

# VisualLinux User's Guide

---

Release 4.0

DOC 00-446-00

Product names mentioned in *VisualLinux User's Guide* are trademarks of their respective manufacturers and are used here for identification purposes only.

Copyright ©1987 - 2002, LynuxWorks, Inc. All rights reserved.  
U.S. Patents 5,469,571; 5,594,903

Printed in the United States of America.

All rights reserved. No part of *VisualLinux User's Guide* may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photographic, magnetic, or otherwise, without the prior written permission of LynuxWorks, Inc.

LynuxWorks, Inc. makes no representations, express or implied, with respect to this documentation or the software it describes, including (with no limitation) any implied warranties of utility or fitness for any particular purpose; all such warranties are expressly disclaimed. Neither LynuxWorks, Inc., nor its distributors, nor its dealers shall be liable for any indirect, incidental, or consequential damages under any circumstances.

(The exclusion of implied warranties may not apply in all cases under some statutes, and thus the above exclusion may not apply. This warranty provides the purchaser with specific legal rights. There may be other purchaser rights which vary from state to state within the United States of America.)

---

# Contents

---

<b>PREFACE</b>	.....	<b>XI</b>
	For More Information .....	xi
	Typographical Conventions .....	xii
	Special Notes .....	xii
	Technical Support .....	xiii
	LynuxWorks U.S. Headquarters .....	xiii
	LynuxWorks Europe .....	xiii
	World Wide Web .....	xiii
<b>CHAPTER 1</b>	<b>INTRODUCTION</b> .....	<b>1</b>
	About VisualLynux .....	1
	Cross Development Hosts and Targets .....	1
	LynuxWorks Cross Development Kits .....	2
	VisualLynux Help .....	3
	VisualLynux Objects .....	3
	Projects .....	3
	Workspaces .....	4
	Workspace and Project Directories .....	4
	Wizards .....	5
	Configurations .....	6
	Terminology .....	7
<b>CHAPTER 2</b>	<b>GETTING STARTED</b> .....	<b>9</b>
	Prerequisites .....	9
	System Requirements .....	9
	Package Contents .....	10
	Configuring VisualLynux .....	10

VisualLynux Configuration Utility .....	11
Invoking the Configuration Utility .....	11
LynxOS CDK Configuration .....	13
BlueCat Linux CDT Configuration .....	15
CYGWIN Installation .....	17
Unregistering VisualLynx 1.x .....	19
Preparing a Mount Table .....	19
Launching the VisualLynux Application and Kernel Wizards .....	20
Overview of Project Creation Process .....	22
Building a Project .....	25
Debugging a Project .....	26

---

<b>CHAPTER 3</b>	<b>VISUALLYNUX BUILD PROCESS.....</b>	<b>27</b>
	Building Project Targets .....	27
	Processing Files .....	28
	Project-Level Build Steps .....	29
	Build Macros .....	30
	Global Macros .....	31
	Project (Configuration)-Level Macros .....	31
	File-Level Macros .....	33

---

<b>CHAPTER 4</b>	<b>VISUALLYNUX MENUS.....</b>	<b>35</b>
	Overview of Menu Structure .....	35
	VisualLynux Menus .....	35
	File Menu .....	36
	Edit Menu .....	36
	View Menu .....	37
	Insert Menu .....	38
	Project Menu .....	38
	Build Menu .....	39
	Tool Menu .....	40
	Window Menu .....	42

---

<b>CHAPTER 5</b>	<b>VISUALLYNUX APPLICATION WIZARD.....</b>	<b>43</b>
	Launching the VisualLynux Application Wizard .....	43
	Selecting a VisualLynux Project Type .....	44
	C User Application Wizard .....	45
	A Custom C LynuxWorks Application .....	47

---

Component Functions .....	49
C++ User Application Wizard .....	52
X/Motif Graphical Application Wizard .....	54
A Custom X/Motif Application .....	56
Device Driver Wizard .....	58
LynxOS Device Drivers .....	59
BlueCat Linux Device Drivers .....	64
Static Library Application Wizard .....	77

---

<b>CHAPTER 6</b>	<b>LYNXOS KERNEL WIZARD .....</b>	<b>79</b>
	Kernel Projects .....	79
	Launching the VisualLynux LynxOS Kernel Wizard .....	80
	Selecting a Board Support Package .....	81
	Preparing a Kernel Executable Project .....	83
	Configuring Kernel Downloadable Image (KDI) .....	86
	Configuring KDI File System .....	88
	Selecting Kernel Library Projects .....	94
	Copying Source Files .....	96

---

<b>CHAPTER 7</b>	<b>VISUALYNUX ADDIN TOOLS .....</b>	<b>101</b>
	IDE Tools .....	101
	Toolbar Icons .....	102
	VisualLynux Import Wizard .....	104
	Import a Makefile .....	104
	Import Wizard Initial Screen .....	105
	Selecting a Makefile to Import .....	105
	Specifying Project Configurations .....	106
	Specifying Build Environment .....	107
	Adding/Editing Environment Variables .....	108
	Adding Source Files to a Project .....	109
	Specifying Build Target .....	110
	VisualLynux Export Wizard .....	111
	Export to Makefile .....	111
	Export Wizard Initial Screen .....	111
	Setting Configurations to Export .....	112
	Export Directories Setup .....	113
	Specifying Directories for New Source Tree .....	114
	Specifying Names of Exported Makefiles .....	115

Project Files to Copy .....	116
Summary Dialog Box .....	117
LynuxWorks GDB .....	118
Total/db .....	120
Upload .....	124
bash Shell .....	124
Telnet .....	125
VisualLynux HTML Help .....	126
Other VisualLynux Tools .....	126

---

<b>CHAPTER 8</b>	<b>VISUALLYNUX PROJECT SETTINGS .....</b>	<b>127</b>
	Overview .....	127
	Project-Level Property Pages .....	129
	File-Level Property Pages .....	130
	General Page .....	131
	Debug Page .....	133
	Browser Page .....	134
	Library Page .....	135
	Pre-Link Step Page .....	136
	Post-Build Step Page .....	137
	Custom Build Steps .....	138
	Project Custom Build .....	138
	Directory Menu .....	140
	File Menu .....	141
	File Custom Build .....	142
	Directory Menu .....	144
	File Menu .....	145
	Compiler Options .....	145
	C/C++ Page .....	145
	C/C++ General .....	148
	C/C++ Code Generation .....	149
	C/C++ Language .....	152
	C/C++ Listings .....	154
	C/C++ Warnings .....	155
	C/C++ Profiling .....	158
	C/C++ Optimization .....	159
	C/C++ Preprocessor .....	164
	C/C++ Special Options .....	165

---

Linker Options .....	166
Linker Option Categories .....	166
Link General .....	167
Link Customize .....	168
Link Debug .....	170
Link Input .....	171
Link Output .....	172
Link Special Options .....	174
Make Page .....	174
Make Project .....	174
General Options .....	175
Environment Options .....	175
Make File .....	177
Kernel Options .....	178
Kernel General Page .....	180
Kernel Configuration Page .....	181
Editing Item Properties .....	183
Kernel Parameters Page .....	192

---

<b>CHAPTER 9</b>	<b>LYNXWORKS FTP .....</b>	<b>195</b>
	Overview .....	195
	LynuxWorks FTP Main Window .....	196
	LynuxWorks FTP Functions .....	197
	Connecting to an FTP Server .....	197
	LynuxWorks FTP Command Line .....	198
	Browsing FTP Folders .....	199
	Keyboard Controls .....	200
	Mouse Controls .....	201
	Menu and Toolbar Commands .....	201
	Displaying Contents of FTP Folders .....	202
	Manipulating FTP Folders .....	202
	Removing an FTP Folder .....	203
	Renaming a Folder .....	203
	Creating an FTP Folder .....	203
	Displaying FTP Folder Properties .....	204
	Manipulating FTP Files .....	204
	Removing FTP Files .....	204
	Renaming an FTP File .....	204
	Displaying File Properties .....	205
	Browsing Local Folders .....	205

Opening Folders .....	205
Displaying the Parent Folder .....	206
Changing the View Mode .....	206
Manipulating Files on a Local Computer .....	207
Creating a New Folder .....	207
Renaming Files or Folders .....	207
Displaying File or Folder Properties .....	208
Removing Files and Folders .....	209
Downloading Files and Folders .....	210
Downloading Files .....	210
Downloading Folders .....	210
Uploading Files and Folders .....	211
Transfer Dialog Box .....	211

---

**CHAPTER 10 VISUALLYNUX TARGET ADMINISTRATION..... 215**

Overview .....	215
Target Configuration .....	216
Invoking Target Administration .....	216
Target Administration Dialog Box .....	217
Target Wizard .....	220
General Information .....	220
Target Identification Step .....	221
Network Connection Step .....	222
Authentication Step .....	224
Utilities Step .....	226
Projects Directory Step .....	228
Debugging Step .....	231
Process Viewer Step .....	233
Target Administration Tasks .....	234
Creating a New Target .....	235
Editing Target Properties .....	235
Removing a Target .....	236
Setting a Default Target .....	236
Specifying Target Selection Mode .....	237
Renaming a Target .....	238
Running Commands on the Target .....	238
Run Command Dialog Box .....	239

---

**CHAPTER 11 LYNEXWORKS CROSS PROCESS VIEWER ..... 241**

Overview .....	241
----------------	-----



---

The CPV Window .....	242
CPV Window Panes .....	243
File Menu .....	245
View Menu .....	247
Options Menu .....	248
Performance Chart Options .....	248
Add to Counters .....	249
Processes Options .....	250
LynxOS Processes Options .....	250
BlueCat Linux Processes Options .....	254
Sort Options .....	256
Help Menu .....	258
Cross Process Viewer Toolbar Buttons .....	258

---

<b>CHAPTER 12</b>	<b>BOOTP-TFTP-PFTP UTILITY .....</b>	<b>261</b>
	BTP Overview .....	261
	BTP Servers .....	261
	Starting BTP .....	262
	BTP Dialog Boxes .....	263
	General BTP Page .....	263
	Bootp Page .....	264
	Tftp Page .....	265
	Pftp Page .....	267
	Bootp DB Entry .....	268
	BTP Tray Icon .....	270

---

<b>INDEX .....</b>	<b>271</b>
--------------------	------------



---

# — *Preface*

---

## For More Information

For more information on the features of VisualLynux, refer to the following printed and online documentation:

- *Release Notes*

This printed document contains late-breaking information about the current release of VisualLynux.

- Online information

The VisualLynux CD-ROM contains an *Online Help System* that includes an online version of this document, man pages for LynxOS and BlueCat Linux, and documentation for GNU tools.

## Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case sensitive and should be typed accurately.

Kind of Text	Examples
Body text; <i>italicized</i> for emphasis, new terms, and book titles	Refer to the <i>LynxOS User's Guide</i> .
Environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data	<code>ls</code> <code>-l</code> <code>myprog.c</code> <code>/dev/null</code>
Commands that need to be highlighted within body text, or commands that must be typed as is by the user are <b>bolded</b> .	<code>login: <b>myname</b></code> <code># <b>cd /usr/home</b></code>
Text that represents a variable, such as a file name or a value that must be entered by the user	<code>cat <i>filename</i></code> <code>mv <i>file1 file2</i></code>
Blocks of text that appear on the display screen after entering instructions or commands	<pre>Loading file /tftpboot/shell.kdi into 0x4000 ..... File loaded. Size is 1314816 Copyright 2000 LynuxWorks, Inc. All rights reserved.  LynxOS (ppc) created Mon Jul 17 17:50:22 GMT 2000 user name:</pre>
Keyboard options, button names, and menu sequences	<b>Enter</b> , <b>Ctrl-C</b>

---

## Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

---

**NOTE:** These callouts note important or useful points in the text.

---



**CAUTION!** Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

---

---

## Technical Support

LynuxWorks Technical Support is available Monday through Friday (holidays excluded) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The LynuxWorks World Wide Web home page provides additional information about our products.

### LynuxWorks U.S. Headquarters

Internet: [support@lnxw.com](mailto:support@lnxw.com)

Phone: (408) 979-3940

Fax: (408) 979-3945

### LynuxWorks Europe

Internet: [tech\\_europe@lnxw.com](mailto:tech_europe@lnxw.com)

Phone: (+33) 1 30 85 06 00

Fax: (+33) 1 30 85 06 06

### World Wide Web

<http://www.lynuxworks.com>



---

## About VisualLynux

VisualLynux is a cross development tool with a graphical user interface (GUI) expands the functionality of Microsoft Visual Studio C++ to provide users the ability to build and debug real-time embedded applications on Windows hosts for target platforms running the LynxOS or BlueCat Linux operating systems from LynuxWorks, Inc.

*Cross development* is required and optimal in situations where targets do not have the necessary development resources. The building and debugging of applications is conducted mainly on a *host*, connected to a target via a network or serial port.

LynuxWorks supports cross development in the form Cross Development Kits (or *CDKs*, for LynxOS) and Cross Development Tools (or *CDTs*, for BlueCat Linux). A CDK is a LynuxWorks configuration for a particular target and host platform. (See “Terminology” on page 7 for definitions.)

---

**NOTE:** Unless otherwise stated, the term *CDK* refers to both CDKs and CDTs.

---

VisualLynux functions are invoked using the Visual C++ main menu and the VisualLynux toolbar. The VisualLynux toolbar is installed the first time Visual C++ is invoked after VisualLynux installation.

---

**NOTE:** The installation procedure is described in the *VisualLynux Release Notes*.

---

## Cross Development Hosts and Targets

A *target*, for this purpose, is a combination of hardware (computer and input/output devices) and software (operating system and applications) that is

intended to perform a specific function. The *target platform* refers to the architecture type, e.g., x86, PowerPC, MIPS, SH3, ARM, XScale.

Embedded target system requirements differ significantly from those of conventional operating systems in several ways. Embedded systems deployed for a specific function, often in a time-critical environment, require increased reliability, small footprint, small kernel overhead, and support for real-time applications.

Visual Studio and VisualLynux, working together, address these needs and the associated accelerated development cycles. LynuxWorks operating systems are specially designed for embedded target systems. LynxOS is used for targets whose real-time requirements are very critical, while the BlueCat Linux is used for targets where the compatibility with Linux (especially the kernel) is more essential than real-time performance.

LynxOS and BlueCat Linux configurations called Board Support Packages (BSPs) are available for several boards within each target platform category.

## LynuxWorks Cross Development Kits

A CDK for a particular target configuration and host platform includes the specific BSP, as well as cross development tools for the host platform.

In addition to the powerful VisualLynux GUI and tools, a CDK itself can be used for cross development in command line mode. You can invoke an MS-DOS window, start bash, and run the SETUP script to create a LynxOS or BlueCat Linux command line environment.

VisualLynux leverages the Microsoft Visual Studio Development System and LynuxWorks CDKs in the following ways:

- It expands the Microsoft Visual C++ GUI and environment to build LynxOS and BlueCat Linux applications, libraries, drivers, and kernels
- It provides the Administration utility to configure and edit target descriptions, automating the setting or editing of development and debugging options for target applications
- It easily transfers development results to the target and starts debugging
- It provides assistance in importing existing projects from a LynxOS or BlueCat Linux environment to the VisualLynux environment.
- It exports VisualLynux projects to UNIX-type Makefiles.
- It provides the GUI FTP tool to transfer files between the host and target



- It provides the GUI BTP utility that runs Bootp, Tftp, and Pftp servers on the host to boot targets via TCP/IP or a parallel port.
- It provides a bash shell if the user needs a command line environment
- It provides a Cross Process Viewer (CPV) to dynamically watch processes and threads on the target
- It provides the detailed Online Help Library

## VisualLynux Help

The *VisualLynux Online Help Library* is an essential resource using LynuxWorks products. Relevant *product documentation* and *reference libraries* are included.

You can navigate online help in either of two ways:

- By using the **Table of Contents** on the left to browse the library, or
- By using the **Index** to search for topics
- By using **Search** to find information by keyword

---

## VisualLynux Objects

### Projects

A key VisualLynux concept is that of a *project*, which is a basic unit of development. VisualLynux projects are created by VisualLynux wizards. (See “Wizards” on page 5.) A VisualLynux project has the same features as Visual C++ project with specific additions and limitations.

VisualLynux projects can be of different types, having one or more goals. Usually, a project has one main *goal* or result (e.g. the program executable) and some secondary goals (e.g. object files). The user can add additional goals to the project. *The main goal defines the type of project.*

The main goal also determines the main *tool* used for the project: the main tool for a C User Application project is the `gcc.exe` cross compiler. A project can have several associated tools: a Static Library project has two tools: the `gcc.exe` compiler to build object files, and the `ar.exe` archiver to build a static library from the object files.

Each project contains *files* that are used to build the project goal(s). The user can edit these files with the Visual C++ editor, or delete or add new files. A project also contains a special group of VisualLynx files. For example, the `.v1m` files contain information about project options and tools. *These files should not be edited or deleted.*

## Workspaces

A group of projects (or a single project) constitutes a *workspace*. A workspace has two main functions:

- It represents several projects as a group in the Visual C++ Workspace window. It allows quick switching between projects in the same group.
- It controls dependencies between projects.

A new project can be created in a new workspace. A project can also be created in an existing workspace. An empty new workspace can be created using the **File/New** menu item. Only one workspace can be active (open) at a given moment. The Visual C++ workspace window displays all projects in the workspace.

The *current project* is highlighted in the Visual C++ workspace window. When you start a building process, Visual C++ builds only the current process. If it depends on other processes in the workspace, first the processes on which it is dependent is built before the current process is built.

A workspace is described by a `.dsw` file. A project is described by a `.dsp` file. These files have formats defined by Visual C++ and *should not be edited*.

---

**NOTE:** VisualLynx 4.0 use a different format for its internal project files from VisualLynx 1.x. If a project created in VisualLynx 1.x is opened (using the File/Open workspace menu item), a warning message appears, explaining how to run the `VLConvert.exe` utility to convert internal file formats.

---

## Workspace and Project Directories

The `.dsw` file resides in the *workspace directory*. The `.dsp` file and other VisualLynx project files reside in the *project directory*. The user source and data files are also usually created in the project directory. While it is recommended that all project files are managed within the same directory, the user can create files in other directories or add files from other directories to the project.

The first project of a workspace is usually created in the workspace directory; in this case the workspace directory and the project directory are the same. All subsequent projects are created in a directory specified by the user, usually a subdirectory of the workspace directory. *The name of the project directory becomes the name of the project.*

The *current directory* is where the build process is initiated. In general, this is the project directory of the project being built. If the current project depends on other projects in the workspace, the current directory is switched to account for the project being built currently.

---

**NOTE:** VisualLynux does not support mixed workspaces, i.e., workspaces that contain both VisualLynux and standard Visual C++ projects.

---

## Wizards

A VisualLynux wizard is a special component that creates a VisualLynux project. The three VisualLynux wizards and the project types they create are listed in the table below:

**Table 1-1: Wizards and Project Types**

Wizard	Project Type	Operating System
VisualLynux Application Wizard	C User Application C++ User Application X11/Motif Application Device Driver Static Library	LynxOS, BlueCat Linux LynxOS, BlueCat Linux LynxOS LynxOS, BlueCat Linux LynxOS, BlueCat Linux
VisualLynux LynxOS Kernel Wizard	Regular LynxOS Kernel (includes Kernel Downloadable Image, KDI)	LynxOS
VisualLynux Import Wizard	VisualLynux make project	LynxOS, BlueCat Linux

A wizard displays dialog boxes that help the user create new projects (of predefined types). The user can create an empty project and add necessary files later. A project can also be created using template files and edited according to requirements. In both cases the VisualLynux wizard creates its own specific project environment depending on the project type and saves it in a `.v1m` file. This file contains information about project goals and configurations.

Never modify the `.v1m` file directly. Instead use the **Project Settings** dialog box by clicking on the **Project/Settings** menu item, or the **Project Settings** icon in the VisualLynux toolbar.

The first two wizards are invoked using the menu sequence to create a new project. The third wizard is invoked by clicking the **Import** button in the VisualLynux toolbar. The Import wizard creates a project from LynxOS or BlueCat Makefile. The VisualLynux make project differs from other VisualLynux projects in that it doesn't provide the **Project Settings** dialog box to set or modify compiler and linker options. Imported projects use the make tool that runs a LynxOS or BlueCat Linux Makefile in a particular VisualLynux/CDK environment. The user can edit the Makefile as needed.

## Configurations

A project may contain one or more configurations. The configuration defines how to build the project goal(s). The user can create and modify configurations using **Project Settings**. Project configurations may differ only in target configurations, project goals, project options, or any combination of these elements. The project configuration specifies the following information:

- The target operating system (LynxOS or BlueCat)
- The target platform (e.g. x86, PPC, ARM)
- The target object format (e.g. coff, elf, xcoff)
- Project goals
- Project options

VisualLynux wizards generate configurations during project creation. The wizards provide a list of supported target configurations, depending on the project type. The target configuration defines the target operating system and the target platform. The object format is also defined because only one object format is used per combination of target configuration and project type.

Project options determine how to build the project goals. Options are available for the project main tool used to build the main goal. There are also options for other project tools, for example, the compiler used to build object files. These options can be modified for all secondary goals, or for each secondary goal.

A project can contain several configurations with different target configurations, or (in the case of Device Driver or kernel projects) all project configurations must have the same target configuration. VisualLynux wizards usually create more than one configuration for each target configuration for Debug and Release mode.

These two configurations differ only in the options for project tools. For example, the Debug configuration adds the options `-g -D_DEBUG` for the compiler.

---

## Terminology

---

**NOTE:** Items are ordered for easy comprehension rather than alphabetically.

---

**Target** - a system (such as a telecommunication switch node or an airplane navigation device) that renders a specific function (from transmitting data over the network to controlling an airplane subsystem) - A target system includes computer hardware with input/output devices, the operating system, and applications.

**Target Platform** - The type of computer architecture used for a particular target - VisualLynux supports various target platforms (x86, PowerPC, MIPS, ARM, SH3, XScale, etc.)

**Board** - Each target platform can have several hardware configurations (combinations of motherboard, I/O devices, buses, etc.) called boards.

**BSP - Board Support Package** -

**Cross Development** - A development paradigm where applications are developed and debugged on a system (host), other than the board/platform (target) that they are deployed on. Development can be performed entirely on the host. Debugging can take place on both the host and target, with the host providing much of the user support, and the target executing a debug server program that communicates with the host debugger.

**CDK - Cross Development Kit** - A LynuxWorks deliverable for a particular target configuration and host platform - It consists of a *Board Support Package (BSP)* and cross development tools for the host platform. The tools include compilers, linkers, shells, utilities, etc. needed for embedded application development. CDKs also include the CYGWIN package, which contains common UNIX tools for Windows systems like `bash`, `make`, `cp`, `ls`, `mount`, and a dynamic link library.

**Project** - The basic unit of application development under VisualLynux consisting a main goal representing the project result

**Goal** - Central or subsidiary purpose of a project, defined in terms of a specific file, e.g., an application, driver, or object file.

**File** - Files included as part of a project (such as C or C++ source files, header files, and script and data files) to build project goals, and additional VisualLynux files containing project summaries and information.

**Workspace** - VisualLynux grouping of projects to manage switching among associated projects and dependencies between them - A workspace can contain none, one, or several projects. However, only one workspace can be active at a time. Thus, at any given point you work with a particular group of related projects.

**Project Configuration** - Information contained in a `.v1m` file pertaining to target configurations, target platform, object format, target operating system, project primary and secondary goals, and project options (tools selected for goals) - A project can contain more than one configuration.

---

## Prerequisites

Users of VisualLynux 4.0 should be familiar with the Microsoft Visual C++, the C programming language, programming in the Windows environment. They should also be familiar with the C++ programming language for developing C++ application programs for target systems.

This guide assumes that you have installed Microsoft Visual C++, the LynxOS or BlueCat Linux cross development system, and VisualLynux. (Please refer to the *LynxOS Installation Guide*, the *LynxOS Installation Guide*, or the *BlueCat Linux User's Guide* for installation instructions.)

Online help is provided with Microsoft's Visual products. For specific questions regarding the operation of the Microsoft Visual Studio 6.0 Development System and Visual C++ Version 6.0, please refer to release-specific help guides.

---

**NOTE:** This document does not cover LynxOS or BlueCat Linux cross development on a Windows machine that does not have VisualLynux installed. For additional information on using the Windows cross development system for LynxOS or BlueCat Linux, please refer to appropriate product documentation.

---

---

## System Requirements

The minimum recommended system configuration for VisualLynux is:

- Windows 98, NT, ME, or 2000
- Microsoft Visual C++, Version 6.0 (or higher), Standard Edition
- 32 MB (or more) RAM

- 250 MB of disk space (This estimate does not include the disk space required for BlueCat Linux or LynxOS cross development systems.)

VisualLynux leverages the framework of the Microsoft Visual Studio 6.0 Development System found in Visual C++, Version 6.0.

VisualLynux requires installation of any of the following versions of Microsoft Visual C++:

- Microsoft Visual C++, Version 6.0, Standard Edition
- Microsoft Visual C++, Version 6.0, Professional Edition
- Microsoft Visual C++, Version 6.0, Enterprise Edition

---

## Package Contents

The VisualLynux package contains the following components and documentation:

A software CD-ROM containing:

- VisualLynux
- *VisualLynux Online Help*
  - *VisualLynux Release Notes*
  - *VisualLynux User's Guide*
  - *LynxOS man pages*
  - *BlueCat Linux man pages*
  - *GNU documentation set*
- *License Agreement*

---

## Configuring VisualLynux

Before working with VisualLynux you need to register target Cross Development Kits (CDKs) installed on your host machine. At least one CDK must be registered before you can use VisualLynux. The registration process is performed by the [VisualLynux Configuration Utility](#) invoked during installation. You might need to run this utility from the **Start** menu in the following cases also:

- When a new CDK has been installed on your host computer



- When an existing CDK has been moved to another directory
- When an existing CDK has been uninstalled

The [VisualLynux Configuration Utility](#) collects necessary data and stores it in the Windows Registry. The data then becomes available to other VisualLynux components (VisualLynux Application wizard, VisualLynux AddIn Tools, Debugger, etc.).

## VisualLynux Configuration Utility

The VisualLynux Configuration Utility sets up necessary configuration data for VisualLynux. This includes information about supported operating systems, available Cross Development Kits (CDKs) installed on your host machine, installation directories for every CDK, and CYGWIN programs used by the build tools (compiler, linker, `make` utility, and so on).

You must also use the Configuration Utility to change the current operating system version used by VisualLynux, to remove configuration data for uninstalled CDKs, or to modify the installation directory (if a CDK has been moved).

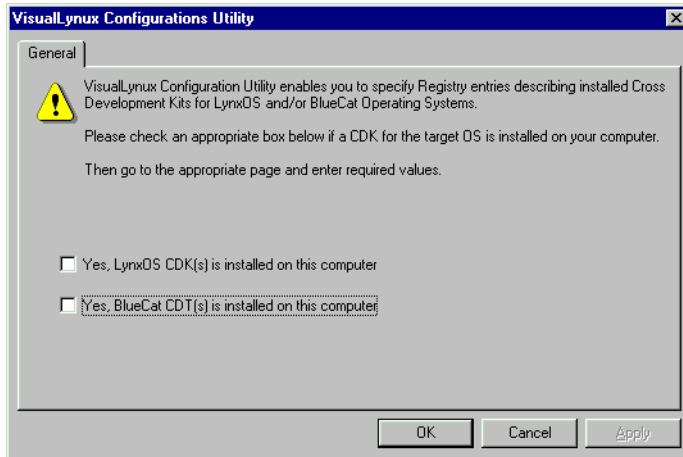
In summary, the Configuration Utility:

- Collects configuration data related to installed LynxOS CDK(s) and/or BlueCat Linux Cross Development Tools (CDTs)
- Writes collected data to the Windows Registry to make it accessible to other VisualLynux components
- Unregisters VisualLynux 1.x, if installed
- Prepares a mount table that includes mountpoints used by CDK tools

## Invoking the Configuration Utility

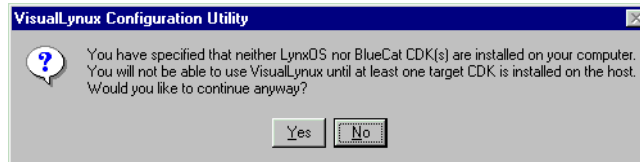
When the VisualLynux installation program first invokes the Configuration Utility, the dialog box shown in the next figure appears. Empty check boxes in the **General** page imply that the Configuration Utility has not found any configuration data in the Registry. Check an appropriate box to set up configuration data for a particular operating system.

If configuration data for an operating system has been found, the appropriate box is checked and the utility displays additional property pages.



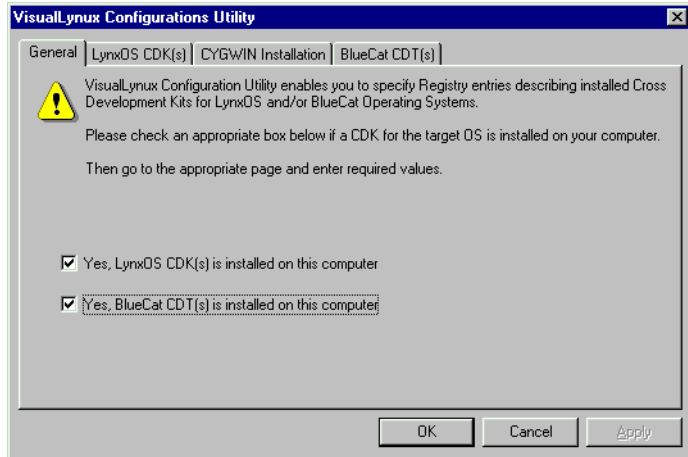
**Figure 2-1: Configuration Utility Initial Screen**

Selecting **Cancel** closes the Configuration Utility. If you click **OK** without checking any boxes, the message in the following figure appears. VisualLynux, however, will not be able to function properly.



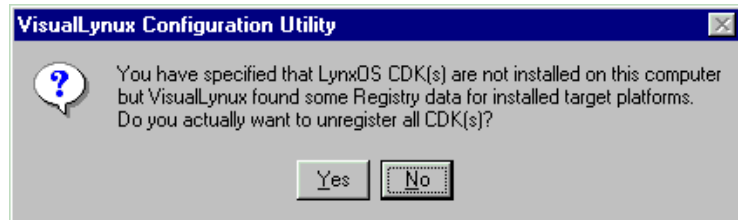
**Figure 2-2: CDK Installation Message**

Selecting **Yes** unregisters CDKs that have been previously registered; selecting **No** returns you to the previous dialog box. Depending on the boxes checked in the Configuration Utility **General** page, additional pages appear. Select an appropriate page to edit corresponding data.



**Figure 2-3: Configuration Data for Target OS**

To unregister all CDKs for a particular operating system (LynxOS or BlueCat Linux), uncheck the appropriate box. To safeguard against the accidental removal of all registration data, when you press **OK**, the Configuration Utility displays the dialog box shown below:



**Figure 2-4: Message Box to Unregister CDKs**

## LynxOS CDK Configuration

To edit configuration data for installed LynxOS CDKs, select the **LynxOS CDK(s)** property page. The following dialog box appears:

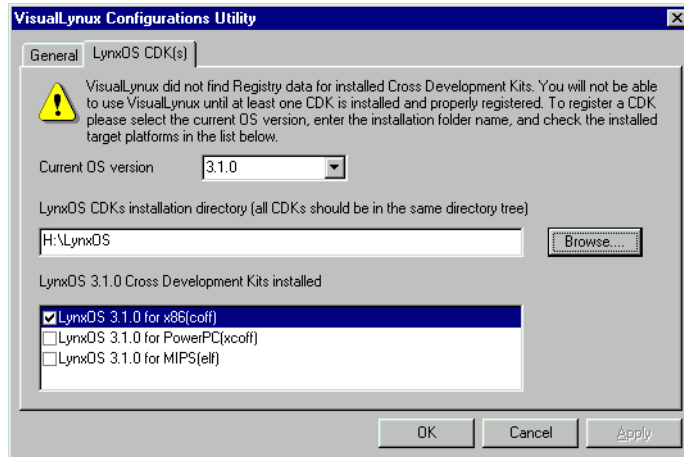


Figure 2-5: Lynx CDK(s) Property Page

**NOTE:** The Configuration Utility recognizes only target platforms, not target configurations.

Because all LynxOS CDK files that depend on a particular target configuration (board) have different names, it is safe to install all LynxOS CDKs under the same root installation directory. You can also install LynxOS CDKs for different OS versions under the same root installation directory, because the CDK tree has branches for different OS versions (3.1.0a and 3.1.1).

However, VisualLynux works with only one version of the operating system for LynxOS CDKs at a time. To specify a different version of the OS, you must run the Configuration Utility again.

CDKs occupy a significant amount of disk space, and therefore, it may not be possible to install all LynxOS CDKs under the same installation tree. If so, you can install some CDKs in another root installation directory.

However, VisualLynux can work with only one LynxOS CDK root installation directory at a time. To switch installation directories, you must run the Configuration Utility again.

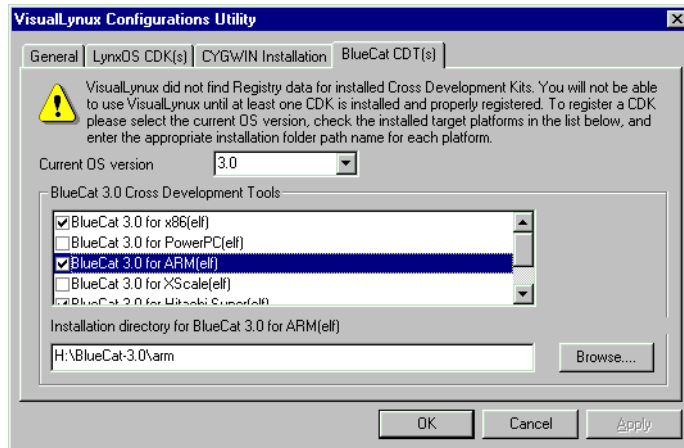
The next table describes the CDK configuration controls.

**Table 2-1: LynxOS CDK Configuration**

Control	Description
<b>Current OS version</b>	Select the current LynxOS version using this drop-down list. VisualLynux can work with only one version of an operating system at a time. You must account for projects built for different versions of LynxOS. Select version 3.1.0 for both LynxOS 3.1.0 and 3.1.0a. Controls change values depending on the version selected.
<b>LynxOS CDKs installation directory</b>	Enter the root installation directory path for installed LynxOS CDKs. Note that the LynxOS installation structure assumes that all CDKs for a particular OS version are installed in the same directory tree and have the same root installation path name. The platform-specific part of the CDK tree begins in <i>root/usr/lynx/platform</i> where <i>root</i> is the root installation name and <i>platform</i> is the platform name. For example, if the LynxOS root installation directory is <i>E:\Lynx</i> , the PowerPC CDK resides in a platform-specific subtree at <i>E:\Lynx\usr\lynx\ppc</i> . Use <b>Browse</b> to search for the LynxOS root installation directory.
<b>LynxOS Cross Development Kits installed</b>	This list box enumerates all supported target platforms for the selected LynxOS version. Check all target platforms installed on your host computer. If you uncheck a previously installed target platform, configuration data for the corresponding CDKs is removed from the Registry and you cannot build LynxOS projects for that platform.

## BlueCat Linux CDT Configuration

To set up configuration data for BlueCat Linux Cross Development Tools (CDTs) installed, check the appropriate box in the Configuration Utility **General** page and select the **BlueCat CDT(s)** property page. The following dialog box appears:



**Figure 2-6: BlueCat Linux CDT(s) Property Page**

The Configuration Utility recognizes only target platforms, not target configurations, implying that VisualLynux uses only one BlueCat Linux CDT for a given platform at a time. Because BlueCat Linux CDTs are installed in separate installation directories, run the Configuration Utility to switch between BlueCat Linux CDTs for a given platform.

VisualLynux works with only one version of an operating system for a given BlueCat Linux CDT at a time. You must rerun the Configuration Utility to switch between OS versions and installation directories of a BlueCat Linux CDT for a given platform.

Because the BlueCat Linux installation tree does not have OS version-dependent folders, you need not change or unregister installed CDTs for other platforms until you have them for different OS versions. In this case, VisualLynux works at a time with only one set of CDTs related to the current OS version.

The table below shows the configuration controls for BlueCat Linux CDKs:

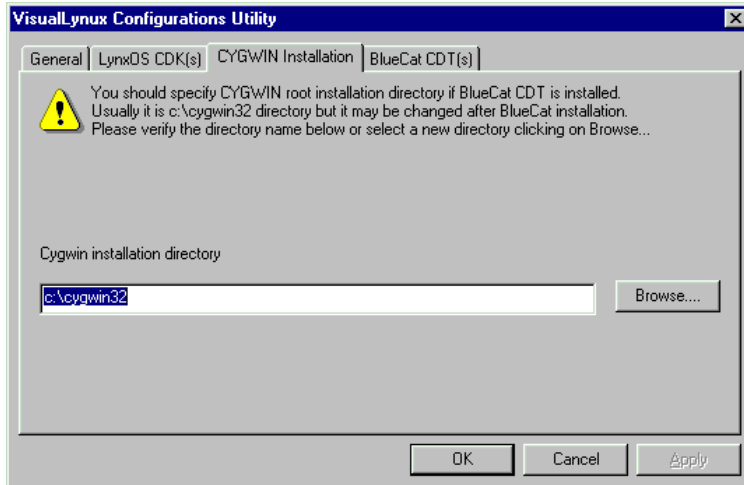
**Table 2-2: BlueCat Linux CDK Configuration**

Control	Description
<b>Current OS version</b>	Select the BlueCat Linux version for which CDTs are installed on your computer. VisualLinux can only work with one version of a given operating system at a time. You must rerun the Configuration Utility to build projects targeted for a different version of BlueCat Linux.
<b>BlueCat 3.0<sup>1</sup> Cross Development Tools</b>	This list box displays supported CDTs for the selected version of BlueCat Linux. Check every installed target platform. To unregister a CDT that has been uninstalled, uncheck the appropriate box.
<b>Installation directory for</b>	Enter the name of the root directory in which the selected BlueCat Linux <u>CDT</u> has been installed. Note that BlueCat Linux configuration requires that the installation directory be specified for each installed target platform. Use the <b>Browse</b> button to look for the installation directory.

1. BlueCat 3.0 and BlueCat 3.1 are both registered as BlueCat 3.0 in the VisualLinux configuration.

## CYGWIN Installation

The BlueCat Cross Development Tools (CDT) installation places the CYGWIN binaries into a different directory from that of the VisualLinux installation. Therefore, you need to specify the root CYGWIN directory. To do so, select the **CYGWIN Installation** property page. Unless the CYGWIN installation has been moved, you need not change the default value (`C:\cygwin32`).



---

**NOTE:** When only LynxOS CDKs are installed, the CYGWIN path name is `%LynxOSInstallDir%\usr\bin`, where `%LynxOSInstallDir%` is the installation directory entered in the LynxOS CDK(s) property page for all LynxOS CDKs.

---

Use the **Browse** button to look for the CYGWIN installation directory.

When you click **OK**, the Configuration Utility verifies that the directory name is valid and that the `cygwin1.dll` library can be found. If the path does not exist, the following message box is displayed (see figure below). Change the directory as appropriate.

---



**CAUTION!** Continuing may break VisualLynux functionality.

---





The table below describes the LynxOS CDK tools mount points:

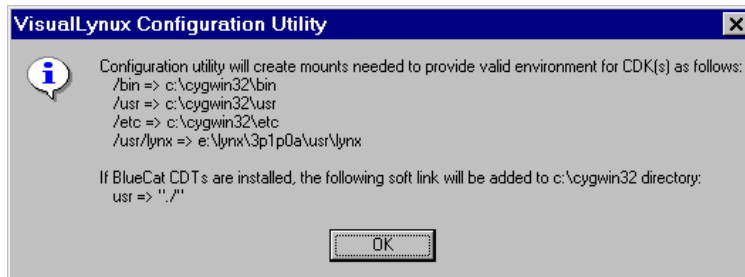
**Table 2-3: Mount Table for CDK Tools**

Mountpoint	Points to
/bin	CYGWIN utilities/commands subdirectory
/usr/bin	CYGWIN utilities/commands subdirectory used by LynxOS CDKs
/etc	CYGWIN etc subdirectory
/usr/lynx	LynxOS CDK subtree

The Configuration Utility sets appropriate mount points and notifies you with the dialog box shown in the next figure.



**CAUTION!** Do not change these mountpoints when working in a command line environment as this may break CDK functionality.



**Figure 2-9: Mountpoints Screen**

## Launching the VisualLynx Application and Kernel Wizards

The [VisualLynx Application Wizard](#) and the VisualLynx Kernel wizard create a new project through a series of dialog boxes or a wizard (see next figure). They

allow a programmer to select the type of project to be developed and specify its parameters.

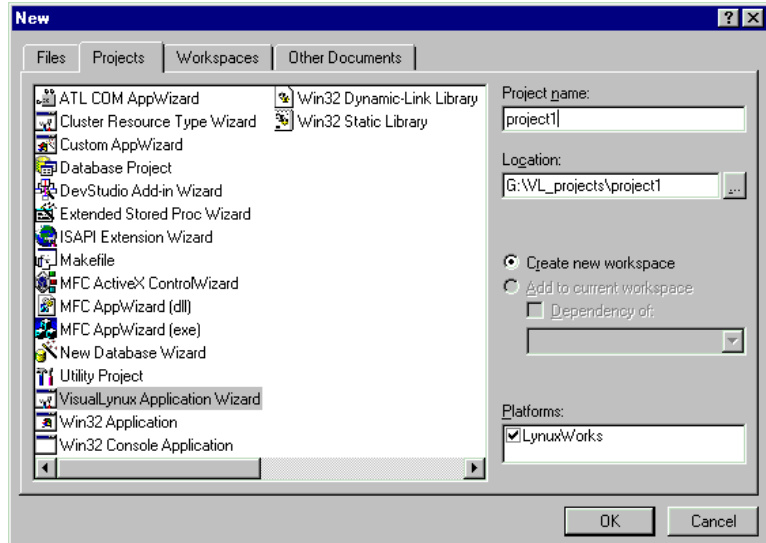


Figure 2-10: VisualLynux Application Wizard

To run the Application wizard or Kernel wizard to create a new project:

1. Select **File/New** in the Visual Studio menu bar.
2. Select the **Projects** tab in the **New** dialog box. The left pane of the dialog box displays the Wizards available.
3. Select **VisualLynux Application Wizard** or **VisualLynux LynxOS Kernel Wizard**.

---

**NOTE:** If the Application wizard or the Kernel wizard do not appear in the left pane, reinstall VisualLynux.

---

4. Next, enter a name for the new project. *Be sure to specify the location for the project.* The project name is used to derive many of the project's C/C++ function names and the project directory name, and should, therefore, conform to ANSI C/C++ naming conventions.

*Do not specify names with blank spaces.*

5. To add the new project to the current workspace, check the **Add to current workspace** radio button. Ensure that the current workspace is empty or contains only VisualLynux projects.

When the Kernel wizard is invoked, the **Create new workspace** radio button is automatically checked, and the **Add to current workspace** button is disabled. This is because the Kernel wizard always creates a separate kernel workspace that can contain several dependent projects.

The **Platforms** control at the bottom right of the dialog box always contains a checked **LynuxWorks** item when the Application wizard or Kernel wizard is selected. It denotes that a new project is targeted to the LynuxWorks operating systems (BlueCat Linux or LynxOS).

6. Click **OK** to launch the Application wizard or Kernel wizard.

The following section describes project creation using the Application wizard. For a detailed description of the Kernel wizard, see Chapter 6, “LynxOS Kernel Wizard” on page 79.

---

## Overview of Project Creation Process

The starting point of a project is selecting a project type using the Application wizard. The project type selected determines project goals and tools. The Application wizard provides a specific set of dialog boxes for each project type. Please refer to Chapter 5, “VisualLynux Application Wizard” for specific details on each project type.

The lower portion of the dialog box displays the list of target configurations (LynxOS or BlueCat Linux, target platform) for which cross development tools are available. Note that the list of configurations depends on the type of project selected in the **Project Type** list. If cross development tools or tools for a particular project type are not installed on your computer, the **Target Configurations** list displays the appropriate message.

Select and check at least one configuration. If not, the Application wizard displays a message and returns to the **Step 1** page.

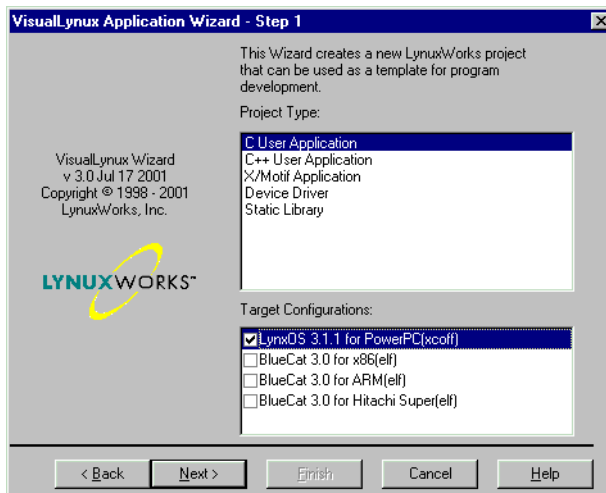
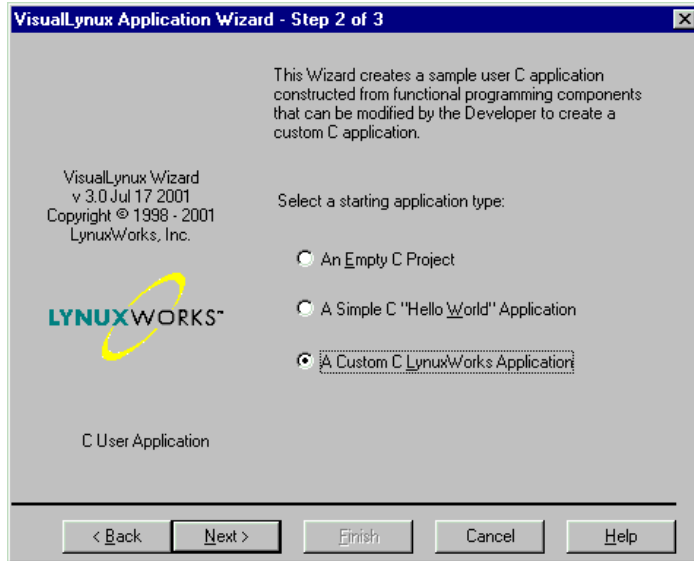


Figure 2-11: Selecting Application Type

Press the **Enter** key or press **Next** to proceed. Follow the instructions in the dialog boxes that follow. The following figure is an example of what appears if you select **C User Application** from the list of project types. For details on other project types, refer to Chapter 5, “VisualLynux Application Wizard”



**Figure 2-12: C User Application Types**

If the **Finish** button is selected on any of these dialog boxes, the Application wizard displays selected settings in a summary dialog box (see next figure). This dialog box offers a final opportunity to go back and change settings.

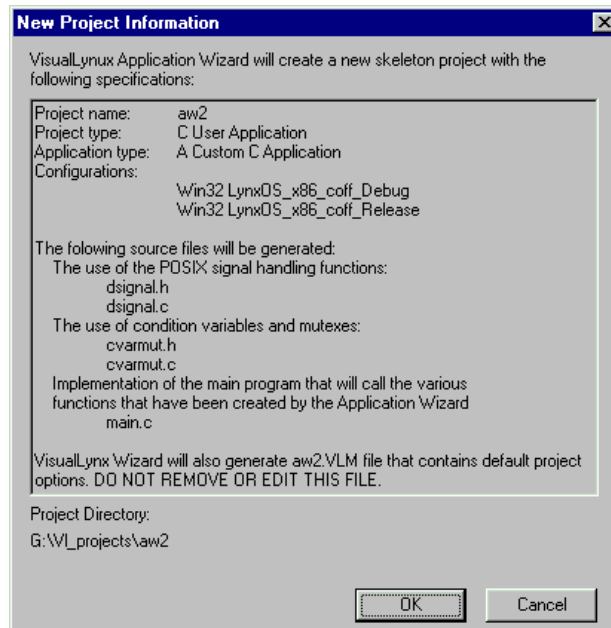


Figure 2-13: Summary Dialog Box

If **Cancel** is selected, the summary dialog box disappears and you are returned to the previous dialog box for more changes. If **OK** is selected, a new project (with template files to modify and compile) is created. The project tree appears in the left pane of the Visual C++ window. The **Class** page shows the structures and functions used in the project, and the **File View** page shows project files you can edit.

