| Optimization Level | minimal (1) | (unspecified) | (unspecified) | minimal (1) |
| Sharing Mode | non–shared | (unspecified) | (unspecified) | non–shared |
| Suppress Runtime Checks | ☐ | ✓ | ✓ | ☐ |
| Compiler Error Output | errors & lines | (unspecified) | (unspecified) | errors & lines |
| Quiet Compiler Info Msgs. | ☐ | ✓ | ✓ | ☐ |
| Quiet Compiler Warnings | ☐ | ✓ | ✓ | ☐ |

**Figure 5-5. Unit Compile Options dialog - General Page**

### Debug Information

This option controls the level of debug information generated for units in a given environment.

| | |
|---|---|
| `none (0)` | No debug information. Debugging tools requiring line numbers or symbolic information will not fully function on modules compiled at this debug level. |
| `lines (1)` | 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. |
| `full (2)` | 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)`.

### 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`.

### Sharing 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 sharing 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. 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` (or `both`) is considered a `shared` unit. All shared units must be placed in a shared object to be used

### Suppress Runtime Checks

This option gives permission to the implementation to suppress all run-time checks.

Its effects are equivalent to a list of `SUPPRESS` pragmas, each naming a different check.

### 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. |
| list in source! | Displays the error messages in the file in the same manner as the list in source option. However, if no errors are present, the source is listed nonetheless. |
| edit in source | 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 `$EDITOR` environment variable. |
| vi in source | Embeds the error messages in the file at the positions in which they occur. The file is subsequently opened in the **`vi(1)`** editor. |

See also:

- *MAXAda Reference Manual* (0890516)

### Quiet Compiler Info Msgs.

Generation of info messages for the selected units is suppressed by the compiler if this option is checked.

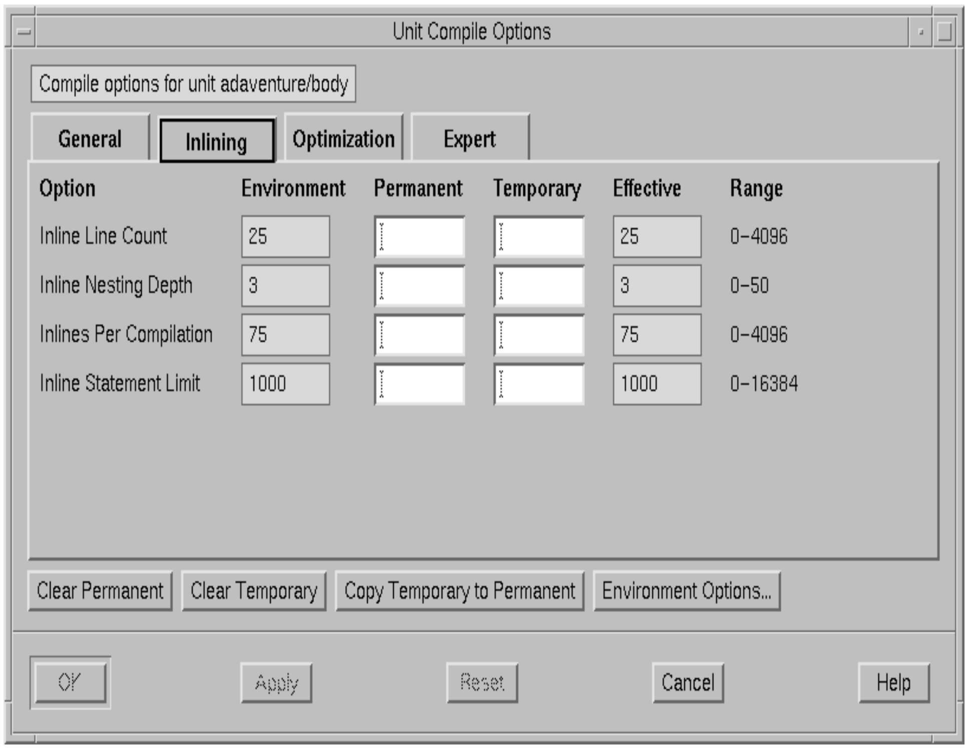
This is usually selected for individual units which generate informational messages that the user desires to ignore.