## Building a Project

To build a project, press the **F7** key or select the **Build->Build LinuxWorks** menu item. The LynuxWorks compiler and the linker create the result (an executable, driver, library, or kernel, depending on the project type) in the output directory.

To view output directories, click the Project Settings icon in the VisualLynux toolbar, use the **Project Settings** menu item, or press the **Alt F7** keys. The **General** page displays the directory names for the project output (intermediate files and results).

The output window, which usually resides at the bottom of the Visual C++ main window, contains information on how the project has been compiled and linked.

If the LynuxWorks compiler or linker finds an error, click on the error message, and the editor displays the origin of the error. For details, refer to Chapter 3, “VisualLynux Build Process”

Now, you are ready to debug the project result on a target machine. First, however, you must define a target machine for VisualLynux. Click on the **Target**

**Administration** button. 


The LynuxWorks debugger can be started after the target machine has been defined.

---

**NOTE:** See the [Target Wizard](#) section in Chapter 10, “VisualLynux Target Administration” for information on defining a target machine.

---

Be sure to set up an account on the target machine. If you do not have one already,

click the **Telnet** button  in the VisualLynux toolbar, and use a telnet session to create an account.

## Debugging a Project

There are two varieties of the LynuxWorks version of the GNU GDB and Insight debuggers: GDB and Total/db. The GDB debugger has a text interface, Total/db has a graphical interface that facilitates the debugging process. Launch the appropriate debugger by clicking on the corresponding VisualLynux toolbar

button: **Debugger**  or **Total/db.**  (see Chapter 7, “VisualLynux AddIn Tools” for descriptions of toolbar icons).

Both debuggers create a session with the target machine, copy the executable to the target machine into project directory defined using the Target Administration wizard, and launch GDBSERVER on the target machine. For additional information, refer to Chapter 7, “VisualLynux AddIn Tools”

If you locate a bug, edit the corresponding source file in the VisualLynux C++ editor and rebuild the project. Then run the debugger again.

---

**NOTE:** See Chapter 7, “VisualLynux AddIn Tools” for descriptions of toolbar items.

---



---

## Building Project Targets

This chapter describes the process used by VisualLynux to create project goals.

- VisualLynux always builds the main project goal in two stages: First, it processes source files, and second, it performs project-level build steps (if all files have been processed successfully). Usually VisualLynux processes source files using the LynuxWorks compiler, and uses the main project tool to build the main project goal.

When a project is created by VisualLynux Application wizard, a default compiler is associated with the source files, depending on their file extension. The project type defines the project-level tool (linker, library, etc.) used to build the project.

- You can customize the build process using the VisualLynux **Settings** dialog box to define **Custom Build** steps for a file or a project in a specific configuration.
- You can build or rebuild the project goal (executable, library, and so on) that a project configuration defines. When you invoke a build (by pressing **F7** or selecting the **Build/Build LynuxWorks** menu command), VisualLynux builds the current project configuration. VisualLynux processes only files that have changed since the last build.

The current project configuration is displayed in the Visual C++ **Build** toolbar. You can switch to another project configuration using the drop-down list on this toolbar, or using the **Build/Set Active Configuration** menu items. VisualLynux wizards create at least two configurations for each project (Debug and Release). You can add configurations using the **Build/Configurations** menu item.

When rebuilding a project (using **Build/Rebuild All** or **Build/Batch Build** menu commands), VisualLynux processes all the files in the project

whether they have or have not been changed. Also, you can either build one project configuration (**Rebuild All**) or choose multiple configurations (**Batch Build**).

---

## Processing Files

To process a file VisualLynux uses either the compiler (the default tool) or the **Custom Build** option. Both tools cannot be specified for a particular file simultaneously. The compiler is associated with a particular source file. This is done by the VisualLynux wizard during project creation or by the VisualLynux add-in tools when you have created or inserted a new source file into the project.

VisualLynux uses file name extensions, as shown in the table below, to assign a default processing tool.

**Table 3-1: File Processing Using File Name Extensions**

File Extension	Default Build Tool
.c	gcc.exe
.cpp	g++.exe
.s	gcc.exe (gcc.exe invokes the assembler)
.cc	g++.exe
.cxx	g++.exe
.i	gcc.exe
.ii	gcc.exe
.C	g++.exe

File name extensions are case sensitive. If a capital “C” is the file extension, the C++ compiler is invoked rather than the C compiler. `.CPP` is not recognized as a valid extension, and causes an error. In cases other than those described in the table above, VisualLynux does not assign a default build tool for a file.

The compiler used to process any or all files in a project can also be changed using the **Settings** dialog box. A custom setting can be specified for any file, even if a default tool is assigned to it. To do so, check the **Always use custom build** box in the file-level **Property** page for a file, and specify the custom build step for it.

VisualLynux always uses the following *sequence* to process project files:

1. Custom build steps are performed for all those files for which a custom build is specified.
2. If all custom build steps are performed successfully or if no custom builds are specified, default builds (compiling) are performed for the remaining files assigned to a default build tool.

---

**NOTE:** VisualLynux checks header/source file dependencies only if you check **Generate dependencies** in the compiler options (C/C++) dialog box.

---

## Project-Level Build Steps

VisualLynux assigns a main build tool to a project that is created by the VisualLynux wizard or imported from an external Makefile. Depending on the project type, VisualLynux assigns these tools as shown in the table below:

**Table 3-2: Project Build Steps**

Application Type	Main Build Step	Tool
C Application	Linking	gcc.exe
C++ Application	Linking	g++.exe
X/Motif Application	Linking	gcc.exe
Device Driver	Linking	ld.exe
Static Library	Building library	ar.exe
LynxOS Kernel	Linking	ld.exe (ar.exe to build intermediate driver and devices subprojects)

You can customize a project build by specifying the additional build steps shown in the next table:

**Table 3-3: Additional Build Steps**

Build Step	Description	Execution
Pre-link	A set of arbitrary batch commands executed by a Windows shell	Right before the main build step (linker or librarian) is invoked
Custom Build	Set of arbitrary batch commands executed by a Windows shell with the ability to set up resulting files and custom dependencies	Right after the main build step is successfully completed
Post-build	A set of arbitrary commands	Right after the build process is finished (including Project Custom Build step, if any)

See detailed descriptions of the Pre-link, Custom Build, and Post-build steps in Chapter 8, “VisualLynux Project Settings”

## Build Macros

VisualLynux defines a set of special build macros for use while customizing file or project builds. Build macros can be used anywhere in the user-defined set of batch commands in the Custom Build, Pre-Link, or Post-Build steps. Make note, however, of the format of the command generated after macro expansion.

To specify the macro value anywhere in a command, use its name in parenthesis with dollar sign (\$) as a prefix. For example, `$(Output_Dir)` is substituted by the relative output directory name before command execution.

Macros used by VisualLynux are distributed in three categories: global, project-(configuration-) level, and file-level macros. Although the names of some project- and file-level macros are the same, their values can change depending on the build level.

## Global Macros

Macros of this group have the same values for all project configurations. The table below shows global build macros and their definitions:

**Table 3-4: Global Build Macros**

Macro	Definition
<code>\$(MSDevDir)</code>	Fully qualified path to the directory in which the Microsoft Visual Studio <code>msdev.exe</code> executable resides
<code>\$(MSVCDir)</code>	Fully qualified path to the directory in which Visual Studio build tools reside
<code>\$(VisualLynuxDir)</code>	Fully qualified path to the directory in which VisualLynux is installed
<code>\$(ToolDir)</code>	Fully qualified path to the directory in which VisualLynux tools reside
<code>\$(WkspName)</code>	Name of the current workspace
<code>\$(WkspDir)</code>	Fully qualified path to the directory in which a workspace file ( <code>.dsw</code> file) resides
<code>\$(ProjDir)</code>	Fully qualified path to the directory in which a project file ( <code>.dsp</code> file) resides
<code>\$(ProjName)</code>	Name of the project (usually the name of a <code>.dsp</code> file without directory name and extension)

## Project (Configuration)-Level Macros

This group of macros have values that change from one configuration to another:

**Table 3-5: Project-Level Build Macros**

Macro	Definition
<code>\$(TargetOS)</code>	Name of the target OS (LynxOS or BlueCat Linux)
<code>\$(TargetOSPrefix)</code>	Target OS (LynxOS or BlueCat Linux) prefix in the directory subtree
<code>\$(TargetOSVersion)</code>	Version number of the target OS (e.g., 3.1.0a)
<code>\$(TARGET_GENERIC)</code>	Generic target platform name (x86, MIPS, etc.)

**Table 3-5: Project-Level Build Macros (Continued)**

Macro	Definition
<code>\$(TargetCPU)</code>	Target processor name used in directory names for cross development tools ( <code>i386</code> , <code>ppc</code> , etc.)
<code>\$(OBJ_FORMAT)</code>	Object format code ( <code>coff\xcoff\elf</code> )
<code>\$(InstallDir)</code>	Fully qualified path to the installation directory of the cross development tools for a particular platform
<code>\$(BUILD_ENV_PREFIX)</code>	Fully qualified path to the root directory of the cross development tools for a particular platform
<code>\$(TargetPlatformCDKDir)</code>	Same as <code>\$(BUILD_ENV_PREFIX)</code>
<code>\$(GCC)</code>	Fully qualified path to the C compiler ( <code>gcc.exe</code> ) for a particular platform
<code>\$(CPP)</code>	Fully qualified path to the C++ compiler ( <code>g++.exe</code> ) for a particular platform
<code>\$(LD)</code>	Fully qualified path to the linker ( <code>ld.exe</code> ) for a particular platform
<code>\$(AR)</code>	Fully qualified path to the library ( <code>ar.exe</code> ) for a particular platform
<code>\$(MAKE)</code>	Fully qualified path to the <code>make</code> utility ( <code>make.exe</code> )
<code>\$(BASH)</code>	Fully qualified path to the bash shell ( <code>bash.exe</code> )
<code>\$(Intermediate_Dir)</code>	Path to the directory in which intermediate project results (e.g., object files) are placed, relative to the project directory
<code>\$(Output_Dir)</code>	Path to the output directory, relative to the project directory
<code>\$(TargetDir)</code>	Path to the directory in which main project goal is created; usually the same as <code>\$(Output_Dir)</code>
<code>\$(TargetPath)</code>	Project main goal file path, relative to the project directory
<code>\$(TargetName)</code>	Project main goal file name (without directory name and extension), relative to the project directory
<code>\$(ProjectPseudoTarget)</code>	Special pseudo-target that may be used to force file/project build step execution

**Table 3-5: Project-Level Build Macros (Continued)**

Macro	Definition
<code>\$(LINKER)</code>	Path to linker used to link main project goal (may not exist for library projects)
<code>\$(Objects)</code>	List of object files generated as a result of file builds (DOS path format)
<code>\$(UnixObjects)</code>	List of object files generated as a result of file builds (UNIX path format)
<code>\$(ExternalInputs)</code>	List of files resulting from other projects on which current project depends
<code>\$(LINK_OPTIONS)</code>	Linker command line options (as they are passed to the linker)
<code>\$(InputPath)</code>	Path to the input file for custom build step, relative to the project directory; always equal to <code>\$(TargetPath)</code>

## File-Level Macros

This is a group of macros that can have different values for different source files.

**Table 3-6: File-Level Build Macros**

Macro	Definition
<code>\$(Intermediate_Dir)</code>	Path to the directory in which file processing result is placed, relative to the project directory
<code>\$(InputPath)</code>	Relative or fully qualified path to the source file
<code>\$(InputName)</code>	Name of the source file (without path and extension)
<code>\$(InputDir)</code>	Fully qualified path to the directory in which source file resides
<code>\$(FileOutputs)</code>	List of files generated as the result of file processing (by default tool or custom build)
<code>\$(FileDependency)</code>	List of files on which source file depends

**Table 3-6: File-Level Build Macros (Continued)**

<b>Macro</b>	<b>Definition</b>
\$( COMPILER )	Fully qualified path to the compiler used to process this file (may be empty if compiler is not associated with the file)
\$( COMPILER_FLAGS )	Compiler command line options (may be empty if compiler is not associated with the file)



---

## Overview of Menu Structure

Under VisualLynux, the default Visual C++ menus retain their Visual Studio default behaviors where appropriate. Because this is true for the majority of Visual Studio menu commands, the following tables detail the differences in menu selection when using VisualLynux.

---

**NOTE:** The VisualLynux Application wizard can guide the developer set up a new project file. The first project file is inserted into the default workspace. Following projects can be inserted into the current workspace. New workspaces are created using the **File->New->Workspace** tabs. This is the default behavior of Visual Studio. The **File->New->Other Documents** sequence allows the creation and normal functioning of files not applicable to VisualLynux projects.

---

## VisualLynux Menus

The following is a list of VisualLynux menus available:

[File Menu](#)

[Edit Menu](#)

[View Menu](#)

[Insert Menu](#)

[Project Menu](#)

[Build Menu](#)

[Tool Menu](#)

[Window Menu](#)

All **Help** menu selections apply only to Microsoft Visual C++ projects. For VisualLynux projects, select **Help** in the VisualLynux Toolbar.

## File Menu

All **File** menu selections and items can be applied to VisualLynux. The **File->New** menu item displays the **New** dialog box where some of the file types in the **Files** page, all project wizards except VisualLynux wizard in the **Project** page, and all documents in the **Other Documents** page are not used in VisualLynux projects.

## Edit Menu

The table below lists the commands available on the **Edit** menu:

**Table 4-1: Edit Menu Commands**

Submenu	Description
<b>Breakpoints</b>	<i>Inapplicable to VisualLynux</i> This command is disabled for VisualLynux projects.
<b>List Members</b>	Default Visual Studio Behavior Part of the IntelliSense options that speed up the creation of C++ code - this feature displays a list of valid member variables or functions for the selected class or structure. Selecting from the list inserts the member into your code.
<b>Type Info</b>	Default Visual Studio Behavior Part of the IntelliSense options that speed up the creation of C++ code - this feature displays the complete declaration for any identifier in your code. If the mouse cursor is reset over the identifier, its declaration appears in a popup window. This feature depends on compiling the application with browser information on.
<b>Parameter Info</b>	Default Visual Studio Behavior This feature displays the complete declaration, including a parameter list, for the function to the left of the cursor. The parameter in bold indicates the next parameter required as you type the function call.
<b>Complete Word</b>	Default Visual Studio Behavior This feature types the rest of a variable or function name once enough characters have been entered to distinguish the term. If typed input has none, or more than one possible match, invoking <b>Complete Word</b> displays the <b>Members</b> list, which finds the term wanted.

## View Menu

The following table shows the commands available on the **View** menu:

**Table 4-2: View Menu Commands**

Submenu	Behavior
<b>ClassWizard</b>	<p><i>Inapplicable to VisualLynux</i></p> <p>This wizard works only with applications that use MFC classes. It makes certain routine tasks easier: such as creating new classes, defining message handlers, overriding MFC virtual functions, and gathering data from controls in a dialog box, form view, or record view.</p>
<b>Resource Symbols</b>	<p><i>Inapplicable to VisualLynux</i></p> <p>This brings up the <b>Resource Symbol Browser</b> for managing Microsoft Studio Resource files. These resource files are not used on LynxOS or BlueCat Linux. Although VisualLynux does not use Microsoft resources, this command functions for native Windows projects.</p>
<b>Resource Includes</b>	<p><i>Inapplicable to VisualLynux</i></p> <p>This brings up a dialog box that prompts for the resource files to be included in a project. LynxOS and BlueCat Linux do not use resource files.</p>
<b>Debug Windows</b>	<p><i>Inapplicable to VisualLynux</i></p> <p>This sliding menu provides a way to bring Microsoft Debugger windows into view. These include windows to view watchpoints, callstacks, memory, variables, registers, and disassembly within the debugger. As VisualLynux does not use these windows, they are disabled for VisualLynux projects.</p>

## Insert Menu

The following table shows the **Insert** menu commands:

**Table 4-3: Insert Menu Commands**

Submenu	Behavior
New Class	Default Visual Studio Behavior This invokes a dialog box that allows you to add a new generic class to a C++ project.
New Form	<i>Inapplicable to VisualLynux</i> This is the database wizard for creating forms. This item is disabled in the VisualLynux workspace.
Resource	<i>Inapplicable to VisualLynux</i> This adds Windows Resources to a project. This item is disabled in the VisualLynux workspace.
Resource Copy	<i>Inapplicable to VisualLynux</i> This copies a Windows Resource. This item is disabled in the VisualLynux workspace.
File As Text	Default Visual Studio Behavior This allows insertion of a selected file at the insertion point in an open source file.
New ATL Object	<i>Inapplicable to VisualLynux</i> This menu item allows creation of a new ATL (ActiveX Template Library). This item is disabled in the VisualLynux workspace.

## Project Menu

This table shows the **Project** menu commands:

**Table 4-4: Project Menu Commands**

Submenu	Behavior
Setting	Default Visual Studio Behavior In the VisualLynux workspace this invokes the <b>VisualLynux Project Settings</b> dialog box instead of the default Visual Studio Settings dialog box.

## Build Menu

The following table shows the **Build** menu commands:

**Table 4-5: Build Menu Commands**

Submenu	Behavior
<b>Compile</b> <i>selected_file</i>	Default Visual Studio Behavior This calls the current compiler to build the <i>selected_file</i> . In the context of a VisualLynux project, the correct version of GCC or G++ is called. In the context of a Microsoft native project, this implies calling the Microsoft compiler.
<b>Build</b> <i>project_target</i>	Default Visual Studio Behavior This builds the entire <i>project_target</i> , bringing the target up to date by compiling and linking as necessary. In the context of a VisualLynux Project, the correct version of GCC or G++ is called to compile each source file, and the proper tool (linker or librarian) is called to create the target file.
<b>Rebuild All</b>	Default Visual Studio Behavior This essentially does a <b>Clean</b> , followed by a <b>Build</b> of the <i>project_target</i> .
<b>Batch Build</b>	Default Visual Studio Behavior This opens a dialog box to select the project configurations to be built in a single <b>Build</b> instance and allows for multiple projects to be built together.
<b>Clean</b>	Default Visual Studio Behavior This removes any existing object files and executables and marks all files as needing to be rebuilt.
<b>Start Debug</b>	Default Visual Studio Behavior This invokes the Microsoft debugger and can direct it to <b>Go</b> , <b>Step Into</b> , <b>Run</b> to cursor, and <b>Attach</b> to process. In the context of a VisualLynux Project, this invokes the GDB debugger for the active project.
<b>Debugger Remote Connection</b>	Default Visual Studio Behavior This allows remote connection setup for the Microsoft debugger. It cannot be used by LynuxWorks debuggers. In a VisualLynux project, it invokes the VisualLynux Target Administration tool.

Table 4-5: Build Menu Commands (Continued)

Submenu	Behavior
<b>Execute</b> <i>file.exe</i>	Default Visual Studio Behavior This executes the project executable on the target machine. This command sends the executable produced by current project to the default target machine (set by the Target Administration tool) and executes it using a remote shell.
<b>Set Active Configuration</b>	Default Visual Studio Behavior <u>By default</u> , the VisualLynux Application wizard creates several configurations for every new project. This dialog box prompts the developer to select the configuration to be current and built by default.
<b>Profile</b>	<i>Inapplicable to VisualLynux</i> This generates profiling information for a Windows application. This item is disabled in a VisualLynux project.

## Tool Menu

The following table displays the **Tool** menu commands:

Table 4-6: Tool Menu Commands

Submenu	Behavior
<b>Setting Source Browser</b>	Default Visual Studio Behavior This launches the Microsoft Class and Symbol browser, allowing the developer to browse the class and symbolic information (like the class definition and reference, member function, structure, member field, and variable). All the behaviors are exactly same as the VC++. To use this feature, the browser information files must be generated for the project sources. To turn on the generation of browser information, use the <b>VisualLynux Project Settings</b> dialog box ( <b>Browser</b> and <b>C/C++</b> pages).
<b>Close Source Browser File</b>	Default Visual Studio Behavior Closes the Source Class and Symbol browser.
<b>Visual Component Manager</b>	<i>Inapplicable to VisualLynux</i> This launches the Visual component manager tool, which maintains a database of reusable components that developers can share.

Table 4-6: Tool Menu Commands (Continued)

Submenu	Behavior
<b>Register Control</b>	<i>Inapplicable to VisualLynux</i> This registers an add-in or plug-in component to the Visual Studio environment.
<b>Error Lookup</b>	<i>Inapplicable to VisualLynux</i> This looks up Microsoft defined error numbers and reports the corresponding textual error message.
<b>ActiveX Control Test Container</b>	<i>Inapplicable to VisualLynux</i> This utility facilitates testing of ActiveX controls.
<b>OLE/COM Object Viewer</b>	<i>Inapplicable to VisualLynux</i> This is a viewer utility to browse OLE/COM objects.
<b>SPY++</b>	<i>Inapplicable to VisualLynux</i> This tool provides a graphical view of the native Win32 systems processes, threads, windows, and window messages. Although it does not look at items on a LynxOS/BlueCat Linux target, it can still be used to monitor tools running on the Windows cross development host.
<b>MFC Tracer</b>	<i>Inapplicable to VisualLynux</i> This allows tracing of MFC calls within the Microsoft Debugger.
<b>InstallShield Wizard</b>	<i>Inapplicable to VisualLynux</i> This helps create an InstallShield version of a native Windows application. This is then used to install the application on Windows systems.

Table 4-6: Tool Menu Commands (Continued)

Submenu	Behavior
<b>Customize</b>	<p style="text-align: center;">Default Visual Studio Behavior</p> <p>Allows for customization of the <b>Toolbars</b> and <b>Tools</b> menu. The developer can add different toolbars and icons to the IDE display as needed. The VisualLynux toolbar can be added using this feature.</p>
<b>Options</b>	<p style="text-align: center;">Default Visual Studio Behavior</p> <p>This pops up a multi-tab dialog box to allow options to be set for the following features:</p> <ul style="list-style-type: none"> <li>• <b>Editor</b></li> <li>• <b>Tabs</b></li> <li>• <b>Debug</b></li> <li>• <b>Compatibility</b></li> <li>• <b>Build</b></li> <li>• <b>Directories</b></li> <li>• <b>Source Control</b></li> <li>• <b>Workspace</b></li> <li>• <b>DataView</b></li> <li>• <b>Macros</b></li> <li>• <b>Help Systems</b></li> <li>• <b>Format</b></li> </ul> <p>All the options function as normal under the VisualLynux IDE except for the <b>Debug</b> options, which apply only to the Microsoft debugger.</p>

## Window Menu

All menu items and selections in the **Window** menu function are the same for both Microsoft C++ and VisualLynux projects.



## Launching the VisualLynux Application Wizard

The VisualLynux Application wizard creates a new project through a series of dialog boxes. You can select the application type and specify new project parameters.

To launch the VisualLynux Application wizard and create a new project:

1. Select **File/New** in the **Visual Studio** menu bar.
2. Select the **Projects** tab in the **New** dialog box.

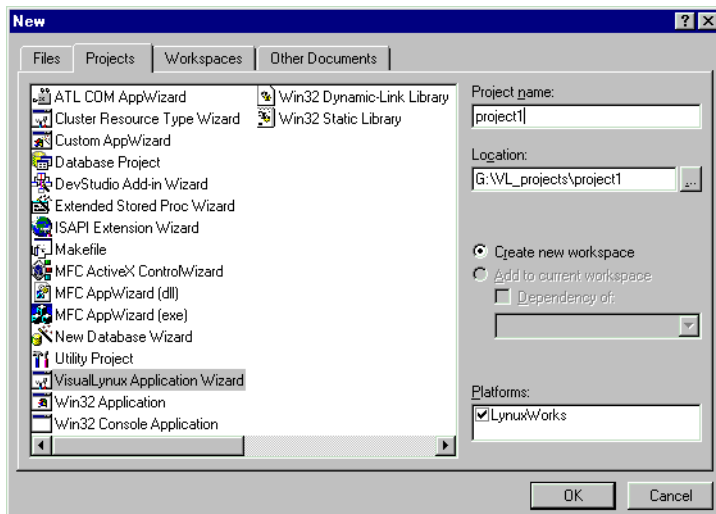


Figure 5-1: VisualLynux Application Wizard

3. The left pane of the dialog box displays the wizards available. Select **VisualLynux Application Wizard**.

---

**NOTE:** *If the VisualLynux Application wizard does not appear in the left pane, reinstall VisualLynux.*

---

4. Next, enter a name for the new project. Be sure to specify a proper location for the project. This is the full path of the project directory. The project name is used to derive many of the project's C/C++ function names and should, therefore, conform to ANSI C/C++ naming conventions. *Do not specify names with blank spaces.*
5. To add the new project to the current workspace, check the **Add to current workspace** radio button. Be sure that the current workspace is empty or contains only VisualLynux projects.

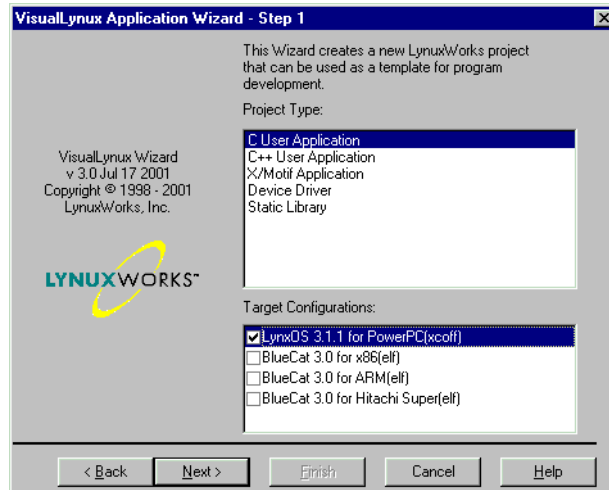
The **Platforms** control at the bottom right of the dialog box always contains a checked **LynuxWorks** item when **VisualLynux Application Wizard** is selected. It shows that a new project is targeted to one or more of the LynuxWorks operating systems.

6. Click **OK** to launch the Application wizard.

---

## Selecting a VisualLynux Project Type

The VisualLynux Application wizard facilitates the creation of a new VisualLynux project based on some sample code and creating a typical VisualLynux project environment depending on the project type. It offers six project types to choose from as a starting point.



**Figure 5-2: Selecting a Project and a Configuration**

The list in the bottom right of the dialog box displays the target configurations for which cross development tools are installed on the host machine. Note that the configurations depend on the type of project selected in the **Project Type** list at the top right of the dialog box.

If no cross development tools or tools for a particular project type are installed on your computer, the **Target Configurations** list displays an appropriate message.

If no target configuration is selected, you are returned to the **Step 1** dialog box of the Application wizard.

## C User Application Wizard

If you select a **C User Application** as the **Project Type**, the **Step 2** dialog box of the VisualLynux Application wizard (see next figure) allows you to choose from three different C user application types.

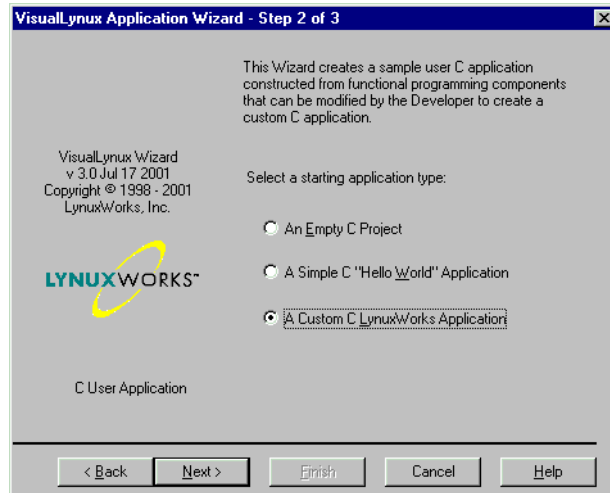


Figure 5-3: C User Application Types

The C User application types and functions are shown in the table below:

Table 5-1: C User Application Types and Functions

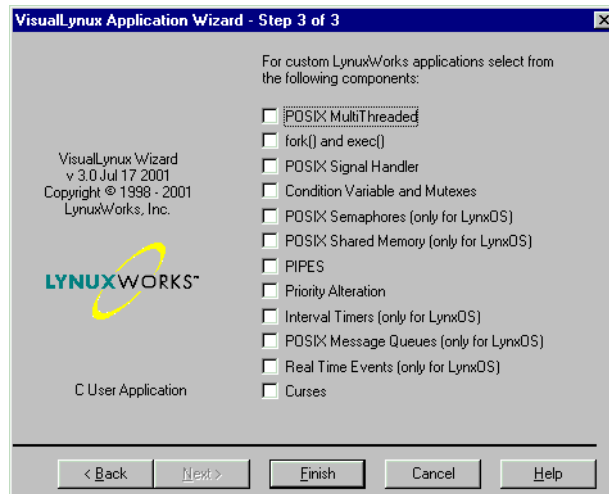
Application Type	Function
<b>An Empty C Project</b>	Creates an appropriate Visual Studio workspace and project files for a LynxOS or BlueCat Linux targeted C user application, but does not create any source code files. This is generally used when existing source code is to be imported into an empty VisualLynux project.
<b>A Simple C "Hello World" Application</b>	Creates a VisualLynux project with a C source file version of the "Hello World" program as the starting source files.
<b>A Custom C LynuxWorks Application</b>	Allows you to create a starting application that contains functioning code for many LynxOS, BlueCat Linux, and POSIX-based programming concepts. An additional dialog box assists in selecting the appropriate template code for your application type.

The first two choices above do not ask for additional information. The VisualLynux Application wizard proceeds to summarize selections made and the project is created with the click of the **Finish** button.

If **A Custom C LynuxWorks Application** is selected, click **Next** to continue customizing the new project.

## A Custom C LynuxWorks Application

The following dialog box appears as **Step 3** of the VisualLynux Application wizard for creating a custom C LynuxWorks application:



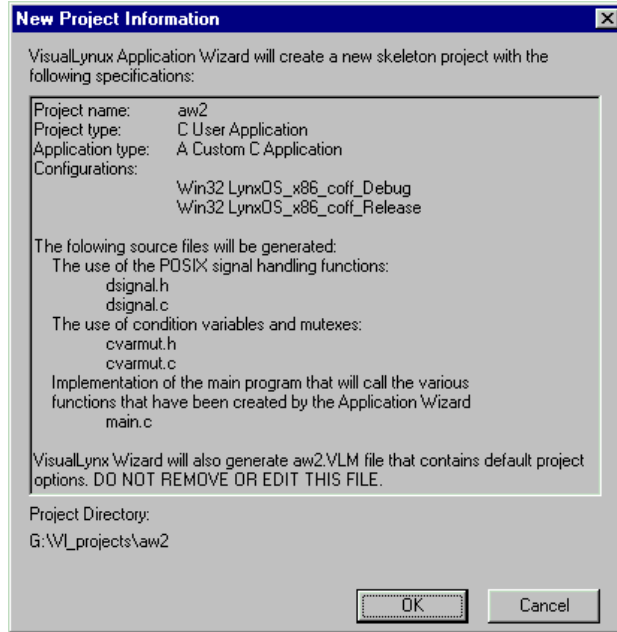
**Figure 5-4: A Custom C LynuxWorks Application**

Any combination of the programming components that appear in the figure above can be selected. Note that not all features are available for both LynxOS and BlueCat Linux operating systems. Your selection should correspond to the configurations selected in the **Project Type** dialog box.

Also, the target operating system must contain appropriate support for selected features. For example, if you select **Curses**, the target BlueCat Linux system must contain the `libncurses.so` and `libtermcap.so` libraries to run the application.

For each component selected, VisualLynux creates sample code that can be compiled and executed. If multiple components are selected, all of them are placed in separate source files within the new project. A main function, which calls all [Component Functions](#) that have been selected, is created.

Selecting **Finish** from the dialog box creates a summary of selected settings and allows you a final opportunity to go back and change them.



**Figure 5-5: C User Application Summary Dialog Box**

Select **OK** to create the new project that contains the template files to be modified and compiled. The **Cancel** button returns you to the previous dialog box.

## Component Functions

The following table describes components of a Custom C Linux Works Application, and the functions they demonstrate.

**Table 5-2: Component Functions**

Component	Description	Functions to be Demonstrated
POSIX MultiThreaded	This generates a section of code that sets up two threads and initializes them to execute a simple function.	pthread_attr_init() pthread_attr_setinheritsched() pthread_attr_setschedpolicy() pthread_create() sigaction() sigemptyset() alarm()
fork() and exec()	This generates a function that forks a child process and allows it to execute some other program, such as /bin/ls.	fork() execvp()
POSIX Signal Handler	This generates a function that demonstrates the use of POSIX signal handling functions. It creates two signal handling functions: one for SIGUSR1, and the other for SIGALRM. The process sends itself SIGUSR1 for 10 seconds and counts how many signals are processed.	The POSIX signal set manipulation function sigemptyset() and the POSIX signal functions kill(), sigaction(), alarm()

Table 5-2: Component Functions (Continued)

Component	Description	Functions to be Demonstrated
Condition Variables and Mutexes	This generates a section of code that demonstrates the use of condition variables and mutexes to synchronize access to a shared resource. A multi-thread application declares a mutex to guard a shared resource. Threads take turns locking the mutex, checking a condition variable, modifying a shared resource, sending a conditional signal to the other thread, and unlocking the mutex.	<pre>pthread_mutexattr_init() pthread_mutex_init() pthread_condattr_init() pthread_cond_init() pthread_mutex_lock() pthread_cond_wait() pthread_cond_signal() pthread_mutex_unlock() pthread_attr_init() and other standard routines to create threads and signal handlers</pre>
POSIX Semaphores (for LynxOS only)	This generates a function that creates a binary semaphore, opens it and calls <code>fork()</code> . The child process waits on the semaphore, and after the parent unlocks it, closes the semaphore and exits.	<pre>sem_wait() sem_open() sem_unlink() sem_close() sem_getvalue() sem_post()</pre>
POSIX Shared Memory (for LynxOS only)	This generates a function to set up a shared memory region that two processes (a parent and child) can read and write messages to.	<pre>shm_open() ftruncate() mmap() fork() sched_yield() close() shm_unlink() munmap()</pre>



Table 5-2: Component Functions (Continued)

Component	Description	Functions to be Demonstrated
PIPES	This generates a function that demonstrates two processes (parent and child) communicating through a pipe. A first pipe is created, a <code>fork()</code> executed, and the child creates an additional pipe to run a sub-process. The child sends all data to the parent through the pipe.	<pre>pipe() fork() fread() close() popen() pclose()</pre>
Priority Alteration	This generates a function that checks the current priority of a process and modifies it to a new priority.	<pre>fork() sched_getparam() sched_setparam()</pre>
Interval Timers (for LynxOS only)	This application demonstrates the POSIX.1b facilities for creating, reading, and deleting an interval timer. When the timer goes off, it calls a signal handler to wake up the application.	<pre>timer_create() timer_settime() timer_gettime() timer_delete()</pre>
POSIX Message Queues (for LynxOS only)	This generates a function that shows two processes (parent and child) communicating through a message queue. The parent sends messages to the child through the queue.	<pre>wait() fork() mq_send() mq_receive() mq_open() mq_close() mq_unlink()</pre>

Table 5-2: Component Functions (Continued)

Component	Description	Functions to be Demonstrated
Real-Time Events (for LynxOS only)	This generates a function that demonstrates the use of real-time events. It sets up a signal handler for a given real-time event, forks a child, and sends the child the real-time event.	sigaction() sigemptyset() sigaddset() fork() sched_yield() wait() sigtimedwait() sigqueue()
Curses	This generates a section of code that uses the curses functions to control and update the screen. A small on-screen animation is created to demonstrate the use of curses functions.	initscr() clear() move() clrtoeol() mvprintw() refresh() endwin()

## C++ User Application Wizard

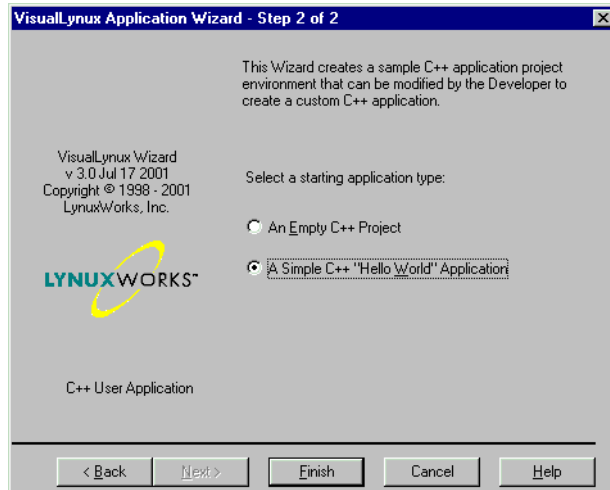


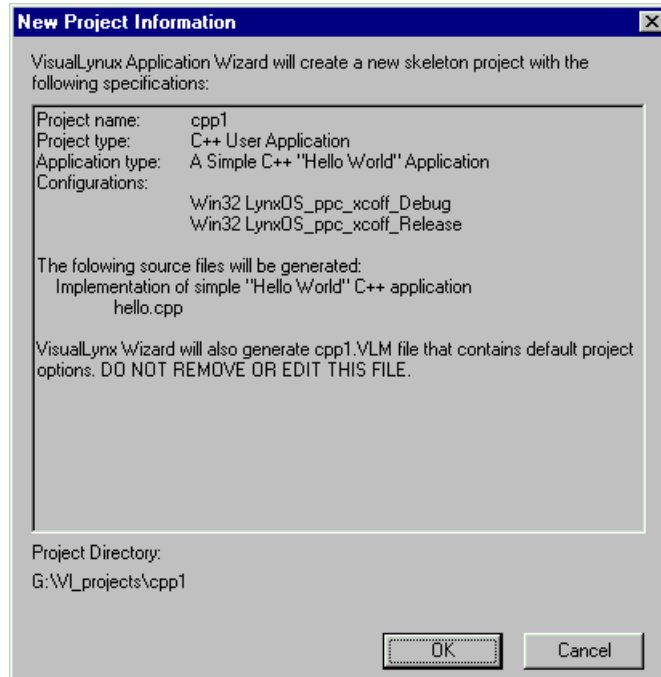
Figure 5-6: C++ User Application Types

The C++ User Application dialog box in the figure above allows you to choose from two different C++ user application types (see table below), each designed to expedite programming.

**Table 5-3: C++ User Application Types and Functions**

Application Type	Function
<b>An Empty C++ Project</b>	Creates an appropriate Visual Studio workspace and project files for a LynxOS or BlueCat Linux targeted C++ user application. Does not create any source code files. This is generally used when existing source code is to be imported into an empty VisualLynux project.
<b>A Simple C++ "Hello World" application.</b>	Creates a VisualLynux project with a C++ source file version of the "Hello World" program as the starting source files.

The VisualLynux Application wizard summarizes selections made and creates the project when you click the **Finish** button. At this stage, you have a final opportunity to go back and change settings.



**Figure 5-7: C++ User Application Summary Dialog Box**

When you select **OK**, the new project with the template files to be modified and compiled is created. The **Cancel** button returns you to the previous dialog box.

## **X/Motif Graphical Application Wizard**

The purpose of the X/Motif wizard is to help the X11/Motif programmer get started with writing a graphical application. This is accomplished by modifying template code to implement many of the common X11 and Motif programming constructs.

The options below are presented if **X/Motif Application** and the **Next** are selected:

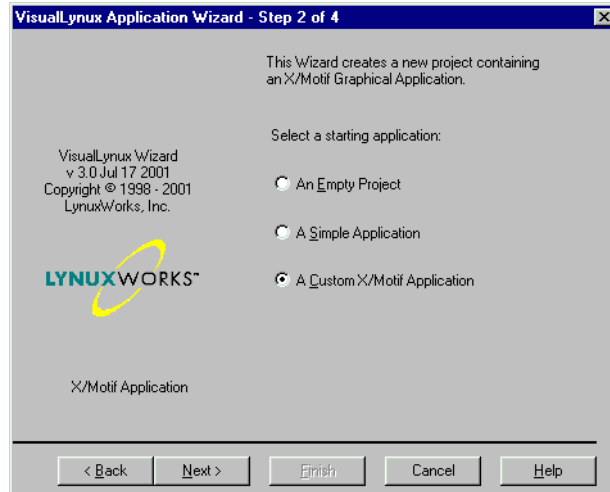


Figure 5-8: X/Motif Graphical Application Types

**NOTE:** The X/Motif wizard is available only for the LynxOS operating system.

The table below lists the X/Motif application types available:

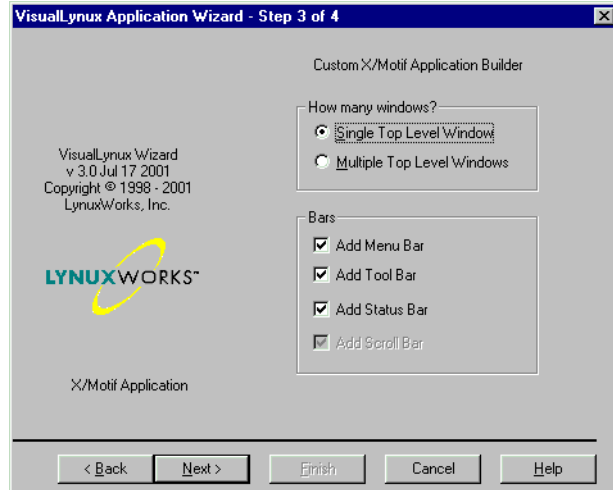
Table 5-4: X/Motif Graphical Application Types

Application Type	Function
<b>An Empty Project</b>	Creates an appropriate Visual Studio workspace and project files for a VisuallyLynux X/Motif graphical user application but does not create any source code files. This is generally with existing source code that is to be imported into an empty VisualLynux project.
<b>A Simple Application</b>	Creates an appropriate Visual Studio workspace and project files for a simple X/Motif graphical application. This is a starting application for a minimal program.
<b>A Custom X/Motif Application</b>	Allows creation of a starting application that contains functioning code for many popular widgets. A series of dialog boxes allow for detailed customization of the X/Motif application, including selection of a toolbar, menu bar and a status bar.

**NOTE:** The first two choices above do not require further information from the developer. The VisualLinux Application wizard proceeds to summarize the information collected and create the project.

## A Custom X/Motif Application

The Custom X/Motif Application wizard (see next figure) allows you to select an application with a single top-level window or an application with multiple top-level windows.



**Figure 5-9: Window Levels on a Custom X/Motif Application**

The following table shows the **Custom X/Motif Application** dialog box controls:

**Table 5-5: Custom X/Motif Application Builder**

Feature	Function
<b>How many windows?</b>	Creates an application with either a single top-level window or multiple top-level windows.
<b>Add Menu Bar</b>	Creates an X11 Menu bar. The menu bar contains example menus such as the common <b>File</b> menu.

Table 5-5: Custom X/Motif Application Builder (Continued)

Feature	Function
Add Tool Bar	Creates an X11 Tool bar.
Add Status Bar	Creates an X11 Status bar.
Add Scroll Bar	Creates an X11 Scroll bar.

**NOTE:** If you select any of the **Menu Bar**, **Tool Bar**, or **Status Bar** items, the **Scroll Bar** option is also selected as defined by X11.

Selecting the **Next** button proceeds to the final X/Motif Custom Application dialog box.

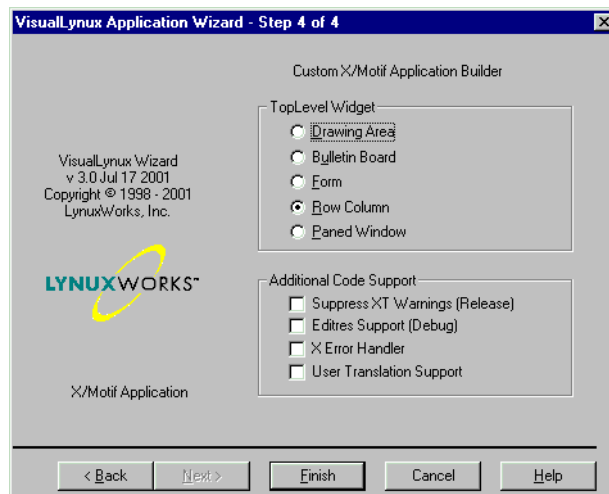


Figure 5-10: Custom X/Motif Application Final Dialog Box

The following table shows additional features of the X11 Application:

**Table 5-6: X11 Application Additional Features**

Feature	Function
<b>Top Level Widget</b>	The following choices are available for the top-level widget: <ul style="list-style-type: none"><li>• Drawing Area</li><li>• Bulletin Board</li><li>• Form</li><li>• Row Column</li><li>• Paned Window</li></ul>
<b>Additional Code Support</b>	The user can choose any combination of the following X11 features: <ul style="list-style-type: none"><li>• Suppress XT Runtime Warnings (good for production code).</li><li>• Edit Support (good for debugging code)</li><li>• X Error Handler (catches X11 error runtime exceptions and handles appropriately)</li><li>• User Translation Support</li></ul>

After making selections from the final X11/Motif dialog box, click the **Finish** button to complete the new project, or the **Cancel** button to continue editing project selections. A summary dialog box that displays project selections and files that are created and added to the new project is shown.

## Device Driver Wizard

The purpose of the Device Driver wizard is to help with writing a new LynxOS or BlueCat Linux driver. The Device Driver wizard creates template driver files that can be used for a dynamic or a static device driver. A dynamic device driver is compiled, linked and loaded separately from the LynxOS or BlueCat Linux kernel image. Because you do not modify the kernel, the system does not need to be rebooted when installing/uninstalling the dynamic device driver. A static device driver can be compiled separately from the LynxOS or BlueCat Linux kernel. To link it with the kernel, you need to import it to a VisuallyLinux Kernel Project (LynxOS only) or use a CDK command line mode to build the kernel (for both LynxOS and BlueCat Linux).

The dynamic device driver leads to a much faster compile/execute/debug cycle than the static device driver.

When selecting the **Device Driver** project type a dialog box offers the driver template choices depending on the operating system.



## LynxOS Device Drivers

When you select a LynxOS configuration for the device driver project type in the Application wizard, the dialog box shown in the next figure appears.

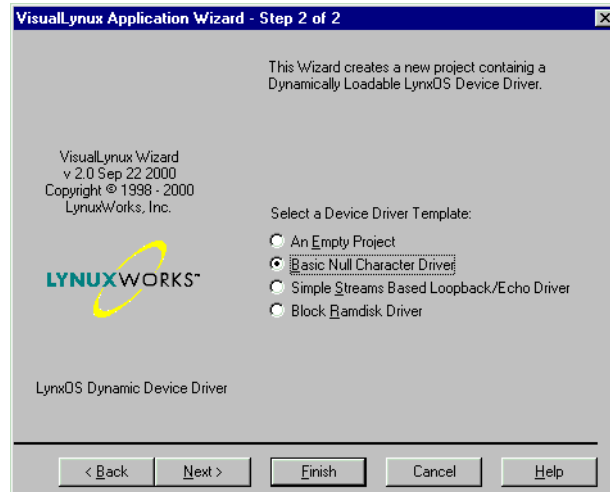


Figure 5-11: LynxOS Device Driver Types

The following table shows LynxOS device driver types:

Table 5-7: LynxOS Device Driver Types

Driver Type	Function
<b>An Empty Project</b>	Creates appropriate Visual Studio workspace and project files for a LynxOS Device Driver. Does not create any source code files. This is generally used when source code exists for importing into the VisualLynux project.
<b>Basic Null Character Device</b>	Creates a Null Character-based driver.
<b>Simple Streams Based Loopback/Echo Driver</b>	Creates a working Streams Loopback/Echo driver.
<b>Block RAMdisk Driver</b>	Creates a dynamically loadable RAMdisk driver.

After choosing a driver template, click **Finish** to see a summary of project settings.

At this point, the Application wizard creates starting project and source files to be compiled, linked, and modified.

The Device Driver wizard also creates `project_DriverInfo` and `project_DriverTest` projects in the same workspace, where `project` is the name of the Device Driver project. The VisualLynux project files define the compile option `-D_DYNAMIC_DRIVER` for all projects to generate the dynamic driver.

To create a static driver do the following:

1. Click on the **Project Settings** icon in the VisualLynux toolbar and select the **C/C++** tab.
2. Select **All configurations** from the **Settings for** drop-down list.
3. Delete `macro_DYNAMIC_DRIVER` from the **Preprocessor definitions** box.
4. Click **OK**.

The `project_DriverInfo` project contains the source files `projectinfo.h` and `projectinfo.c` containing the device information necessary to install a major device. The dynamic driver version of `projectinfo.c` also contains a `main()` program to generate the driver device information as a binary file.

The `project_DriverTest` project contains the test program `projecttest.c` for the driver. The VisualLynux Device Driver wizard always makes the `project_DriverTest` project active (current). Because `project_DriverTest` depends on the device driver project, which in its turn depends on the `project_DriverInfo` project, all projects are built when you press the **F7** key.

Click the **File/View** tab in the workspace window to see the projects and files tree.

## Testing a Dynamic Driver

To test a dynamic driver you should define a target machine for VisualLynux, copy all necessary files to the target machine, install the driver, create a device, and run the test. The whole procedure is described below assuming that `project` is the project name specified in the Application wizard **New** dialog box:

1. Click on the **Target Administration** icon in the VisualLynux toolbar to display the VisualLynux Target Administration dialog box.
2. Click on the **New target** button in the VisualLynux Target Administration dialog box. This invoked the VisualLynux Target wizard. Follow its instructions to define a target name, a target host name, an account name,

a password, a project directory on the target, etc. Be sure you have an account on the target machine with the same name and password. See the detailed description in Chapter 10, “VisualLynux Target Administration”

3. Be sure that the target machine is a default target. If its name is not highlighted in the VisualLynux Target Administration dialog box, select it and click the **Make default** button.
4. Be sure that the `project_DriverTest` is still the current project (i.e. it is highlighted in the workspace window). If it is not, set the test project as active using the **Project/Set Active Project** menu item. Or click on the **File/View** tab in the workspace window, select the `project_DriverTest` project, use the right mouse button to display the context menu, and select the **Set As Active Project** menu item.
5. Click on the **Upload** icon in the VisualLynux toolbar menu. This copies the driver test to the target machine.
6. Make the `project_DriverInfo` project active. Click on the **Upload** icon to copy the device information to the target machine.
7. Make the driver project active. Click on the **Upload** icon to copy the device driver to the target machine.
8. Now you are ready to install the dynamic driver on the target machine. Click on the **Telnet** icon in the VisualLynux toolbar and enter your account name and password on the target machine. Go to the project directory specified while creating the target description. If you haven't specified a project directory, your home directory is the project directory. The following three subdirectories are seen:

- `project`
- `project_DriverInfo`
- `project_DriverTest`

9. Go to the `project` directory. Enter `su` to become a superuser.

For a character or STREAM device driver:

```
drintall -c project
```

For a block device driver:

```
drintall -b project
```

If successful, the `drintall` command returns the unique driver ID that is defined internally by the LynxOS kernel. LynxOS block drivers are

always composite drivers including the code for raw (character) and block drivers. Therefore for a block driver, the driver ID is a logical OR of the character-driver ID in the low 16 bits and the block-driver ID in the upper 16 bits.

---

**NOTE:** You can run the **drivers** command to see driver IDs for installed drivers.

---

10. Go to the `project_DriverInfo` directory and run the device information program:

```
cd ../project_DriverInfo
./project_DriverInfo > projectinfo
```

This creates the device information file `projectinfo`.

11. Install the device.

For the character device:

```
devinstall -c -d driver_id project_nameinfo
```

For the block device:

```
devinstall -c -d raw_driver_id projectinfo
devinstall -b -e raw_major_no -d block_driver_id \
projectinfo
```

The **devinstall** command returns a major number of the installed device. The `driver_id` is the driver identification number returned in Step 9. This installs the appropriate device with the corresponding driver and assigns a major device number. For the block driver, the first **devinstall** command returns a `major_no` of the raw device. Use it in the second command with option **-e** to logically connect the raw and block devices.

12. Run the **devices** command to get the major number `major_no` assigned to the device. Use this number to create the device node.

For a character device:

```
mknod /dev/project c major_no 0
```

For a block device:

```
mknod /dev/rproject c raw_major_no 0
mknod /dev/project b major_no 0
```

The `mknod` command above specified `project` as a device name because this name is used in the test generated by the VisualLynux Device Driver wizard. To specify another device name, ensure that it is unique, and change the file name in the test. Although you can create a device anywhere in the file system, the `/dev` directory is recommended. The last parameter is the minor device number, which is specified as `0` for testing purposes. You can create several devices with minor device numbers in the range of `0-255`.

13. Go to the `project_DriverTest` directory and run the test.

For Basic Null Character and Simple Streams Echo drivers:

```
cd ../project_DriverTest
./project_DriverTest
```

For Block Ramdisk drivers you need first to create a file system and a mountpoint:

```
mkfs /dev/project
mkdir /mnt/project
mount /dev/project /mnt/project
cd ../project_DriverTest
./project_DriverTest
```

14. To uninstall the driver when it is no longer needed, uninstall the device.

For a character driver enter:

```
devinstall -u -c major_no
```

For a block driver enter:

```
devinstall -u -b major_no
devinstall -u -c raw_major_no
```

The `major_no` and `raw_major_no` are the same numbers used in Step 12 above.

After the device is uninstalled the driver can be uninstalled using the command:

```
drinstall -u driver_id
```

where `driver_id` is the number used in Step 11 above.

## BlueCat Linux Device Drivers

When you select a BlueCat Linux configuration for the type of device driver in the Application wizard, the following dialog box appears:

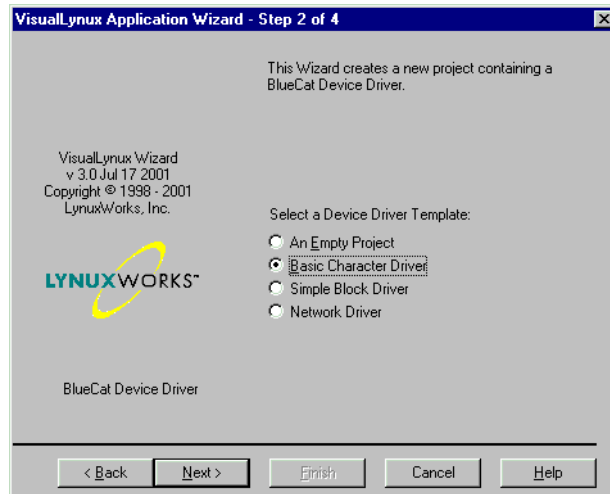


Figure 5-12: BlueCat Linux Device Driver Types

The BlueCat Linux device driver types are shown below:

Table 5-8: BlueCat Linux Device Driver Types

Driver Type	Function
An Empty Project	Creates appropriate Visual Studio workspace and project files for BlueCat Linux Device Driver. Does not create any source code files. This is generally used when there is existing source code that is to be imported into an empty VisualLynux project.
Basic Character Driver	Creates a working <a href="#">Character Driver</a> .
Simple Block Driver	Creates a working <a href="#">Block Driver</a> .
Network Driver	Creates a working, dynamically loadable network driver.

After choosing the driver template, click **Next** to proceed.

## Character Driver

When you select the **Basic Character Driver** type from the [BlueCat Linux Device Drivers](#) dialog box, the following screen appears:

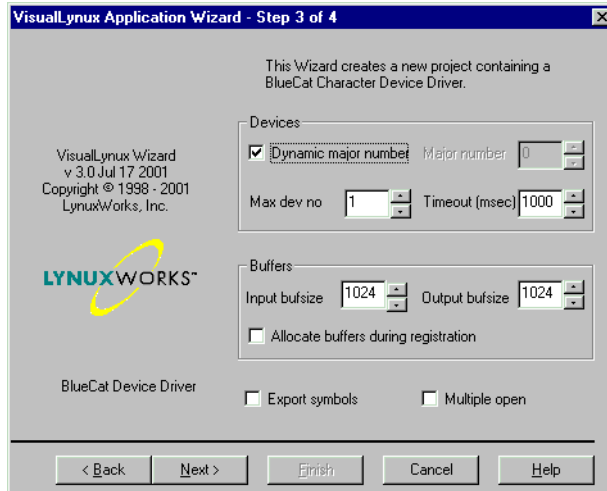


Figure 5-13: BlueCat Linux Character Driver Wizard

The following table shows Character Driver wizard controls:

Table 5-9: Character Driver Wizard

Control	Description
<b>Dynamic major number</b>	Uncheck this box to specify the major number explicitly. It is a good practice to keep this box checked to force dynamic assignment of the major number because BlueCat Linux driver major numbers depend on the kernel configuration.
<b>Major number</b>	Specifies major number value. Use this control to explicitly specify the driver major number. Else, check the <b>Dynamic major number</b> box.
<b>Max dev no</b>	Specifies the maximum number of devices supported by the driver.

Table 5-9: Character Driver Wizard (Continued)

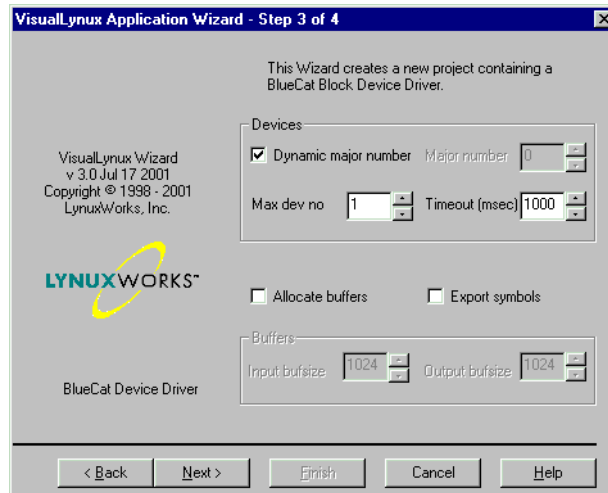
Control	Description
<b>Timeout (sec)</b>	Specifies the timeout value in milliseconds. Drivers use timeouts to recover from operations where operation with the device lasts too long or the device hangs. Leave as is if there is no use for this value.
<b>Input bufsize</b>	Specifies the input buffer size. If input buffer is not needed, set size to 0. Use spin control to modify the value.
<b>Output bufsize</b>	Specifies the output buffer size. If output buffer is not needed, set size to 0. Use spin control to modify the value.
<b>Allocate buffers during registration</b>	Check this box to allocate buffers during driver registration (i.e., during kernel booting for static drivers, or during dynamic driver installation). It is not recommended that you allocate buffers during registration because this wastes kernel memory when driver is unused.
<b>Export symbols</b>	Check this box to export some driver functions or variables. The wizard creates a skeleton table. Edit this later on to specify particular functions and/or variables for exporting.
<b>Multiple open</b>	Check this box if the driver must support multiple concurrent sessions with a device.

Click **Next** to display the [BlueCat Linux Driver Ports and IRQs](#) dialog box.

## Block Driver

When you select **Simple Block Driver** from the [BlueCat Linux Device Drivers](#) dialog box, the following screen appears:





**Figure 5-14: Block Driver Dialog Box**

The following table shows Character Driver wizard controls and their descriptions:

**Table 5-10: Block Driver Wizard**

Control	Description
<b>Dynamic major number</b>	Uncheck this box to explicitly specify a major number. It is a good practice to keep this box checked to force dynamic assessment of the major number. This is because BlueCat Linux driver major numbers depend on the kernel configuration.
<b>Major number</b>	Specifies major number value. Use this control to explicitly specify the driver major number. Otherwise, check the <b>Dynamic major number</b> box.
<b>Max dev no</b>	Specifies maximum number of devices supported by the driver.
<b>Timeout (msec)</b>	Specifies timeout value in milliseconds. Drivers use timeouts to recover from situations where operations with a driver lasts too long or the driver hangs. Leave as is if this value is not needed.
<b>Export symbols</b>	Check this box to export driver functionalities or variables. The wizard creates a skeleton table. Edit this table later on to specify particular functions and variables for exporting.

Table 5-10: Block Driver Wizard (Continued)

Control	Description
<b>Allocate buffers</b>	Check this box to allocate buffers in the <code>open ( )</code> method.
<b>Input bufsize</b>	Specifies the input buffer size. If you do not need input buffer, set its value to 0. Use spin control to modify the value.
<b>Output bufsize</b>	Specifies the output buffer size. If you do not need output buffer, set its value to 0. Use spin control to modify the value.

Click **Next** to display the [BlueCat Linux Driver Ports and IRQs](#) dialog box.

## Network Driver

When you select **Network Driver** from the [BlueCat Linux Device Drivers](#) dialog box, the following screen appears:

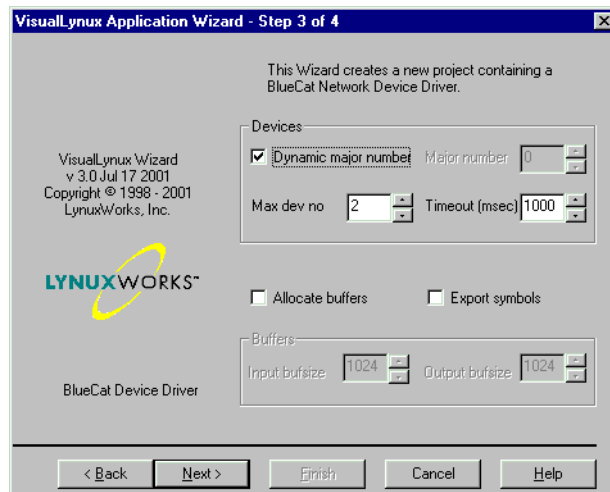


Figure 5-15: Network Driver Dialog Box

The following table shows Network Driver wizard controls:

**Table 5-11: Network Driver Wizard**

Control	Description
<b>Dynamic major number</b>	Uncheck this box if you want to specify major number explicitly. It is a good practice to keep this box checked to force dynamic assignment of the major number. This is because BlueCat Linux driver major numbers depend on kernel configuration.
<b>Major number</b>	Specifies major number value. Use this control to explicitly specify the driver major number. Otherwise, check the <b>Dynamic major number</b> box.
<b>Max dev no</b>	Specifies the maximum number of devices supported by the driver.
<b>Timeout (msec)</b>	Specifies timeout value in milliseconds. Drivers use timeouts to recover from situations where operation with a device lasts too long or the device hangs. Leave as is if this value is not needed.
<b>Export symbols</b>	Check this box to export some driver functions or variables. The wizard creates a skeleton table. Edit this later on to specify particular functions and/or variables for exporting.
<b>Allocate buffers</b>	Check this box to allocate buffers in the <code>open()</code> method.
<b>Input bufsize</b>	Specifies the input buffer size. If you do not need input buffer, set its value to 0. Use spin control to modify the value.
<b>Output bufsize</b>	Specifies the output buffer size. If you do not need output buffer, set its value to 0. Use spin control to modify the value.

Click **Next** to display the [BlueCat Linux Driver Ports and IRQs](#) dialog box.

## BlueCat Linux Driver Ports and IRQs

This is the final dialog box that appears when creating a BlueCat Linux [Character Driver](#), [Block Driver](#), or [Network Driver](#) project. By default, the **Board uses ports** and **Board generates IRQ** boxes are unchecked. Click **Finish** to create a project for a

pseudo-driver, that is, a driver that does not work with a real device. VisualLynux creates a *pseudo-driver project* depending on the selected driver type:

- **Basic Character Driver:** VisualLynux creates a working character driver that simulates the real device working with main memory.
- **Simple Block Driver:** VisualLynux creates a working RAM block driver that uses the kernel buffers simulating a block device.
- **Network Driver:** VisualLynux builds a skeleton driver project that can be used as a starting point to develop a real network driver.

If creating a real device driver, check the **Board uses ports** and **Board generates IRQ** boxes. The following dialog box appears:

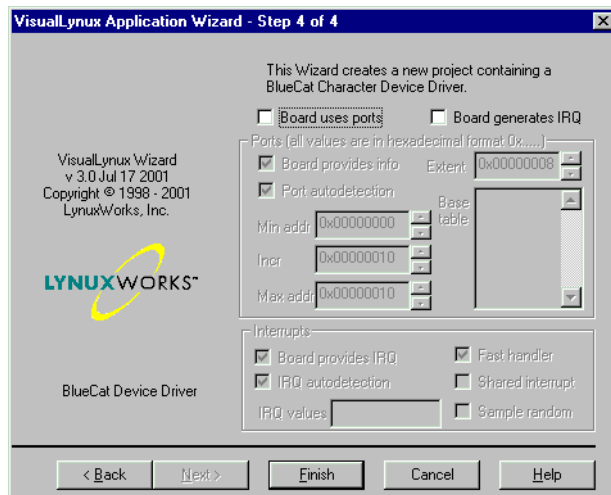


Figure 5-16: Ports and IRQ Dialog Box

The following table shows Ports and IRQ wizard controls:

Table 5-12: Ports and IRQ Wizard

Control	Description
Board uses ports	Check this box to specify information about port addresses.
Board generates IRQ	Check this box to specify information about IRQ lines.

Table 5-12: Ports and IRQ Wizard (Continued)

Control	Description
<b>Board provides info</b>	<p>If this box is checked, the board provides the device base port address(es). Leave box checked for PCI boards. The PCI standard specifies how the board is mapped to the address space and also describes the common header providing information on device ports and IRQ numbers. Add the code specific to your device later.</p> <p>Uncheck the box to probe the device.</p> <p><b>NOTE:</b> Sometimes, the probing procedure may be inappropriate, for example, for ISA boards. In this case, leave the box checked and edit the resulting source code later to explicitly specify information about device port addresses.</p>
<b>Extent</b>	<p>Specifies the address range the device uses for port numbers. Enter appropriate value in C-hexadecimal format (0x10, for example).</p>
<b>Port autodetection</b>	<p>If checked, the wizard generates skeleton code to automatically probe the device during registration. You need to specify minimum, maximum, and incremental values for port addresses.</p> <p>The probing procedure starts with the port base address specified in the <b>Min addr</b> box. If probing is unsuccessful, the next attempt is made for the base port address incremented by value specified in the <b>Incr</b> box. This continues till the base address reaches the maximum address specified in the <b>Max addr</b> edit box. Uncheck this box to probe a device using a list of base port addresses.</p>
<b>Min addr</b>	<p>Specifies the minimum base port address used for the autoprobng feature. Enter the value in C-hex format (0x100, for example).</p>
<b>Incr</b>	<p>Specifies the incremental value for the autoprobng procedure. Enter the value in C-hex format (0x40, for example).</p>
<b>Max addr</b>	<p>Specifies the maximum base port address for the autoprobng procedure. Enter the value in C-hex format (0x300, for example).</p>

Table 5-12: Ports and IRQ Wizard (Continued)

Control	Description
<b>Base table</b>	Specifies list of base port addresses to probe the device. Enter the values in C-hex format (0x300, for example). You can enter up to eight values.
<b>Board provides IRQ</b>	If checked, the board provides the IRQ line number. Leave unchecked for PCI boards. The PCI standard specifies how the board is mapped to the address space and describes the common header that provides information on device ports and IRQ numbers. Add code specific to your device later. Uncheck to probe the device for IRQ number. <b>NOTE:</b> Sometimes, probing is inappropriate, for example, for ISA boards. In this case, leave the box checked. Edit the resulting source code later to explicitly specify information about device IRQ numbers.
<b>IRQ autodetection</b>	If checked, the wizard generates skeleton code to automatically probe the device for IRQ during registration. You need to specify a list of IRQ values in the <b>IRQ values</b> box. The probing procedure tries them for each board in the order they appear. Uncheck the box if you want to explicitly specify a list of IRQ values, one for each board. If the number of IRQ values is less than the maximum number of boards, the last IRQ value is used to probe all other boards.
<b>IRQ values</b>	Specifies list of IRQ values for IRQ probing. Enter values separated by a comma ( , ). <b>NOTE:</b> The <b>IRQ values</b> list cannot be empty.
<b>Fast handler</b>	Specifies the type of interrupt handler routine (SA_INTERRUPT). Although (Linux 2.12 onwards) there is no difference between fast and slow handlers, the flag can still be useful.

**Table 5-12: Ports and IRQ Wizard (Continued)**

Control	Description
<b>Shared interrupt</b>	Specifies that the driver processes shared interrupts. It means that the IRQ line number can be used by different devices and the kernel propagates interrupts to all devices. You need to add code that determines whether the interrupt was produced by the board supported by the driver.
<b>Sample random</b>	Specifies whether the generated interrupts can contribute to the entropy pool used by <code>/dev/random</code> and <code>/dev/urandom</code> .

Click **Finish** to create a BlueCat Linux driver project.

---

**NOTE:** You also need to edit the resulting source files to implement the specific device code. Look for `// TODO:` lines in the code where you should add or modify code for the particular device.

---

The Device Driver wizard also creates a `project_DriverTest` project in the same workspace, where `project` is the name of the Device Driver project. The VisualLynux project files define the compile option `-D_DYNAMIC_DRIVER` for all projects to generate the dynamic driver. To create a static driver do the following:

1. Click on the **Project Settings** icon in the VisualLynux toolbar and select the **C/C++** tab.
2. Select **All configurations** from the **Settings for** drop-down list.
3. Delete `macro_DYNAMIC_DRIVER` from the **Preprocessor definitions** box.
4. Click **OK**.

The `project_DriverTest` project contains the test program `projecttest.c` for the driver. The VisualLynux Device Driver wizard always makes the `project_DriverTest` project active (current). Because the `project_DriverTest` project depends on the device driver project, both project are built when you press **F7**.

Click the **File/View** tab in the workspace window to see the projects and files tree.

## Testing a Dynamic Driver

To test a dynamic driver you should define a target machine for VisualLynux, copy all necessary files to the target machine, install the driver, create a device, and run the test. The whole procedure is described below assuming that *project* is the project name specified in the Application wizard **New** dialog box:

1. Click on the **Target Administration** icon in the VisualLynux toolbar to display the VisualLynux Target Administration dialog box.
2. Click on the **New target** button in the VisualLynux Target Administration dialog box. This invokes the VisualLynux Target wizard. Follow its instructions to define a target name, a target host name, an account name, a password, a project directory on the target, etc. Be sure to have an account on the target machine with the same name and password. See the detailed description in Chapter 10, “VisualLynux Target Administration”
3. Be sure that the target machine is a default target. If its name is not highlighted in the VisualLynux Target Administration dialog box, select it and click the **Make default** button.
4. Be sure that the *project\_DriverTest* is still the current project (i.e. it is highlighted in the workspace window). If it is not, set the test project as active using the **Project/Set Active Project** menu item. Or click on the **File/View** tab in the workspace window, select the *project\_DriverTest* project, use the right mouse button to display the context menu, and select the **Set As Active Project** menu item.
5. Click on the **Upload** icon in the VisualLynux toolbar menu. This copies the driver test to the target machine.
6. Make the driver project active. Click on the **Upload** icon to copy the device driver to the target machine.
7. Now you are ready to install the dynamic driver on the target machine. Click on the **Telnet** icon in the VisualLynux toolbar and enter your account name and password on the target machine. Go to the project directory specified while creating the target description. If you haven't specified a project directory, your home directory is the project directory. The following subdirectories are seen:
  - *project*
  - *project\_DriverTest*
8. Go to the *project* directory. Enter **su** to become a superuser.
9. Enter the following command to install the dynamic driver:



```
/sbin/insmod ./project arguments
```

where *arguments* are additional parameters passed to the driver during installation. The template drivers created by the Application wizard do not need any additional parameters to run tests.

If the target operating system version differs from the BlueCat Linux CDT version you get a warning message. In such a case there is no guarantee that your driver will be functional.

10. Run the following command to see major number assigned to the driver:

```
cat /proc/devices
```

11. The output contains a list of character and block drivers with their major numbers and names. Use the major number for the installed driver and enter the following command to create the device node:

For a character device enter:

```
mknod /dev/project c major_no 0
```

For a block device enter:

```
mknod /dev/project b major_no 0
```

The **mknod** command above specified *project* as a device name because this name is used in the test generated by the VisualLinux Device Driver wizard. To specify another device name, be sure it is unique, and change the file name in the test. Although you can create a device anywhere in the file system, the */dev* directory is recommended. The last parameter is the minor device number, which is specified as **0** for testing purposes. You can create several devices with minor device numbers in the range of 0-255.

12. Go to the *project\_DriverTest* directory and run the test.

```
cd ../project_DriverTest
```

A)For a Basic Character driver:

```
./project_DriverTest
```

B)For a Simple Block driver you need first to create a file system and a mountpoint:

```
mkfs /dev/project
```

```
mkdir /mnt/project
```

```
mount /dev/project /mnt/projectr
```

```
./project_DriverTest
```

C) For a Network driver you need first to prepare a simulated network environment to test the driver on your target. Enter the following commands to create additional interfaces:

```
ifconfig project0 local0
```

```
ifconfig project1 local1
```

Run an editor (e.g. vim) and add the following lines to the file `/etc/hosts`:

```
192.168.2.88 local0
```

```
192.168.2.99 remote0
```

```
192.168.3.99 local1
```

```
192.168.3.88 remote1
```

The IP addresses shown above are selected from the range of public addresses and should not interfere with those used in your local network. If they do, select different addresses provided the following conditions are met:

- The subnet addresses for hosts ending with **0** must differ from hosts ending with **1** only in lowest bit of the third octet. For example, instead of subnets `192.168.2` and `192.168.3`, you can specify `172.17.6` and `172.17.7`.
- The forth octet for `local0` and `remote1` addresses must be the same, as well as for `remote0` and `local1`.

The above conditions are related to the implementation of the template network driver that simulates two subnets by changing the lowest bit in the IP headers of the packets transmitting through the driver.

13. Now you can run `ping` to see if the simulated subnets work:

```
ping -c 2 remote0
```

```
ping -c 2 remote1
```

14. If the `ping` commands transmit packets successfully, run the test:

```
./project_DriverTest
```

To uninstall the driver when it is no longer needed:

```
rmmod project
```

Before removing a network driver you need to close the interfaces:

```
ifconfig project0 down
```

```
ifconfig project1 down
```

## Static Library Application Wizard

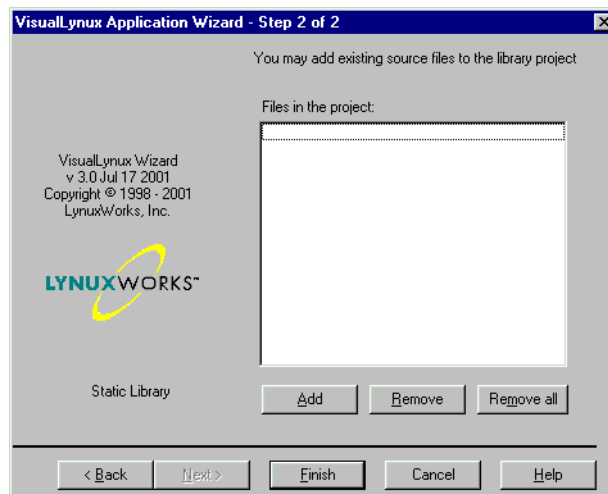
The purpose of the Static Library Application wizard is to help the programmer get started with building a static library.

---

**NOTE:** You must provide a project name beginning with `lib...` characters.

---

When you select **Static Library** and click **Next**, the following dialog box appears:



**Figure 5-17: Static Library Dialog Box**

You can add or remove source and header files that are used to build a static library. You can also add or remove files after the project has been created. To add files to the project, press **Add** and select files to the list box. To remove files select them in the list box and click **Remove**. **Remove all** removes all files from the list box.

Clicking the **Finish** button completes the new project. **Cancel** is used to continue editing project selections. A summary dialog box that displays your selections and the files that are created and added to the new project is shown. An additional test project is also built to test your library.



---

## Kernel Projects

To create LynxOS Kernel projects, use the VisualLynux LynxOS Kernel wizard (Kernel wizard). The Kernel wizard provides a set of dialog boxes that help create a workspace to build LynxOS 3.1.0a kernels, and optionally, Kernel Downloadable Images (KDIs). The Kernel wizard allows you to select a Board Support Package (BSP), specify projects to be included in the workspace, select source code to be copied into the working directory for modification, and set up an initial kernel configuration parameters as well as KDI properties.

A new kernel workspace contains one to five interdependent projects:

- The main *Kernel Executable* project that builds a kernel executable (usually `a.out`), `nodetab`, and optionally, *Kernel Downloadable Image (KDI)* files
- The *Generic Devices Library* project that builds a generic device library for the selected target platform (x86, PowerPC, or MIPS)
- The *Generic Device Drivers Library* project that builds a generic device driver library for the selected BSP
- The *Specific BSP Device Library* project that builds a device library for the selected BSP
- The *Specific BSP Device Drivers Library* project that builds a device drivers library for the selected BSP

The actual number of projects depends on the selected BSP and user needs. The Kernel wizard always creates the main Kernel Executable project, and allows you to specify the projects you need. If you do not create any Kernel Library projects, the appropriate library from the CDK is used to link the kernel executable. In this case, you cannot edit any driver source code, but you have the ability to configure the kernel by setting up drivers and kernel parameters.

If you do create one or more Kernel Library projects, the Kernel wizard allows you to select the source code that you are going to change. Selected source and header files are copied to the working directory from the CDK installation. The Library project builds the modified kernel library, used to link the Kernel Executable. All unmodified components (drivers, devices, and so on) are copied from the source library.

The Kernel wizard makes the main Kernel Executable project dependent on all Kernel Library projects. Every time the source code is changed, the appropriate library and kernel are rebuilt.

---

## Launching the VisualLynux LynxOS Kernel Wizard

To run the Kernel wizard and create a new kernel workspace

1. Select **File->New** from the Visual Studio menu bar.
2. Then select the **Projects** tab in the **New** dialog box. The left pane of the dialog box displays the wizards available.
3. Select **VisualLynux LynxOS Kernel Wizard**.

---

**NOTE:** If this item does not appear in the left pane, reinstall VisualLynux.

---

4. Enter a name for the new project. The project name is used as the workspace name.

Note that the Kernel wizard disables the **Add to current workspace** radio button and selects the **Create new workspace** button. This is because the Kernel wizard can create several interdependent projects (up to five), each of which must be placed in the same separate workspace.

5. Select a directory where the new workspace is to be located using the **Location** edit box.

You can also use the button to the right of this field to browse for a directory in which to create a kernel workspace.

Note that Visual Studio automatically appends the project name to the selected directory and creates a new workspace directory. To create a project in another directory, manually edit data in the **Location** box.

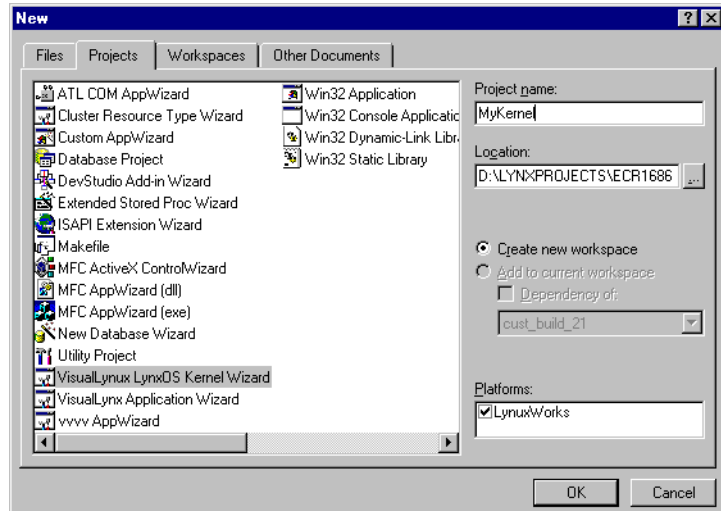


Figure 6-1: LynxOS Kernel Wizard

6. Click **OK** to launch the VisualLynx LynxOS Kernel wizard.

## Selecting a Board Support Package

The first step of the Kernel wizard is shown in the figure below. Select a target platform and a Board Support Package (BSP). When you choose a platform using the **Target Platform** drop-down list, the **Supported Kernels** list box displays available BSPs. Select one to create a the kernel workspace. When finished with selection, press **Next** to continue. The Kernel wizard creates the main Kernel Executable project.

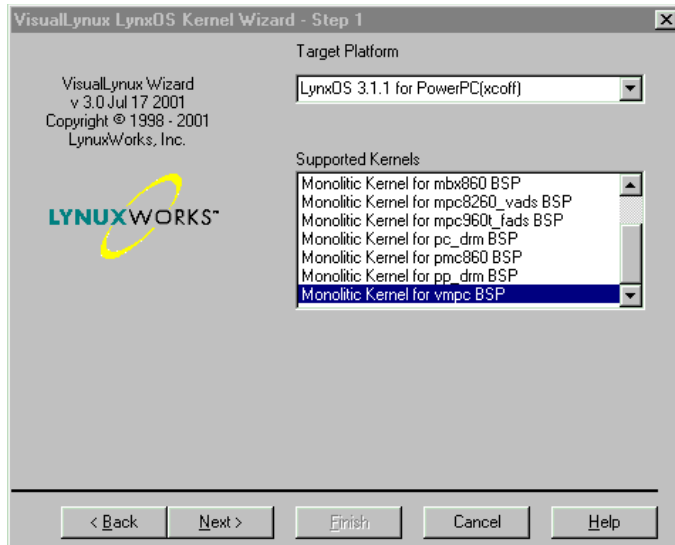


Figure 6-2: Selecting a BSP

The Kernel wizard fills the **Supported kernels** list box by looking in the kernel subdirectory of the appropriate CDK. If there is no BSP subdirectory for a selected target platform, the Kernel Wizard displays the following dialog box. You will not be able to continue creation of the Kernel workspace and will need to invoke the **Platform Administrator** to modify installation data.



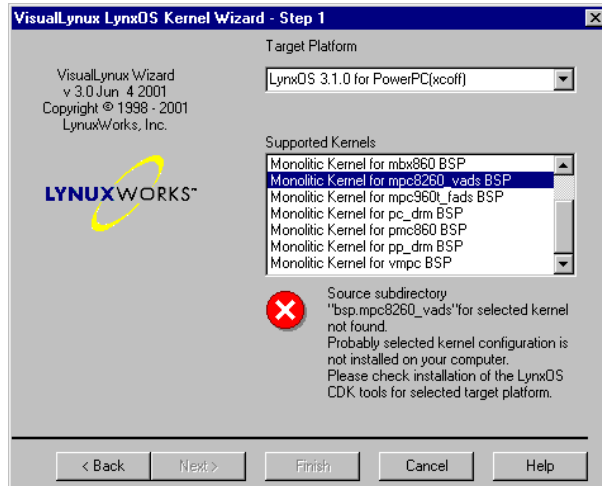


Figure 6-3: Invalid BSP Error Message

The Kernel wizard always creates the same build configuration for all projects in the workspace. By default, the Kernel wizard assigns the name “*Release*” for the build configuration. Change it as needed by editing the **Configuration** edit box. Note that the actual build configuration name contains a platform-dependent prefix. For example, if you create a kernel project for the LynxOS PowerPC platform and do not change the configuration name, the build configuration is named `LynxOS_ppc_xcoff_Release`. This convention provides consistency in case you wish to add other projects to this workspace.

## Preparing a Kernel Executable Project

The Kernel wizard starts this process after you select a BSP and press the **Next** button in the BSP Selection step. The kernel executable project preparation consists of the following steps:

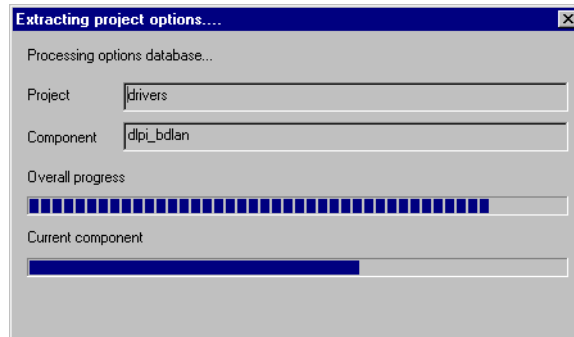
1. Extracting linker options used to link the kernel
2. Extracting compiler options used to compile configuration source files (`conf.c`, `info.c`).
3. Creating a working copy of the kernel configuration data: `CONFIG.TBL` and driver configuration files (`cfg`)

#### 4. Extracting source file dependencies

The Kernel wizard specially processes the Makefile that comes with a BSP to extract *linker and compiler options*.

The Kernel wizard runs the compiler and collects *source file dependency information* from generated `.d` files.

Throughout the process, the Kernel wizard displays a progress dialog box. Errors are displayed as appropriate.



**Figure 6-4: Kernel Wizard Progress**

The Kernel wizard creates a working copy of the *kernel configuration files* (`CONFIG.TBL` and driver configuration files). These files are copied from the CDK directory to the Configuration Data subdirectories.

---

**NOTE:** Project configurations allow the use of different options to build different versions of a project goal (e.g., a kernel in a kernel project) for the same set of files, for example, Debug and Release versions of a kernel. Kernel projects, unlike ordinary application projects, also use specific configuration data to build a kernel. To allow handling of configuration data depending on a project configuration, a separate Configuration Data subdirectory is created, and has a name derived from the configuration name with a “\_CFG” suffix. For example, the Configuration Data subdirectory for the Release kernel configuration on a PPC platform is named `LynxOS_ppc_xcoff_Release_CFG`.

---

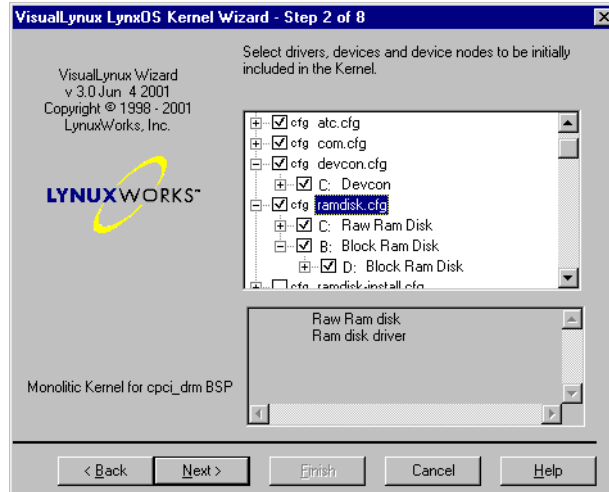
After preparation is complete, the Kernel wizard displays the **Kernel Configuration** dialog box. This allows you to select a primary kernel configuration (drivers, devices, device nodes, and so on). The dialog box displays `CONFIG.TBL` and

driver configuration files in a tree view. To include a component, check the corresponding box.

---

**NOTE:** The Kernel wizard copies all configuration files, regardless of their initial selection state, to provide the ability to edit the configuration at a later time.

---



**Figure 6-5: Kernel Configuration Dialog Box**

When you have finished selection, press **Next** to display the **Kernel Parameters** dialog box. A table representing lines from the corresponding `uparam.h` file shows parameter names, current values, and default values. To edit a value, select the appropriate line in the table. Edit the parameter value that appears in the **Value** edit box.

Some lines in the table define parameter values as C expressions. You do not normally need to edit these. However, the Kernel wizard does not prevent you from doing so.

---

**NOTE:** Modifying such values may cause an invalid kernel build.

---

VisualLynux creates and stores the `uparam.h` file for every project configuration. The file name is `configuration_name_uparam.h`, where `configuration_name` stands for the build configuration. For example,

VisualLynux creates the file `LynxOS_x86_coff_Release_uparam.h` for the project configuration `Win32_LynxOS_x86_coff_Release`.

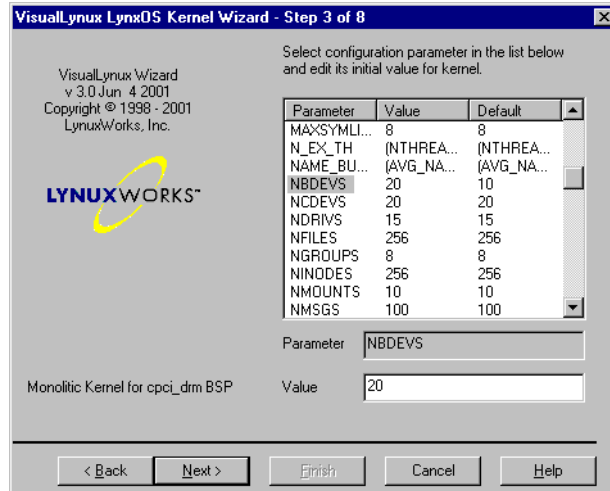


Figure 6-6: Kernel Parameters Dialog Box

Press **Next** to continue.

## Configuring Kernel Downloadable Image (KDI)

The fourth step of the Kernel wizard is shown in figure below. To create a *Kernel Downloadable Image (KDI)* based on the kernel project, check the appropriate box. To skip KDI creation, press **Next** and the Kernel wizard skips all dialog boxes that configure the KDI project. You can, however, add the KDI to the kernel project in the future.

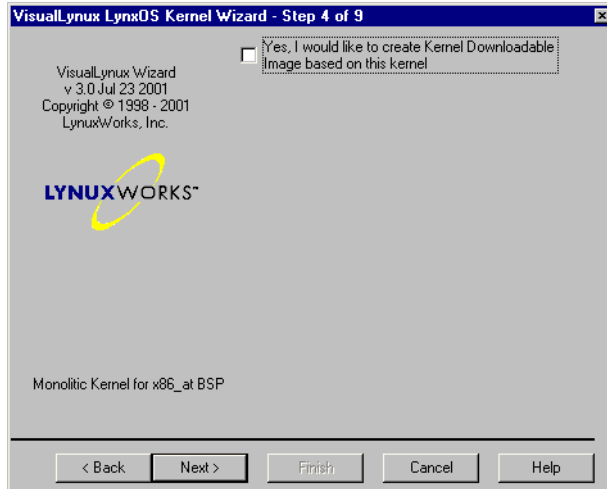


Figure 6-7: Kernel Wizard Step 4

If the box is checked, the dialog box displays additional controls that allow you to configure a KDI. The Kernel wizard also adds an additional step to configure the target file system to be included in the KDI. When finished with the dialog box, press **Next**.

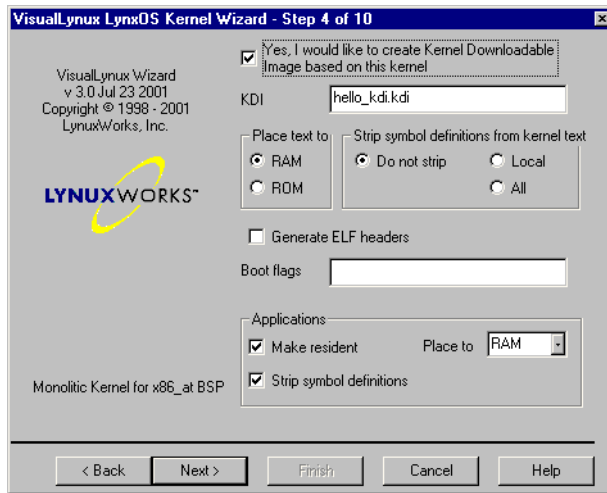


Figure 6-8: KDI Configuration Box

The following table explains the controls in the KDI configuration box:

**Table 6-1: KDI Configuration Controls**

Control	Description
<b>KDI</b>	Name of the Kernel Downloadable Image file - The Kernel wizard creates the default KDI name by appending a <code>.kdi</code> extension to the project name. The KDI file is always created in the <i>output</i> project directory.
<b>Place text to</b>	These radio buttons designate where the kernel is to reside in the running system, RAM, or ROM
<b>Strip symbol definitions from kernel text</b>	Select <b>Do not strip</b> if the symbol definitions should not be stripped from the kernel text file. Check <b>Local</b> if only local symbol definitions should be stripped, and <b>All</b> if all symbols should be stripped.
<b>Generate ELF headers</b>	Check if ELF headers must be generated for netbooting by the firmware that requires ELF format files.
<b>Boot flags</b>	Enter boot flags to be passed to the kernel. “-” flag (single-user) is not allowed. See the <code>reboot(1)</code> man page for valid values.
<b>Applications</b>	Allows you to specify properties of the application text files added to the KDI.
<b>Make resident</b>	Check if the application programs should be resident in memory.
<b>Strip symbol definitions</b>	Check this box if application files are to have symbols stripped before being placed into the ROM kit file.
<b>Place to</b>	Select the location of the resident application programs (RAM or ROM)

## Configuring KDI File System

Next, the Kernel wizard allows you to specify the initial configuration of the KDI root file system.

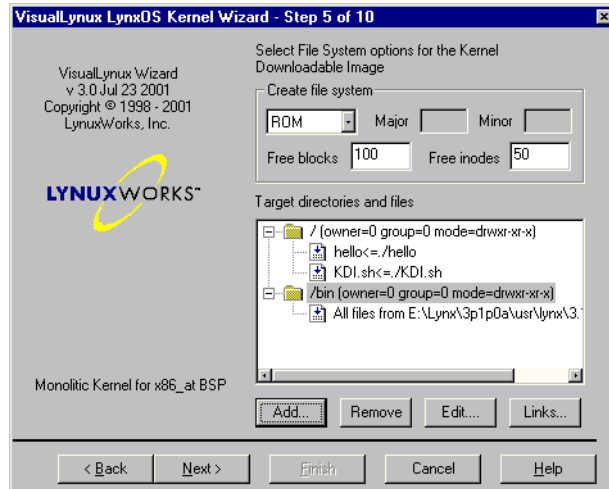


Figure 6-9: KDI Root File System Creation

The set of controls in the table below allow you to specify general properties of the file system to be created in the KDI. Select the location of the root file system using the drop-down list in the upper left corner of these controls.:

Table 6-2: Create File System Controls

Control	Description
Do not create	Do not create the file system - When this option is selected, all other controls are disabled.
ROM	A ramdisk resident in Read Only Memory
RAM	A ramdisk resident in Random Access Memory
Other device	A disk as specified by the device's <b>Major</b> and <b>Minor</b> numbers

The following table describes other controls in the KDI root file system creation dialog box:

**Table 6-3: KDI Root File System Controls**

Control	Description
<b>Major, Minor</b>	These boxes are enabled only if <b>Other device</b> is selected from the drop-down list. Enter the major and minor device numbers of the disk on which the root file system is to be created.
<b>Free blocks</b>	Number of free blocks to be generated for the file system
<b>Free inodes</b>	Number of free inodes to be generated for the file system
<b>Target directories and files</b>	Displays the set of directories and files to be created on the root file system.
<b>Add</b>	Invokes the dialog box to add a new directory to the target file system.
<b>Remove</b>	Removes the currently selected item in the <b>Target directories and files</b> control.
<b>Edit</b>	Invokes the dialog box to edit data associated with item currently selected in the tree.
<b>Links</b>	Invokes the dialog box to edit the list of hard and soft links to be created on the target file system.

The **Target directories and files** control displays the directory path, owner, group, and mode set for every target directory. The directory child items display files to be placed in the directory. The buttons below the tree view allow you to add a new directory, remove a directory or file, edit properties, or create hard or soft links.

The **Links** button invokes the following dialog box:



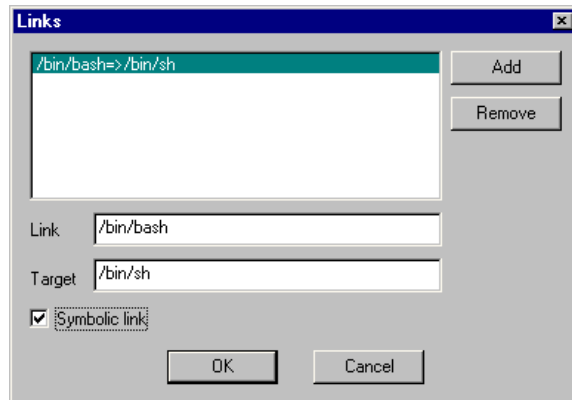


Figure 6-10: KDI RFS Links Dialog Box

To add a new link to the list, press the **Add** button. Then enter the path name of the link file in the **Link** edit box. Enter the full target file path name for the link in the **Target** edit box. If creating a symbolic link, check the **Symbolic link** box.