### 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.

## Inlining



Compile options for unit adaventure/body

| Option | Environment | Permanent | Temporary | Effective | Range |
|---|---|---|---|---|---|
| Inline Line Count | 25 | | | 25 | 0–4096 |
| Inline Nesting Depth | 3 | | | 3 | 0–50 |
| Inlines Per Compilation | 75 | | | 75 | 0–4096 |
| Inline Statement Limit | 1000 | | | 1000 | 0–16384 |

**Figure 5-6. Unit Compile Options dialog - Inlining Page**

### 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.

### 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.

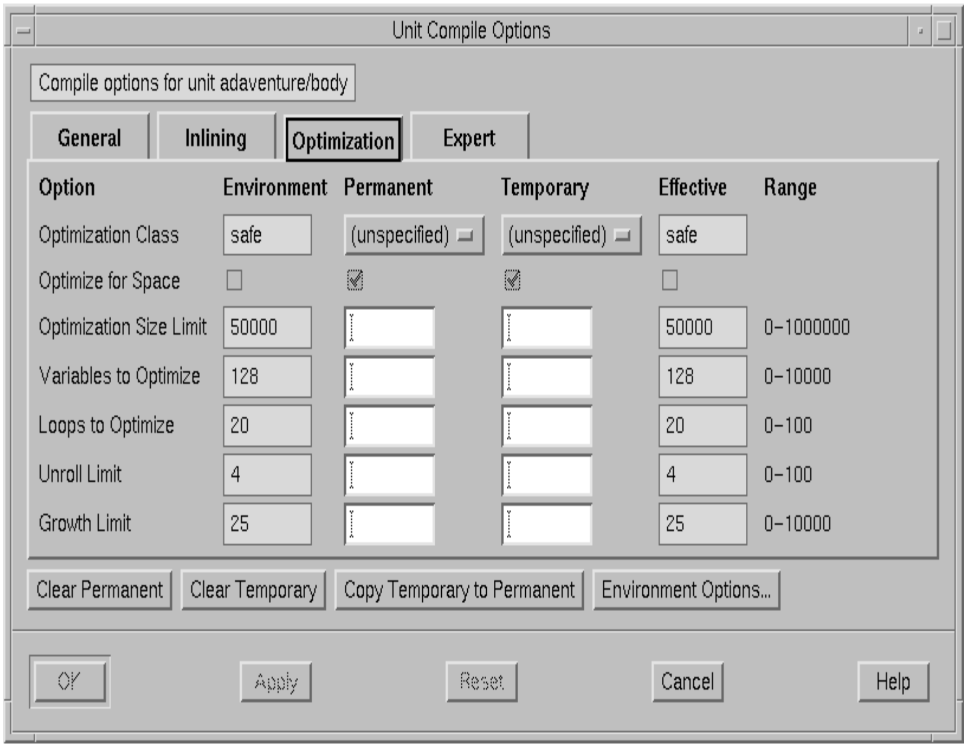
### Inlines Per Compilation

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.

### 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.

## Optimization



Compile options for unit adaventure/body

| Option | Environment | Permanent | Temporary | Effective | Range |
|---|---|---|---|---|---|
| Optimization Class | safe | (unspecified) | (unspecified) | safe | |
| Optimize for Space | ☐ | ✓ | ✓ | ☐ | |
| Optimization Size Limit | 50000 | | | 50000 | 0-1000000 |
| Variables to Optimize | 128 | | | 128 | 0-10000 |
| Loops to Optimize | 20 | | | 20 | 0-100 |
| Unroll Limit | 4 | | | 4 | 0-100 |
| Growth Limit | 25 | | | 25 | 0-10000 |

**Figure 5-7. Unit Compile Options dialog - Optimization Page**

### 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).

### Optimize for Space

A boolean value that determines whether <u>all</u> routines in a compilation will be optimized for space regardless of the values of other compiler directives. By default, this parameter is `false`.