To edit a link definition, select it in the list and edit data in the **Link**, **Target** and **Symbolic link** fields. To delete a link, select it in the list and click the **Remove** button.

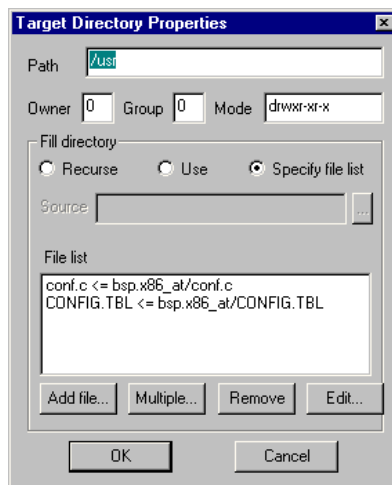


Figure 6-11: Target Directory Properties Edit Box

The table below describes the controls in the **Target Directory Properties** edit box:

**Table 6-4: KDI Target Directory Properties**

Control	Definition
<b>Path</b>	The full UNIX-like path name of the directory to be created on the target
<b>Owner</b>	The numeric owner ID of the target directory
<b>Group</b>	Numeric group ID of the target directory
<b>Mode</b>	The file mode of the target directory - The mode should be specified as a 10-character string in the “ <code>ls -l</code> ” style. To change any character, place the cursor at that point and type. Blanks are treated as a dash “-”
<b>Fill directory</b>	Designates the mode in which the directory is to be filled.
<b>Recurse</b>	Fills the directory with all the files from the <b>Source</b> directory, including subdirectories.
<b>Use</b>	Fills directory with all files from the <b>Source</b> directory only.
<b>Specify file list</b>	Explicitly specifies list of files to be created in the directory.
<b>Source</b>	The path name of the directory used in the <b>Recurse</b> and <b>Use</b> modes - Disabled if the <b>Specify file list</b> option is selected. Enter an absolute or relative DOS path name to the source directory or browse to select a directory.
<b>File list</b>	Displays the list of files to be created in the target directory. Disabled if the <b>Recurse</b> or <b>Use</b> options are selected.
<b>Add file</b>	Invokes the dialog box to add a new file to the file list.
<b>Multiple</b>	Invokes the dialog box that allows adding multiple files to the list
<b>Remove</b>	Removes the selected file from the list.
<b>Edit</b>	Invokes the dialog box to edit file properties for selected file.

The **File list** displays the files that are to be created in the target directory in the following format:

```
file_name<=source_file_path
```

where *file\_name* is the name in the target directory, and *source\_file\_path* is the absolute or relative DOS path name of the source file.

The **Add file** button invokes the following dialog box. The same dialog box is invoked by the **Edit** button.

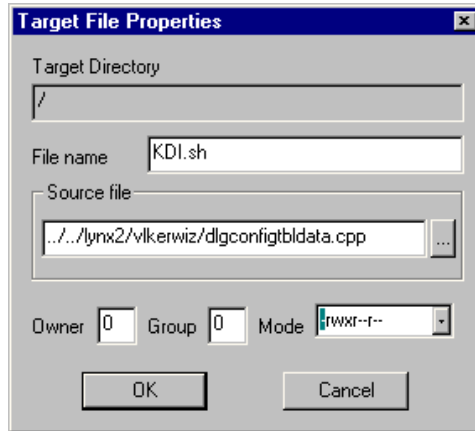


Figure 6-12: Adding a File to the KDI Target Directory

The following table explains the controls in the previous figure:

Table 6-5: KDI Target File Property Controls

Control	Description
<b>Target Directory</b>	Displays the target directory path.
<b>File name</b>	File name as it should appear in the target directory - If the field is empty while you use the <b>Browse</b> button, it is filled with the name of the source file.
<b>Source file</b>	The full or relative DOS path name of the source file to be copied to the target directory - You may use <b>Browse</b> to search for the file.
<b>Owner</b>	Numeric owner ID of the file
<b>Group</b>	Numeric group ID of the file
<b>Mode</b>	Permission mode of the file - The mode should be specified as a 10-character string in the "ls -l" style. To change any character, place cursor at that point and type. You can also use the drop-down list to select a permission mode.

To add several files from the same directory, use the **Multiple** button. It invokes the Windows **Open File** dialog box with three additional controls that allow you to set

an Owner ID, a Group ID and a Mode for all files selected. The **Mode** drop-down list also allows you to select an initial value from a predefined set of values.

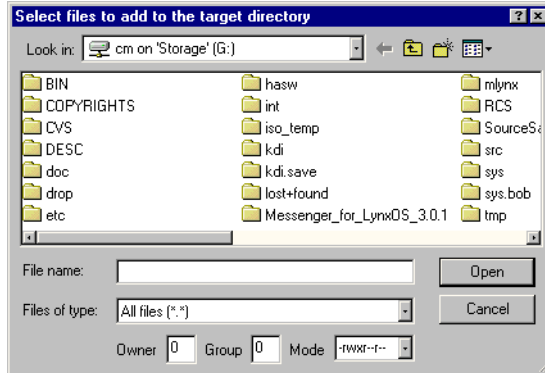
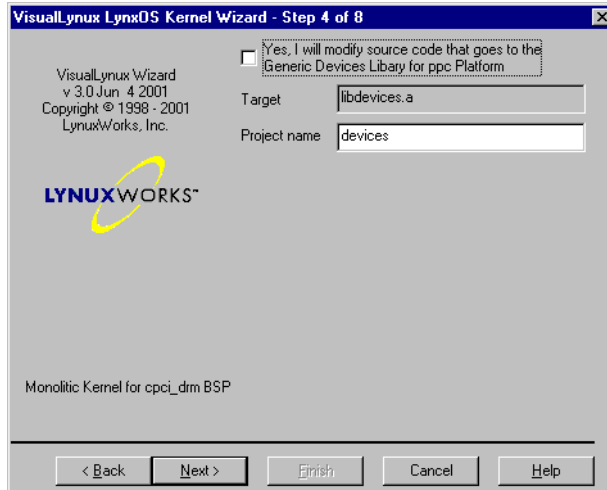


Figure 6-13: Adding Multiple Files to KDI Target Directory

---

## Selecting Kernel Library Projects

After you have finished configuring the *Kernel Executable* and *KDI* project, the Kernel wizard takes you through a set of dialog boxes to select *Kernel Library* projects to be included in the kernel workspace. Each dialog box corresponds to a library type. By default, the Kernel wizard does not create any library projects. You need to create such sub-projects only if you are going to modify source code that comes with the CDK (for example, if you wish to create your own version of a particular driver). Also you may create an empty Kernel Library project to add your own drivers/devices to the kernel in the future.



**Figure 6-14: Library Project Creation**

To create a library project, check the box at the top of the dialog box. The **Target** edit box displays the name of library to be built. The **Project name** edit box shows the default project name assigned by the Kernel wizard. You can change the project name as appropriate.

For proprietary reasons, CDKs may not include source code for several drivers/devices, causing errors while processing such components. In most cases, this means that you cannot modify the source code of the erratic component in VisualLynux or in the `bash` command line environment.

When the Kernel wizard finishes preparing for a Kernel Library project, it updates the dialog box and shows the components, source and header files in a tree view. You should either check **Create empty kernel library project** or select at least one component to be included in the project. Otherwise, the Kernel wizard displays an error message when you press **Next** button. Make your selections by checking the boxes next to the appropriate files. Note that when you select an upper-level component, all components in the appropriate subtree are also selected. When **Create empty kernel library project** is checked, the selection tree is grayed out and the Wizard ignores all selections.

When you select a component in the tree, the Kernel Wizard starts preparation for this component. If there is an error, the Kernel wizard displays an appropriate

message box and marks the erratic components with red icons to denote errors. Do not include such components in the project.

Usually, devices and drivers libraries are closely interrelated; you probably need to modify the corresponding source code for both libraries.

If you do not want to create a project, uncheck the box at the top of the dialog box. When you finished with component selection press the **Next** button. The Kernel wizard displays the dialog box for the next project or the **Copy files** dialog box.

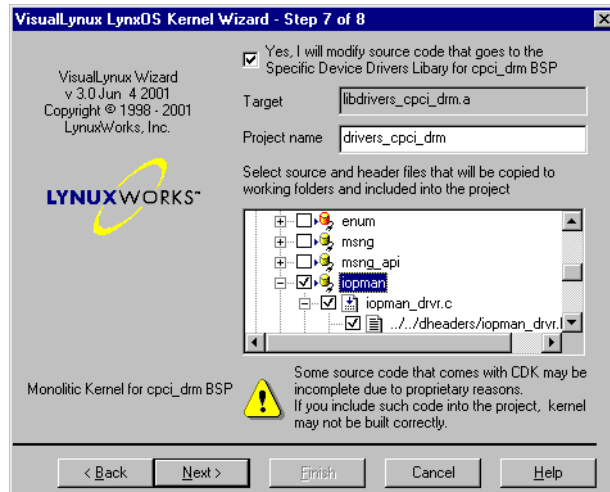


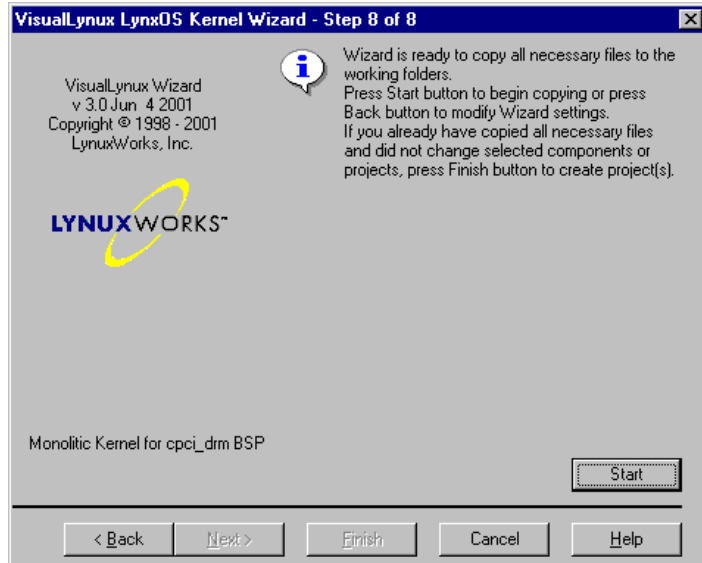
Figure 6-15: Library Project Content Selection

## Copying Source Files

Finally, the Kernel wizard asks permission to start the process of copying files. Press **Start** to copy files from CDK directories to your working directories. Or, press **Back** to return to previous steps.

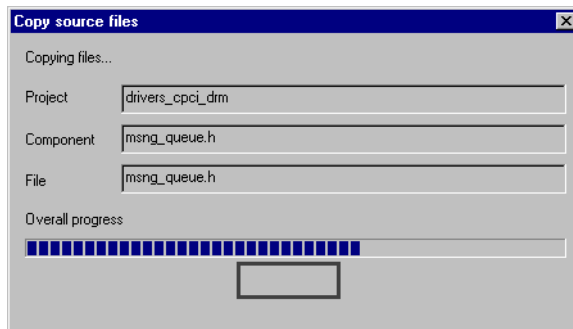
**NOTE:** The **Next** and **Finish** buttons are disabled when the dialog box appears for the first time. They are enabled if you have copied files at least once. If you have not made any changes that require copying files, you can skip this step and press **Next** or **Finish** to complete creating the project.

If you repeat the process of copying files, the Kernel wizard removes all files previously copied before proceeding.



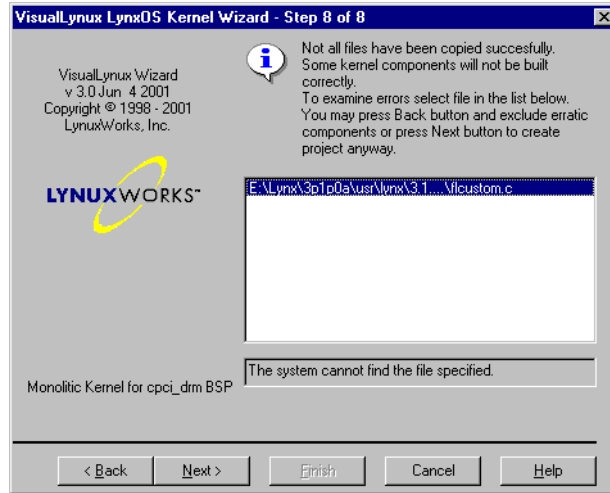
**Figure 6-16: Kernel Wizard Copy Files Dialog Box**

The number of files copied depends on selected projects and library components. The Kernel wizard displays a progress indicator box.



**Figure 6-17: File Copy Progress**

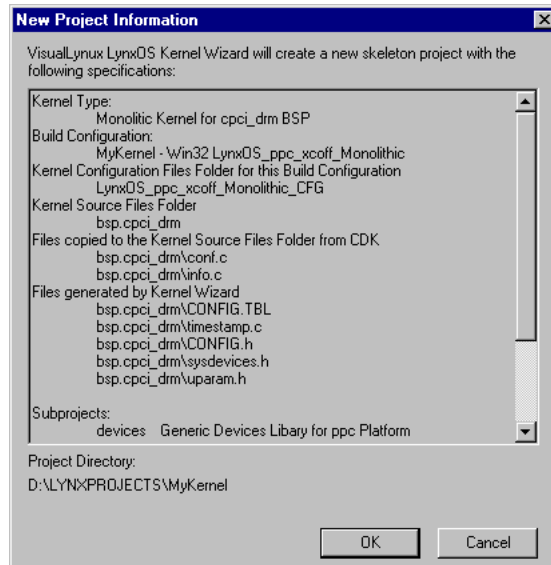
After copying files, the Kernel wizard updates the dialog box above. The dialog box contains a list of source files copied with errors. Such errors may occur if you accidentally select an erratic component. For an error description, select a source file from the list of files copied with errors. Press the **Back** button to modify your selections. If you ignore errors, the kernel does not build correctly. You can exclude invalid components later.



**Figure 6-18: Error Message While Copying Files**

If there are no copy errors, the message at the top of dialog box indicates that all files have been copied. Press the **Next** or **Finish** button to view the **New Project Information** dialog box. This box displays information about the *Kernel Executable* project to be created, the name of the *Configuration Data Subdirectory*, and names and descriptions of each *Kernel Library* project included in the workspace. Press **OK** to create all selected projects, or press **Cancel** to return to the previous steps.





**Figure 6-19: New Project Information**

A sample workspace created for the x86\_at Board Support Package is displayed in the following figure. It is assumed that the project has been created to modify a high level SCSI hard disk driver.

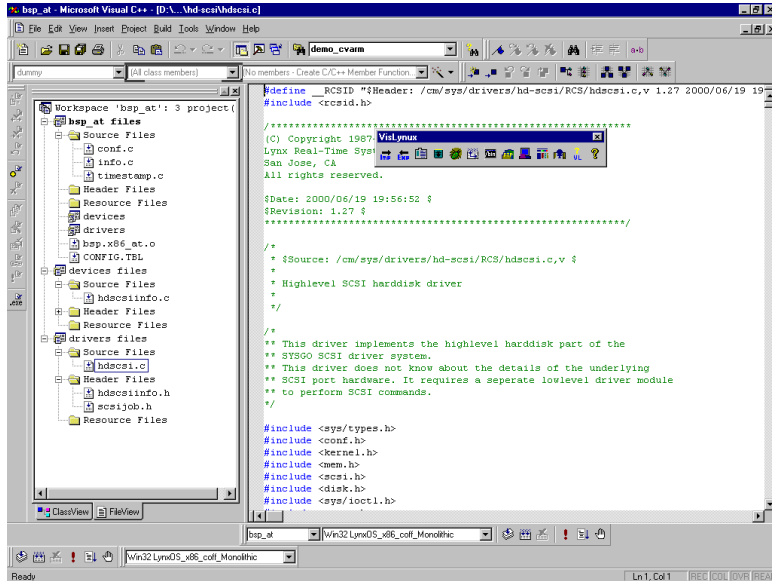


Figure 6-20: Sample Workspace

---

## IDE Tools

The VisualLynux IDE Tools provide additional functionality to the standard Visual Studio environment and help increase productivity for real-time embedded systems development. VisualLynux AddIn Tools include the following functions:

- [Import a Makefile](#)
- [Export to Makefile](#)
- [VisualLynux Project Settings](#)
- [LynuxWorks FTP](#)
- [LynuxWorks GDB](#)
- [LynuxWorks Total/db](#)
- [Upload](#)
- [LynuxWorks Cross Process Viewer](#)
- [Telnet](#)
- [VisualLynux Target Administration](#)
- [VisualLynux HTML Help](#)

The VisualLynux **Toolbar** provides integrated access to LynuxWorks tools and features. LynuxWorks tools that are unique from those already available in the Studio IDE are added to this toolbar. The add-in toolbar is a small, movable, icon-based toolbar on the desktop (see following figure).







Figure 7-1: VisualLynux AddIn Toolbar

If the VisualLynux toolbar is not available on your screen, check to see if you have a VisualLynux project open. A VisualLynux project is one created using the VisualLynux Application wizard. If a VisualLynux project is open and the VisualLynux Addin Toolbar is still not visible, it can be installed manually by using the **Tools->Customize** selection in Visual Studio. Use the **Add-ins and MacroFiles** tab to make sure the **VisualLynux Add-in** is installed. Use the **Toolbars** tab to make sure it is set to **Display**.

## Toolbar Icons

Table 7-1: VisualLynux Toolbar Icons

Icon	Function
	<p><b>Import Makefile</b> This icon launches a VisualLynux Import wizard that helps create a Makefile-based project. This is especially useful for bringing existing UNIX, LynxOS, or BlueCat Linux code and Makefiles into the Visual Studio environment.</p>
	<p><b>Export to Makefile</b> This icon launches a VisualLynux Export wizard that allows the developer to save existing Visual Studio project information as a LynxOS/BlueCat Linux Makefile. It is useful for switching back to executing builds in a <code>bash</code> shell environment.</p>
	<p><b>VisualLynux Project Settings</b> This icon launches the project settings dialog boxes for setting up preferences for LynuxWorks tools. Preferences can be set independently for each configuration or for several selected configurations. The initial window displays the source files for the project on the left. By selecting an individual file, the developer can change options for just that file. Or else, the options apply to the entire project. This is the default look of the initial dialog box.</p>
	<p><b>GDB</b> This icon invokes the LynuxWorks GDB debugger in command line mode. If a default target is defined in the VisualLynux Target Administration wizard, it may cause the GDBSERVER to be automatically launched on the target. Otherwise, VisualLynux prompts for the selection of a target on which the debugged program should run.</p>

**Table 7-1: VisualLynux Toolbar Icons (Continued)**











Icon	Function
	<p><b>LynuxWorks Total/db</b>                      This icon launches Total/db, the LynuxWorks version of Insight Debugger. This is a Tcl/Tk version of the GDB debugger.</p>
	<p><b>Upload</b>                      This icon copies the project result (an executable, library, or kernel image) to the default target set by the VisualLynux Target Administration wizard.</p>
	<p><b>Bash</b>                      This icon opens an MS-DOS window, launches the <code>bash</code> shell, and sets the environment for the current project.</p>
	<p><b>LynuxWorks FTP</b>                      This icon launches the LynuxWorks FTP (File Transfer Protocol) application. This program facilitates the transfer of files to and from the LynxOS or BlueCat Linux target. The LynuxWorks FTP program can be used to transfer the binary to the target. LynuxWorks FTP automatically connects to the default target (set by the VisualLynux Target Administration wizard).</p>
	<p><b>Telnet</b>                      This icon launches the standard Windows telnet application, providing a convenient way to log in to a LynxOS or BlueCat Linux target and run applications on the target. Telnet automatically connects to the default target.</p>
	<p><b>LynuxWorks Cross Process Viewer</b>                      This icon launches the LynuxWorks Cross Process Viewer. This application runs on a Windows cross development host and communicates with a LynxOS or BlueCat Linux target machine to display all processes and threads running on the target. The LynuxWorks Cross Process Viewer automatically connects to the default target.</p>

Table 7-1: VisualLynux Toolbar Icons (Continued)

Icon	Function
	<p><b>VisualLynux Target Administration Wizard</b>  This icon launches the VisualLynux Target Administration wizard, which facilitates and manages Windows connections to LynxOS and BlueCat Linux targets. The wizard can be used to establish a connection, set up target names and login names, pick default directories for FTP, and also test connections. Only TCP/IP connections are supported. The information set up by this wizard is used by the other VisualLynux tools described above.</p>
	<p><b>VisualLynux Help</b>  This icon launches the VisualLynux online HTML help facility, which provides an indexed, searchable HTML Help version of the <i>VisualLynux User's Guide</i>, <i>LynxOS man pages</i>, <i>BlueCat Linux online documentation</i>, and <i>GNU Tools Documentation</i>.</p>
	<p><b>About VisualLynux</b>  This icon displays the <b>About VisualLynux</b> dialog box.</p>
	<p><b>Bootp-Tftp-Pftp Utility</b>  This icon launches the program that provides servers for booting targets remotely through an Ethernet network or a parallel port connection. BTP uses the TCP/IP protocol for communication with remote targets and Parallel Port Protocol for targets connected to the host via parallel port.</p>

## VisualLynux Import Wizard

### Import a Makefile

This launches the VisualLynux Import wizard, which helps in the creation of Makefile-based VisualLynux projects. If the current workspace is not a VisualLynux workspace, this button is disabled. The Import wizard provides a series of dialog boxes to gather information about existing Makefiles, source files to be included in the project, project configurations, and Makefile targets.

## Import Wizard Initial Screen

The Import wizard starts with an initial screen. Press **Next** to go to the next step.

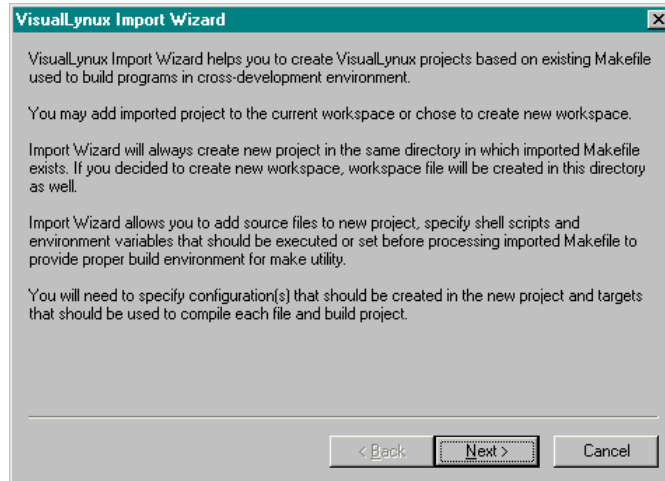


Figure 7-2: Import Wizard Initial Screen

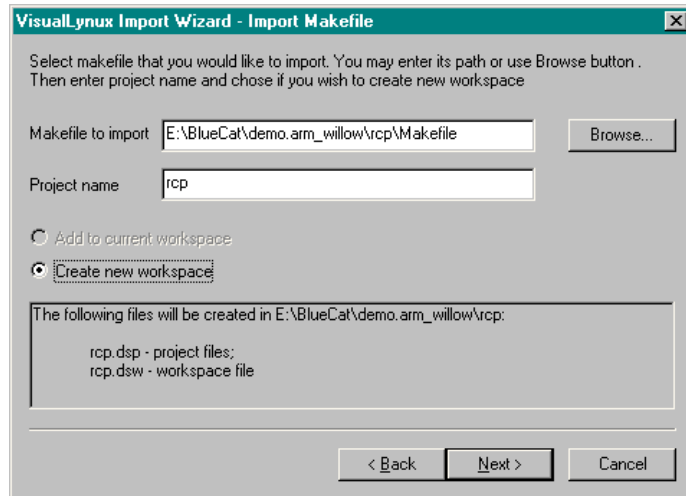
## Selecting a Makefile to Import

In the second step, the wizard prompts for the selection of an existing Makefile. Enter the path in the **Makefile to import** edit box or click **Browse** to select a Makefile.

The edit box at the bottom of the dialog box displays the files created as a result of the import. The Import wizard always creates a project file (with a `.dsp` extension) in the directory containing the Makefile. If you create a new workspace, a workspace file (with a `.dsw` extension) is created in the new workspace directory.

Add the new project to the current workspace or create a new workspace by clicking the appropriate radio button. If there is no open workspace, or the current workspace is not a VisualLynux workspace, the **Add to current workspace** radio button is disabled.

Enter a new project name in the **Project name** edit box. By default, the Import wizard sets the project name to the name of the directory containing the imported Makefile.



**Figure 7-3: Import: Makefile and Project Names**

After entering required data, press **Next** to proceed.

## Specifying Project Configurations

To create a Makefile-based VisualLinux project you must specify at least one project configuration. In the next step, the Import wizard displays a dialog box with a choice of platforms in the **Platforms** drop-down list. This list displays the cross development tools installed on your computer. Select a target platform from this list.

Enter the name of the configuration in the **Configuration name** edit control and click **Add**. The Import wizard constructs a full configuration name that reflects the target platform and adds it to the **Create configurations** list. To create more than one configuration, repeat target platform selection and enter a configuration name.

To remove a configuration from the list, make the **Create configurations** list active, select the configuration to be removed, and click **Remove**.



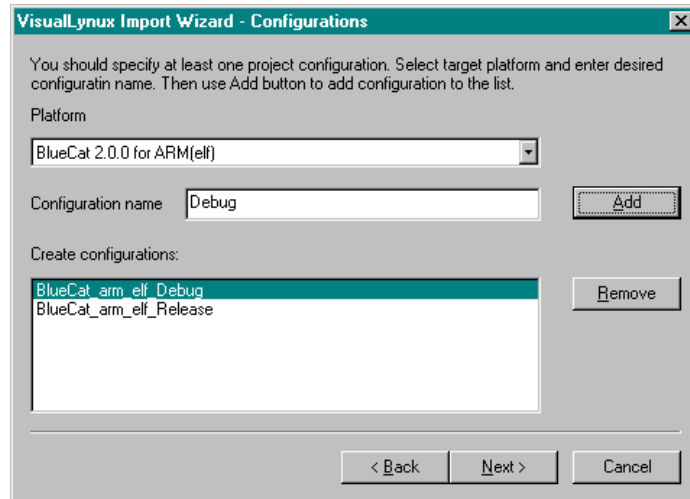


Figure 7-4: Makefile Import: Specifying Project Configuration

## Specifying Build Environment

In the next step, the Import wizard asks you to specify a build environment, which should be established before processing the imported Makefile. You must specify a build environment for every configuration. Use the **Configuration** drop-down list to select the configuration to which data belongs.

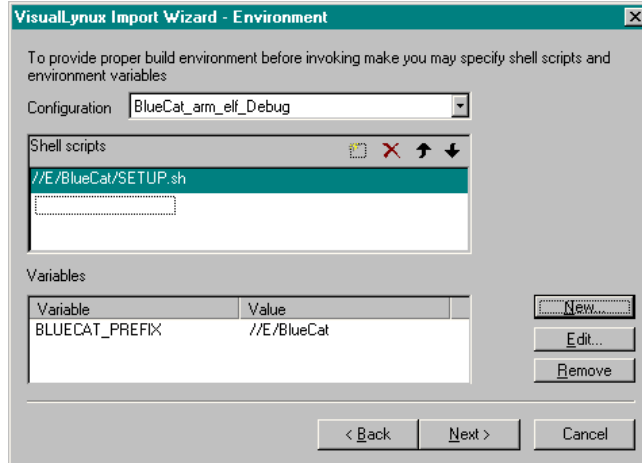
A build environment can include:

- Shell scripts to be executed before processing the Makefile
- A set of environment variables to be exported before processing the Makefile

VisualLynux executes shell scripts specified (using `bash`) and exports environment variables before invoking the `make` utility to process the imported Makefile.

Usually, the scripts and environment variables required for every target configuration are set during VisualLynux installation, and the wizard displays them in the proper controls. You can add or remove additional scripts/variables, and modify values of variables.

You can use the small toolbar in the **Shell scripts** control to add new script, remove a selected script, and arrange scripts in the list box.

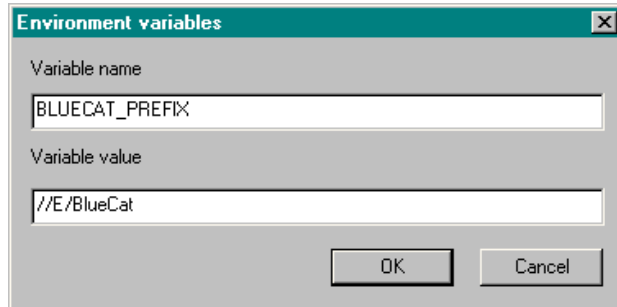


**Figure 7-5: Makefile Import: Build Configurations.**

**NOTE:** The script path must be entered in UNIX format using the CYGWIN convention for mounts and drives. The Import wizard checks the path and displays an error message if a script file does not exist.

## Adding/Editing Environment Variables

To add a new variable, click the **New...** button. This invokes a dialog box to enter the new variable's name and value. To edit the name or value of the variable, select it in the **Variables** list and click the **Edit...** button. A dialog box that allows you to make the changes appears. To remove an existing variable, select it in the **Variables** list and click **Remove**.



**Figure 7-6: Editing Environment Variables**

When you have finished with the build environment settings for all configurations, click **Next** to proceed to the next step.

## Adding Source Files to a Project

The next step of the Import wizard allows you to specify source files to be included in the new project. (You can add files later using the Visual Studio **Insert files in project** function.) Remember that while adding files to the project, there are no additional build steps or tools to process these files. The Makefile itself is responsible for performing a build step for a particular source file. Nevertheless, adding files to the project may be useful because it provides additional editing and project maintenance capabilities.

You can add only existing files to the project. To do so, click **Add...** and select file(s) in the dialog box. To remove file(s) from the project, select them in the list box and click **Remove**.

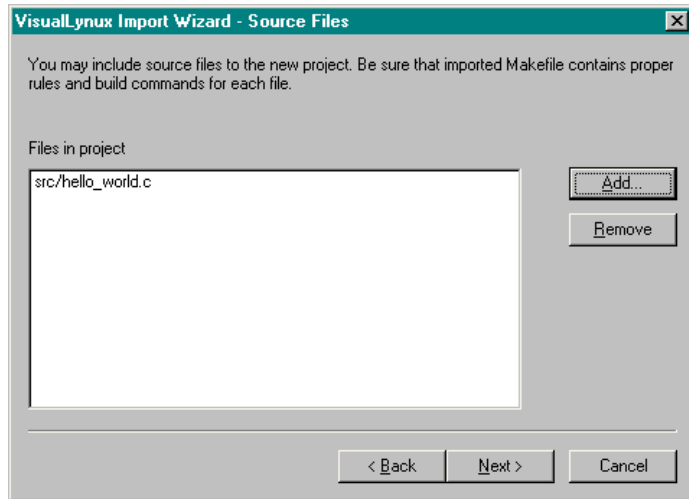


Figure 7-7: Makefile Import: Adding Source Files

## Specifying Build Target

The last step of the Import wizard allows you to specify build targets (project goal) used by the imported Makefile. The **Project target** edit control contains, by default, the name of the project without an extension. Enter the name of the valid Makefile target. For example, if a Makefile uses the pseudo-target `all` to build a project, you should set the default target name to `all`.

If source files have been added to the project, the **File targets** list contains the files and default targets assigned by the wizard. To change the target name, select the appropriate line in the list and click the **Edit...** button. The edit control in the **Target** column appears, allowing you to change the target name.

When finished, click **Enter**.

If a file does not have a target, begin editing and clear the target name. After you hit **Enter**, the text `none` appears in place of the target name. The same text is assigned by default for all non-compilable files (such as header files).

Click **Finish** to complete the wizard and create a new imported project.

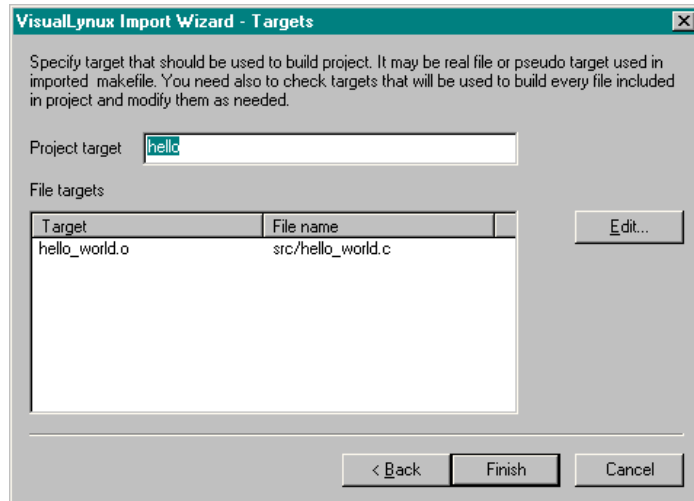


Figure 7-8: Build Target for Imported Project

## VisualLynux Export Wizard

### Export to Makefile

This toolbar button launches the VisualLynux Export wizard, which helps convert VisualLynux projects into the Makefiles to be used in cross- or native-development environments. If the current workspace is not a VisualLynux workspace, the appropriate toolbar button is disabled.

The Export wizard allows creation of Makefiles for every configuration defined for projects in the current workspace. You can export an entire workspace or select projects and configurations to be exported. If you need to, you can create a new source tree. In such a case, the Export wizard copies all source files to specified directories.

### Export Wizard Initial Screen

The Export wizard starts with an initial screen. Click **Next** to continue exporting.

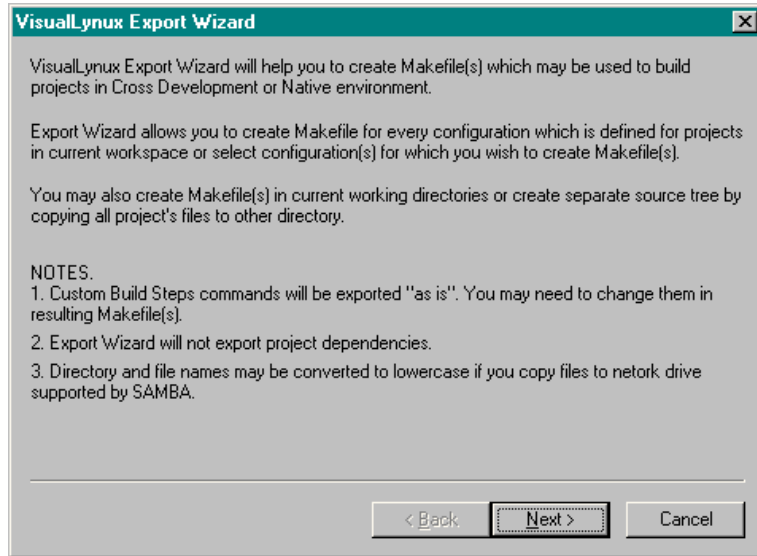
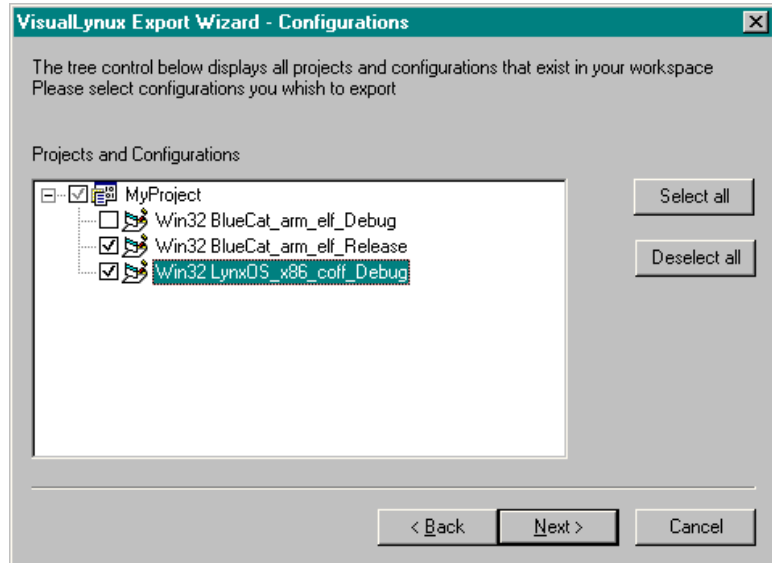


Figure 7-9: Makefile Export Wizard Initial Screen

## Setting Configurations to Export

In the second step the Export wizard displays a dialog box with a control tree listing the projects and configurations in the current workspace. Select configurations for which you wish to create Makefiles. Click on the check mark next to a configuration, or next to a project icon to select all project configurations. You can also use the **Select all** and **Deselect all** buttons.



**Figure 7-10: Makefile Export Configurations**

When done with selection, press **Next** to proceed. The wizard displays the **Export Directories** page.

## Export Directories Setup

The Export Directories page allows you to specify directories where the new source tree is to be placed. To generate a new source tree, check the **Create new source tree** box. Appropriate controls appear in the dialog box.

If you do not wish to create a new source tree, simply click **Next** for the next screen. In this case, all Makefiles are created in the `Output` directories as specified in VisualLynux **Settings** dialog box.

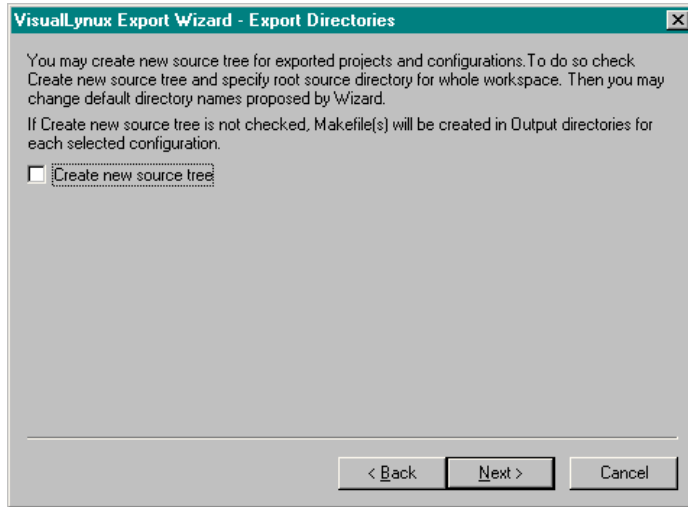


Figure 7-11: Export Directories Setup

## Specifying Directories for New Source Tree

When the **Create new source tree** box is checked, additional controls appear in the dialog box. Enter a root directory name for the new source tree in the **Root workspace source directory** box.

You can also use **Browse...** to select an existing directory. In this case the Export wizard appends the name of the current workspace directory to the name of the selected directory and displays it in the **Root workspace source directory** edit field. You can modify this name as needed.

The **Subdirectories** tree displays the subdirectory structure that is to be created for each exported project and configuration. To rename the subdirectory, select the appropriate item and click **Rename**. An edit control appears with a directory name that you can change as appropriate.

---

**NOTE:** The Export wizard uses project and configuration names as default directory names.

---



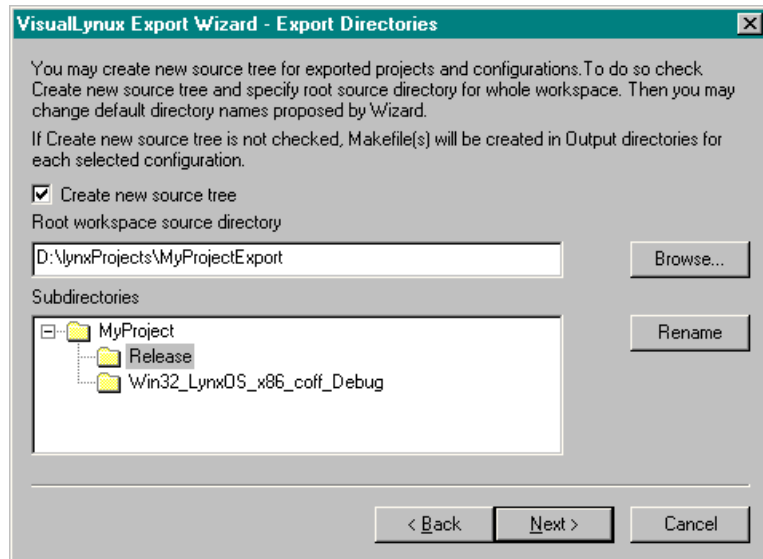


Figure 7-12: Export Directories for New Source Tree

To continue click **Next**. The Export wizard displays the **Makefiles** page.

## Specifying Names of Exported Makefiles

The **Makefiles** page allows you to specify names for exported Makefiles. It contains tree controls that display projects and configurations to be exported. When you select a configuration, a Makefile name appears in the **Makefile** edit control. The Makefile directory is displayed at the bottom of the dialog box. Edit the Makefile name as needed. To rename a directory, click **Back** to edit the name in the **Export Directories** dialog box.

---

**NOTE:** Directory names cannot be changed if you have not created a new source tree.

---

If you have not selected **Generate new source tree**, the **Finish** button appears instead of **Next**. Otherwise, the **Project Files** step dialog box appears.

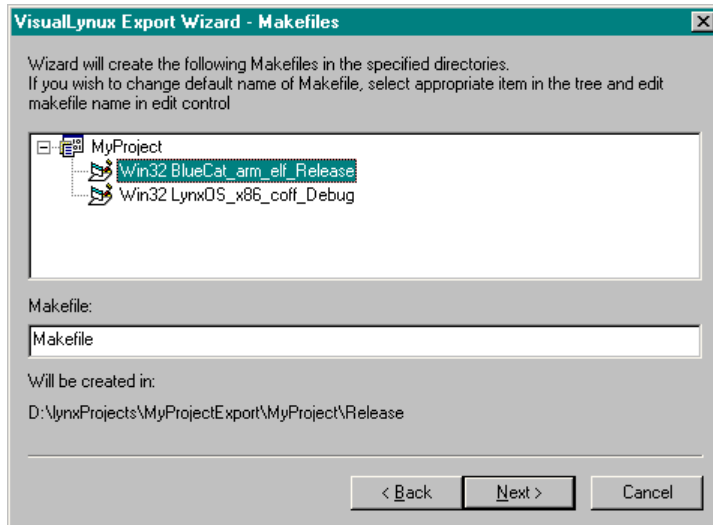


Figure 7-13: Exported Makefile Names

## Project Files to Copy

The **Project Files** dialog box displays projects to be exported and files to be copied in a tree view. You can view the list of files copied, directories in which project files are placed, and source file paths. This page is for your own information. Press **Next** to copy files.

Files to be copied are organized into two folders. The **Project Files** folder contains files included directly in the project. The **External Dependencies** folder includes files not included in the project but on which some source files may depend.

The Export wizard uses dependency files (.d files) to obtain dependency information, so you should rebuild all exported configurations before exporting the project with the compiler's **Generate Dependency** switch set for all source files.

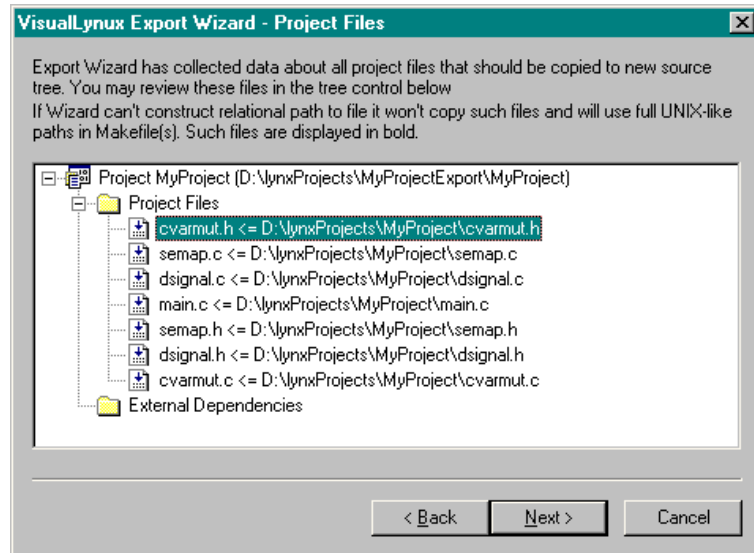


Figure 7-14: Makefile Export Summary

## Summary Dialog Box

Finally, the Export wizard displays the results of file copying. If a file has been copied successfully, a check mark appears against its name. If any error occurs, an appropriate message pops up and the file is marked as not copied. This dialog box is for your information.

To complete the export, click **Finish**. The wizard then creates Makefiles for the selected configurations.

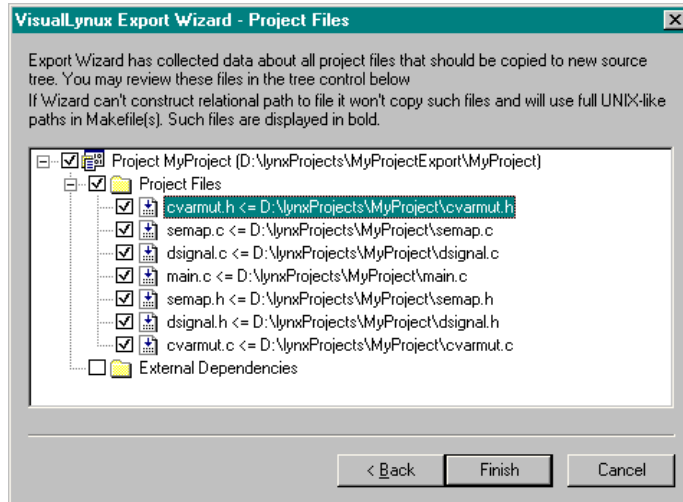


Figure 7-15: Makefile Export Summary Dialog Box

## LynuxWorks GDB



This toolbar button launches the standard LynuxWorks GDB debugger.

The LynuxWorks GDB debugger creates a session with the target machine, copies the executable to the target machine project directory (defined using the Target wizard), and launches GDBSERVER on the target machine.

The GDBSERVER text mode window appears. It displays GDBSERVER information as well as program output:

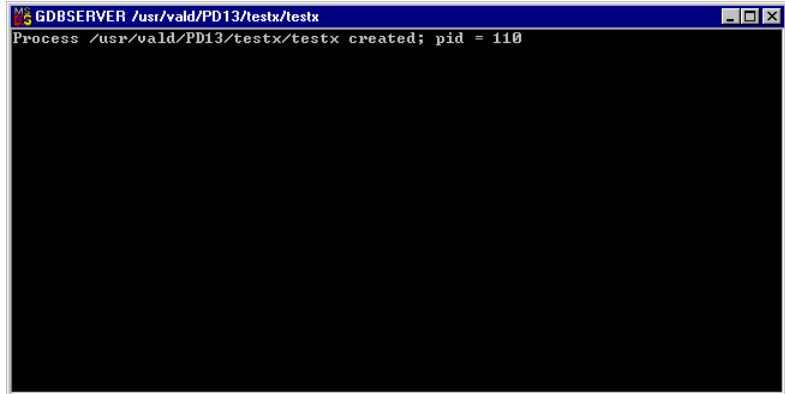



Figure 7-16: GDBSERVER Text-Mode Window

Use the LynuxWorks GDB text-mode window to enter GDB commands:



Figure 7-17: Entering GDB Commands

Note that GDBSERVER has already started the main process of your program and has stopped it at the first instruction. Therefore, after defining break points, enter the **GDB continue** command (not **run**). If you need to start a program with

arguments, use the [VisualLynx Project Settings](#)  dialog box and set program arguments in the **Debug** page. GDBSERVER starts the program with these arguments.

After you have executed the debugged program, GDBSERVER restarts the process. Enter the `run` command in the GDB window to restart debugging. To exit, enter the `quit` command.

If you have found a bug, edit the corresponding file in the VisualC++ editor window, rebuild the project, and start the debugger again.

## Total/db



This toolbar button launches the LynuxWorks Total/db debugger.

VisualLinux includes two LynuxWorks versions of the GNU GDB debugger: **GDB** and **Total/db**. **Total/db** has a graphical interface that significantly facilitates the debugging process.

The LynuxWorks Total/db debugger creates a session with the target machine, copies the executable to the target machine project directory (defined using the Target wizard), and launches GDBSERVER on the target machine. The GDBSERVER text-mode window displays GDBSERVER information as well as program output as shown in the figure [GDBSERVER Text-Mode Window](#).

Next, the LynuxWorks Total/db window appears:

```

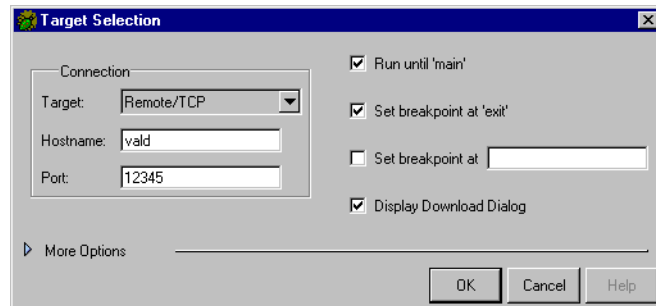
testx.c - Source Window
File Run View Control Preferences Help
0x3657 30%
290 va_start(ap);
291 format = va_arg(ap, char *);
292 vsprintf(mytrans, format, ap);
293 va_end(ap);
294
295 translations = XtParseTranslationTable(mytrans);
296 XtOverrideTranslations(w, translations);
297 XtFree((void*)translations);
298 }
299
300 ///////////////////////////////////////////////////////////////////
301 // main - main program
302 int main(int argc, char** argv) {
303
304 Dimension width = DefaultWidth, height = DefaultHeight;
305 Dimension shell_height = 0, scroll_height;
306 Window topWindow;
307 Dimension tb_height = 0;
308 Window tbWindow;
309 Dimension st_height = 0, st_sep_height;
310 Window stWindow;
311
312 // Set locale environment
313 XtSetLanguageProc (NULL, (XtLanguageProc)NULL, NULL);
314
315 // Initialize at once to use Xt facilities to process options and resources
316 topShell = XtVaOpenApplication(&app_context,
317 appl_class,
318 options, XtNumber(options)).
Program stopped at 0x003657
testx.c | main | SOURCE

```

Figure 7-18: LynuxWorks Total/db Window

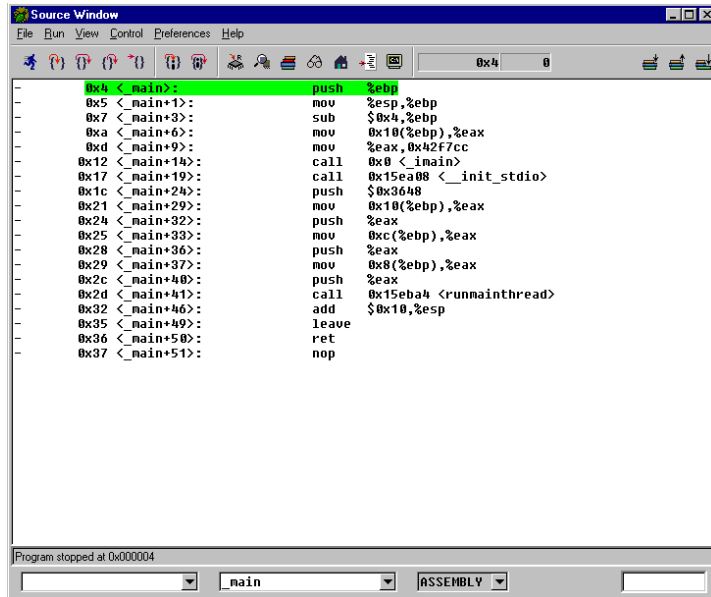
Total/db has its own help system. Click on the **Help** menu item to get additional information on how Total/db works.

To start debugging, click on **File->Target Settings**. The **Target Selection** dialog box appears (see the following figure).



**Figure 7-19: Total/db Target Selection Dialog Box**

Note that the dialog box already contains correct settings for the target set by the Target wizard. Click **OK** and the LynxOS source window displays the starting assembler commands of your program. For BlueCat Linux, Total/db displays the starting source code of your program.

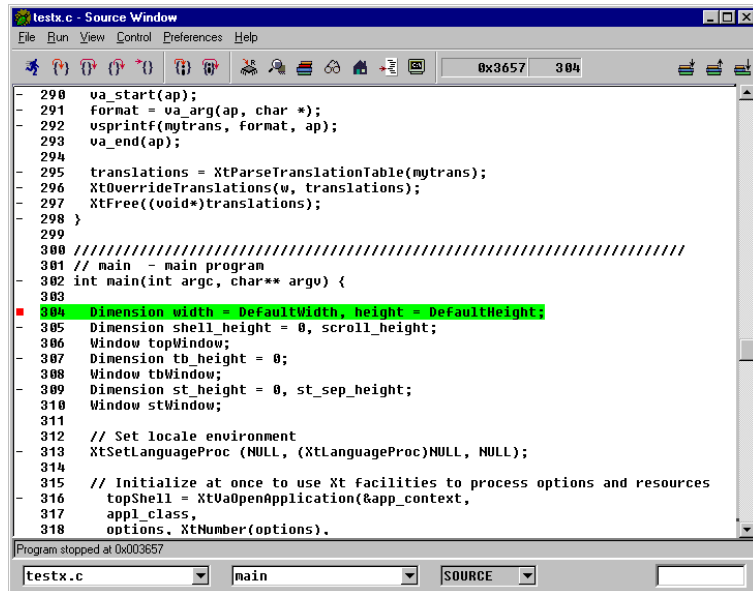


```
Source Window
File Run View Control Preferences Help
0x4 0
-
0x4 <_main>:      push  %ebp
0x5 <_main+1>:    mov   %esp,%ebp
0x7 <_main+3>:    sub   $0x4,%ebp
0xa <_main+6>:    mov   0x10(%ebp),%eax
0xd <_main+9>:    mov   %eax,0x42f7cc
0x12 <_main+14>:  call  0x0 <_main>
0x17 <_main+19>:  call  0x15ea08 <__init_stdio>
0x1c <_main+24>:  push  $0x3648
0x21 <_main+29>:  mov   0x10(%ebp),%eax
0x24 <_main+32>:  push  %eax
0x25 <_main+33>:  mov   0xc(%ebp),%eax
0x28 <_main+36>:  push  %eax
0x29 <_main+37>:  mov   0x8(%ebp),%eax
0x2c <_main+40>:  push  %eax
0x2d <_main+41>:  call  0x15eba4 <runmainthread>
0x32 <_main+46>:  add   $0x10,%esp
0x35 <_main+49>:  leave
0x36 <_main+50>:  ret
0x37 <_main+51>:  nop
-
Program stopped at 0x000004
main ASSEMBLY
```

Figure 7-20: Starting Assembler Commands

Click on **Continue**. The execution stops at the first line of your main function, and the source window shown in the next figure is displayed.





```
testx.c - Source Window
File Run View Control Preferences Help
0x3657 304
- 290 va_start(ap);
- 291 format = va_arg(ap, char *);
- 292 vsprintf(mytrans, format, ap);
- 293 va_end(ap);
- 294
- 295 translations = XtParseTranslationTable(mytrans);
- 296 XtOverrideTranslations(w, translations);
- 297 XtFree((void*)translations);
- 298 }
- 299
- 300 ////////////////////////////////////////////////////
- 301 // main - main program
- 302 int main(int argc, char** argv) {
- 303
- 304 Dimension width = DefaultWidth, height = DefaultHeight;
- 305 Dimension shell_height = 0, scroll_height;
- 306 Window topWindow;
- 307 Dimension tb_height = 0;
- 308 Window tbWindow;
- 309 Dimension st_height = 0, st_sep_height;
- 310 Window stWindow;
- 311
- 312 // Set locale environment
- 313 XtSetLanguageProc (NULL, (XtLanguageProc)NULL, NULL);
- 314
- 315 // Initialize at once to use Xt facilities to process options and resources
- 316 topShell = XtVaOpenApplication(&app_context,
- 317 appl_class,
- 318 options, XtNumber(options),
Program stopped at 0x003657
testx.c main SOURCE
```

Figure 7-21: Total/db Source Window

Use the Total/db GUI to debug your program. (The online VisualLynux Help System contains a guide for Total/db.) To rerun your program for LynxOS, click on the **Console** button to pop up the Total/db console window and enter **run**. For BlueCat Linux, simply click **Continue**.

**NOTE:** For LynxOS 3.1.0a CDKs and BlueCat Linux 3.0 CDTs, you need to edit the `targetselection.if1` file to be able to rerun your program. See the *VisualLynux Release Notes* for instructions on how to do this.



```
Console Window
(gdb) run
Starting program: //H/testx/Debug/testx
0x4 in _main ()

Breakpoint 1, main (argc=1, argv=0x2fffffff8) at testx.c:304


(gdb) |
```

Figure 7-22: Total/db Console Window

If you have found a bug, edit the corresponding file in the VisualC++ editor window, rebuild the project, and start the debugger again.

---


## Upload

This button  launches the VisualLynux FTP program to copy the executable to the target machine. The project result must exist, that is, it must be built without errors. The target machine must be defined using [VisualLynux Target Administration](#).

If you debug a program, you do not need this function because the LynuxWorks GDB and Total/db debuggers always refresh the executable on the target machine. This function is useful if you have created the Release version of your executable and want to launch it using a Telnet window. In this case, ensure that you are in Release mode. If not, use the **Build->Set Active Configuration** menu item to switch to Release mode.

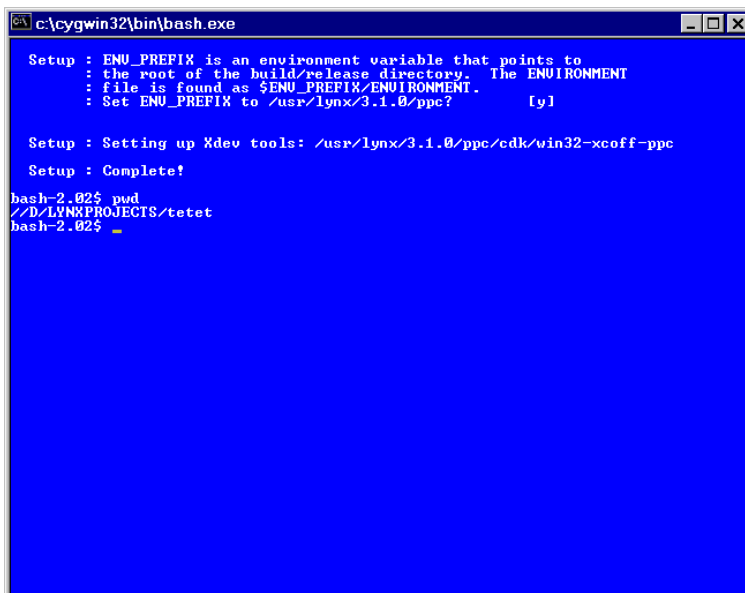
---

## bash Shell

This button  opens the MS-DOS console window and starts the `bash` shell. The button is active only if a VisualLynux project is loaded. While starting the `bash` shell, VisualLynux invokes the appropriate `bash` script to establish the build environment for the command line tools in accordance with the current build

configuration. The current working directory is set to the project directory of the active VisualLinux project.

To change the appearance (font, color, size, etc.) of the window, press the **Alt-Space** keys and invoke the **Properties** command.



```
c:\cygwin32\bin\bash.exe

Setup : ENU_PREFIX is an environment variable that points to
       : the root of the build/release directory. The ENVIROMENT
       : file is found as $ENU_PREFIX/ENVIROMENT.
       : Set ENU_PREFIX to /usr/lynx/3.1.0/ppc?      [y]


Setup : Setting up Xdev tools: /usr/lynx/3.1.0/ppc/cdk/win32-xcoff-ppc
Setup : Complete!

bash-2.02$ pwd
/D/LYNXPROJECTS/tetet
bash-2.02$
```


Figure 7-23: bash Shell Screen

## Telnet



The VisualLinux Toolbar contains a button  to launch the Windows version of Telnet. Telnet is launched with no parameters. Ideally, the Telnet session is started by default with the name of the target connection.

## VisualLynux HTML Help

This button  invokes an HTML-based set of documents for VisualLynux.

Documents include:

- *VisualLynux User's Guide*
- *LynxOS man pages*
- *LynuxWorks GNU Tools*

---

## Other VisualLynux Tools

Project settings are discussed in Chapter 8, “VisualLynux Project Settings” FTP functions in greater detail, in Chapter 9, “LynuxWorks FTP” Target Administration in Chapter 10, “VisualLynux Target Administration” the Cross Process Viewer in Chapter 11, “LynuxWorks Cross Process Viewer” and the BTP utility in Chapter 12, “Bootp-Tftp-Pftp Utility”.

---

## Overview

The VisualLynux **Project Settings** dialog box allows you to specify options for LynuxWorks tools (compiler, linker, library, for example) used to build a project target. You can open the VisualLynux **Settings** dialog box using the toolbar button



or by clicking **Settings** on the **Project** menu.

---

**NOTE:** If the current workspace is not a VisualLynux workspace, this command invokes the Visual Studio settings dialog box. The VisualLynux **Project Settings** dialog box is very similar to that of Visual Studio, so you can use Visual Studio Help to learn how to work with the dialog box.

---

Using the VisualLynux **Project Settings** dialog box, you can specify settings for the tool used to process a particular source file, any one or several configurations, or the project as a whole.

You can also edit options common to several items (files or projects). To determine what items the edited options belong to, you should check in the drop-down list in the left side of the **Project Settings** dialog box.

To select configurations for which you wish to apply options, use the drop-down list at the top left of the **Settings** dialog box. You can select a particular configuration, all configurations, or a specific set of the configurations from the tree view.

---

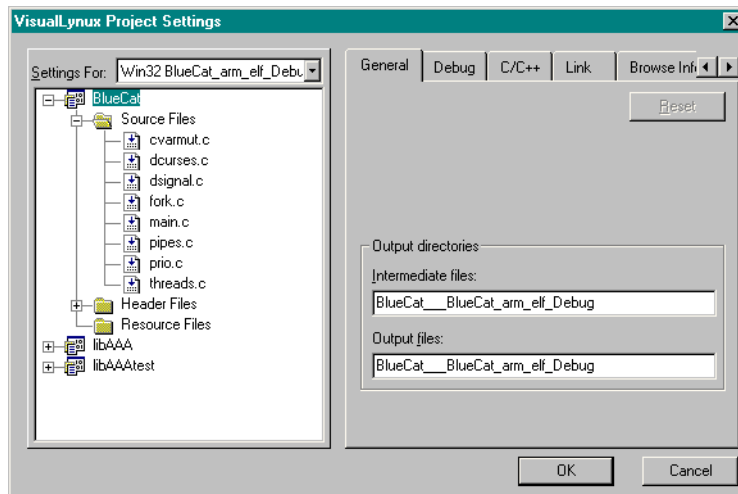
**NOTE:** When you have selected the configuration to be edited, the tree view may change. This is because the tree view contains items for projects that have at least one of the configurations selected to edit.

---

Depending on what is selected, the right part of the **Settings** dialog box changes, representing the set of properties that belong to all selected items.

If several items or configurations are selected, tabs in the right part of the dialog box display common properties. Properties that are different for selected items have empty or indeterminate values. When several items are selected, all changes made are applied to all items.

If there are no common options for selected items, the right part of the dialog box contains no tabs and displays an appropriate message.



**Figure 8-1: VisualLynux Project Settings**

The **Reset** button discards option changes made to project or file settings, and reverts to the settings of the project or file when it was created. This button is enabled only if a single project or file is selected, and the settings of the selection have been changed.

## Project-Level Property Pages

When one or several project items are selected from the tree, the right part of the dialog box displays the following pages:

- [General](#): Allows editing of general project properties.
- [Debug](#): Allows editing of options used while debugging the target executable.
- [C/C++](#): Allows editing of compiler options used as defaults for compiling all source files in the project.
- [Link](#): Allows editing of linker options.
- [Library](#): Allows for editing options of the `ar.exe` utility.
- [Browse Info](#): Allows you to specify options used to build the Source Browser Database.
- [Custom Build](#): Allows you to specify a Custom Build step for the project as a whole.
- [Pre-Link Step](#): Allows you to specify a set of arbitrary commands, executed just before linking the target.
- [Post-Build Step](#): Allows you to specify a set of arbitrary commands, executed after the project build.
- [Make](#): Allows you to edit options used to build a project based on an imported Makefile.
- [Kernel](#): Allows editing of LynxOS kernel properties, including selection of drivers to be included in the kernel, and selection of kernel parameters.

Depending on the type of application, the pages displayed at the project level vary. If the result of a project is a LynxOS/BlueCat Linux application or driver, the [Link](#) page appears. If a project builds a static library, the [Library](#) page is displayed instead. For a project based on an imported Makefile, only the [Make](#) page is displayed.

For LynxOS projects, both the [Kernel](#) page and the [Link](#) page are displayed.

## File-Level Property Pages

When one or more source files is selected from the tree, a combination of the following pages appears in the right part of the **Settings** dialog box:

- **General**: Allows you to specify general properties of the file build.
- **C/C++**: Allows editing of compiler settings for a particular file or group of selected files.
- **Custom Build**: Allows you to specify special commands used to build a file.
- **Make**: Allows you to edit properties of the file build step for Makefile-based projects.

The **C/C++** page at the file level appears only for files that should be compiled. VisualLinux assigns a compiler to a file with one of the following extensions: `.c`, `.cpp`, `.cc`, `.cxx`, `.i`, or `.ii`. If the compiler is assigned to a file, you can use either the **C/C++** or **Custom Build** page, but not both. If file is not compilable, the **Custom Build** page appears by default.

If the project is based on an imported Makefile, only the **Make** page appears when a single file is selected from the tree.

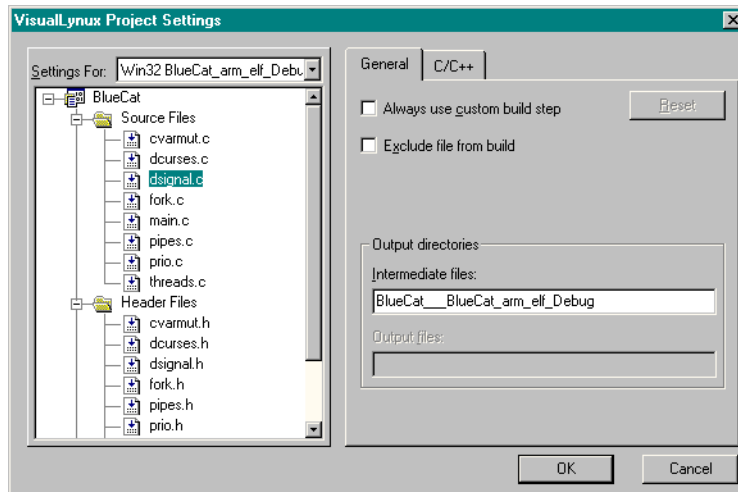
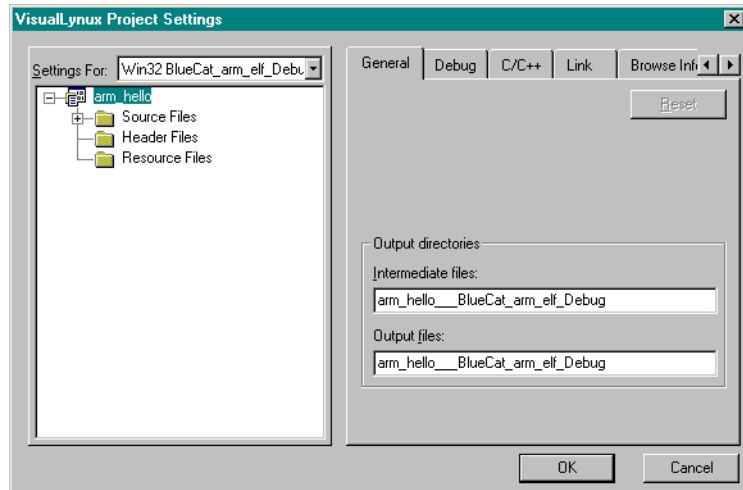


Figure 8-2: File-Level Property Page



## General Page

The **General** page for a *project* allows you to change directories used during a build to place output and intermediate files.



**Figure 8-3: General Page for a Project**

The following table lists **General** page commands for a project:

**Table 8-1: General Page Commands**

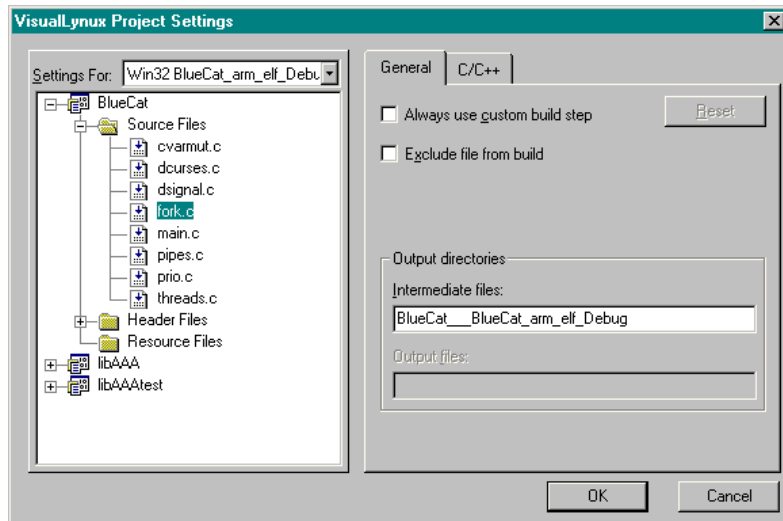
Command	Description
<b>Intermediate files</b>	Specifies the name of the folder in which to place results of processing project files. Usually, these are object files and dependency files, but can be any other files that result from building a particular file. This field specifies a default directory for new files, and the directory used if the intermediate directory has not been changed for a particular file.
<b>Output files</b>	Specifies the name of a folder for a file resulting from a project build. Usually an executable or a library.

When a single file or a group of files is selected in the left part of the dialog box, the **General** property page in the figure below appears. The **Output files** field is

disabled because it specifies project-level options. You can set up additional properties for a *file*; the following table lists file commands:

**Table 8-2: Single File Command Options**

Command	Description
<b>Always use custom build step</b>	Click on this control to disable default build step (compilation) and use a Custom Build step for a file. If clicked, the <b>Custom Build</b> property page appears instead of the <b>C/C++</b> property page.
<b>Exclude file from build</b>	Click to exclude file from build. VisualLynux does not process this file at all. This is useful when a file should not participate in a build or when different configurations require different sets of source files.
<b>Intermediate files</b>	Specifies the folder in which results of processing a selected file are placed. The value in this field overrides settings made at the project level in the same field.



**Figure 8-4: General Page for a File**

## Debug Page

The **Debug** page allows you to specify options for debugging the *project executable*. These options apply to both GDB and Total/db, as appropriate.

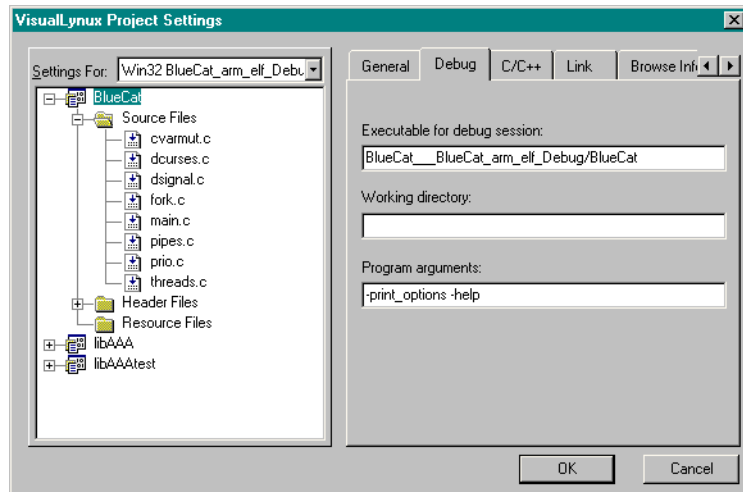


Figure 8-5: Debug Page

The following table lists **Debug** page commands for a project:

Table 8-3: Debug Page Commands

Command	Description
<b>Executable for debug session</b>	This is the name of the executable used to debug the application. By default it is set by VisualLynx to the resulting executable of the <u>project</u> . If you change the value in the field, be sure that the executable exists and is valid for a target platform when the debugger is invoked.
<b>Working directory</b>	This field is not supported in the current implementation.
<b>Program arguments</b>	By default, this field is empty. If your program uses command line arguments, enter them in this field.

## Browser Page

The **Browser** page is used by VisualLynux to enter options used to build the source browser database. This database readily accesses program definitions and references, facilitates source code editing, outlines source file contents, or browses class definitions (for C++ programs). To use the browser database, be sure that the **Generate browse info** box is checked for each source file in the “C/C++ Page” on page 145.

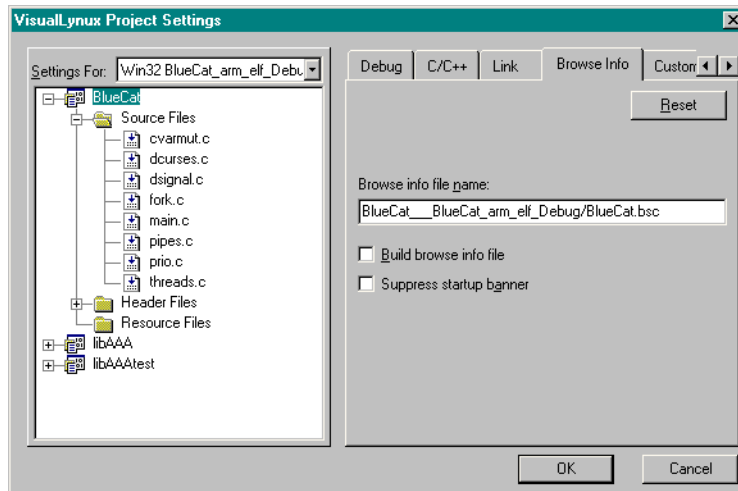


Figure 8-6: Browser Page

The following table displays the **Browser** page commands:

Table 8-4: Browser Page Commands

Command	Description
<b>Browser info file name</b>	Specify the name of the browser database file if it differs from the default. By default, VisualLynux generates a browser database name by appending a <code>.bsc</code> extension to the project name and places it in the <code>Output</code> directory.
<b>Build browse info file</b>	Check this box to browse the database to be built.
<b>Suppress startup banner</b>	Check this box if you do not wish the <code>bscmake.exe</code> utility to display startup information during a build.

---

**NOTE:** To build the `Browse Info` file (browser database), VisualLynx uses the Microsoft compiler (`cl.exe`) and browser builder (`bscmake.exe`). Because of differences between Windows header files and LynxOS/BlueCat Linux header files, VisualLynx source files may not always be compiled by `cl.exe` without error. In such cases, you may get an incomplete browser database that may not allow you to browse system structures and/or files.

---

---

## Library Page

The **Library** page allows you to edit options used to build static libraries with the `ar.exe` utility. This page appears only for Static Library projects.

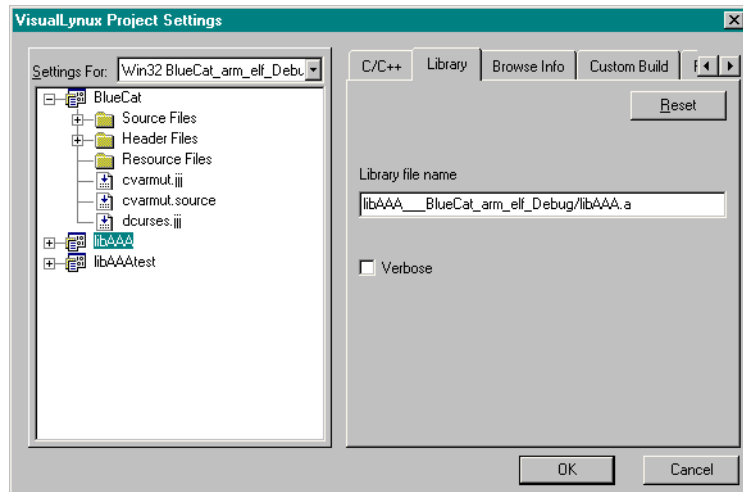


Figure 8-7: Library Page

The following table shows the **Library** page commands:

**Table 8-5: Library Page Commands**

Command	Description
<b>Library file name</b>	Enter or edit the library file name. By default, VisualLynux creates the library name by adding an <code>.a</code> extension to the project name and places it in the <code>Output</code> directory.
<b>Verbose</b>	Check this box to obtain <code>ar.exe</code> messages in the output window during library build.

---

## Pre-Link Step Page

You can specify a set of commands that is always executed *before linking a project target*. The term “linking” is used here in general and refers to a build step that is performed by the main build tool (for example, linker, library builder) for a project.

The **Pre-link** step page contains a **Pre-link description** edit box and a **Pre-link commands** list. Fill the first box with appropriate text to be displayed in the output window during the build. The **Pre-link commands** list contains the batch commands to be executed.

- To add a new command, double click on the empty line with a focus rectangle and enter a command.
- To edit an existing command, double click the appropriate line in the grid and edit the text.
- To delete a command, select it in the list and press **Del** on your keyboard.

You can also use internal toolbar buttons to add a command, or delete or arrange commands in the grid.

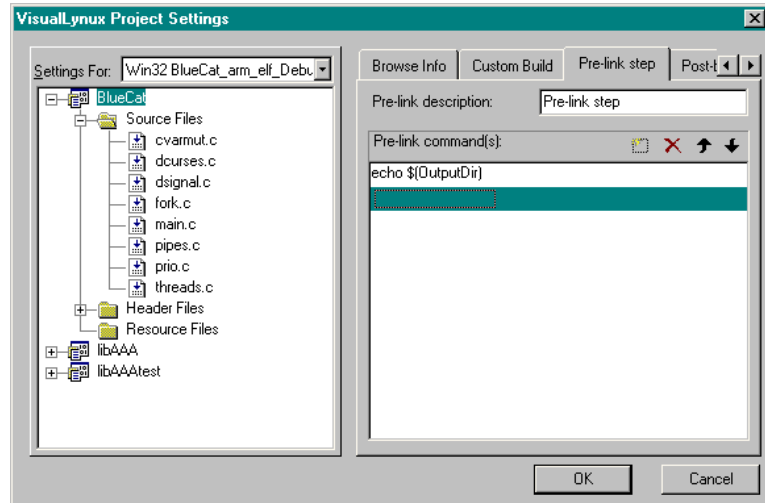


Figure 8-8: Pre-Link Step Page

While entering commands or descriptions, you can use **Build macros** as part of the command/description text. At build time these macros are replaced by appropriate values.

---

## Post-Build Step Page

This page allows you to specify a set of commands to be executed after every successful completion of the normal build process, including successful execution of the commands specified in the [Custom Build](#) step.

The **Post-build** page contains a **Post-build description** control and a **Post-build commands** list. Fill the first box with appropriate text to be displayed in the output window during step execution. The **Post-build commands** list contains the batch commands to be executed.

- To add a new command, double click on the empty line with a focus rectangle and enter a command.
- To edit an existing command, double click the appropriate line in the list and edit the text.
- To delete a command, select it in the list and press **Del** on your keyboard.

You can use internal toolbar buttons to add new commands, delete commands, or arrange commands in the list.

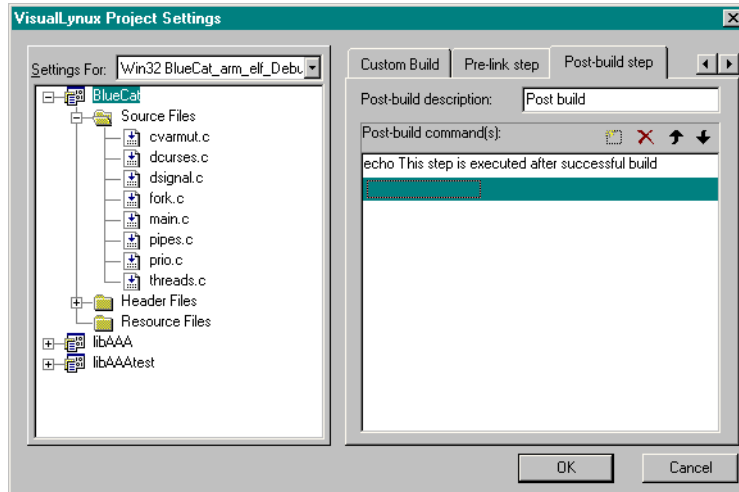


Figure 8-9: Post-Build Step Page

While entering commands or descriptions, you can use **Build macros** as part of the command/description text. At build time these macros are replaced by appropriate values.

---

## Custom Build Steps

### Project Custom Build

The *Project Custom Build* step is executed just after the normal build process. Custom build commands run only if the project target file is rebuilt, or if any of the output files specified for Custom Build step do not exist.



The project **Custom Build** page contains the fields in the table below:

**Table 8-6: Project Custom Build Page Commands**

Command	Description
<b>Input file</b>	Displays name of project target file.
<b>Description</b>	You can enter a description of the Custom Build step. By default, VisualLynux uses the text “Performing custom build step on <code>\$(InputPath)</code> .” All <b>Build macros</b> used in this field are replaced by their values before running commands.
<b>Commands</b>	Enter commands to be executed in this box. If you need more than one command, press <b>Enter</b> to start a new line and enter the next command. Commands are executed in the sequence in which they appear in the edit box. You can use <b>Build Macros</b> in the commands (see also the <b>Directory</b> and <b>Files</b> menus).
<b>Outputs</b>	Enter names of the files generated as a result of the Custom Build step execution. If more than one file is generated, enter the name of every file on a separate line. You can use <b>Build Macros</b> in the commands (see also <b>Directory</b> and <b>Files</b> menus).
<b>Directory</b>	Press this button to invoke the <b>Directory</b> menu, which allows you to insert the <code>Directory</code> macro at the current insertion point.
<b>Files</b>	Press this button to invoke the <b>File</b> menu, which allows you to insert the <code>File</code> macro at the current insertion point.

---

**NOTE:** Do not specify pseudo-targets in the **Outputs** control. The Custom Build step must create a real file. You cannot debug the application or send the executable to the target because the resulting file does not exist.

---

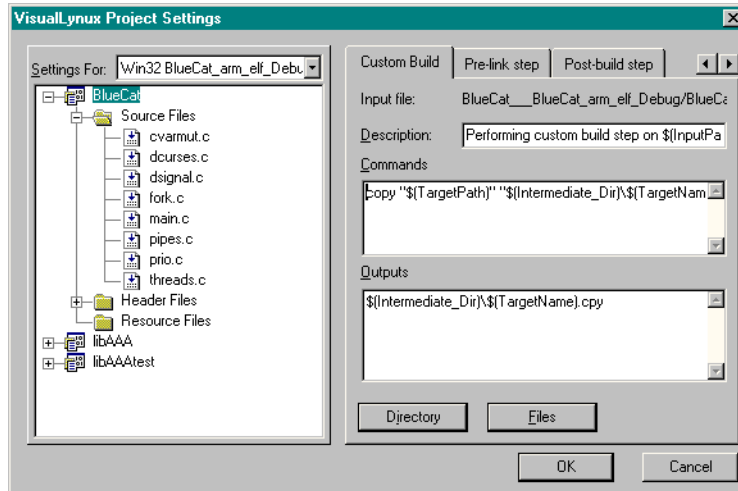


Figure 8-10: Project Custom Build Page

## Directory Menu

The **Directory** menu allows you to insert the `Directory` macro at the current insertion point (command text, Custom Build step description, or list of output files). You can insert the macros from the following table:

Table 8-7: Directory Menu (Project Custom Build)

Menu Item	Macro	Description
Intermediate	<code>\${Intermediate_Dir}</code>	The value of the <b>Intermediate directory</b> control in the project <b>General</b> page
Output	<code>\${Output_Dir}</code>	The value of the <b>Output directory</b> control in the project <b>General</b> page
Target	<code>\${TargetDir}</code>	Directory in which the project target file is placed - It may differ from <b>Output directory</b> if you have changed <b>Output file name</b> in the <b>Linker</b> or <b>Library</b> pages.
Input	<code>\${InputDir}</code>	Always has an empty value for Project Custom Build step.

Table 8-7: Directory Menu (Project Custom Build) (Continued)

Menu Item	Macro	Description
<b>Project</b>	<code>\$(ProjDir)</code>	Fully qualified path to the directory in which the project file (.dsp file) resides
<b>Workspace</b>	<code>\$(WkspDir)</code>	Fully qualified path to the directory in which the workspace file (.dsw file) resides - The value of this macro is changed if the project is inserted into a different workspace.
<b>Microsoft Developer</b>	<code>\$(MSDevDir)</code>	Fully qualified path to the directory in which the msdev.exe executable resides
<b>Cross- Development Tools</b>	<code>\$(InstallDir)</code>	Fully qualified path to the root installation directory of cross development tools
<b>Build Environment</b>	<code>\$(BUILD_ENV_PREFIX)</code>	Fully qualified path to the directory considered as the root directory of cross-development tools for a particular configuration
<b>CDK Directory</b>	<code>\$(TargetPlatformCDKDir)</code>	Fully qualified path to the directory containing build tools for a particular configuration

## File Menu

The File menu allows insertion of the `File` macro at the current insertion point (command text, Custom Build step description, or list of output files). You can insert the macros from the table below:

Table 8-8: File Menu (Project Custom Build)

Menu Item	Macro	Description
<b>Target path</b>	<code>\$(TargetPath)</code>	Path to the target file relative to the project directory
<b>Target name</b>	<code>\$(TargetName)</code>	Name of the target file without directory and extension (if any)
<b>Input path</b>	<code>\$(InputPath)</code>	Always the same as <b>Target path</b> for a project

**Table 8-8: File Menu (Project Custom Build) (Continued)**

Menu Item	Macro	Description
<b>Input name</b>	<code>\$( InputName )</code>	Always empty for a project
<b>Workspace name</b>	<code>\$( WkspName )</code>	Name of workspace file ( . dsw file) without extension

## File Custom Build

The Custom Build step for a file is executed before the normal build process starts, that is, all custom builds defined for particular files are performed before compiling any other file. VisualLynux executes the Custom Build step for a particular file if the source file has been changed or any of the output files need to be rebuilt.

If a compiler is assigned to a file, you must check the **Always use custom build** box in the **General** page to specify a Custom Build step. Instead of a compile, the Custom Build step is executed for a source file.

The **Custom Build** page for a file contains the fields in the table below:

**Table 8-9: File Custom Build Page**

Command	Description
<b>Input file</b>	Displays source file path relative to the project directory.
<b>Description</b>	You can enter a description of the Custom Build step in this field. By default, VisualLynux uses the text “Performing custom build step on <code>\$( InputPath )</code> .” All <b>Build Macros</b> used in this field are substituted by their values before commands are run.
<b>Commands</b>	Enter commands to be executed in this edit control. If you need more than one command, press <b>Enter</b> to start a new line. Commands are executed in the sequence they appear in the edit box. You can use <b>Build Macros</b> in the commands (See also the <b>Directory</b> and <b>Files</b> menus).

Table 8-9: File Custom Build Page (Continued)

Command	Description
<b>Outputs</b>	Enter names of the files generated as a result of the Custom Build execution. If more than one file is generated, enter the name of every file on a separate line. You can use <b>Build Macros</b> in the commands (See also the <b>Directory</b> and <b>Files</b> menus).
<b>Directory</b>	Press this button to invoke the <b>Directory</b> menu, which allows you to insert the <code>Directory</code> macro at the current insertion point.
<b>Files</b>	Press this button to invoke the <b>File</b> menu, which allows you to insert the <code>File</code> macro at the current insertion point.

**NOTE:** If you specify a pseudo-target as the output of a file custom build, this step is executed twice. First, it is executed during file build and second, during project build. Avoid using `pseudo-targets` in Custom Build steps for a file.

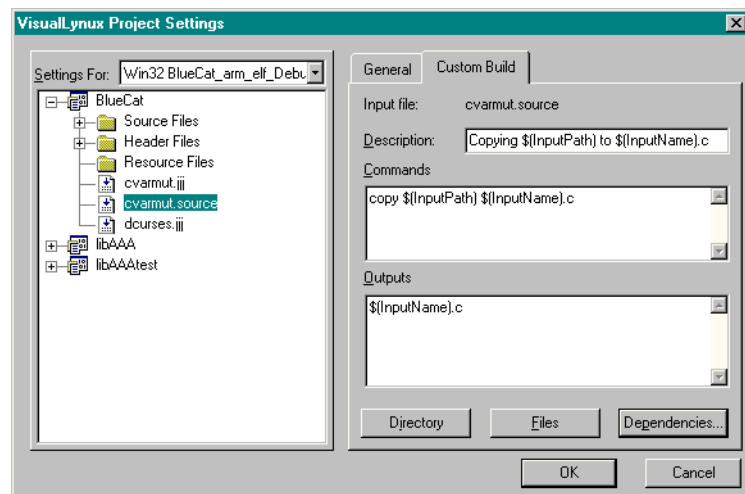


Figure 8-11: File Custom Build Page

## Directory Menu

The **Directory** menu allows insertion of the *Directory* macro at the current insertion point (command text, Custom Build step description, or list of output files). You can insert the macros listed in the table below:

**Table 8-10: Directory Menu (File Custom Build)**

Menu Item	Command	Description
<b>Intermediate</b>	<code>\$(Intermediate_Dir)</code>	The value of the <b>Intermediate directory</b> control in the <b>General</b> page for a <i>file</i>
<b>Output</b>	<code>\$(Output_Dir)</code>	The value of the <b>Output directory</b> control in the <i>project</i> <b>General</b> page
<b>Target</b>	<code>\$(TargetDir)</code>	Directory in which project target file is placed - It may differ from the <b>Output directory</b> if you have changed the <b>Output file name</b> in the <b>Linker</b> or <b>Library</b> pages.
<b>Input</b>	<code>\$(InputDir)</code>	Name of fully qualified directory where the source file resides
<b>Project</b>	<code>\$(ProjDir)</code>	Fully qualified path to the directory in which the project file ( <code>.dsp</code> file) resides
<b>Workspace</b>	<code>\$(WkspDir)</code>	Fully qualified path to the directory in which the workspace ( <code>.dsw</code> file) file resides - The value of this macro is changed if the project is inserted into a different workspace.
<b>Microsoft Developer</b>	<code>\$(MSDevDir)</code>	Fully qualified path to the directory in which the <code>msdev.exe</code> executable resides
<b>Cross- Development Tools</b>	<code>\$(InstallDir)</code>	Fully qualified path to the root installation directory of cross-development tools
<b>Build Environment</b>	<code>\$(BUILD_ENV_PREFIX)</code>	Fully qualified path to the directory considered as the root directory of cross-development tools for a particular configuration
<b>CDK Directory</b>	<code>\$(TargetPlatformCDKDir)</code>	Fully qualified path to the directory containing build tools for a particular configuration

## File Menu

The **File** menu allows insertion of the `File` macro at the current insertion point (command text, Custom Build step description, or list of output files). You can insert the macros from the table below:

**Table 8-11: File Menu (File Custom Build)**

Menu Item	Macro	Description
Target path	<code>\$(TargetPath)</code>	Path to the target file relative to the project directory
Target name	<code>\$(TargetName)</code>	Name of the target file without directory and extension (if any)
Input path	<code>\$(InputPath)</code>	Path to the source file relative to the project directory
Input name	<code>\$(InputName)</code>	Name of source file without path and extension
Workspace name	<code>\$(WkspName)</code>	Name of the workspace file ( <code>.dsw</code> file) without extension

## Compiler Options

### C/C++ Page

The **C/C++** page allows you to set options for the LynuxWorks C/C++ cross compiler. Depending on the selection from the tree, this page displays *project* settings (if a project is selected) or *file* settings (if a particular file is selected). If several files or a file folder is selected, this page displays common options for all selected items.

The following figure shows the **C/C++** page when a project is selected. The edit box in the bottom right corner of the dialog box is called the **Project Options** box, and is open for editing. This edit box displays compiler options in a command line style. When any option is changed in the dialog box, the corresponding command line switch is added (or removed) from this edit box. You can use this control to enter compiler command line options that are not presented in dialog boxes.

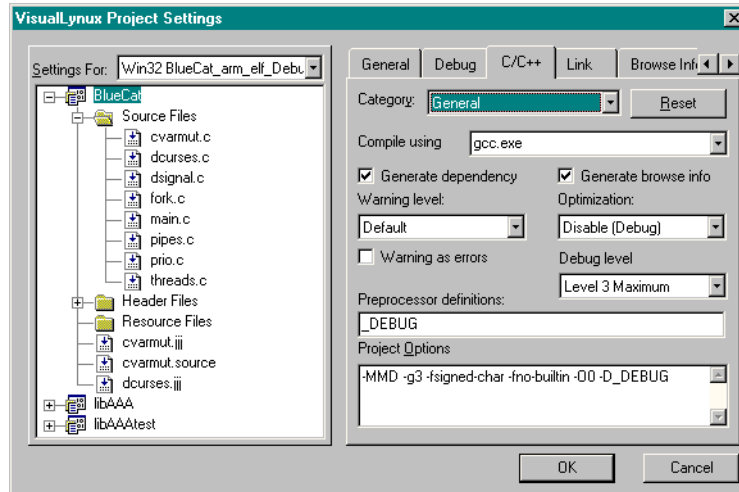


Figure 8-12: C/C++ Page - Project

Compiler options for a project are used as defaults in the following cases:

- When a new source file is added to the project and inherits all the compile options set at the project level
- When **Reset** is pressed while editing compiler options for a particular file

The C/C++ page for a particular file is shown in the next figure. Note that the edit box in the bottom right of the dialog box is titled **Source File Options** and is closed for editing. This edit box displays differences between compile options for a project and for a file. If there are no differences, it displays the text `Project Settings`. All differences are shown in command line terms.

For example, if the **Generate dependency** option is checked for a project but unchecked for a file, the control displays the text `Project settings`, and not `-MMD`. If the **Warning as errors** option is checked for a file but not for a project, the control displays `Project Settings`, and `-Werror`.

If the **Debug level** for a project is **Level 2 Default**, but is **Level 1 Minimal** for a file, the control contains the text `Project Settings`, and `-g1` replaces `-g`.

Because the settings edit control is closed for editing when a single file is selected, use the **Special options** page to enter compile options not represented by the **Settings** dialog box.



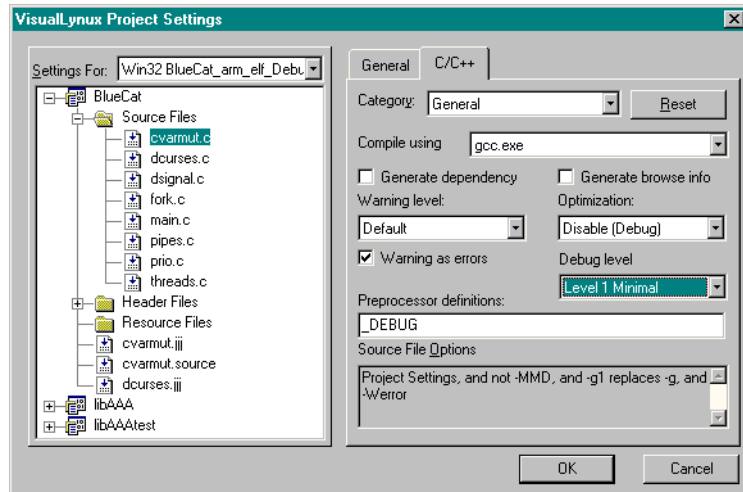


Figure 8-13: C/C++ Page - File

The C/C++ page organizes compile options into several categories. To edit options belonging to a category, use the **Category** drop-down list.