### 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.

### 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.

### 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.

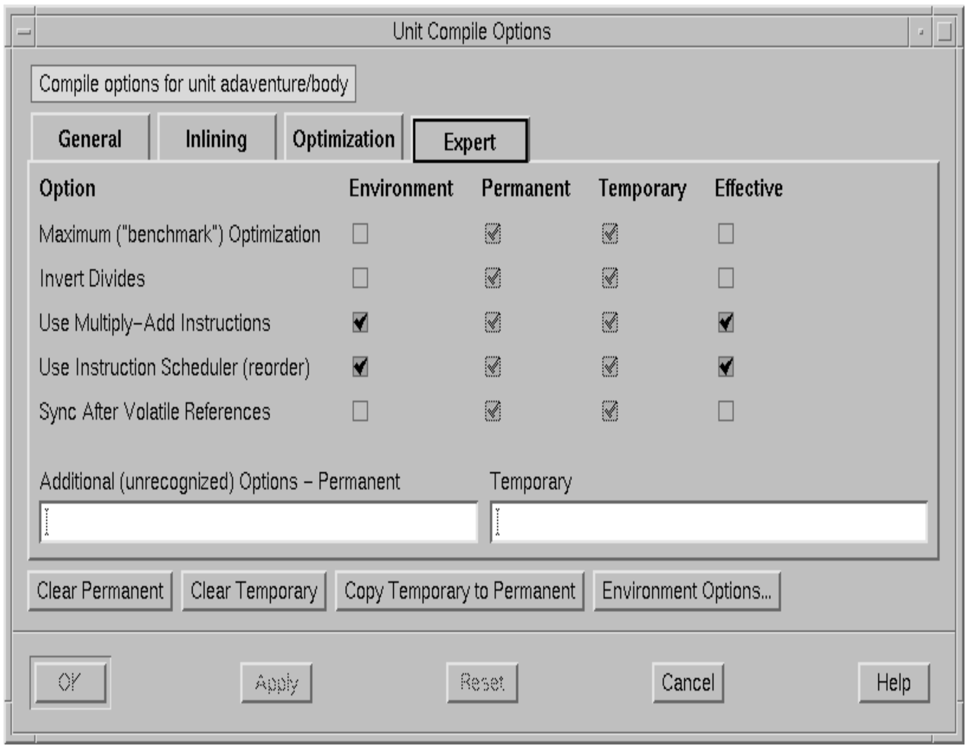
### Unroll Limit

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.

### 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.

## Expert



Compile options for unit adaventure/body

| Option | Environment | Permanent | Temporary | Effective |
|---|---|---|---|---|
| Maximum ("benchmark") Optimization | ☐ | ✓ | ✓ | ☐ |
| Invert Divides | ☐ | ✓ | ✓ | ☐ |
| Use Multiply–Add Instructions | ✓ | ✓ | ✓ | ✓ |
| Use Instruction Scheduler (reorder) | ✓ | ✓ | ✓ | ✓ |
| Sync After Volatile References | ☐ | ✓ | ✓ | ☐ |

Additional (unrecognized) Options – Permanent

Temporary

**Figure 5-8. Unit Compile Options dialog - Expert Page**

### 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.

### 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.

### 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.

### Use Instruction Scheduler (reorder)

The Instruction Scheduler attempts to fill the instruction pipeline by reordering instructions.

### Sync After Volatile References

Before any reference to a volatile object, an `eieio` instruction is inserted to enforce in-order execution of I/O.

### 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.

# 6
# NightBench Builder

## Overview

The NightBench Builder is the mechanism used for compilation and linking of Ada units and partitions. It can be used in conjunction with the Ada Environment 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 6-9)
- Targets (see page 6-14)
- Settings (see page 6-16)
- Notification (see page 6-18)
- Expert (see page 6-20)

At the top of every page of the NightBench Builder window is listed:

#### 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 6-8
- "Tools Menu" on page 1-17
- "Help Menu" on page 1-19

## Builder Menu

The Builder menu appears on the Builder menu bar.