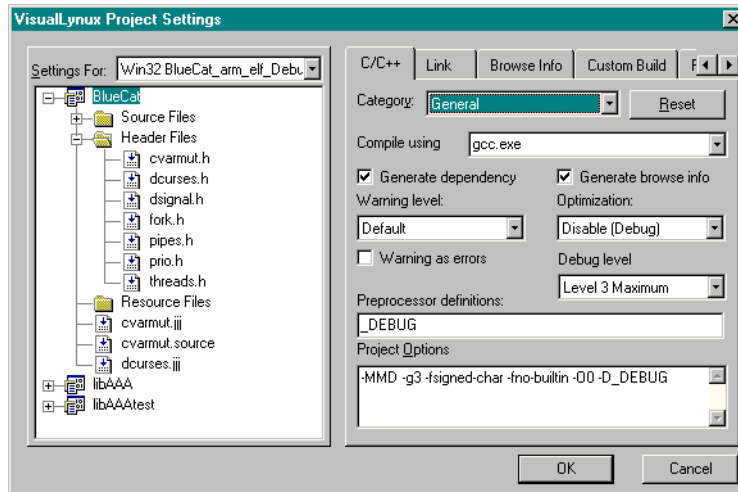
There are nine categories of compiler options as shown in the table below:

Table 8-12: Compiler Option Categories

Category	Description
<a href="#">C/C++ General</a>	General compiler options
<a href="#">C/C++ Language</a>	Language-specific compiler options
<a href="#">C/C++ Warnings</a>	Custom warnings options
<a href="#">C/C++ Code Generation</a>	Options for code generation conventions
<a href="#">C/C++ Profiling</a>	Options controlling profiling
<a href="#">C/C++ Listings</a>	Options controlling listings generated during compilation
<a href="#">C/C++ Optimization</a>	Custom optimization options
<a href="#">C/C++ Preprocessor</a>	Options controlling the preprocessor
<a href="#">C/C++ Special Options</a>	Special options that cannot be edited using the C/C++ page

## C/C++ General

The **General** compiler settings are shown in the figure below:



**Figure 8-14: C/C++ General Compiler Settings**

This page provides the functionalities shown in the following table:

**Table 8-13: C/C++ General Settings**

Control Item	Description
<b>Compile using</b>	Allows for selection of the compiler used to process source files ( <code>gcc.exe</code> or <code>g++.exe</code> ). By default, VisualLinux assigns compilers according to file extension.
<b>Generate dependency</b>	Check to generate the dependency file ( <code>.d</code> file) during compilation ( <code>-MMD</code> ). The dependency file is used to check dependencies before build.
<b>Generate browse info</b>	Check to generate browse information for a file (or by default).

Table 8-13: C/C++ General Settings (Continued)

Control Item	Description
<b>Warning Level</b>	Allows the selection of the warning level during compilation. The following levels can be selected: <ul style="list-style-type: none"> <li>• <b>Default</b> (default compile warning level)</li> <li>• <b>Inhibit all warnings</b> (<code>-W</code>)</li> <li>• <b>Syntax only</b> (<code>-fsyntax-only</code>)</li> <li>• <b>Customize</b> (use <a href="#">C/C++ Warnings</a> page to set specific warnings)</li> </ul>
<b>Optimization</b>	Allows selection of the optimization level. There are five optimization levels: <ul style="list-style-type: none"> <li>• <b>Disable</b> (<code>-O0</code>)</li> <li>• <b>Minimal Size</b> (<code>-O1</code>)</li> <li>• <b>Maximum Speed</b> (<code>-O2</code>)</li> <li>• <b>Extended Optimization</b> (<code>-O3</code>)</li> <li>• <b>Customize</b> (use <a href="#">C/C++ Optimization</a> page to set specific optimization flags)</li> </ul>
<b>Warnings as errors</b>	When checked, forces the compiler to treat all warnings as errors ( <code>-Werror</code> ).
<b>Debug level</b>	Controls debug information produced by compiler. There are four debug levels: <ul style="list-style-type: none"> <li>• <b>Disable</b> (generate no debug information)</li> <li>• <b>Level 1 Minimal</b> (<code>-g1</code>)</li> <li>• <b>Level 2 Default</b> (<code>-g</code>)</li> <li>• <b>Level 3 Maximum</b> (<code>-g3</code>)</li> </ul>
<b>Preprocessor Definitions</b>	You can enter definitions that should be passed to the compiler in this field using the <code>-D</code> directive. To separate definitions, use a comma ( , ).

## C/C++ Code Generation

The Code Generation page settings are shown in the figure below:

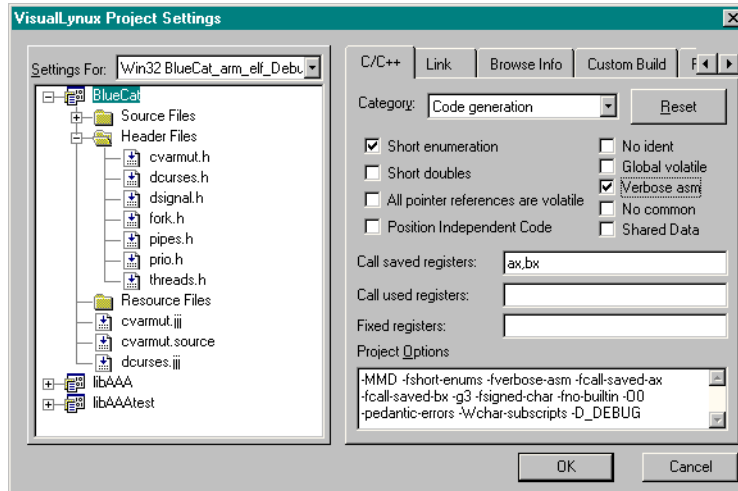


Figure 8-15: C/C++ Code Generation Page

The user to change the code generation options in the table below.:

Table 8-14: C/C++ Code Generation Commands

Control	Command Line Option	Description
Short enumeration	<code>fshort-enums</code>	Allocates to an enum type only as many bytes as it needs for the declared range of possible values.
Short doubles	<code>-fshort-doubles</code>	Use the same size for double as for float.
All pointer references are volatile	<code>-fvolatile</code>	Considers all memory references through pointers to be volatile.
Position Independent Code	<code>-fpic</code>	Generates Position-Independent Code (PIC) suitable for use in a shared library, if supported on the target machine.
No ident	<code>-fno-ident</code>	Ignores the <code>#ident</code> directive.
Global volatile	<code>-fvolatile-global</code>	Considers all memory references to <code>extern</code> and <code>global</code> data items to be volatile.

**Table 8-14: C/C++ Code Generation Commands (Continued)**

<b>Control</b>	<b>Command Line Option</b>	<b>Description</b>
<b>Verbose asm</b>	<code>-fverbose-asm</code>	Puts extra commentary information in the generated assembly code to make it more readable.
<b>No common</b>	<code>-fno-common</code>	Allocates even uninitialized global variables in the <code>bss</code> section of the object file, rather than generating them as common blocks.
<b>Shared Data</b>	<code>-fshared-data</code>	Requests that the data on non-constant variables of this compilation be shared rather than private.
<b>Call saved registers</b>	<code>-fcall-saved-</code>	Treats the registers listed in the field as allocatable registers saved by functions. Differentiate multiple registers in this field with a comma (,).
<b>Call used registers</b>	<code>-fcall-used-</code>	Treats the registers listed in the field as allocatable registers, clobbered by function calls. Differentiate multiple registers in the field with a comma (,).
<b>Fixed registers</b>	<code>-ffixed-</code>	Treats the registers listed in the field as fixed registers. Differentiate multiple registers in this field with a comma (,).

## C/C++ Language

The C/C++ Language category settings are shown in the figure below:

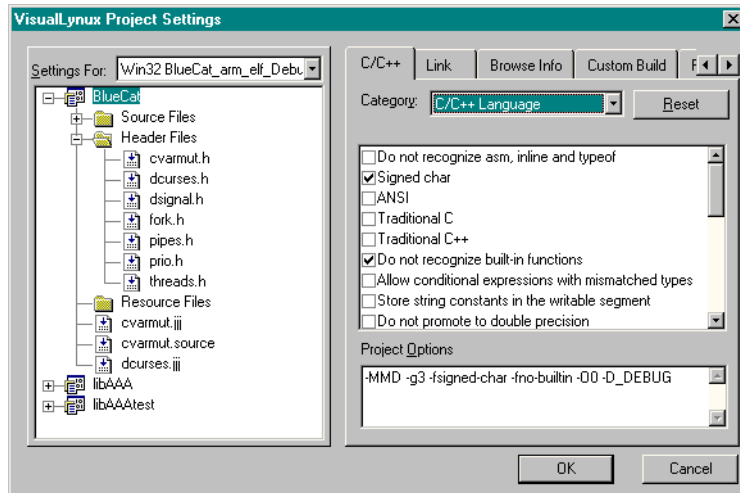


Figure 8-16: C/C++ Language Page

You can enable or disable options using check marks in the list box.

Correspondence between compiler command line options, and options listed in the list box is shown in the table below:

Table 8-15: C/C++ Language

List Item	Command Line Option
Do not recognize asm, inline and typeof	<code>-fno-asm</code>
Signed char	<code>-fsigned-char</code>
ANSI	<code>-ansi</code>
Traditional C	<code>-traditional</code>
Traditional C++	<code>-traditional-cpp</code>
Do not recognize built-in functions	<code>-fno-builtin</code>
Allow conditional expressions with mismatched types	<code>-fcond-mismatch</code>

Table 8-15: C/C++ Language (Continued)

List Item	Command Line Option
Store string constants in the writable segment	-fwritable-strings
Do not promote to double precision	-fallow-single-precision
Turn off all access control	-fno-access-control
Treat all possible functions as virtual	-fall-virtual
Conserve space	-fconserve-space
External templates	-fexternal-templates
External templates based on place	-falt-external-templates
Enable automatic templates instantiation	-frepo
Operator names	-foperator-names
Check pointer returned by new	-fcheck-new
Do not recognize GNU keywords	-fno-gnu-keywords
No implicit templates	-fno-implicit-templates
Limit scope of vars to the for loop	-ffor-scope
Strict function prototype	-fstrict-prototype
Enable \$ sign in identifiers	-fdollars-in-identifiers

## C/C++ Listings

The C/C++ Listings category settings are shown in the figure below:

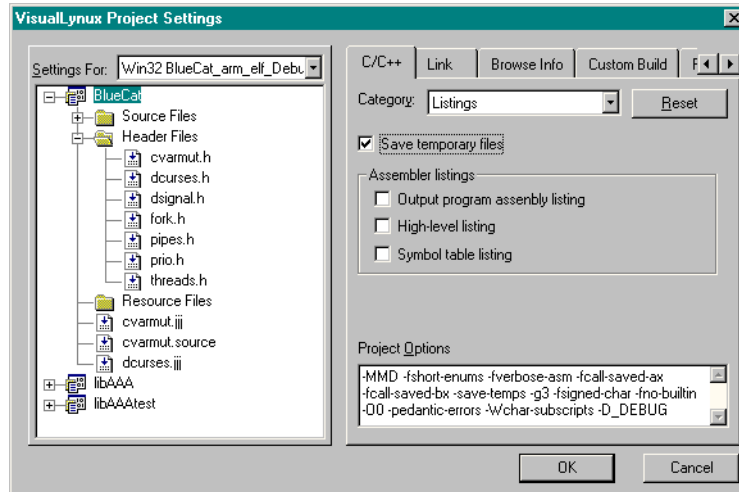


Figure 8-17: C/C++ Listings Page

All assembler listings are displayed in the output window.

Table 8-16: C/C++ Listings

Control Item	Command Line Option	Description
Save temporary files	<code>-save-temps</code>	Stores the usual temporary intermediate files permanently, that is, it places them in the current directory, and names them based on the source file.
Output program assembly listing	<code>-Wa, -al</code>	Requests an output program assembly listing.
High level listing	<code>-Wa, -ah</code>	Requests a high-level language listing.
Symbol table listing	<code>-Wa, -as</code>	Requests a symbol table listing.



## C/C++ Warnings

The C/C++ Warnings category settings are shown in the figure below:

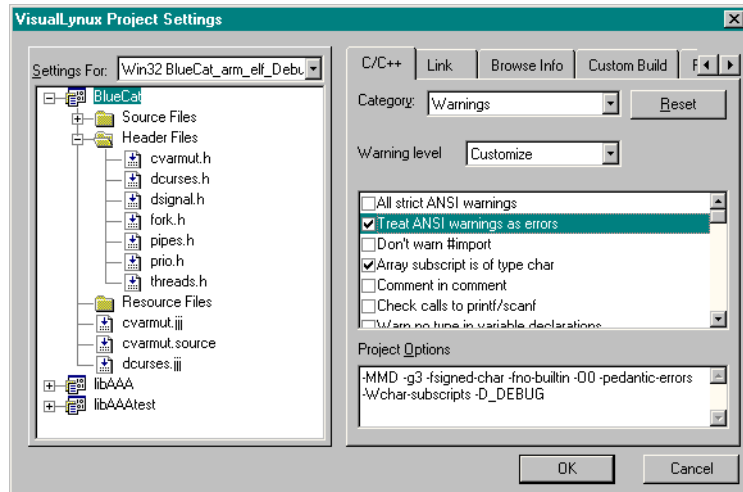


Figure 8-18: C/C++ Warnings Page

The **Warning level** drop-down list allows you to select one of the following levels:

- **Default** (compiler issues warnings in accordance with its configuration)
- **Inhibit all warnings** (`-W`)
- **Syntax only** (`-fsyntax-only`)
- **Customize** (custom options)

The list box below this control is enabled only if the **Warning level** is set to **Customize**. You can turn the option on by checking the appropriate item. Correspondence between items and command line options is presented in the table below:

Table 8-17: C/C++ Warnings

List Item	Command Line Option
All strict ANSI warnings	<code>-pedantic</code>
Treat ANSI warnings as errors	<code>-pedantic-errors</code>
Don't warn #import	<code>-Wno-import</code>

Table 8-17: C/C++ Warnings (Continued)

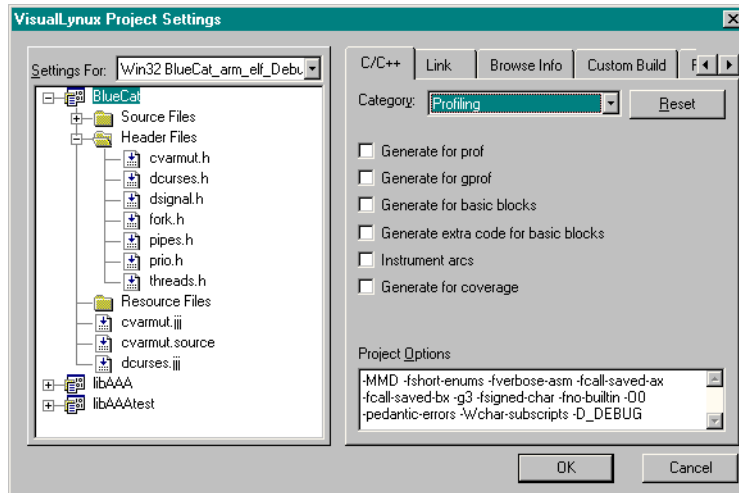
List Item	Command Line Option
Array subscript is of type char	-Wchar-subscripts
Comment in comment	-Wcomment
Check calls to printf/scanf	-Wformat
Warn no type in variable declarations	-Wimplicit-int
Warn no type in function declaration	-Wimplicit-function-declaration
Treat no type in function declaration as error	-Werror-implicit-function-declaration
Warn implicit declarations	-Wimplicit
Check declaration of main	-Wmain
Warn if multichar constants are used	-Wmultichar
Check if parentheses are omitted	-Wparentheses
Check function return type	-Wreturn-type
Check enumerated types in switch	-Wswitch
Warn trigraphs	-Wtrigraphs
Warn unused variables	-Wunused
Warn uninitialized automatic variables	-Wuninitialized
Check order of member initialization	-Wreorder
Warn unknown pragmas	-Wunknown-pragmas
Turn on all warnings shown above	-Wall
Print extra warnings	-W
Warn traditional constructs	-Wtraditional
Warn undefined identifiers evaluation	-Wundef
Warn shadowing of local variables	-Wshadow
Warn pointer arithmetic	-Wpointer-arith
Warn casting to non-matching type	-Wbad-function-cast
Warn removing type qualifier in casting	-Wcast-qual
Warn pointer alignment in casting	-Wcast-align

Table 8-17: C/C++ Warnings (Continued)

List Item	Command Line Option
Warn copying constant strings	-Wwrite-strings
Warn type conversion in function calls	-Wconversion
Warn comparison of signed/unsigned values	-Wsign-compare
Warn returning aggregate values	-Waggregate-return
Warn if argument type isn't defined	-Wstrict-prototypes
Warn if function prototype missing	-Wmissing-prototypes
Warn if function declaration missing	-Wmissing-declarations
Warn multiple declarations	-Wredundant-decls
Warn extern within function	-Wnested-externs
Warn if function can't be inlined	-Winline
Warn possible errors in overloaded virtuals	-Woverloaded-virtual
Warn g++ synthesis behavior	-Wsynth
Warn if long long type is used	-Wlong-long

## C/C++ Profiling

The **Profiling** category settings dialog box is shown in the figure below:



**Figure 8-19: C/C++ Profiling**

The C/C++ Profiling commands are shown in the table below:

**Table 8-18: C/C++ Profiling Commands**

Control Coverage	Command Line Option	Description
Generate for prof	<code>-p</code>	Generates extra code to write profile information suitable for the <code>prof</code> analysis program.
Generate for gprof	<code>-pg</code>	Generates extra code to write profile information suitable for the <code>gprof</code> analysis program.
Generate for basic blocks	<code>-a</code>	Generates extra code to write profile information for basic blocks, which record the number of times each basic block is executed, the basic block start address, and the name of the function containing the basic block.

Table 8-18: C/C++ Profiling Commands (Continued)

Control Coverage	Command Line Option	Description
Generate extra code for basic blocks	-ax	Generates extra code to profile basic blocks. Your executable produces output that is a superset of that produced by the previous option. This includes the source and target address of the basic blocks where a jump takes place, and the number of times a jump is executed.
Instrument arcs	-fprofile-arcs	Instruments arcs during compilation.
Generate for coverage	-ftest-coverage	Generates data files for the <code>gcov</code> code-coverage utility.

## C/C++ Optimization

This dialog box allows you to select an optimization level and set custom optimization options using the **Optimization** drop-down list. You can select an optimization level from the following:

- **Disable** (-O0)
- **Minimal Size** (-O1)
- **Maximum Speed** (-O2)
- **Extended Optimization** (-O3)
- **Customize**

When the optimization level is not set to **Customize**, the optimization level list box is disabled.

---

**NOTE:** You can also select the optimization level using the [C/C++ General](#) page.

---

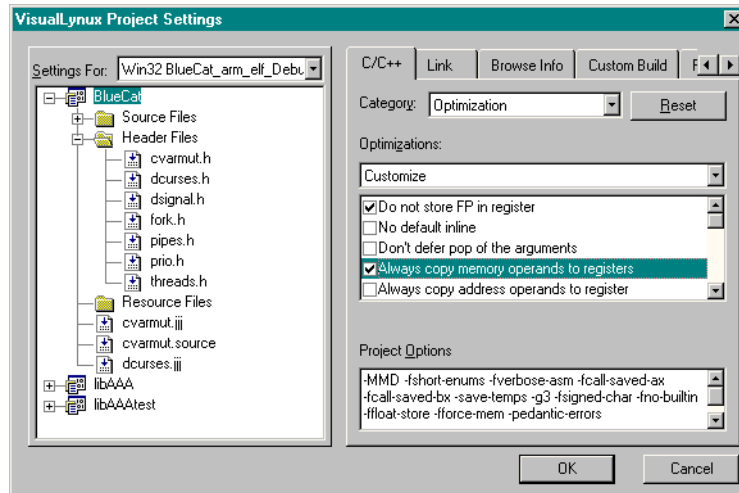


Figure 8-20: C/C++ Custom Optimization

The items in the custom options list are described in the table below:

Table 8-19: C/C++ Custom Option Commands

Customize Option	Command Line Option	Description
Do not store FP in register	<code>-ffloat-store</code>	Does not store floating point variables in registers, and inhibits other options that might change, whether a floating point value is taken from a register or from memory.
No default inline	<code>-fno-default-inline</code>	Does not make member functions inline by default, merely because they are defined inside the class scope (C++ only). Otherwise, when you specify <code>-O</code> , member functions defined inside class scope are compiled inline by default, that is, you do not need to add <b>inline</b> in front of the member function name.

Table 8-19: C/C++ Custom Option Commands (Continued)

Customize Option	Command Line Option	Description
Don't defer pop of the arguments	<code>-fno-defer-pop</code>	Always pops the arguments to each function call as soon as that function returns. For machines that must pop arguments after a function call, the compiler normally lets arguments accumulate on the stack for several function calls and pops them all at once.
Always copy memory operands to registers	<code>-fforce-mem</code>	Forces memory operands to be copied into registers before performing arithmetic operations on them. This produces better code by making all memory references potential common subexpressions. When they are not common sub-expressions, the instruction combination should eliminate the separate register-load. The <code>-O2</code> option turns on this option.
Always copy address operands to registers	<code>-fforce-addr</code>	Forces memory address constants to be copied into registers before performing arithmetic on them. This may produce better code just as <code>-fforce-mem</code> may do likewise.
Keep static consts	<code>-fkeep-static-consts</code>	Emits variables declared <code>static const</code> when optimization is not turned on, even if the variables are not referenced. This option is enabled by default. <code>-fno-keep-static-consts</code> forces the compiler to check if the variable is referenced, whether or not optimization is turned on.
Omit frame pointer	<code>-fomit-frame-pointer</code>	Does not keep the frame pointer in a register for functions that do not need one. This avoids instructions to save, set up, and restore frame pointers; it also makes an extra register available for many functions. <i>Additionally, it also makes debugging impossible on some machines.</i>

Table 8-19: C/C++ Custom Option Commands (Continued)

Customize Option	Command Line Option	Description
<b>Ignore inline keyword</b>	<code>-fno-inline</code>	Does not pay attention to the <code>inline</code> keyword. Normally, this option is used to keep the compiler from expanding any functions inline. If you are not optimizing, no functions can be expanded inline.
<b>Inline functions</b>	<code>-finline-functions</code>	Integrates all simple functions into their callers. The compiler heuristically decides which functions are simple enough to be integrated in this way. If all calls to a given function are integrated and the function is declared <code>static</code> , then the function is normally not output as assembler code in its own right.
<b>Keep inline functions</b>	<code>-fkeep-inline-functions</code>	Even if all calls to a given function are integrated and the function is declared <code>static</code> , this command outputs a separate run-time callable version of the functions.
<b>Strength reduce</b>	<code>-fstrength-reduce</code>	Performs the optimization of loop strength reduction and elimination of iteration variables.
<b>Thread jumps</b>	<code>-fthread-</code>	Performs <code>jump</code> optimizations to check if a <code>jump</code> branches to a location where another comparison subsumed by the first is found. If so, the first branch is redirected to either the destination of the second branch or to a point immediately following it, depending on whether the condition is true or false.
<b>CSE follow jumps</b>	<code>-fcse-follow-jumps</code>	In common subexpression elimination, scans through <code>jump</code> instructions when the target of the jump is not reached by any other path. For example, when CSE encounters an <code>if</code> statement with an <code>else</code> clause, CSE follows the <code>jump</code> when the condition tested is false.



Table 8-19: C/C++ Custom Option Commands (Continued)

Customize Option	Command Line Option	Description
<b>CSE skip blocks</b>	<code>-fcse-skip-blocks</code>	This is similar to <code>-fcse-follow-jumps</code> , but causes CSE to follow jumps that conditionally skip over blocks. When CSE encounters a simple <code>if</code> statement with no <code>else</code> clause, <code>-fcse-skip-blocks</code> causes CSE to follow the jump around the body of the <code>if</code> .
<b>CSE after loop</b>	<code>-frerun-cse-after-loop</code>	Reruns common subexpression elimination after loop optimization has been performed.
<b>Expensive optimization</b>	<code>-fexpensive-optimizations</code>	Performs a number of minor optimizations that are relatively expensive.
<b>No peephole</b>	<code>-fno-peephole</code>	Disables any machine-specific peephole optimization.
<b>Function sections</b>	<code>-ffunction-sections</code>	Places each function into its own section in the output file if the target supports arbitrary sections.
<b>Caller saves</b>	<code>-fcaller-saves</code>	Enables values to be allocated in registers that are clobbered by function calls, by issuing extra instructions to save and restore the registers around such calls. Such allocation is done only when it seems to result in better code than would otherwise be produced.
<b>Unroll loops</b>	<code>-funroll-loops</code>	Performs the optimization of loop unrolling. This is only done for loops whose number of iterations can be determined at compile time or run time. <code>-funroll-loop</code> implies both <code>-fstrength-reduce</code> and <code>-frerun-cse-after-loop</code> .

Table 8-19: C/C++ Custom Option Commands (Continued)

Customize Option	Command Line Option	Description
Unroll all loops	<code>-funroll-all-loops</code>	Performs the optimization of loop unrolling. This is done for all loops and usually makes programs run more slowly. <code>-funroll-all-loops</code> implies <code>-fstrength-reduce</code> as well as <code>-frerun-cse-after-loop</code> .
Branch probabilities	<code>-fbranch-probabilities</code>	After running a program compiled with <code>-fprofile-arcs</code> (see <a href="#">C/C++ Profiling</a> ), you can compile it a second time using the option <code>-fbranch-probabilities</code> to improve optimizations based on guessing the path a branch might take.

## C/C++ Preprocessor

The Preprocessor options are shown in the figure below:

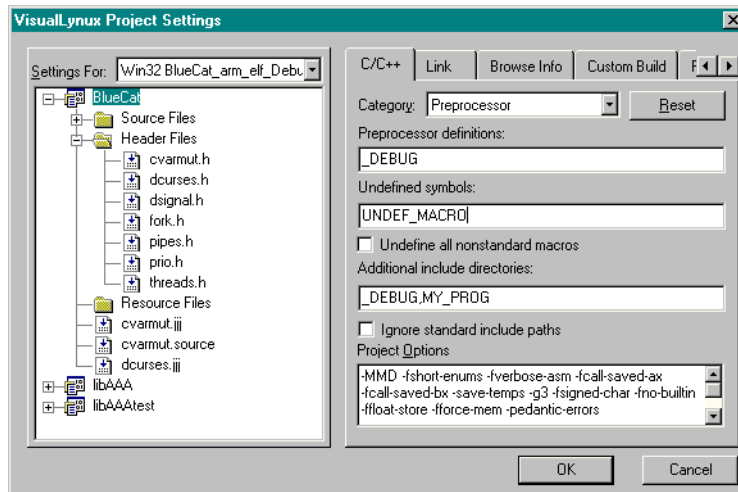


Figure 8-21: C/C++ Preprocessor

Table 8-20: C/C++ Preprocessor Options

Control	Command Line Option	Description
Preprocessor definitions	<code>-Dsymbol</code>	You can set up preprocessor definitions to pass to the C/C++ preprocessor in the form <code>-Dsymbol</code> . Delimit different symbols with a comma (,).
Undefine all nonstandard macros	<code>-undef</code>	Does not predefine any nonstandard macros (including architecture flags).
Undefine Symbols	<code>-Usymbol</code>	You can set up preprocessor undefinitions to pass to the C/C++ preprocessor in the form <code>-Usymbol</code> . Delimit different symbols with a comma (,).
Ignore standard include paths	<code>-nostdinc</code>	Does not pass any of the standard include file paths.
Additional include directories	<code>-Iinclude_dir</code>	Uses the specified include paths. Delimit different paths with a comma (,).

## C/C++ Special Options

The **Special options** dialog box is intended for use when you need to specify compiler options that cannot be edited using the **Settings** dialog box. For example, you can use this dialog box to enter specific hardware-dependent compile options.

The dialog box is shown in the following figure. Enter compile options in the **Special options** edit box. Remember that all options are used as entered, and appended to the compiler command line. Options entered are displayed in the **Project Options** edit box after you leave the **Special options** field.

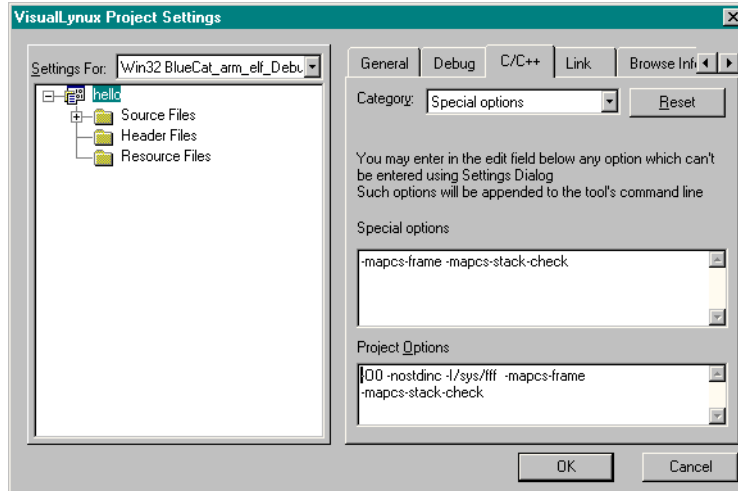


Figure 8-22: C/C++ Special Options

## Linker Options

This page allows you to set preferences for the LynuxWorks cross linker. There are six groups of linker options. You can select a group of options to edit using the **Category** drop-down list.

At the bottom of every category dialog box is a **Project Options** edit box that displays the selected command line options corresponding to the options from the **Settings** dialog box. The **Project Options** box can be manually modified at any time by typing options directly into the box. While entering options in the edit box, follow the rules for `ld.exe` command lines.

Note that when `gcc.exe` or `g++.exe` is used to link the program executable, linker options are passed to the linker using the `-w1` prefix. The **Settings** dialog box does not display this prefix in the **Project Options** edit control, and you should not type it in while entering specific options.

## Linker Option Categories

The following is a list of linker option categories available in VisualLynux:

- [Link General](#)
- [Link Customize](#)
- [Link Debug](#)
- [Link Input](#)
- [Link Output](#)
- [Link Special Options](#)

## Link General

The Linker General category settings are shown in the figure below:

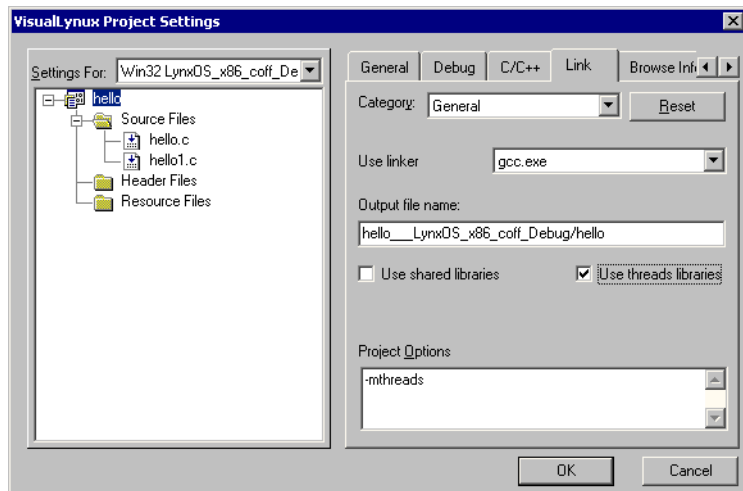


Figure 8-23: Linker Options Dialog Box

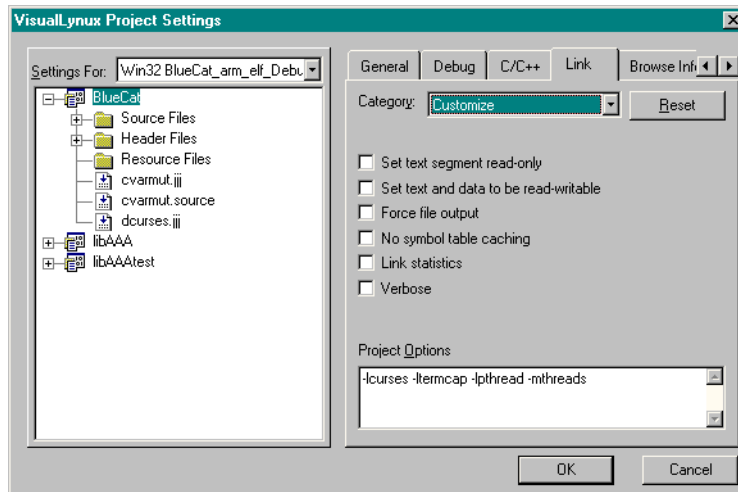
The Link General category allows you to change the control items shown in the table below:

**Table 8-21: Linker General Options**

Control Item	Description
<b>Use linker</b>	Use the drop-down list to explicitly select a linker ( <code>gcc.exe</code> , <code>g++.exe</code> , or <code>ld.exe</code> ). VisualLynux assigns a default linker based on the type of project.
<b>Output file name</b>	In this edit box, the path relative to the executable produced by the linker is displayed. By default, VisualLynux uses the output directory and project name to construct the project target name.
<b>Use shared libraries</b>	If clicked, forces the linker to link with shared instead of static libraries ( <code>-mshared</code> ) and <code>-shared</code> for BlueCat Linux.
<b>Use threads libraries</b>	Links an application written to use <code>pthread</code> s with special libraries and startup code that enable the application to run in a multi-threaded environment.

## Link Customize

The Linker Customize category settings are shown in the figure below:



**Figure 8-24: Linker Customize Options Dialog Box**

You can change the linker customization options shown in the table below:

**Table 8-22: Linker Customize Options**

Control Item	Description
<b>Set text segment read-only</b>	Sets the text segment to be read only, and marks the output as <code>NMAGIC</code> if possible.
<b>Optimization Set text and data to be read-writable</b>	Sets the text and data sections to be readable and writable. Does not page align the data segment. If the output format supports UNIX-style magic numbers, marks the output as <code>OMAGIC</code> .
<b>Force file output</b>	Retains the executable output file whenever it is still usable. Normally, the linker does not produce an output file if it encounters errors during the link process; it exits without writing an output file.
<b>No symbol table caching</b>	<b>ld</b> normally optimizes for speed over memory usage by caching the symbol tables of input files in memory. This option tells <b>ld</b> to optimize for memory usage instead by rereading the symbol tables as necessary. This may be required if <b>ld</b> runs out of memory space while linking a large executable.
<b>Link statistics</b>	Computes and displays statistics about the operation of the linker, such as execution time and memory usage.
<b>Verbose</b>	Displays the version number for <b>ld</b> . The <code>-v</code> option also lists the supported emulations.

## Link Debug

The linker debug category allows you to change options that control debugging data.

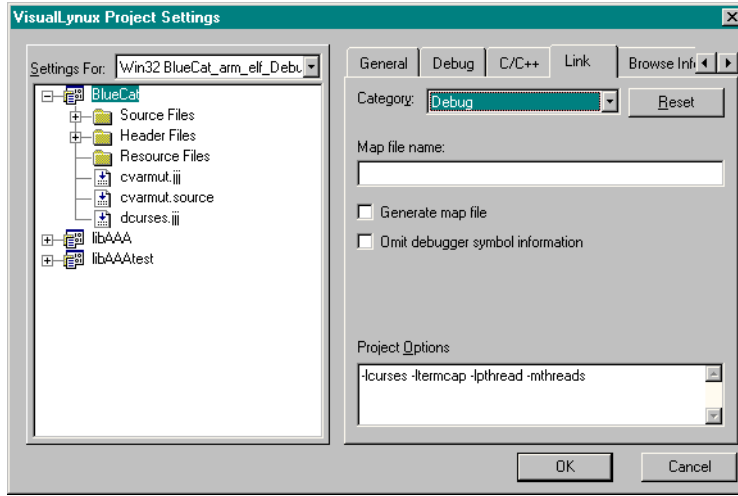


Figure 8-25: Linker Debug Options Dialog Box

You can change the following control items on the linker **Debug** page:

Table 8-23: Linker Debug Options

Control Item	Command Line Option	Description
Map file name	<code>-Map <i>mapfile</i></code>	Prints to the file <i>mapfile</i> a link map: diagnostic information about where symbols are mapped by <code>ld</code> , and information on global common storage allocation.
Generate map file	<code>-M</code>	Prints (to the standard output) a link map: diagnostic information about where symbols are mapped by <code>ld</code> , and information on global common storage allocation.
Omit debugger symbol information	<code>-S</code>	Omits debugger symbol information (but not for all symbols) from the output file.



## Link Input

The Linker Input category settings are shown in the figure below:

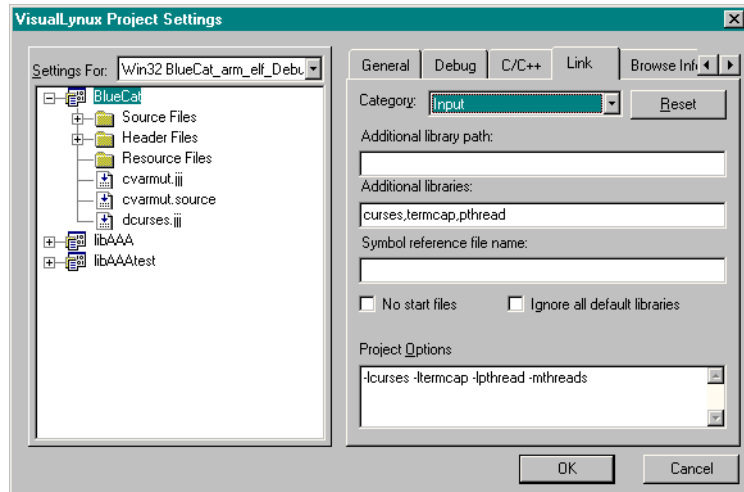


Figure 8-26: Linker Input Options Dialog Box

The control items in the table below appear on the linker Input page:

Table 8-24: Linker Input Options

Control Item	Command Line Option	Description
Additional library path	<code>-Llibrarypath</code>	Enter additional paths in which the linker should look for libraries in this field. Library paths should be delimited with a comma ( , ).
Additional libraries	<code>-l<math>ar</math></code>	Adds archive file <code>lib<math>ar</math>.a</code> to the list of files to link. This option can be used any number of times. <code>ld</code> searches its path list for occurrences of <code>lib<math>ar</math>.a</code> for every archive specified. Different libraries should be delimited with a comma ( , ).

Table 8-24: Linker Input Options (Continued)

Control Item	Command Line Option	Description
Debug Level Symbol reference file name	<code>-Rfilename</code>	Reads symbol names and their addresses from the filename, but does not relocate it or include it in the output. This allows your output file to refer symbolically to absolute locations of memory defined in other programs.
No start files	<code>-nostartfiles</code>	Does not use the standard system startup files when linking. The standard system libraries are used normally, unless <code>-nostdlib</code> or <code>-nodefaultlibs</code> is used.
Ignore all default libraries	<code>-nostdlib</code>	Does not use the standard system startup files or libraries when linking. No startup files, and only the libraries you specify are passed to the linker.

## Link Output

The Linker Output category options are shown in the figure below:

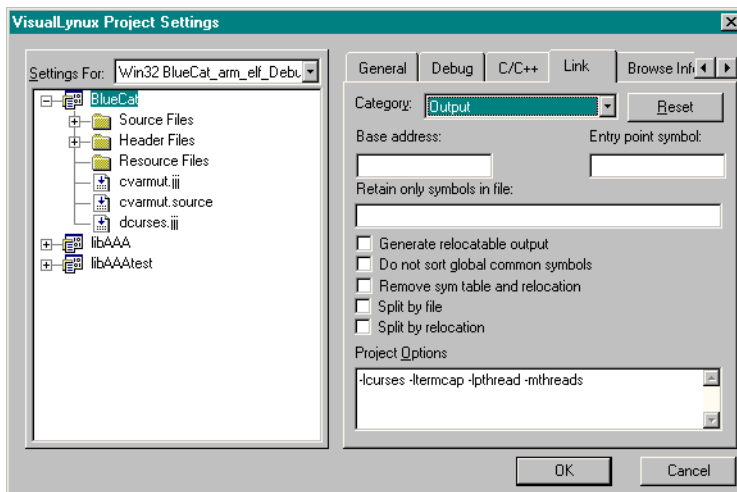


Figure 8-27: Link Output Options Dialog Box

You can change the following linker output control items:

**Table 8-25: Linker Output Options**

Control	Command Line Option	Meaning
Base address	<code>-Ttext org</code>	Uses <i>org</i> as the starting address for the text segment of the output file. <i>org</i> must be a single hexadecimal integer; for compatibility with other linkers, you can omit the leading <b>0x</b> usually associated with hexadecimal values.
Entry point symbol	<code>-e entry</code>	Uses <i>entry</i> as the explicit symbol for beginning execution of your program, rather than the default entry point.
Retain only symbols in file	<code>-retain-symbols-file filename</code>	Retains only the symbols listed in the file <i>filename</i> , discarding all others. <i>filename</i> is simply a flat file, with one symbol name per line. This option is especially useful in environments where a large global symbol table is accumulated gradually to conserve runtime memory.
Generate relocatable output	<code>-r</code>	Generates relocatable output, that is, generates an output file that can, in turn, serve as input to <b>ld</b> .
Do not sort global common symbols	<code>-sort-common</code>	Normally, when <b>ld</b> places the global common symbols in the appropriate output sections, it sorts them by size. First come all the one byte symbols, then the two byte symbols, then the four byte symbols, and then everything else. This is to prevent gaps between symbols from alignment constraints. This option disables that sorting.
Remove sym table and relocation	<code>-s</code>	Omits all symbol information from the output file.
Split by relocation	<code>-split-by-reloc</code>	Tries to create extra sections in the output file.
Split by file	<code>-split-by-file</code>	Similar to <code>-split-by-reloc</code> but creates a new output section for each input file.

## Link Special Options

The **Special options** dialog box shown in the following figure is used to specify compiler options that cannot be edited using the **Settings** dialog box. For example, you can use this box to enter specific hardware-dependent linker options.

Enter linker options in the **Special options** edit box. All options are used as entered by the user and appended to the linker command line. Options entered are displayed in the **Project Options** edit box after you leave the **Special options** field.

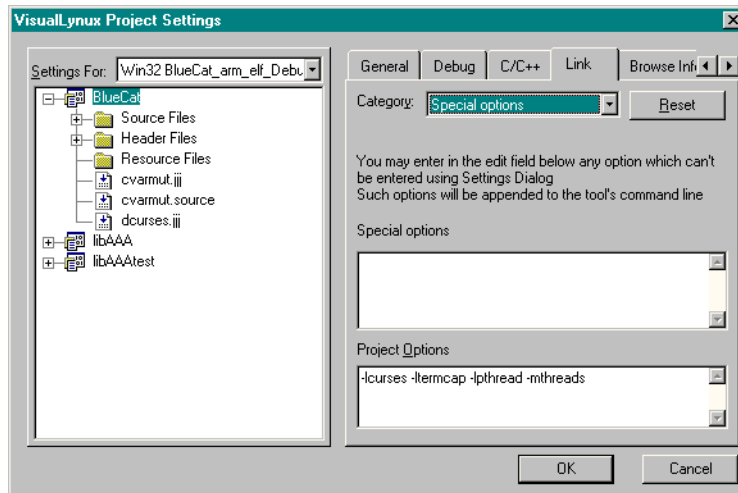


Figure 8-28: Linker Special Options Dialog Box

## Make Page

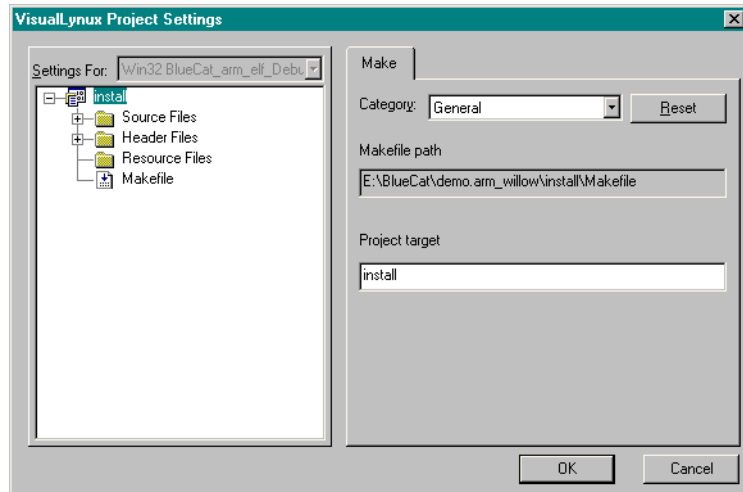
### Make Project

The **Make** page, shown in the next figure, allows for editing of project options for a project based on an external Makefile. The **Make** page for a project provides two dialog boxes that appear depending on the category selected in the **Category** drop-down list.

- The **General** category allows editing of general make options.

- The **Environment** category allows editing of data used to establish a build environment before invoking the `make` utility.

## General Options



**Figure 8-29: Make Project General Options**

The following control items appear on **General Make** page for a project:

**Table 8-26: Make Project General Options**

Command	Description
<b>Makefile path</b>	This field is for information purposes only. It displays the full path to Makefile used to build a project.
<b>Project target</b>	Enter or modify the name of the target (project goal) built by the <code>make</code> utility. If your Makefile contains several targets, you can use this field to specify the target to be built.

## Environment Options

A build environment can include:

- Shell scripts to be executed before processing the Makefile

- A set of environment variables that must be exported before processing the Makefile

VisualLynux executes shell scripts specified (using `bash`) and exports environment variables before invoking `make` to process the Makefile.

Usually, the VisualLynux installation process sets up scripts and environment variables required for every target configuration and displays them in the proper controls. You can add/remove scripts or variables, or modify values of variables.

## Setting up Scripts

To manipulate shell scripts use the small toolbar in the **Scripts** control (see next figure). You can then add a new script, remove a selected script, and arrange scripts in the list box. Remember that the script path must be entered in a UNIX-like form, using the CYGWIN convention for mounts and drives. The **Settings** dialog box checks the path and displays an error message if a script file does not exist.

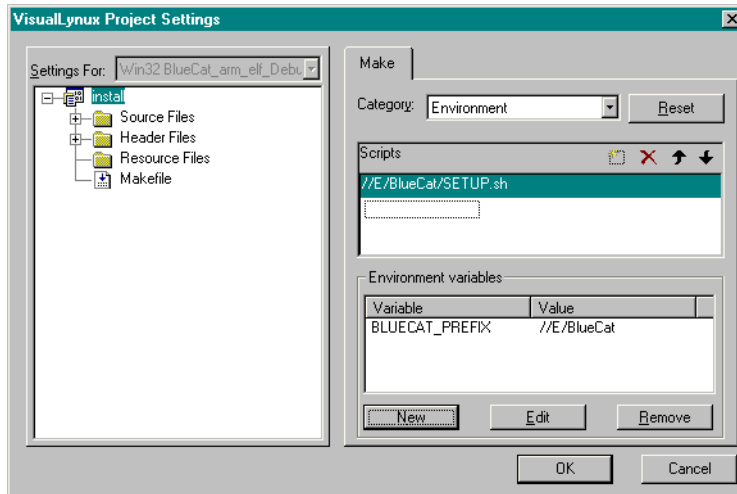


Figure 8-30: Setting Up Scripts

## Adding/Editing Environment Variables

To add a new variable press the **New** button. It invokes a dialog box (see next figure) in which you enter the variable name and value. To edit the value or name of a variable, select it in the **Variables** list and press the **Edit** button. To remove an

existing variable, select it in the **Environment variables** list and press the **Remove** button.

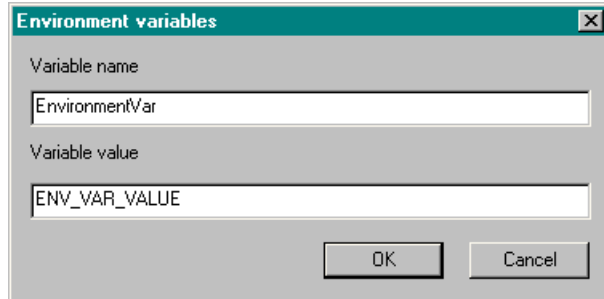


Figure 8-31: Editing Environment Variables

## Make File

The file **Make** page (see figure below) allows you to edit the name of the target to be built when the **Compile** command is invoked. Also, Visual Studio uses this name to remove intermediate files when the **Clean** command is executed.

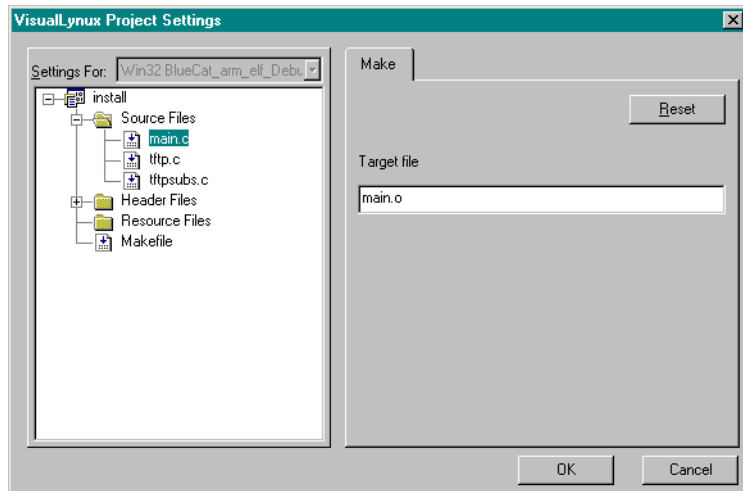


Figure 8-32: Make File Page

## Kernel Options

The **Kernel** page (see following figure) allows you to enter options used by VisualLynux to configure a LynxOS Kernel project. The **Kernel** page appears when the **Kernel Project** item is selected from the tree in left pane. There are three different groups of Kernel configuration options. You select a group of options to edit using the **Category** drop-down list.

The following are the kernel options categories:

- [Kernel General Page](#)
- [Kernel Configuration Page](#)
- [Kernel Parameters Page](#)

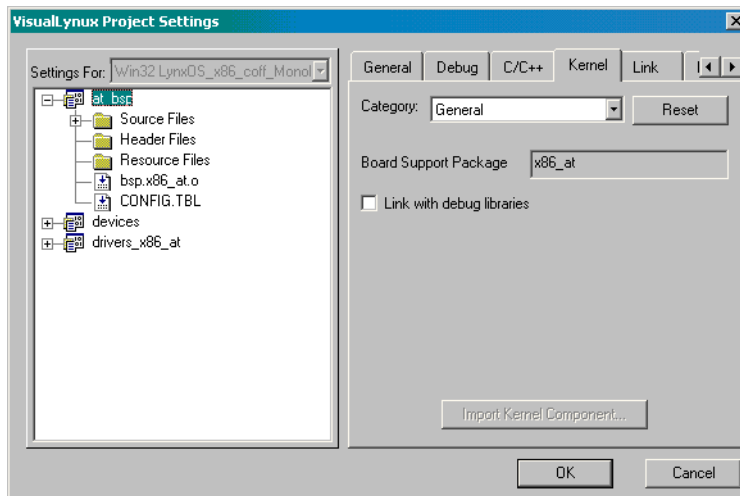


Figure 8-33: Kernel Page

If the `CONFIG.TBL` file is selected in the left tree view, the **Configuration** page appears instead of the **Custom Build** page. This page enables you to edit some of the kernel build parameters.



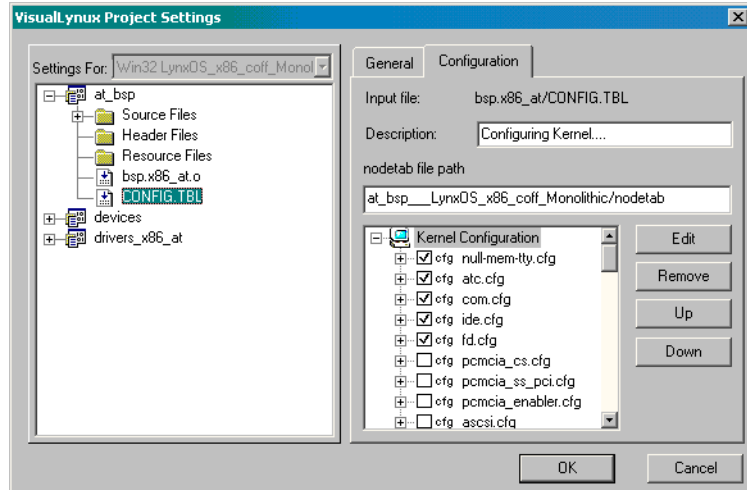


Figure 8-34: Configuration Page

The following table shows the control items on the **Kernel Configuration** page:

Table 8-27: Kernel Options Configuration Page

Item	Description
<b>Input file</b>	This field contains the name of the source file whose properties are displayed. Note that VisualLynx always saves a copy of kernel configuration data for a particular project configuration in the <i>configuration_name_CFG</i> subdirectory. <i>configuration_name</i> is a VisualLynx project configuration name. For example, for the project configuration Win32 LynxOS_x86_coff_Monolithic, VisualLynx creates a subdirectory named LynxOS_x86_coff_Monolithic_CFG and places CONFIG.TBL and all driver configuration files into this directory.
<b>Description</b>	This is a verbal description of the build step that appears in the Visual Studio <b>Output</b> window during the build.

Table 8-27: Kernel Options Configuration Page (Continued)

Item	Description
<b>nodetab file path</b>	Use this edit box to change the default path to the generated <code>nodetab</code> file. By default, the <code>nodetab</code> file is placed into the output directory.
<b>Configuration tree</b>	Represents driver configuration data to be included in the kernel. It can be edited exactly in the same way as in the <a href="#">Kernel Configuration Page</a> dialog box.

## Kernel General Page

The **Kernel General** page provides general information about the kernel project.

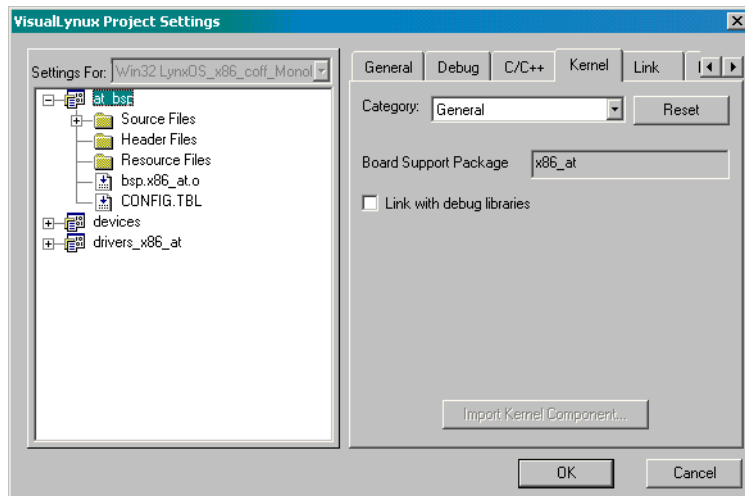


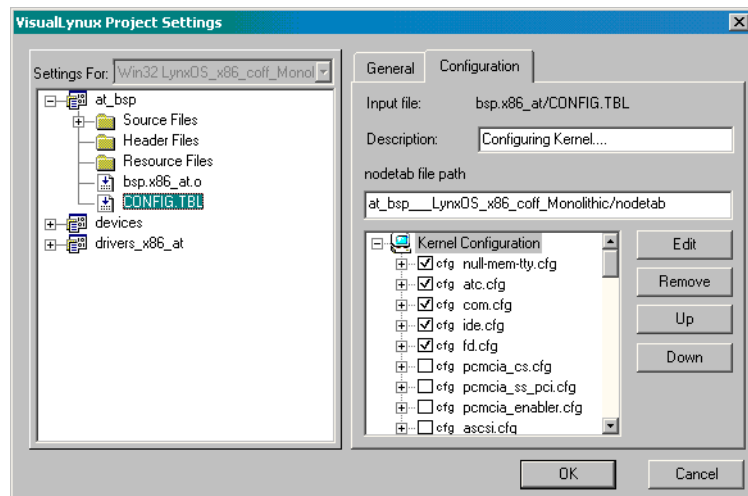
Figure 8-35: Kernel General Page

The following table shows the control items on the **Kernel General** page:

**Table 8-28: Kernel General Page Options**

Item	Description
<b>Board Support Package</b>	Displays the name of the Board Support Package from the Cross Development Kit that has been used to create the kernel project. This field is for information purposes only.
<b>Link with debug libraries</b>	If checked, VisualLynx uses the debug version of the drivers and devices libraries (libraries with <code>_d</code> suffix). Otherwise, optimized libraries are used.

## Kernel Configuration Page



**Figure 8-36: Kernel Configuration Page**

These kernel options allow you to edit kernel configuration data, select drivers to be included in the kernel, and specify driver and device properties. The dialog box presents kernel configuration data (`CONFIG.TBL` and driver configuration files) in a tree view. The root of the tree corresponds to the `CONFIG.TBL` file for a selected project configuration. Other items represent driver configuration files, driver

properties, devices, device nodes, etc. The tree contains the item types shown in the following table:

**Table 8-29: Kernel Configuration Tree Items**

Item	Description
<b>cfg</b>	Driver configuration file ( <code>.cfg</code> file) included in <code>CONFIG.TBL</code> or another configuration file.
<b>C:</b>	Character driver definition
<b>B:</b>	Block driver definition
<b>D:</b>	Device definition
<b>N:</b>	Device node definition
<b>sys</b>	System device definition
<b>n:</b>	Directly specified <code>nodetab</code> entry
<b>L:</b>	Cloned device node
<b>M:</b>	Module definition
<b>c:</b>	Direct output to the <code>config.h</code> file

During the build, VisuallyLinux processes `CONFIG.TBL` and driver configuration files using the `config.exe` utility. This build step produces the `nodetab`, `config.h`, and `sysdevices.h` files. Generated header files are used to compile the `conf.c` and `info.c` files.

---

**NOTE:** Working with configuration data using the VisuallyLinux **Setting** dialog box is very similar to editing `CONFIG.TBL` and appropriate driver configuration files manually. The user is expected to know how to configure kernels using the `CONFIG.TBL` file. For more details refer to the `config` and `config.spec` man pages.

---

You can specify different kernel configurations for different VisuallyLinux project configurations. This dialog box allows you to use the same kernel project and workspace to build kernels containing different sets of drivers or to configure drivers in a different way.

VisualLinux always stores configuration data and the `CONFIG.TBL` file for a particular project configuration in a separate directory. You can edit common kernel configuration data for several/all project configurations. If several configurations are selected in the **Settings For** drop-down list, the configuration data tree displays only items present in all project configurations. By selecting/deselecting an item, you include/exclude it from all selected project configurations.

VisualLinux always saves a copy of kernel configuration data for a particular project configuration in a `configuration_name_CFG` subdirectory, where `configuration_name` is a VisualLinux project configuration name. For example, for the project configuration `LynxOS_x86_coff_Monolithic`, VisualLinux creates a `LynxOS_x86_coff_Monolithic_CFG` subdirectory and places all driver configuration files in this directory.

To include any component in a kernel, check the corresponding box in the tree. To include an IDE driver in the kernel, check the box against the `ide.cfg` item. To exclude this driver from the kernel, uncheck the `ide.cfg` item. You can include or exclude other components (drivers, devices, device nodes) in the same way. Note that VisualLinux does not check for driver interdependency, so make sure that you have selected all necessary drivers, devices, and device nodes.

---

**NOTE:** If you have created a *new component*, VisualLinux does not mark it as selected automatically.

---

To remove a component permanently, select it and press **Remove**. VisualLinux does not remove an item immediately but marks it with a special overlapped image. The **Remove** button changes to a **Restore** button. You can return the item by selecting it and pressing the **Restore** button. Marked items are deleted from the configuration file when you close the dialog box or move to another page. Note that if you have just created a new item, the **Remove** button deletes it immediately.

To rearrange items at the same level, use the **Up** and **Down** buttons. The sequence in which items appear on the tree determines the major and minor device numbers in the generated `nodetab` file.

## Editing Item Properties

**Edit** button functions depend on the item selected from the tree. When pressed, the button displays a submenu listing available operations. If the submenu contains only one item, that operation is invoked implicitly.

The actions available for each kind of item are described in the table below:

**Table 8-30: Edit Options for Kernel Items**

Item Type	Action	Description
Root	<a href="#">Importing Configuration File</a>	Inserts a reference to the existing driver configuration file in the <code>CONFIG.TBL</code> file.
	<a href="#">Inserting Configuration File</a>	Creates a new driver configuration file and inserts a reference to it in the <code>CONFIG.TBL</code> file.
	<a href="#">Adding nodetab Entry</a>	Creates a configuration entry that directly specifies the <code>nodetab</code> entry ( <code>n: item</code> ).
	<a href="#">Adding config.h Line Entry</a>	Creates a configuration entry that directly dumps a <code>config.h</code> line.
cfg	<a href="#">Adding Driver Entry</a>	Creates a new driver specification line and adds it to the selected configuration file.
	<a href="#">New Module Entry</a>	Creates a new module specification line and adds it to the selected configuration file.
	<a href="#">Adding nodetab Entry</a>	Creates a configuration entry that directly specifies the <code>nodetab</code> line ( <code>n: item</code> )
	<a href="#">Adding config.h Line Entry</a>	Creates a configuration entry that directly dumps a <code>config.h</code> line.
C:	<a href="#">New Device Entry</a>	Creates a new device specification entry and adds it to the selected <i>character driver</i> .
	<a href="#">Editing Driver Entry</a>	Edits properties of the selected <i>character driver</i> .
B:	<a href="#">New Device Entry</a>	Creates a new device specification entry and adds it to the selected <i>block driver</i> .
	<a href="#">Editing Driver Entry</a>	Edits properties of the selected <i>block driver</i> .
D:	<a href="#">New Device Node Entry</a>	Creates a new device node and adds it for the selected device.
	<a href="#">New System Device Entry</a>	Creates a <i>system device</i> entry for the selected device.
	<a href="#">Editing Device Entry</a>	Edits selected device properties.
N:	<a href="#">Editing Device Node Properties</a>	Automatically invokes a dialog box to edit <i>device node properties</i> .

Table 8-30: Edit Options for Kernel Items (Continued)

Item Type	Action	Description
sys:	<a href="#">Editing System Device Entry</a>	Automatically invokes a dialog box to edit <i>system device properties</i> .
n:	<b>Properties...</b> <a href="#">Adding nodetab Entry</a>	Automatically invokes a dialog box to edit <i>nodetab entry properties</i> .
L:	<b>Properties...</b> <a href="#">Device Node Entry</a>	Automatically invokes a dialog to edit <i>module properties</i> .
M:	<b>Properties...</b> <a href="#">Editing Module Properties</a>	Automatically invokes a dialog box to edit <i>module properties</i> .
c:	<b>Properties...</b> <a href="#">Editing .config.h Line</a>	Automatically invokes a dialog box to edit <i>config.h line properties</i> .

## Importing Configuration File

Perform this operation to add an existing driver configuration file to the kernel configuration data. Select the root item from the configuration tree, press the **Edit** button, and select the **Import cfg file** menu item. VisualLinux displays a dialog box containing a list of driver configuration files. After you have selected a file, VisualLinux loads it and adds a subitem to the end of the tree. To move this file to another position in the tree, use the **Up** and **Down** buttons.

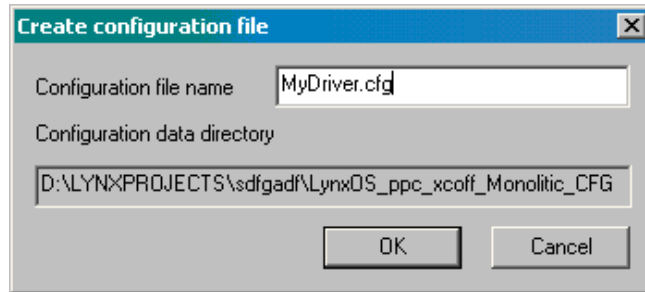
---

**NOTE:** VisualLinux creates a copy of the imported driver configuration file and saves it in the configuration data directory for a selected project configuration. If several project configurations are selected, a separate copy of the imported configuration file is created for every project configuration.

---

## Inserting Configuration File

This operation creates a new empty driver configuration file (or files, if several project configurations are being edited). Select the root item from the tree, press the **Edit** button and select the **Insert cfg file** menu command. The following dialog box appears:



**Figure 8-37: Creating Configuration File**

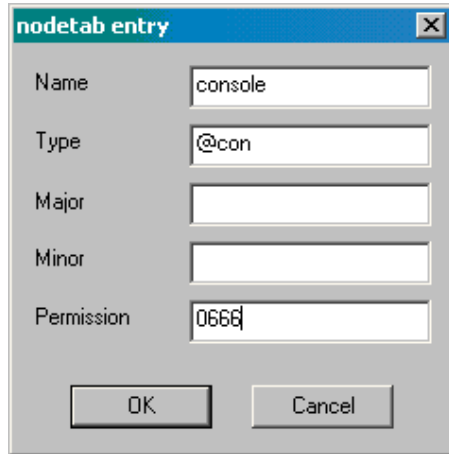
Enter the name of a configuration file (including extension) in the **Configuration file name** edit box and press **OK**. The **Configuration data directory** edit box shows the full path to the directory in which the new file is to be placed.

An entry for the new configuration file is inserted into the tree under the root item. The entry is the last item in the tree. Use the **Up** and **Down** buttons to change its position.

### Adding nodetab Entry

This operation allows you to add an entry to generate the `nodetab` line directly. Select the root item or the driver configuration file (`cfg` item) from the tree, press the **Edit** button and select the **Add nodetab entry** menu command. The following dialog box appears:





**Figure 8-38: Nodetab Entry**

Fill in the dialog box fields as appropriate and press **OK**. The new `nodetab` entry is added as the last item in the subtree. Use the **Up** and **Down** buttons to position it as desired. For information on `nodetab` lines, refer to the `nodetab` man page.

### config.h Line Entry

To create a line that is dumped directly to the `config.h` file, select the root tree item or the appropriate `cfg` item, press the **Edit** button and select the **Add config.h line** menu item. The following dialog box appears. Enter the line to add to the `config.h` file as is and press the **OK** button. The appropriate tree item is added to the end of the list.

---

**NOTE:** All such lines are written to the `config.h` file before all other lines generated by the `config.exe` utility.

---

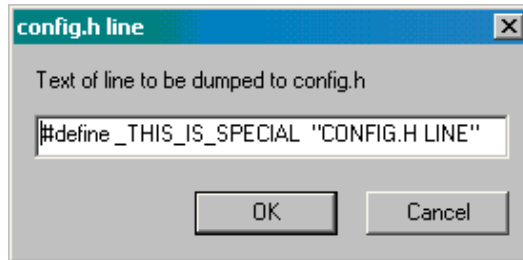


Figure 8-39: config.h Line

### Editing .config.h Line

To edit a `config.h` line entry, select it from the tree and press the **Edit** button. The dialog box above pops up. Edit the line and press **OK**.

### Driver Entry

To create a driver entry, select the appropriate driver configuration file (`cfg` item), press the **Edit** button, and select the **Add driver** menu item. The following dialog box appears:

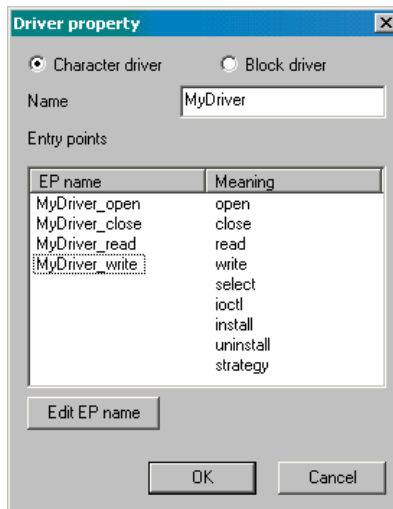


Figure 8-40: Driver Properties

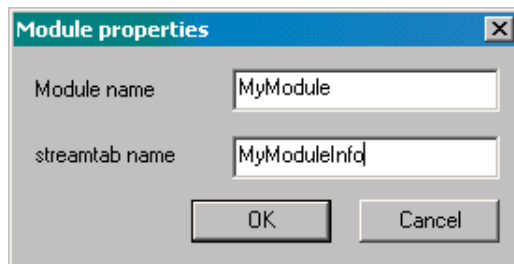
Select a driver type by checking the appropriate radio button, then enter entry point names in the **Entry points** list box. Select the appropriate line in the list and press the **Edit EP name** button. Then enter or edit the entry point name in the edit box. You can leave some entry point names empty. In such case the `ionull` entry point name is used. A new driver entry is added to the end of the driver configuration file. To move it, use the **Up** and **Down** buttons.

## Editing Driver Entry

To *edit* a driver entry, select it from the tree, press the **Edit** button, and select the **Properties** menu item. The **Driver property** dialog box appears. You may change driver *properties*, but note that you may not change driver *type*.

## New Module Entry

To create a new module, select the appropriate driver configuration file, press **Edit**, and select the **Add module** command. The following dialog box appears:



**Figure 8-41: Kernel Module Properties**

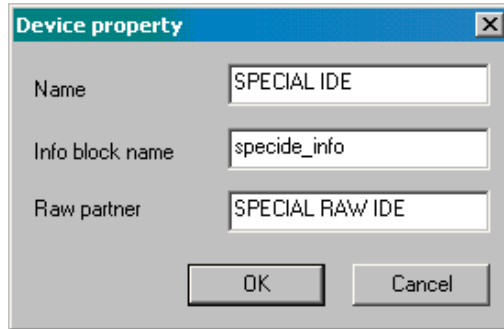
Fill the text boxes and press **OK**. A new module entry is added to the end of the configuration file. Use the **Up** and **Down** buttons to move the module entry.

## Editing Module Properties

To *edit* module properties, select the module from the tree view and press **Edit**. The **Module properties** dialog box appears. You can change appropriate values.

## New Device Entry

To create a new device entry, select the appropriate driver (C: or B:), press the **Edit** button and select the **Add device** menu item. The **Device property** dialog box pops up.



**Figure 8-42: Kernel Device Properties**

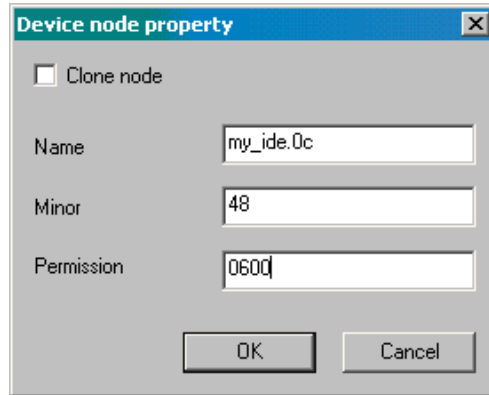
Fill in edit controls (the **Name** must be filled in) and press **OK**. A new device entry is added to the tree under the selected driver. If needed, move the entry using the **Up** and **Down** buttons.

## Editing Device Entry

To *edit* a device entry, select it from the tree, and press the **Edit** button. Select the **Properties** menu item. The **Device property** dialog box appears.

## Device Node Entry

To create a device node entry, select the device from the tree, press the **Edit** button and select the **Add device node** menu item. The **Device node property** dialog box appears:



**Figure 8-43: Device Node Properties**

Select a device node type using the **Clone node** check box. Note that this type of node does not use the **Minor** field and is disabled if **Clone node** is checked. Enter a device node name, minor number, and permissions in the corresponding field. If left empty, the **Permission** field defaults to `0600`. A new device node entry is added at the end of the device node list for the selected device.

---

**NOTE:** If the **Clone node** is checked, an entry of the type `L:` is created. Otherwise the `N:` entry is created.

---

## Editing Device Node Properties

To *edit* device node properties, select a device in the tree (`L:` or `N:`) and press the **Edit** button. VisualLynux automatically displays the **Device node property** dialog box. Change the appropriate data. Note that the device node type cannot be changed and the **Clone node** check box is disabled.

## System Device Entry

To create a system device entry, select the device from the tree, press the **Edit** button and select the **Add system device** menu item. The **System device property** dialog box appears.

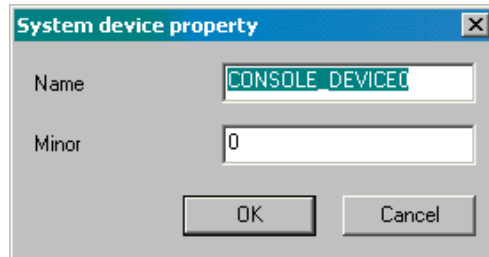


Figure 8-44: System Device Properties

Enter a name and minor number for the new system device and press the **OK** button. A new system device entry is placed at the end of the device node list.

## Editing System Device Entry

To *edit* a system device entry, select it from the tree and press the **Edit** button. VisuallyLinux automatically displays the **System device property** dialog box. Modify the desired values and press **OK**.

## Kernel Parameters Page

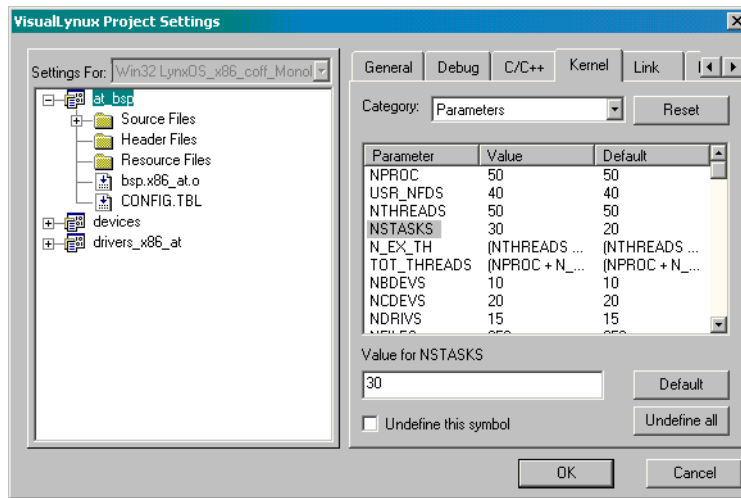


Figure 8-45: Kernel Parameters Page

This option allows you to change some of the kernel build parameters. It displays the table representing lines from the corresponding `uparam.h` file. The table shows parameter names, current values, and default values (that is, when the project was created). To edit a value, select the appropriate line in the table. Its current value appears in the edit box. Other controls are described in the [Kernel Parameter Controls](#) table below.

Note that some of the lines define parameter values as C expressions. Usually you need not edit them, but VisualLynux does not prevent you from doing so.

---

**NOTE:** Modifying such values can cause an invalid kernel build.

---

VisualLynux creates and stores a `uparam.h` file for every project configuration. The actual name of the file used for a particular configuration is `configuration_name_uparam.h`, where `configuration_name` is the VisualLynux project configuration name. For example, VisualLynux creates the file `LynxOS_x86_coff_Monolithic_uparam.h` for the project configuration `Win32 LynxOS_x86_coff_Monolithic`.

If you have selected several configurations in the **Settings For** drop-down list, VisualLynux displays empty values for parameters that differ for some configurations. Changing a value for such parameters sets the new value for all configurations.

The following table shows control items for changing kernel parameters:

**Table 8-31: Kernel Parameter Controls**

Item	Description
<b>Default</b>	This command replaces the current value for a selected parameter with a default value.
<b>Undefine this symbol</b>	This command inserts the appropriate <code>#undef name</code> statement in the <code>uparam.h</code> file before the corresponding <code>#define</code> statement. Use this to reduce the number of warning messages displayed during compile time.
<b>Undefine all</b>	This command inserts <code>#undef name</code> statements for all symbols in the <code>uparam.h</code> file.





---

## Overview

The LynuxWorks FTP program (`v1ftp.exe`) has been developed to facilitate access to FTP servers for VisualLynux users. It provides a Windows Explorer-like interface for easy transfer, creation, removal and renaming of files and directories on both target and a host machines. The LynuxWorks FTP application contains its own online help facility.

LynuxWorks FTP is designed and implemented with VisualLynux user needs in mind. LynuxWorks FTP contains special functionalities, such as creation, deletion, and renaming of FTP directories and files. Also, LynuxWorks FTP assumes high speed connections to target FTP servers through a local area network. Therefore, it does not use asynchronous implementation of the file/directory transfer or other FTP operations.

LynuxWorks FTP provides the following functionality:

- [Connecting to an FTP Server](#) on the target computer
- [Browsing FTP Folders](#) on a target computer
- [Displaying Contents of FTP Folders](#)
- [Browsing Local Folders](#) and directories
- [Manipulating FTP Folders](#) - creating, renaming, and removing
- [Manipulating FTP Files](#) - renaming and removing on target
- [Manipulating Files on a Local Computer](#) - creating folders, removing directories
- [Downloading Files and Folders](#) from FTP server to local computer
- [Uploading Files and Folders](#) to an FTP server

To perform a particular operation, you can use toolbar buttons, menu items, and keyboard shortcuts.

## LynuxWorks FTP Main Window

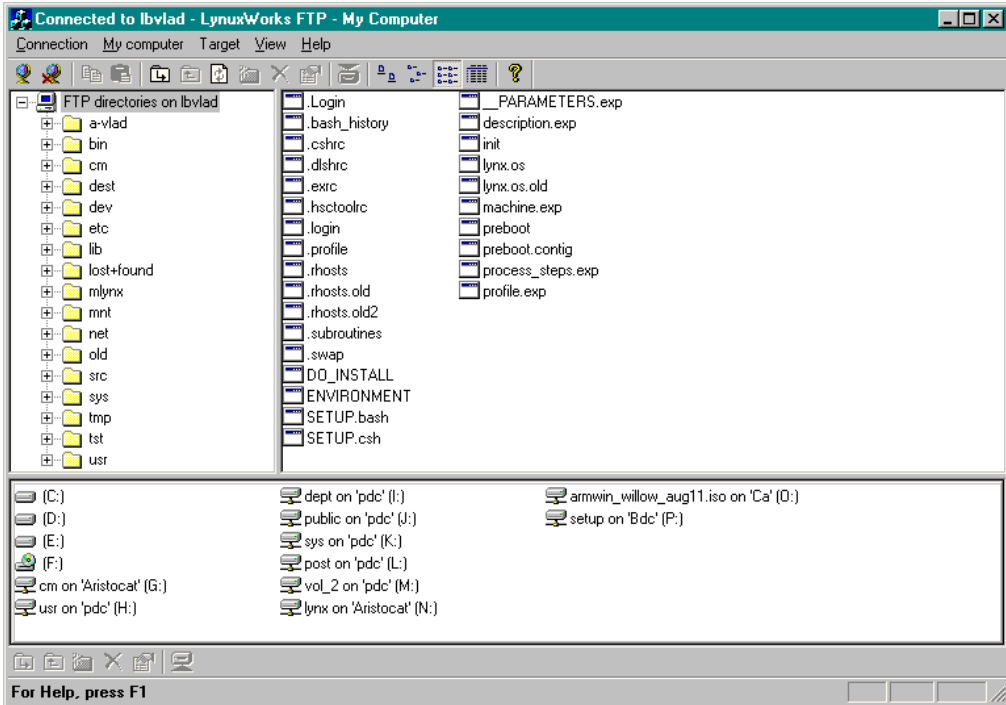


Figure 9-1: Main FTP Window

The LynuxWorks FTP main window contains three panes (shown in the previous figure) displaying an FTP directory tree (top left pane), contents of the current FTP directory (top right pane), and contents of the current directory on the local computer. The active pane is highlighted visually by a border. Both the top right and bottom panes can display file lists in four different modes (large icons, small icons, list, and details). The user selects the desired mode using the menu item or toolbar button.

In addition to panes, two toolbars reside at the top and the bottom of the main window. The top toolbar is used to perform operations with directories and files

located on the target computer and to select the display mode of the active pane. The bottom toolbar pertains to the local computer and is used to perform operations on files displayed in the bottom pane.

---

## LinuxWorks FTP Functions

### Connecting to an FTP Server

You can establish a connection to the target FTP server using the `Connect` command or by passing [command line](#) parameters while launching the LinuxWorks FTP program. To invoke the `Connect` command, open the **Connection** submenu and select the **Connect** item, or click on this toolbar button



to invoke the command.

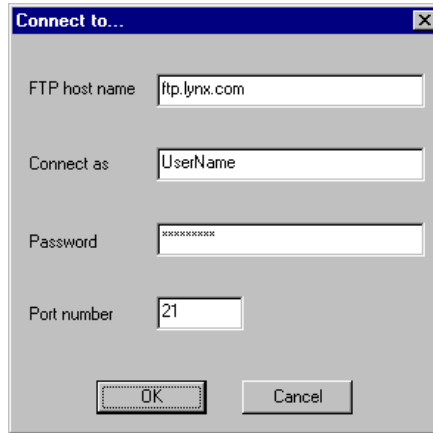
When the **Connect** command is invoked, the LinuxWorks FTP utility brings up the dialog box shown in the following figure. Fill in the fields and press **OK** to connect to the target FTP server.

If LinuxWorks FTP has already been connected to a target, it asks for permission to close the existing connection before the dialog box appears.

After the connection has successfully been established, LinuxWorks FTP fills the FTP directory view and sets the root FTP directory on the target machine as the current directory. The contents of the root directory are displayed in the FTP view.

An error occurring while connecting to the FTP server is displayed in an error box.

To disconnect from the server, open the **Connection** submenu and select **Disconnect**. LinuxWorks FTP disconnects from the server automatically when you close the main window or invoke the **Exit** command.



**Figure 9-2: LynuxWorks FTP connect window**

The following table describes the FTP connection window controls:

**Table 9-1: FTP Connect Window Controls**

Control Item	Definition
FTP host name	Enter the name of the target computer as defined in your TCP/IP network.
Connect as	Enter your name. You can leave this field blank and LynuxWorks FTP registers you on the FTP server as Anonymous.
Password	Enter a password if it is required to connect to the server. Usually, when connecting to the server as Anonymous, a password is not required.
Port number	You can change the default FTP port number if the FTP service running on the target machine uses a non-default TCP/IP port.

## LynuxWorks FTP Command Line

You can connect to the target FTP service automatically, launching the LynuxWorks FTP program from a command line that specifies the target machine. The command line contains switches that provide values for connection

parameters. Any switch should start with “-” or “/” followed by a letter (see table below) and parameter value without blank spaces. LynuxWorks FTP recognizes the switches shown in the following table:

**Table 9-2: FTP Command Line Switch Letters**

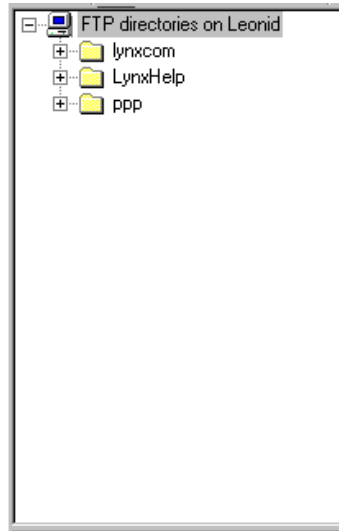
Command Line Switch Letter	Parameter Value
h or H	FTP host computer name
u or U	User name (can be omitted)
w or W	User password (can be omitted)
p or P	Port number (can be omitted)

For example, the following command line connects *UserName* to the FTP server [ftp.lynx.com](http://ftp.lynx.com) using the password *pswUser* and the default FTP port number (21).

```
vlftp.exe -hftp.LynuxWorks.com -uUserName \  
-WpswUser
```

## Browsing FTP Folders

LynuxWorks FTP displays the FTP folder hierarchy in a tree as shown in the figure below. The root item in the tree corresponds to the FTP home directory on the target machine. Its name contains the name of the target machine to which the program is connected. If there is no active connection to the FTP server, the tree contains only the root item named `not connected`.



**Figure 9-3: FTP Folder Hierarchy**

You can browse FTP directories on the target using the keyboard, mouse or menu/toolbar commands. Every time you change the selected item in the tree, the right pane (a file list) displays contents of the selected FTP folder. The folder whose contents are currently displayed in the list view is highlighted and has a **File**

**Open**  icon.

After you have selected an FTP folder from the tree, you can [remove](#) or [rename](#) it, or [download](#) its contents to the local computer.

## Keyboard Controls

The following keyboard controls can be used to browse FTP folders:

- |                   |  |
|-------------------|--|
| <b>Down Arrow</b> | Moves the selection to the next directory at the current level, or to the first directory on the next level (if the current item is expanded). |
| <b>Up Arrow</b>   | Moves the selection to the previous directory at the current level or to the parent directory.   |

<b>Left Arrow</b>	Collapses the current subtree and moves the selection to the parent item.
<b>Right Arrow</b>	Expands the current subtree and selects the first directory in the subtree.
<b>Home</b>	Moves the selection to the root item.
<b>End</b>	Moves the selection to the last item in the tree.



## Mouse Controls

Use the following mouse controls to browse FTP folder:

- Click an item label or icon to select the item.
- Double-click on an item to open the folder.
- Click on the “+” sign to open the folder.

## Menu and Toolbar Commands

To use the following menu or toolbar commands, activate the FTP directory view:

- To expand a currently selected subtree and select the first child item, select the **Target/Open folder** menu item. You can use the **Open Folder**  button in the top toolbar. This command is unavailable if you have already tried to expand an item that didn't contain a subfolder.
- To select the parent folder, use the **Target/Parent folder** menu command, or press the **Parent folder**  button in the top toolbar. This command is unavailable if the top tree item is currently selected.

You can also invoke the **Parent folder** command from the popup menu (right mouse click).

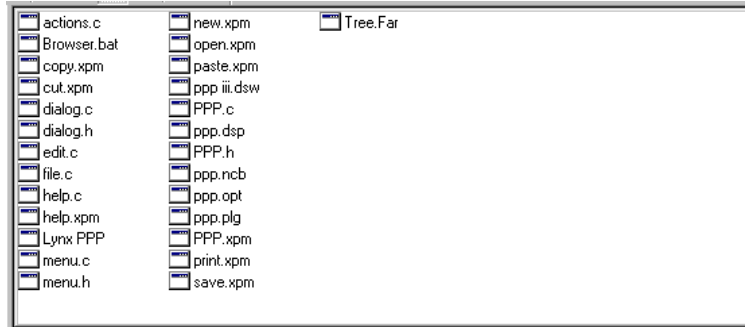
---

**NOTE:** Initially, all items in the tree have a “+” sign indicating that the item has subfolders. In fact, the program checks if the item really has subfolders when you try to expand it. If the item doesn't have subfolders, the “+” sign disappears.

---

## Displaying Contents of FTP Folders

The contents of FTP folders on the target machine are displayed in the FTP file list view (see figure below). LynuxWorks FTP updates this view every time you move the selection to another folder in the [FTP Directory Tree](#). The view displays the file list in four different modes: large icons, small icons, list, and details.





**Figure 9-4: FTP Directory Tree**


To change the display mode using the menu, do the following:


1. Make the FTP file list view active.
2. Select a desired command in the **View/List** submenu.

You can also use the following toolbar buttons in the top toolbar:

 Displays the file list using large icons

 Displays the file list using small icons

 Displays the files in list mode

 Displays the file list in detailed mode


## Manipulating FTP Folders

Before performing any operation on an FTP folder you should make the FTP folder tree view active and select the folder you want.



## Removing an FTP Folder

To remove an FTP folder do the following:

1. Make the FTP folder tree view active.
2. Select the folder to remove. Ensure that the folder is empty. If it isn't, delete all files and subfolders.
3. Select **Target/Remove** from the main menu, *or*
  - Press the **Delete**  button in the top toolbar, *or*
  - Press the right mouse button and select **Remove** from the popup menu.


## Renaming a Folder

To rename a folder do the following:

1. Make the FTP folder tree view active.
2. Select the folder to rename.
3. Right-click the mouse and select **Rename**, or edit the folder name in-place.

## Creating an FTP Folder

To create a new FTP folder on the target computer, do the following:

1. Make the FTP folder tree view active.
2. Select the subdirectory under which you are going to create a new folder.
3. Select **Target/New folder** in the main menu, *or*
  - Press the right mouse button and select **New folder** in the popup menu, *or*
  - Press the **Create Folder**  button in the top toolbar.

LynuxWorks FTP creates a new folder and adds a new item to the FTP folder tree. The new item is selected and LynuxWorks FTP goes into label edit mode, allowing you to change the folder name.

## Displaying FTP Folder Properties

To display folder properties, do the following:

1. Make the FTP folder tree view active.
2. Select the folder you wish to look up.
3. Select **Target/Properties** in the main menu, *or*

Press the **Properties**  button in the top toolbar.

LinuxWorks FTP displays a dialog box that contains the description of the selected folder.


## Manipulating FTP Files

Before performing any operation on FTP files, activate the FTP file list view.

### Removing FTP Files

To remove FTP files from a folder, do the following:

1. Select the FTP folder in the tree view to display its contents in the FTP file list view.
2. Make the FTP file list view active.
3. Select one or more files in the file list view.
4. Select **Target/Remove** in the main menu, *or*

- Press the **Delete**  button in the top toolbar, *or*
- Press the right mouse button and select **Remove** in the popup menu.

LinuxWorks FTP asks for confirmation of the deletion and removes files only if you reply **Yes**.

### Renaming an FTP File

To rename an FTP file, do the following:

1. Select the FTP folder in the tree view to display its contents in the FTP file list view.

2. Make the FTP file list view active.
3. Select exactly one file in the list.
4. Select **Target/Rename** in the main menu, *or*
  - Press the right mouse button and select **Rename** in the popup menu, *or*
  - Click once more on the label of the selected item.

LynuxWorks FTP goes into label edit mode for the selected file. When finished editing, press the **Enter** or click outside of the edit field.


---

**NOTE:** This command is available only if you select exactly one item in the list.

---

## Displaying File Properties

To display file properties, do the following:

1. Select the FTP folder in the tree view to display its contents in the FTP file list view.
2. Make the FTP file list view active.
3. Select exactly one file in the list.
4. Select **Target/Properties** in the main menu, *or*
  - Press the **Properties**  button in the top toolbar, *or*
  - Press the right mouse button and select **Properties**.

---

**NOTE:** This command is available only if you have selected just one file in the list.



---

## Browsing Local Folders

Initially, the bottom pane in the LynuxWorks FTP main window displays the list of both local and remote drives on your machine.

## Opening Folders

You can browse the contents of the folders on the drives. To open any folder and display its contents, do the following:

1. Make the local file list view active.
  2. Select the folder whose contents you wish to display. Folder items have the **Folder**  icon on the left. When the local file view displays the list of drives, the items have the appropriate drive icon.
  3. Double-click on the folder item, *or*
    - Invoke **My computer/Open folder** using the menu, *or*
    - Press **Open Folder**  in the bottom toolbar, *or*
    - Press the right mouse button and select **Open folder** from the menu.
- LynuxWorks FTP changes the contents of the local file list view.


---

**NOTE:** The **Open folder** command is inaccessible if you select non-folder items from the view, or select more than one item.

---

## Displaying the Parent Folder

To return to the parent folder and to display its contents, do the following:

1. Make the local file list view active.
2. Select **My computer/Parent folder** in the menu, *or*
  - Press **Parent folder**  in the bottom toolbar, *or*
  - Press right mouse button and select **Parent folder**.

LynuxWorks FTP displays contents of the parent folder.

---

**NOTE:** The command is inaccessible if the root folder is displayed (drives list).


---


## Changing the View Mode


To change the display mode, do the following:

1. Make the local file list view active;
2. Select a desired mode in the **View/List** submenu.

You can also use the following toolbar buttons in the top toolbar:

 Displays the file list using large icons

 Displays the file list using small icons

 Displays the files in list mode


 Displays the file list in detailed mode

## Manipulating Files on a Local Computer

Before performing any operation on files on the local computer, you should make the local file list view active. Then [Browse to the folder](#) on which you are going to perform the operation.

### Creating a New Folder

To create a new folder, do the following:

1. Make the local file list view active.
2. Go to the folder on which you are going to perform the operation.
3. Select **My computer/New folder** in the main menu, *or*
  - Press the right mouse button and select **New folder** from the popup menu, *or*
  - Press the **New folder**  button in the bottom toolbar.

LynuxWorks FTP creates a new folder as a child of the folder currently displayed in the local file view and goes into label edit mode. You can edit the name of the new folder.

---

**NOTE:** You cannot create a new folder if the disk drive list is displayed.

---

### Renaming Files or Folders

To rename a file or a folder on the local machine, do the following:


1. Make the local file list view active.
2. Go to the folder on which you are going to perform the operation.

3. Select exactly one file or folder.
4. Invoke **My computer/Rename** from the main menu, *or*
  - Press the right mouse button and select **Rename** from the menu, *or*
  - Click once more on the label of selected item.

LynuxWorks FTP goes into label edit mode for the selected file. When finished editing, press **Enter** or click outside of the edit field.

## Displaying File or Folder Properties

To display file or folder properties, do the following:

1. Make the local file list view active.
2. Browse to the folder that you want.
3. Select exactly one file or folder.
4. Select **My computer/Properties** in the main menu, *or*
  - Press the right mouse button and select **Properties** from the menu, *or*
  - Press the **Properties**  button in the bottom toolbar.

LynuxWorks FTP displays a dialog box containing the description of the selected file or folder:

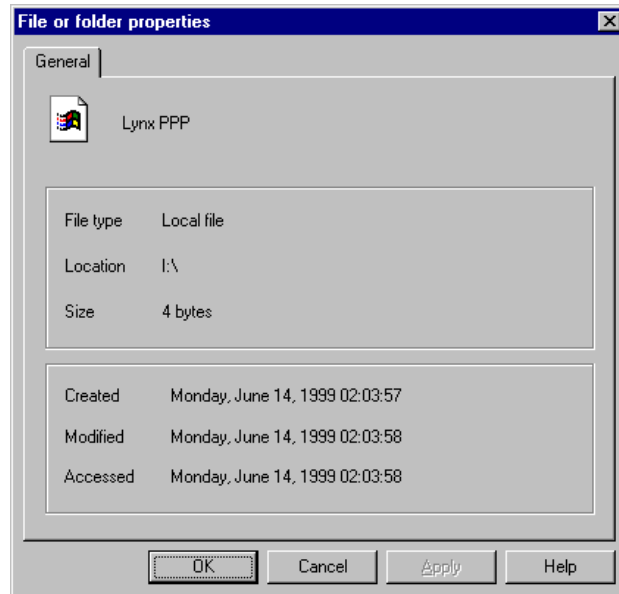



Figure 9-5: File Properties

## Removing Files and Folders

To remove one or more files or folders on the local machine, do the following:

1. Make the local file list view active.
2. Go to the folder from which you wish to delete files or folders.
3. Select files and folders you are going to remove.
4. Select **My computer/Remove** in the main menu, *or*
  - Press the right mouse button and select the **Remove**, *or*
  - Press the **Delete**  button in the bottom toolbar.

LynuxWorks FTP asks for your permission to remove the selected files or folders.

---


**NOTE:** Use the `Recycle Bin` folder to restore removed files and folders.

---

## Downloading Files and Folders

### Downloading Files


To download one or more files, do the following:

1. [Browse to the folder](#) on the local computer to which you wish to download the file from the FTP server.
2. Select the folder to be downloaded using the [FTP Directory Tree](#).
3. In the FTP file list view select files to download.
4. Select the **Target/Download** menu item, *or*
  - Press the right mouse button and select the **Download** command in the popup menu, *or*
  - Press the **Download**  button in the top toolbar.

LynuxWorks FTP brings up the [Transfer Dialog Box](#) that displays the progress of the download. If some files already exist in the current local folder, LynuxWorks FTP asks for permission to replace them. You can permit replacing, skip transferring files, or cancel the operation.

### Downloading Folders

To download a folder from the target machine to the local host, do the following:

1. [Browse to the folder](#) on the local computer to which you wish to download the folder from the FTP server.
2. Using the [FTP Directory Tree](#), select the folder to be downloaded.
3. Select **Target/Download** in the main menu, *or*
  - Press the right mouse button and select **Download** from the menu, *or*
  - Press the **Download**  button in the top toolbar.

LynuxWorks FTP brings up the [Transfer Dialog Box](#) that displays the progress of the download. If a folder with the same name already exists in the current local directory, LynuxWorks FTP asks for permission to replace the folder.

---


**NOTE:** If the selected folder contains subfolders, they are also downloaded.

---



## Uploading Files and Folders

To upload files or folders from the local host to the target, do the following:

1. Select the folder in the [FTP Directory Tree](#) to which you wish to transfer files and folders from the local computer.
2. [Browse to the folder](#) on the local computer where the files and folders you wish to transfer to the target machine reside.
3. Select all files and folders you wish to upload.
4. Select **My computer/Upload** in the main menu, *or*
  - Press the right mouse button and select **Upload** from the menu, *or*
  - Press the **Upload**  button in the bottom toolbar.

LynuxWorks FTP brings up the [Transfer Dialog Box](#) that displays the progress of upload. If a folder or file with the same name already exists on the target, LynuxWorks FTP asks for permission to replace it.