See also:

- "Builder Menu Bar" on page 6-2

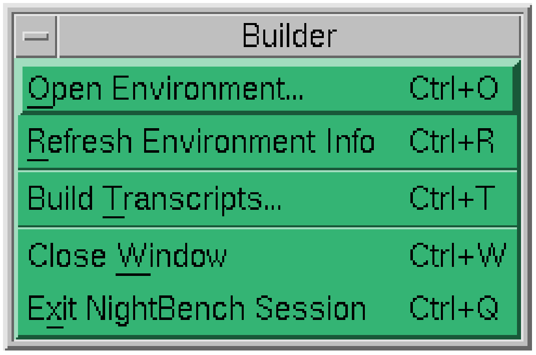

Builder
Open Environment... Ctrl+O
Refresh Environment Info Ctrl+R
Build Transcripts... Ctrl+T
Close Window Ctrl+W
Exit NightBench Session Ctrl+Q

**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 6-5

#### 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 utilites.

## 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:

- "Overview" on page 6-1
- "Builder Menu" on page 6-2
- "Build" on page 6-9

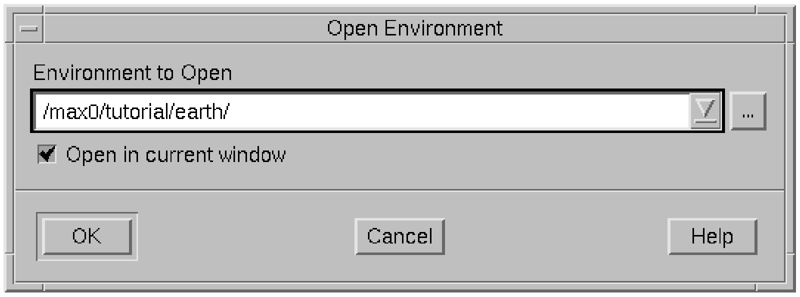

Environment to Open
/max0/tutorial/earth/
☑ Open in current window

**Figure 6-2. Open Environment dialog**

#### Environment to Open

The environment which is 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.

### 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.

## Build Transcripts

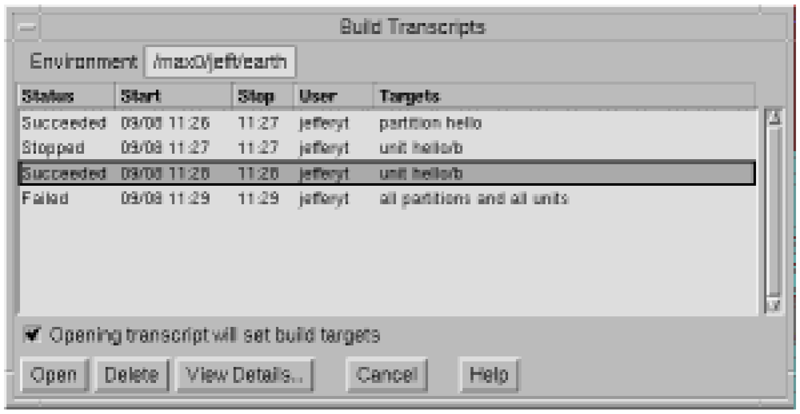
The Build Transcripts dialog lists the transcripts of previous builds that have occurred in the current environment. This dialog allows the user to view details regarding a particular transcript and also permits the user to delete selected transcripts.

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 and 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:

- "Preferences" on page 1-7
- "Builder Menu" on page 6-2



Environment /max0/jeff/earth

| Status | Start | Stop | User | Targets |
|---|---|---|---|---|
| Succeeded | 09/08 11:26 | 11:27 | jefferyt | partition hello |
| Stopped | 09/08 11:27 | 11:27 | jefferyt | unit hello/b |
| Succeeded | 09/08 11:28 | 11:28 | jefferyt | unit hello/b |
| Failed | 09/08 11:29 | 11:29 | jefferyt | all partitions and all units |

☑ Opening transcript will set build targets

**Figure 6-3. Build Transcripts dialog**

### 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.

### Opening transcript will set build targets

The targets on the Build and Targets page are set as specified by this transcript. This option is selected by default.

See also:

- "Build" on page 6-9
- "Targets" on page 6-14

### Open

When a transcript is opened, it is displayed 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.

See also:

- “Build” on page 6-9

### 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.

## Options Menu

The Options menu appears on the Builder menu bar.

See also:

- "Builder Menu Bar" on page 6-2

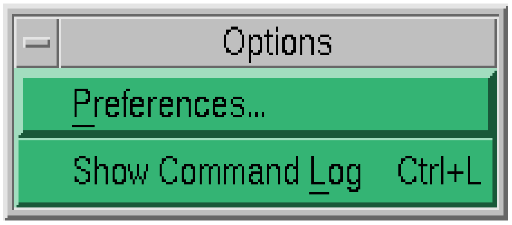


**Figure 6-4. Options menu**

### 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 1-7

### Show Command Log

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

See also:

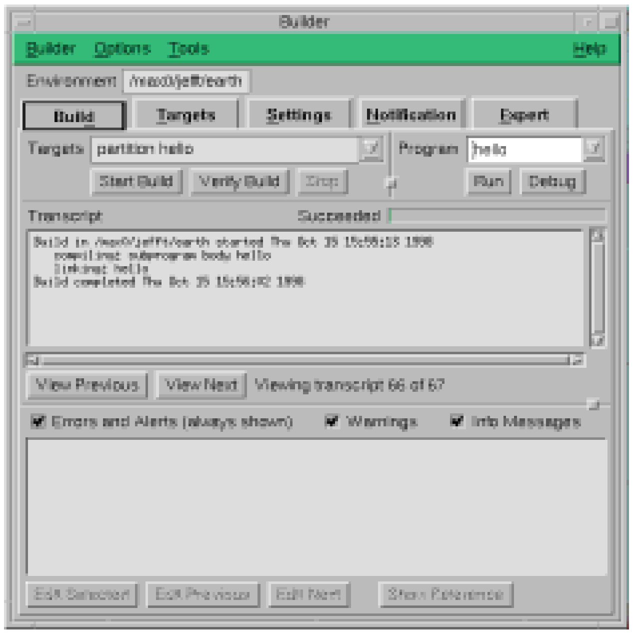
- *MAXAda Reference Manual* (0890516)
- "Command Log" on page 1-13

## 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:

- "Overview" on page 6-1


Builder
Builder Options Tools Help
Environment /mzx0/jeffl/earth
Build Targets Settings Notification Expert
Targets partition hello
Program hello
Start Build Verify Build Stop Run Debug
Transcript Succeeded
View Previous View Next
Errors and Alerts (always shown) Warnings Info Messages
Edit Selected Edit Previous Edit Next Show Reference

```
Build in /mzx0/jeffl/earth started Thu Oct 15 15:58:15 1998
   compiling: subprogram body hello
   linking: hello
Build completed Thu Oct 15 15:58:32 1998
```

Viewing transcript 66 of 67

**Figure 6-5. Builder window - 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:

- "Targets" on page 6-14

### 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.

### Program

The name of the executable program to Run or Debug.

### Run

Executes the specifed Program in a terminal window.

See also:

- “Preferences” on page 1-7

### Debug

Runs the specified Program under the NightView debugger.

See also:

- *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 6-5

### View Previous

This button allows the user to view the previous build transcript of those that are saved.

See also:

- “Build Transcripts” on page 6-5

### 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 6-5

### Viewing transcript ... of ...

This area specifies which transcript in the sequence of saved transcripts is currently displayed.

See also:

- “Build Transcripts” on page 6-5

### Errors and Alerts (always shown)

Errors and alerts occurring during the build are always shown in the Errors Window.

### 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:

- "Environment Compile Options - General" on page 5-5
- "Unit Compile Options - General" on page 5-15

### 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:

- "Environment Compile Options - General" on page 5-5
- "Unit Compile Options - General" on page 5-15

### Errors Window

The Errors Window shows errors from the Builder only. Specific error messages from the compiler or linker are sent to the Build Transcript window.

### Edit Selected

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

See also:

- "Preferences" on page 1-7

### 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" on page 1-7

### 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” on page 1-7

### Show Reference

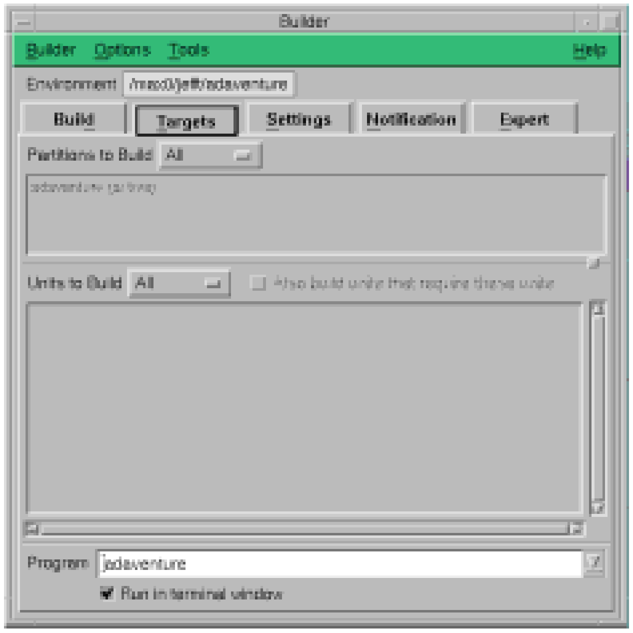
Opens the HyperHelp online Ada 95 Reference Manual or MAXAda Reference Manual to the location specified by the selected error, warning or informational message. If the selected error, warning, or informational message lists more than one reference, pressing the Show Reference button repeatedly cycles through the references.

## 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:

- "Overview" on page 6-1


Builder
Builder Options Tools Help
Build Targets Settings Notification Expert

Environment /mao0/jeff/adaventure

Partitions to Build All

adaventure (active)

Units to Build All

☐ Also build units that require these units

Program adaventure

✓ Run in terminal window

**Figure 6-6. Builder window - 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.

### Program

The name of the executable to be used by the Run and Debug buttons on the Build page.

See also:

- “Build” on page 6-9

### Run in terminal window

When the executable is Run from the Build page, a terminal window is opened for its execution.

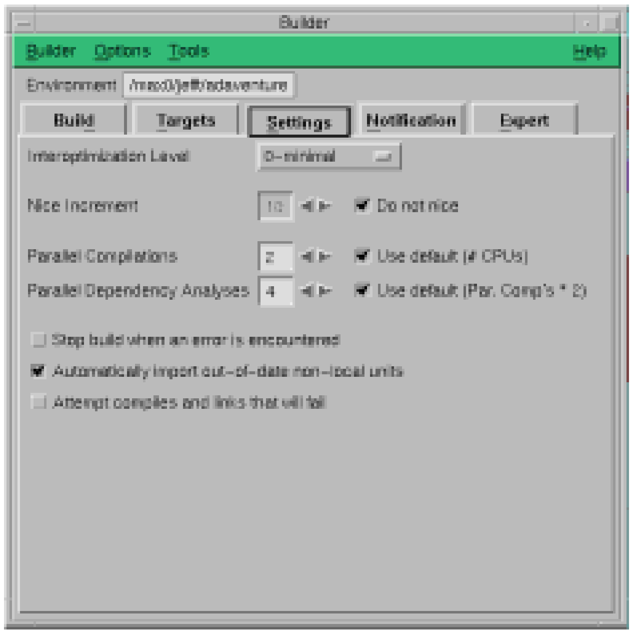
See also:

- “Build” on page 6-9

## Settings

See also:

- "Overview" on page 6-1



Environment /mxo0/jeffb/adaventure

Interoptimization Level 0–minimal

Nice Increment 10 ✓ Do not nice

Parallel Compilations 2 ✓ Use default (# CPUs)

Parallel Dependency Analyses 4 ✓ Use default (Par. Comp's * 2)

☐ Stop build when an error is encountered

✓ Automatically import out–of–date non–local units

☐ Attempt compiles and links that will fail

Figure 6-7. Builder window - 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.

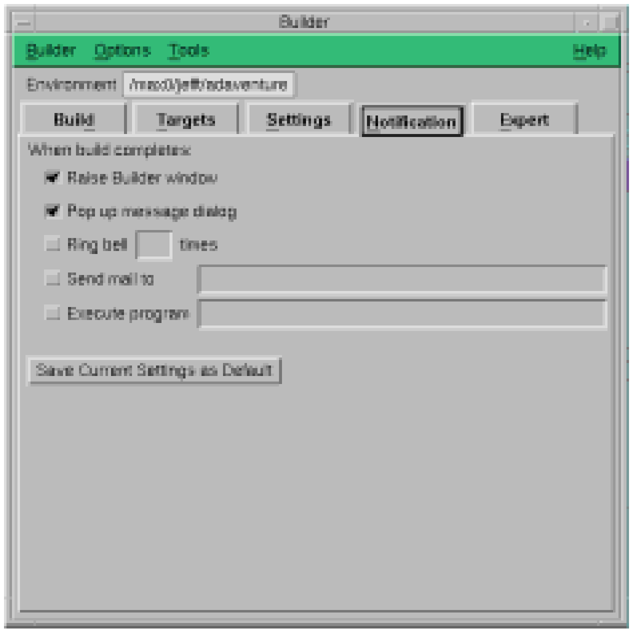
The default for this checkbox is unchecked.

## 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:

- "Overview" on page 6-1
- "Build" on page 6-9



Environment /mxp0/jeffb/adaventure

When build completes:

- [✓] Raise Builder window
- [✓] Pop up message dialog
- [ ] Ring bell ___ times
- [ ] Send mail to
- [ ] Execute program

**Figure 6-8. Builder window - 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).

### Execute program

This program will run at the completion of the build.

### Save Current Settings as Default

The current settings may be made the default notification operations by using this button.

## Expert

See also:

- "Overview" on page 6-1

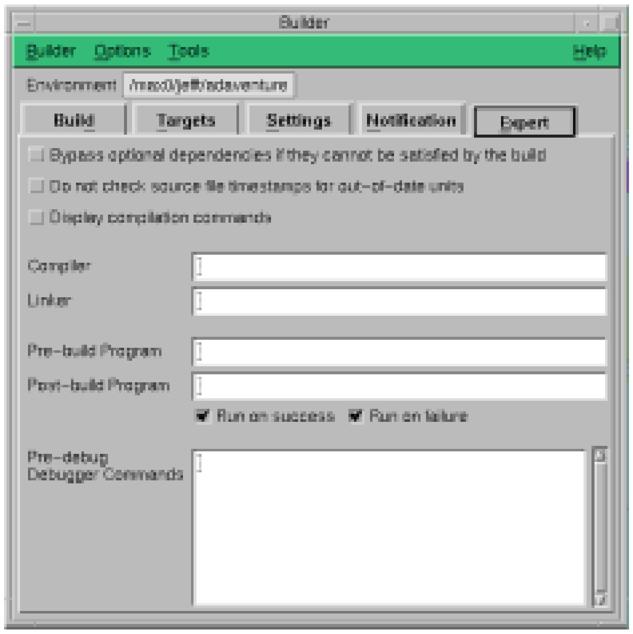

Builder
Builder
Options
Tools
Help
Environment
/mxx0/jeffb/adaventure
Build
Targets
Settings
Notification
Expert
Bypass optional dependencies if they cannot be satisfied by the build
Do not check source file timestamps for out-of-date units
Display compilation commands
Compiler
Linker
Pre-build Program
Post-build Program
Run on success
Run on failure
Pre-debug Debugger Commands

**Figure 6-9. Builder window - 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:

- "Settings" on page 6-16

### 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 MAXAda commands used for compilation and linking in the Transcript window of the Builder.

See also:

- "Build" on page 6-9
- *MAXAda Reference Manual* (0890516)

### 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 6-12

#### Run on failure

Execute the Post-build Program only if there are errors in the Errors Window.

See also:

- "Errors Window" on page 6-12

### Pre-debug Debugger Commands

Enter commands to be sent to the NightView debugger prior to running the executable when the user presses the `Debug` button on the Build page.

See also:

- "Build" on page 6-9
- *NightView User's Guide* (0890395)

# 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 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 sharing mode `non-shared` (implying the use of static code).

**archive link method**

A link method which instructs the linker to search all archive partitions for included units.

**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. Also, the unit's options may have not changed so as to make it inconsistent.

**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 *Ada 95 Reference Manual* for a complete definition.

**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.

**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. A generic unit is a template, which can be parameterized, and from which corresponding (nongeneric) subprograms or packages can be obtained.

**instance**

A subprogram or package obtained as the result of instantiating a generic unit with appropriate generic actual parameters for the generic formal parameters.

**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 requried 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).

Similarily, 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.

**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 link method**

A link method which instructs the linker to use the object of a unit directly when linking.

**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.

**shared object**

A shared object is a shared collection of routines and data associated with a user's application during the link and execution phases of program generation. Shared objects are dynamically built (i.e. shared) objects that contain position independent code.

**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.

**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.

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**