---

**NOTE:** If the selected folder contains subfolders, they are also uploaded.

---

## Transfer Dialog Box

The transfer dialog box enables the user to control the process of downloading or uploading files and folders. This dialog box is shown in the following figure. When a file or folder transfer is occurring, the dialog box displays the name of the source directory, the name of the file being transferred, as well as the full name of the destination file. The progress bar shows the status of the current file processing.

In addition, the **Status** tree view displays the status of the files and folders in the transfer process (see figure below).

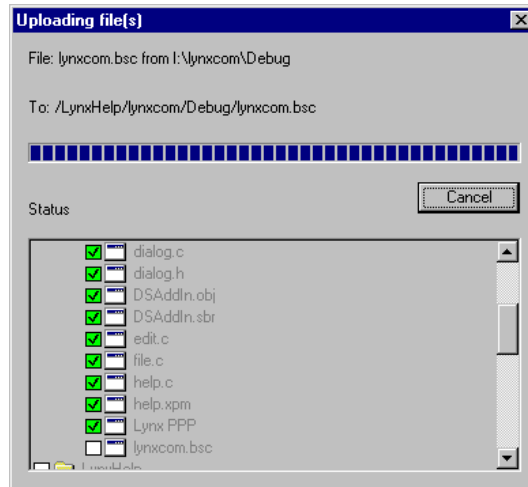


Figure 9-6: Transfer Dialog Box

You can stop the transfer process by pressing the **Cancel** button. When the transfer procedure is completed, the name of the button changes to **Close** and the **Status** window becomes available. You can browse the tree in the **Status** window to verify the results of the transfer. To display the results of processing, the transfer dialog box uses icons (shown in the following table) in the **Status** tree:

Table 9-3: FTP Status Tree Icons





Icon	Description
	In Progress
	Folder Copied Partially

Table 9-3: FTP Status Tree Icons

Icon	Description
	Not Copied (Error or Skipped)
	Copied Successfully



# *VisualLynux Target Administration*

---

## Overview

The purpose of the VisualLynux Target Administration tool is to provide support in defining, configuring, and troubleshooting targets.

A target is a computer or device for which the user develops LynxOS or BlueCat Linux applications using VisualLynux, and which is connected to the developer computer through a network or serial line.

VisualLynux uses TCP/IP protocols as the basis for communication with targets and FTP protocols as a primary tool for file transfers between the local computer and the target.

It uses the target definition to establish a connection while running VisualLynux tools such as Telnet, LynuxWorks FTP, or the Cross Process Viewer. In addition, VisualLynux uses target definitions while running the debugger and transferring project files and executables to and from the target.

You can define any number of targets with unique readable names. Each target definition is composed of configuration data, including the target computer network name, communication mode, FTP/Telnet configuration, project root directories, and communication parameters to establish connections with the debugger and process viewer servers. The user can specify different target definitions for the same target computer or device, which enables a different setup from the target properties for another project.

The Target Administration tool provides the default target from a set of defined targets. You can change default behavior and specify that VisualLynux enables target selection whenever the tool is invoked. The user can invoke VisualLynux without a target and specify the correct target later on.

## Target Configuration

To communicate properly with a computer running VisualLynux, target configuration should be in accordance with the following requirements:

- It must support TCP/IP.
- It must run FTP service.
- It must run the remote shell daemon, which has to be configured to allow the user to run programs on the target (that is, there must be proper user rights).
- The FTP service should allow the user full access to directories under the home FTP directory (including creating and removing directories/files).
- The home FTP directory for the user should be the same as the root directory of the target local file system or its name.
- It should run the Telnet service.
- It should allow for remote debugging. The target should have the GDBSERVER program and allow it to be run remotely.
- It should allow the Cross Process Viewer to run. It should have the PCDSRVR program and allow the user to run it remotely.

Furthermore, VisualLynux Target Administration assumes that all LynxOS or BlueCat Linux configuration tasks have been performed.

## Invoking Target Administration

To invoke the Target Administration Wizard, press the **Target Administration** button



in the VisualLynux toolbar. The Target Administration dialog box appears with the following options:

- [Creating a New Target](#) definition
- [Editing Target Properties](#)
- [Renaming a Target](#) definition
- [Removing a Target](#) definition
- [Setting a Default Target](#)
- [Specifying Target Selection Mode](#)
- [Running Commands on the Target](#)

## Target Administration Dialog Box

The Target Administration dialog box allows you to define new targets, modify or remove existing target definitions, or set default targets. If no target is defined, the Target Administration dialog box invokes the *Target Wizard* to automatically allow you to define the first target.

The Target Administration dialog box displays existing target definitions in a tree view (see next figure). The name of the tree root is formatted using the following syntax:

```
VisualLinux Targets on computer_name
```

where *computer\_name* is the name of the local host machine as specified in the TCP/IP configuration.

The default target is highlighted in bold. You can navigate the target tree using the **Arrow** keys or the mouse. The target currently selected is the target on which the next operation is performed. To perform a particular operation on a target, use the dialog buttons or press the right mouse button and select a command from the popup menu.

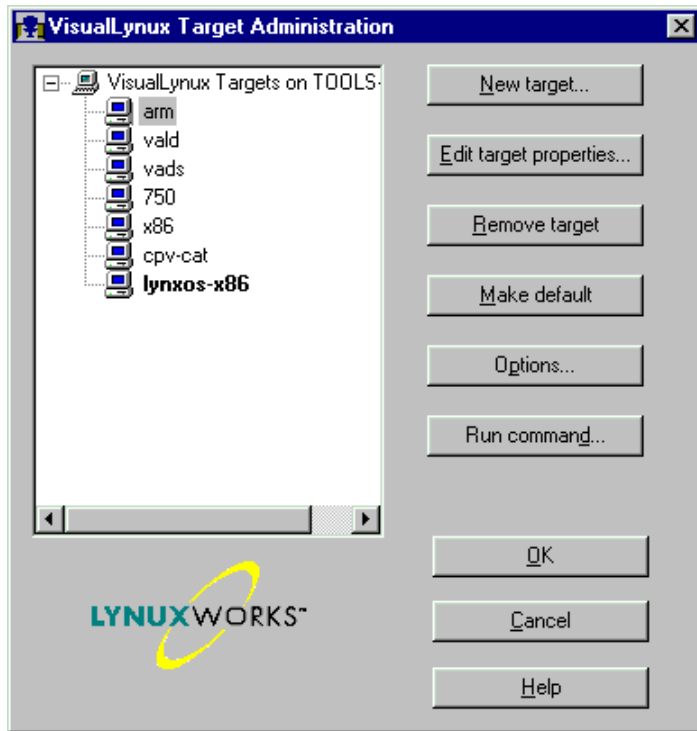


Figure 10-1: Target Administration Dialog Box



Table 10-1: Target Administration Dialog Box

Control Item	Definition
<b>New target</b>	This button invokes the <a href="#">Target Wizard</a> , which helps to define a new target, and define a network connection with the target, to specify and check Telnet/FTP configurations, and to define communication parameters for the GDBSERVER and PCDSRVR programs running on the target. After successful definition of the new target, its name appears in the tree view. If the new target is the only target defined, it automatically becomes the default.
<b>Edit target properties</b>	This button is enabled only if a target is selected in the tree view. It displays the <a href="#">Editing Target Properties</a> dialog box, which facilitates changing the properties of the highlighted target.
<b>Remove target</b>	Removes the target selected in the tree view. Because target removal cannot be undone, you are alerted before the operation is performed. The button is enabled only if a target is selected.
<b>Make default</b>	This button sets the current target as the default. The name of the target is immediately highlighted in bold. This button is enabled only if a target that is not already the default is selected.
<b>Options</b>	This button invokes the <a href="#">Target Selection</a> dialog box, which is used to set a target selection mode. This specifies whether the default target is used for VisualLynux tools, or the user selects a target every time a VisualLynux tool is invoked.
<b>Run command</b>	This button invokes the <a href="#">Running Commands on the Target</a> dialog box, which allows you to run a command on the selected target. This button is enabled only if a non-root item is selected in the target tree.
<b>OK</b>	This button closes the dialog box and saves changes.
<b>Cancel</b>	This button closes the dialog box and rejects changes.
<b>Help</b>	This button invokes the help system.

## Target Wizard

VisualLinux invokes the Target Wizard when you press the **New target** button in the [Target Administration Dialog Box](#). The Target Wizard displays a sequence of pages that help to define target properties.

Target Administration also invokes the Target Wizard automatically if it needs a target definition, and no target is defined yet. For example, if VisualLinux is launching a tool and needs to connect to the target but none has been defined, the Target Wizard is invoked.

The Target Wizard includes the following steps to process target definition. You can navigate the steps using the **Next** and **Back** buttons.

1. [General Information Step](#)
2. [Target Identification Step](#)
3. [Network Connection Step](#)
4. [Authentication Step](#)
5. [Utilities Step](#)
6. [Projects Directory Step](#)
7. [Debugging Step](#)
8. [Process Viewer Step](#)

### General Information

The General information step provides general information about the Target Wizard.

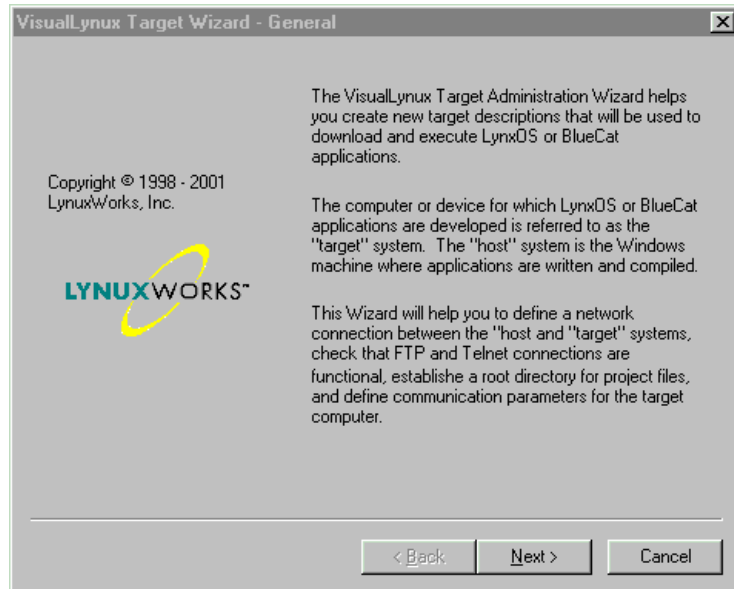


Figure 10-2: General Information Step

## Target Identification Step

Using the Target Identification step, you provide the target name and select the mode used to connect with the target (see next figure).

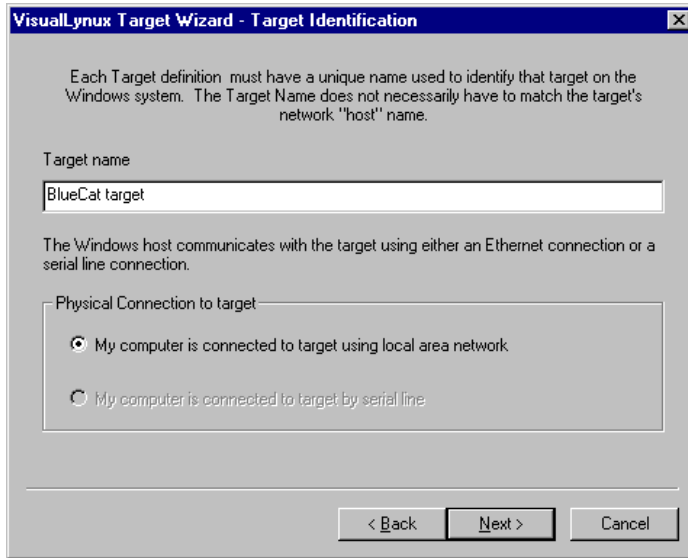


Figure 10-3: Target Identification Step

The following table lists the controls for the target identification step:

Table 10-2: Target Identification Step

Control Item	Description
Target name	This field has to be filled with a unique target name. The program checks to see if the name is unique. <b>Target name</b> allows you to specify the one physical target as multiple logical targets with different properties.
Connection mode	This radio button specifies the mode used by Target Administration to communicate with a particular target. The current implementation supports only connection through a local area network.

## Network Connection Step

The Network Connection step allows you to specify a target network name and check the TCP/IP connection with the target.

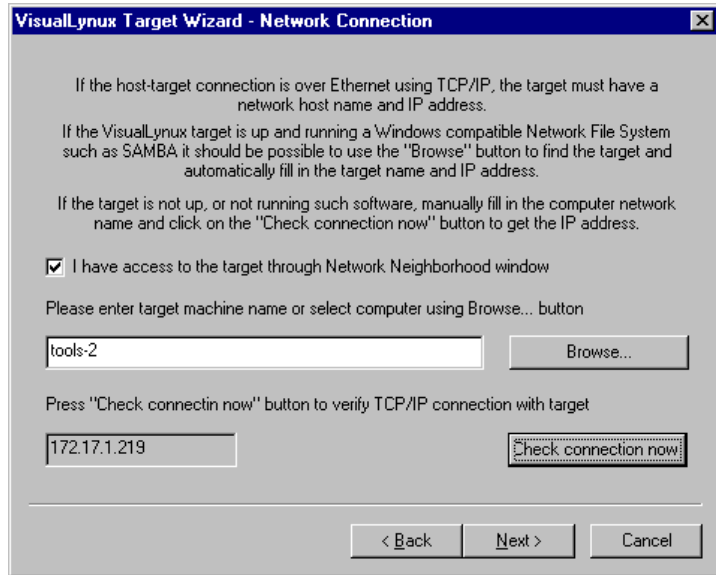


Figure 10-4: Network Connection Step

The following table lists the controls in the network connection step:

Table 10-3: Network Connection Step

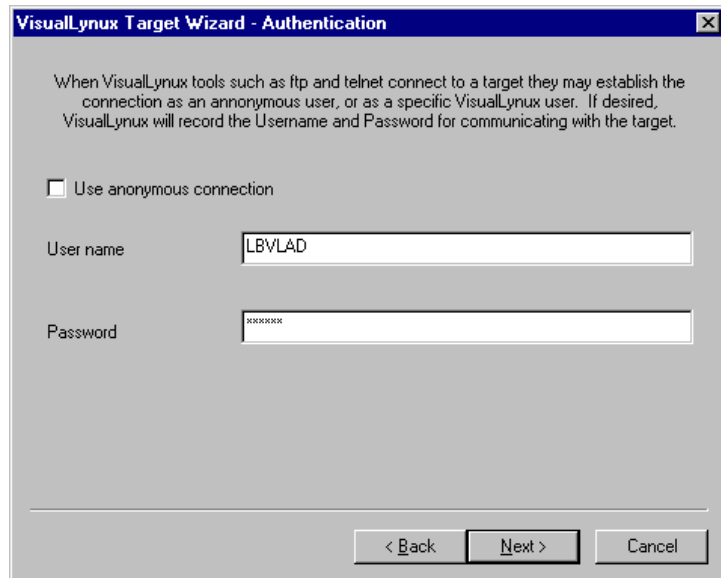
Control Item	Description
<b>I have access to the target through the Network Neighborhood window</b>	Check this box if the target computer is visible in the <b>Network Neighborhood</b> window. This means that the target is running Network File System support compatible with the Windows Network browser and you can select the target using the <b>Browse</b> button. If you uncheck the box, the <b>Browse</b> button is disabled and you must enter the target name manually.

**Table 10-3: Network Connection Step (Continued)**

Control Item	Description
<b>Target computer name</b>	This field displays the target name as defined in the TCP/IP configuration of your network. The program fills this field automatically if you have selected the target computer using the <b>Browse</b> button. Otherwise you must enter a valid computer name in this field manually. You can also enter a valid IP address in this field.
<b>Check connection now</b>	Verifies the TCP/IP connection with the target immediately. The program verifies connection with the target when you press <b>Next</b> as well. If connection with the target cannot be established, the program displays an error message and offers to verify the computer name entered in the <b>Target computer name</b> field.

## Authentication Step

To maintain proper connection with the services running on the target, VisualLinux may need authentication data (user name and password). You should enter this data during the Authentication Step.



**Figure 10-5: Authentication Step**

The following table lists the controls in the target authentication step:

**Table 10-4: Target Authentication Step**

Control Item	Description
<b>Use anonymous connection</b>	This box is unchecked by default. If checked, VisualLynux does not use the user name and password while connecting to services on the target computer. For example, while connecting to the target FTP service, VisualLynux uses <code>Anonymous</code> as the user name and a default password (usually the user's e-mail address). Be sure that the services on the target are configured to allow anonymous access.

**Table 10-4: Target Authentication Step (Continued)**

Control Item	Description
<b>User name</b>	If the <b>Use anonymous connection</b> button is not checked the user must fill this field with a valid user name. This name is used by Target Administration while connecting to services on the target. LynxOS or BlueCat Linux running on the target and all other services should be configured to allow the user to access required services (FTP, Telnet). By default the program fills this field with the current user name provided by Windows.
<b>Password</b>	If you leave this field empty, Target Administration uses an empty password while connecting to the target. In this case, the target must allow user access without a password. If this is not the case, you must enter a valid password.

## Utilities Step

The Utilities step is used to specify connection parameters with services running on the target. VisualLinux uses Telnet and FTP as the main communication protocols with the target. Usually you do not need to change the data displayed on this page by default.



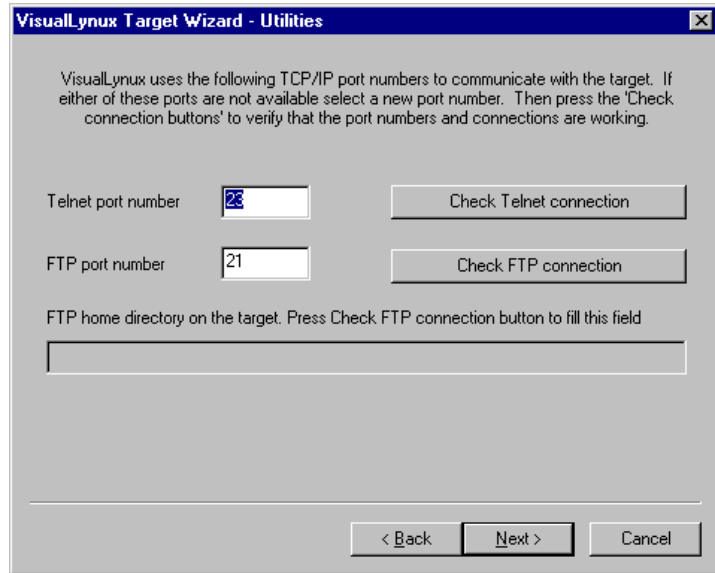


Figure 10-6: Utilities Step

The following table shows the controls in the utilities step:

Table 10-5: Utilities Step

Control Item	Description
Telnet port number	You can change the default Telnet port number if the Telnet service on the target uses a non-standard port number. By default the program uses a standard Telnet port number (23).
FTP port number	You can change the default FTP port number if the FTP service on the target uses a non-standard port number. By default the program uses a standard FTP port number (21).
Check Telnet connection	Press this button to check the Telnet connection to the target immediately. The program establishes a connection with the Telnet service on the target using the <a href="#">Network Connection Step</a> and the Telnet port number. In case of an error, a message box explains the error.

**Table 10-5: Utilities Step (Continued)**

Control Item	Description
<b>Check FTP connection</b>	Press this button to establish an FTP connection with the target using the <a href="#">Network Connection Step</a> and FTP port number. If a connection is established successfully, the program fills the <b>FTP home directory</b> field with the full name of the user's home directory. If any error occurs, a message box explaining the error appears. The FTP connection is also checked when you change the page using <b>Next</b> .
<b>FTP home directory</b>	Contains the full name of the user's home directory. This field is filled by the program as a result of a successful FTP connection checking.

## Projects Directory Step

The Projects Directory step is used to create or select a directory on the target that is used by VisualLynux for files and executables while debugging/running applications. VisualLynux creates a directory for each project under the root projects directory and places files and executables in this directory.

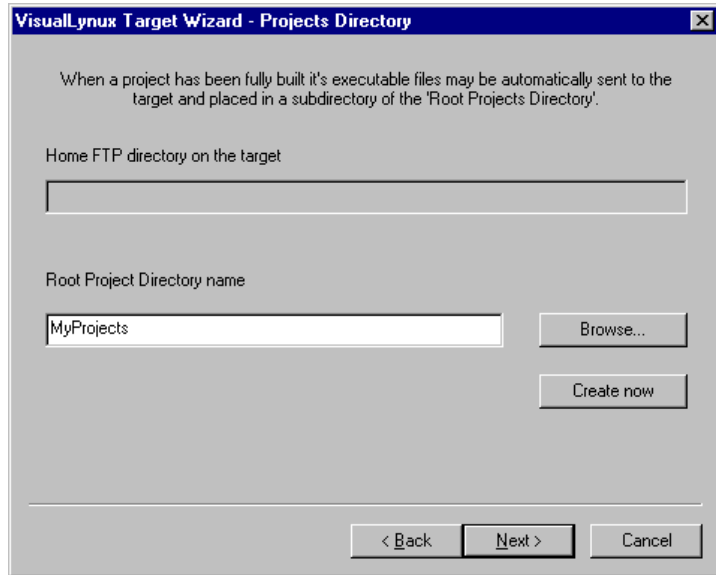


Figure 10-7: Projects Directory Step

The following table lists the projects directory step controls:

**Table 10-6: Projects Directory Step**

Control Item	Description
<b>Home FTP directory on target</b>	This field is filled by the program as a result of checking the FTP connection to the target. It is not accessible and is displayed only for your information.
<b>Root Project Directory name</b>	Enter the root project directory name in this field. If the directory name is entered without forward (/) or back slash (\), the program considers the entered name as a subdirectory of the home FTP directory. If the entered name starts with a slash, the program considers it as a full directory name on the target. Use <b>Browse</b> to select the directory that serves as a root project directory on the target. If you have entered the name of a nonexistent directory, you can create it by pressing the <b>Create now</b> button. The program also checks if the directory exists when you press the <b>Next</b> button and it prompts you to create a new directory immediately.
<b>Browse</b>	This button displays the <b>Browse for FTP folders</b> dialog box. You can select the existing folder on the target as the root project directory.
<b>Create now</b>	Press this button to force the program to create a new FTP directory on the target. The new directory has a name specified in the <b>Root Project Directory name</b> field. Again, the program considers the directory name in this field as a relative of the home directory if it doesn't start with a slash, and as a full directory name if it does.

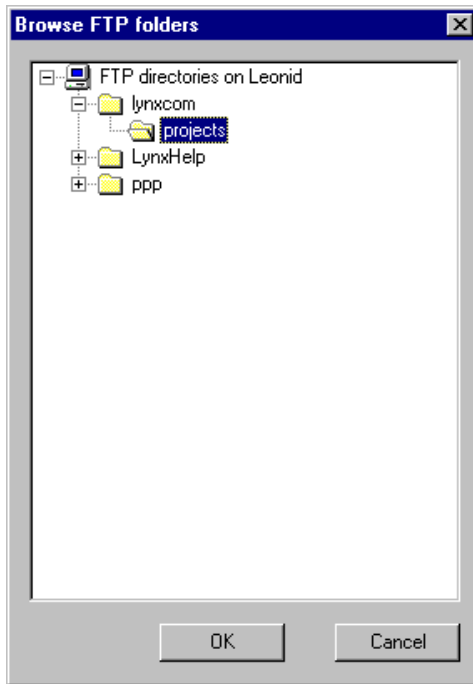


Figure 10-8: Browse for FTP Folders Dialog Box

## Debugging Step

The Debugging step (see next figure) is used to specify the location of the GDBSERVER program and the default port number with which it communicates.

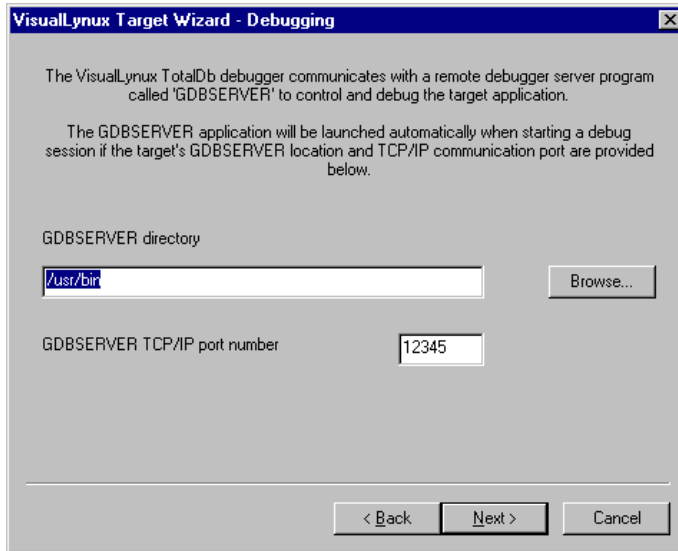


Figure 10-9: Debugging Step

The following table lists the debugging step controls:

Table 10-7: Debugging Step


Control Item	Description
<p>GDBSERVER directory</p> 	<p>Enter the full path to the directory in which the GDBSERVER program is placed (if it differs from default <code>/usr/bin</code> directory). The path must be specified using the local target file system directory tree.</p> <p><b>CAUTION!</b> The FTP directory tree and filesystem directory tree on the target should be the same because the full directory name is used to invoke GDBSERVER on the target.</p>

Table 10-7: Debugging Step (Continued)

Control Item	Description
Browse	Invokes the <b>Browse FTP folders</b> dialog box.
GDBSERVER TCP/IP port number	You should specify the TCP/IP port number to communicate with the GDBSERVER. If you leave this field empty, the program signals that remote debugging is not available.

## Process Viewer Step

The Process Viewer step allows you to specify the directory on the target where the Process Viewer server (PCDSRVR) is placed and the TCP/IP port number VisualLynux uses to communicate with the server.

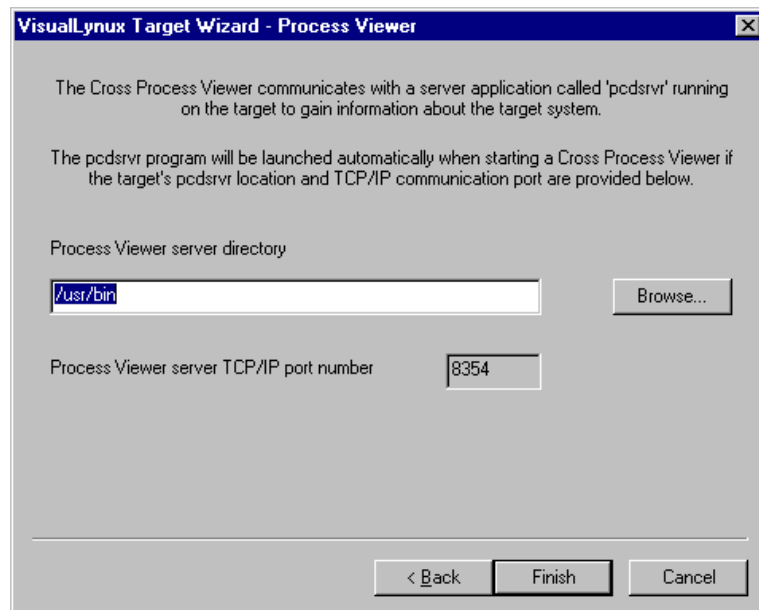



Figure 10-10: Process Viewer Step

**NOTE:** The current version does not allow changing the default TCP/IP port number used by the PCDSRVR and, therefore, the appropriate field is grayed out.

---

The following table lists the process viewer step controls:

**Table 10-8: Process Viewer Step**

Control Item	Description
Process Viewer server directory	Change the name of the directory in which PCDSRVR is placed if it differs from the default <code>/usr/bin</code> directory name. The directory name has to be specified using the local target file system directory tree.
Browse  	Invokes the <b>Browse for FTP folders</b> dialog box. Select the existing directory where PCDSRVR resides.  <b>CAUTION!</b> The FTP directory tree and file system directory tree on the target should be the same because the full directory name is used to invoke PCDSRVR on the target.
Process Viewer server TCP/IP port number	Because the current implementation does not allow changing the default TCP/IP port used by PCDSRVR, this field is grayed out.

---

## Target Administration Tasks

VisualLinux provides capabilities for performing the following Target Administration tasks:

- [Creating a New Target](#)
- [Editing Target Properties](#)
- [Renaming a Target](#)
- [Removing a Target](#)
- [Setting a Default Target](#)



- [Specifying Target Selection Mode](#)
- [Running Commands on the Target](#)

## Creating a New Target

To create a new target:

1. Invoke the [Target Administration Dialog Box](#).
2. Press the **New target** button.

Target Administration invokes the [Target Wizard](#), which guides you through defining new target properties.

## Editing Target Properties

To edit target properties, do the following:

1. Invoke the **Target Administration** dialog box.
2. Select the target from the target tree.
3. Press the **Edit target properties** button, *or*

Press the right mouse button and select the **Edit target** command from the popup menu.

This invokes a tabbed **Target Properties** dialog box. Select the page or pages you want to edit and follow the rules described in the [Target Wizard](#) section.

- [Target Identification Step](#)
- [Network Connection Step](#)
- [Authentication Step](#)
- [Utilities Step](#)
- [Projects Directory Step](#)
- [Debugging Step](#)
- [Process Viewer Step](#)

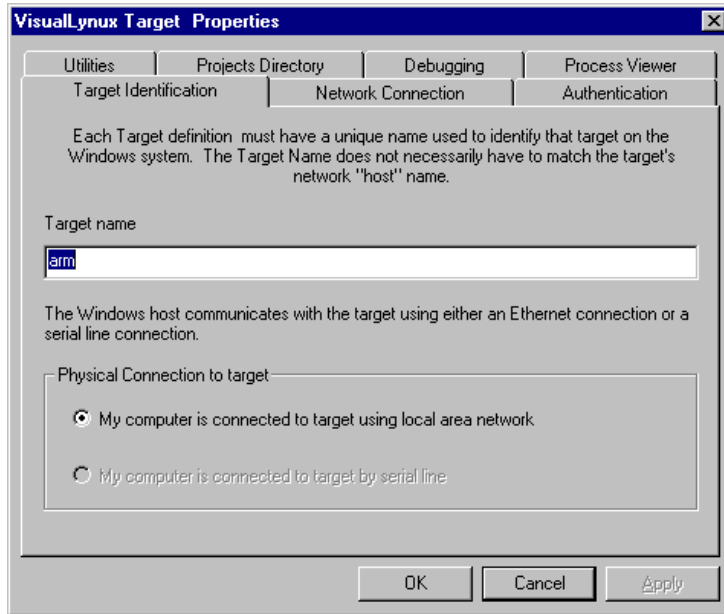


Figure 10-11: Target Properties Dialog Box

## Removing a Target

To remove a target do the following:

1. Invoke the [Target Administration Dialog Box](#).
2. Select the target you wish to remove from the target tree.
3. Press the **Remove target** button, *or*

Press the right mouse button and select the **Remove** command from the popup menu.

Target Administration asks if you really want to remove the target because this operation cannot be undone.

## Setting a Default Target

To set the default target, do the following:

1. Invoke the [Target Administration Dialog Box](#).
2. Select the target you want to set as the default.
3. Press the **Make default** button, *or*

Press the right mouse button and select the **Make default** command from the popup menu.

If you select a target that is already set as default, the **Make default** button is grayed out.

## Specifying Target Selection Mode

To specify the target selection mode used to connect to the target while VisualLynux is being launched, do the following:

1. Invoke the [Target Administration Dialog Box](#).
2. Press the **Options** button.

This invokes the **Target selection** dialog box (see following dialog box). Select the target selection mode you wish to use when VisualLynux invokes tools.

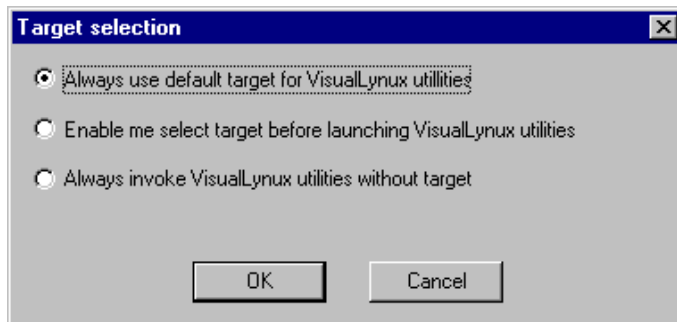


Figure 10-12: Target Selection Dialog Box

The following controls appear in the target selection dialog box:

**Table 10-9: Target Selection**

Control Item	Description
<b>Always use default target for VisualLynux tools</b>	If this option is selected, VisualLynux automatically tries to connect to the default target while a tool is starting. This is the default mode.
<b>Enable me to select target before launching VisualLynux tools</b>	If this option is selected, VisualLynux always brings up the dialog box that allows you to select the target to which a tool has to be connected.
<b>Always invoke VisualLynux utilities without target</b>	If this option is selected VisualLynux does not try to connect automatically to any target. You should connect to the target after a tool has been launched using the tool's commands.

## Renaming a Target

To rename target do the following:

1. Invoke the [Target Administration Dialog Box](#).
2. Select the target you want to rename from the target tree.
3. Press the right mouse button and select the **Rename** command from the popup menu, *or*

Click once more on selected item label.

Target Administration changes to label edit mode. After editing, press **Enter** or select another item in the tree. Target Administration verifies that the new name you have entered in the edit box is unique, and rejects changes if it is not unique.

## Running Commands on the Target

To run a command on a target, do the following:

1. Invoke the [Target Administration Dialog Box](#).
2. Select the target on which you are going to run a command.

3. Press the **Run command** button.

Target Administration invokes the [Running Commands on the Target](#) dialog box, which allows you to run commands using the `Remote` shell (see next figure).

## Run Command Dialog Box

The **Run command** dialog box allows you to run commands on the target using the `Remote` shell. This window also pops up if Target Administration runs servers on the target to support VisualLynux utilities (for example, Cross Process Viewer), and an error occurs.

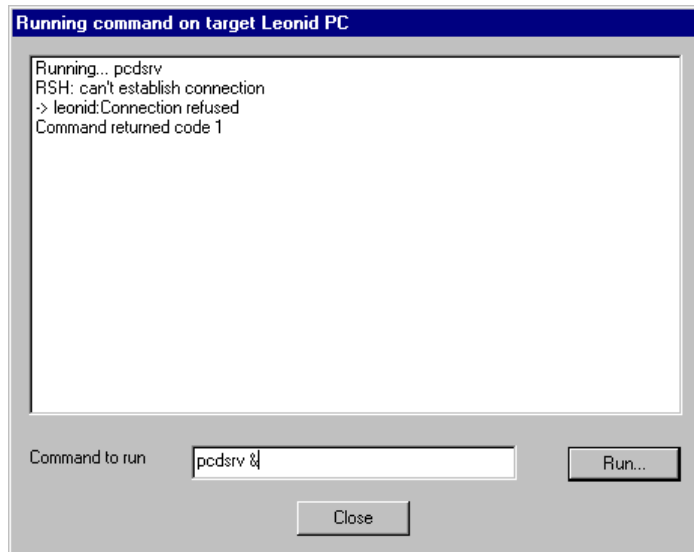


Figure 10-13: Run Command Dialog Box

The following controls appear in the **Run Command** dialog box:

**Table 10-10: Run Command Dialog Box**

Control Item	Description
<b>Command to run</b>	Enter the command you want to run on the target. If the dialog box is invoked (e.g., when starting Cross Process Viewer on the selected target) this field is filled in by the program.
<b>Run</b>	Press this button to run the command on the target. The button is enabled only if the previous field is filled in. The program disables this button after command execution has begun.
<b>Close</b>	When the command is being executed, this button is called <b>Cancel</b> . Pressing it interrupts current command execution. After the command has been executed, the button name changes to <b>Close</b> . This button closes the <b>Run command</b> dialog box.
<b>Command output</b>	The command output window is placed at the top of the dialog box. This window shows output from the Remote Shell utility. The first line of output always displays "Running <i>command_text</i> " and the last line shows the return code from the RSH utility. You can use this line to check the success of command execution.

# *LynuxWorks Cross Process Viewer*

---

## Overview

This LynuxWorks Cross Process Viewer (CPV) is a part of the VisualLynux Integrated Development Environment (IDE). CPV runs on Windows cross development hosts and communicates with LynxOS or BlueCat Linux target machines to display all processes and threads running on the target.

The LynuxWorks CPV program (`cpv.exe`) monitors a target system for information on load average, memory usage, and processes and threads statistics.

Listed below are the main CPV functionalities:

- [The CPV Window](#)
- [File Menu](#)
- [View Menu](#)
- [Options Menu](#)
- [Help Menu](#)
- [Cross Process Viewer Toolbar Buttons](#)

---

**NOTE:** While CPV can be called from VisualLynux, it can also be executed separately to monitor target systems with LynxOS or BlueCat Linux after the development cycle. This use of CPV does not require VisualLynux. To run CPV from an MS-DOS window, enter `cpv -t targetname` where `targetname` is a name or IP address of the target. You can also enter just `cpv` and specify a target or IP address using the **File->New** menu item.

---

## The CPV Window

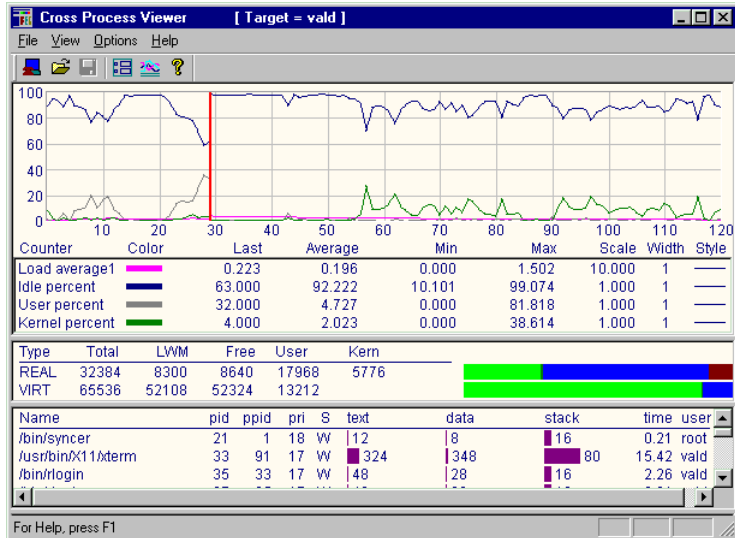


Figure 11-1: CPV Window

The CPV Window displays four panes:

- Performance pane
- Memory Usage pane
- Processes pane
- System Tasks pane (for LynxOS only)

The information in all four panes reflects the state of the current target with which CPV has a session with and is updated after periodic transactions between CPV and the target. The duration of the interval for periodic transactions is set by the user. The default is 1 second. The possible range is between 0.5 seconds and 30 seconds.



## CPV Window Panes

The following table describes the panes in the CPV window:

**Table 11-1: CPV Panes**

Pane	Display
Performance pane	The modes of display and display items are listed below:
	<p><i>Text</i></p> <ul style="list-style-type: none"> <li>• Load average values for 1-, 5-, and 15-minute intervals</li> <li>• CPU state (<code>idle/user/kernel</code>) in percentages</li> <li>• Number of processes selected for display, including the number of sleeping processes and those on the CPU</li> </ul>
	<p><i>Chart</i></p> <p>A chart of performance counters and the list of selected performance counters</p>
	<p><i>Histogram</i></p> <p>A histogram of performance counters and the list of selected performance counters - Performance counters include:</p> <ul style="list-style-type: none"> <li>• Load average1</li> <li>• Load average5</li> <li>• Load average15</li> <li>• Idle percent</li> <li>• User percent</li> <li>• Kernel percent</li> <li>• LWM real (LynxOS only)</li> <li>• Free real</li> <li>• User real (Used real for BlueCat Linux)</li> <li>• Kernel real (LynxOS only)</li> <li>• Buffers real (BlueCat Linux only)</li> <li>• Cached real (BlueCat Linux only)</li> <li>• LWM virtual (LynxOS only)</li> <li>• Free virtual</li> <li>• User virtual (Used virtual for BlueCat Linux)</li> <li>• Process number (LynxOS only)</li> <li>• Waiting processes (LynxOS only)</li> <li>• Paged in (BlueCat Linux only)</li> <li>• Paged out (BlueCat Linux only)</li> <li>• Swapped in (BlueCat Linux only)</li> <li>• Swapped out (BlueCat Linux only)</li> <li>• Interrupts (BlueCat Linux only)</li> <li>• Context switches (BlueCat Linux only)</li> </ul>

**Table 11-1: CPV Panes (Continued)**

Pane	Display
<b>Memory Usage pane</b>	<p data-bbox="486 295 1099 349">Displays the text and graphical bars representing memory usage information in the target system:</p> <p data-bbox="486 383 1089 497"><i>LynxOS:</i> Displays values for the total amount of real and virtual (if any) memory in the target system, low water mark (LWM), free and user real and virtual memory, and real kernel memory.</p> <p data-bbox="486 530 1089 673"><i>BlueCat Linux:</i> Displays values for the total amount of real and virtual (if any) memory in the target system, free memory, memory used for system buffers, cached memory, shared memory, free and used swapped memory.</p>

Table 11-1: CPV Panes (Continued)

Pane	Display
Processes pane	<p>Displays the following information for each selected process and each additional thread:</p> <ul style="list-style-type: none"> <li>• name</li> <li>• pid</li> <li>• ppid</li> <li>• pgrp</li> <li>• sid</li> <li>• pri</li> <li>• S (state)</li> <li>• tid</li> <li>• total time</li> <li>• system time</li> <li>• user time</li> <li>• last percentage</li> <li>• total percentage</li> <li>• user name</li> <li>• semaphore on wait</li> <li>• number of files</li> <li>• flags</li> <li>• device name</li> <li>• memory information (graphical bars and text values)</li> </ul>
System Tasks pane (LynxOS only)	<p>Displays the following information:</p> <ul style="list-style-type: none"> <li>• name</li> <li>• tid</li> <li>• pri</li> <li>• S (state)</li> <li>• stack</li> <li>• system time</li> <li>• last percentage</li> <li>• total percentage</li> <li>• semaphore on wait</li> <li>• flags</li> </ul>

---

## File Menu

The **File** menu initiates the following actions:

- Initiates a new session with a target running LynxOS or BlueCat Linux.
- Makes a snapshot of the current session and saves it in a file.

- Opens a file containing a snapshot and displays it in the panes.
- Specifies the most recent sessions.
- Specifies the most recent files (with snapshots).
- Exits from the CPV.

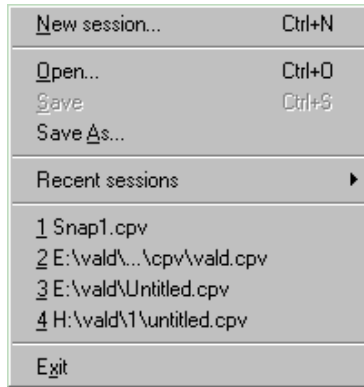


Figure 11-2: CPV File Menu

The following table lists the CPV File menu controls:

Table 11-2: CPV File Menu

Command	Description
New session	Displays a dialog box to specify a target machine name or IP address.
Open	Displays a <b>File Selection</b> dialog box to select a Cross Process Viewer snapshot.
Save	Makes a snapshot and saves it in the file specified in the <b>Save As</b> dialog box.
Save As	Displays a dialog box to specify a file name for the snapshot.
Recent sessions	Displays a cascade menu to specify a target for a new session.
File list	Lists the latest snapshots. You can select one of the latest snapshots using this menu item instead of <b>Open</b> .
Exit	Terminates the Cross Process Viewer session.

## View Menu

The **View** menu (see figure below) controls the visibility of the toolbar, the status bar, and the four panes.

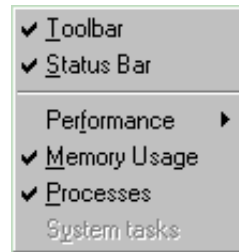


Figure 11-3: CPV View Menu

The following table shows the CPV **View** menu controls:

Table 11-3: CPV View Menu Controls

Control Item	Description
<b>Performance</b>	Displays a cascade menu that allows you to select one of three modes to represent the target system parameters in the <b>Performance</b> pane.
	<i>Text</i> Displays system load averages for the last 1, 5, and 15 minutes; percent of time the system has spent in idle, user, and kernel states during the last quantum; the number of running or sleeping processes and threads.
	<i>Chart</i> Displays target system parameters (counters) as a dynamic chart. You can add or remove counters and modify their look.
	<i>Histogram</i> Displays target system parameters (counters) as a dynamic histogram.
<b>Memory Usage</b>	Toggles the visibility of the <b>Memory Usage</b> pane.

Table 11-3: CPV View Menu Controls(Continued)

Control Item	Description
Processes	Toggles the visibility of the <b>Processes</b> pane.
System tasks	Toggles the visibility of the <b>System tasks</b> pane (LynxOS only).

## Options Menu

The **Options** menu displays one of the options dialog boxes shown below:

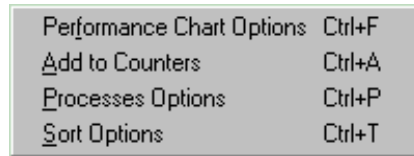


Figure 11-4: CPV Options Menu

The following table describes the CPV **Options** dialog boxes:

Table 11-4: CPV Option Dialog Boxes

Command	Description
Performance Chart Options	Displays <a href="#">Performance Chart Options</a> dialog box.
Add to Counters	Displays <a href="#">Add to Counters</a> dialog box.
Process Options	Displays <a href="#">Processes Options</a> dialog box.
Sort Options	Displays <a href="#">Sort Options</a> dialog box.

### Performance Chart Options

The CPV **Performance Chart Options** item displays the **Performance Options** dialog box for changing **Performance** pane settings

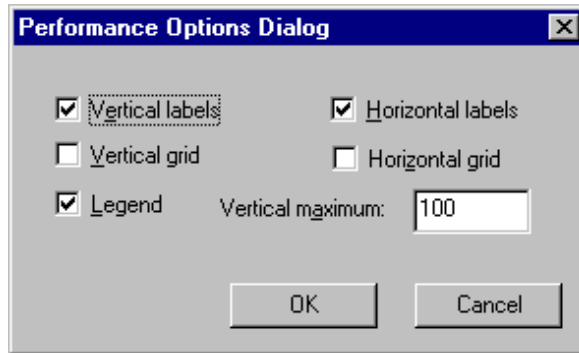


Figure 11-5: Changing Performance Pane Options

The Performance Options dialog box controls are shown below:

Table 11-5: Performance Options Dialog Box

Command	Description
Vertical labels	Displays the vertical scale to represent counter values.
Horizontal labels	Displays the horizontal time scale.
Vertical grid	Displays vertical lines representing time intervals.
Horizontal grid	Displays horizontal lines representing counter values.
Legend	Displays counter information such as type, color, minimal, last and maximal values, scale and line style.
Vertical maximum	Sets the maximum visible counter value. Used for vertical labels representing scaled counter values.

## Add to Counters

The CPV **Add to Counters** dialog box allows you to change Performance pane counter options:

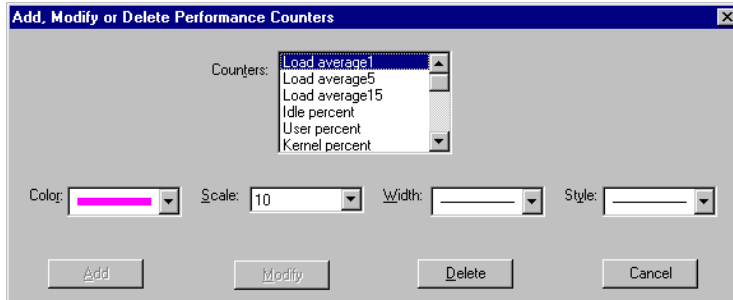


Figure 11-6: Add to Counters Dialog Box

The controls for modifying performance counters are shown below:

Table 11-6: Add, Modify or Delete Performance Counters

Control	Description
Counters	Used to add or delete a counter from the active set, or to select a counter for modifications.
Color	Selects a color for the counter.
Scale	Modifies scale factor for the counter value.
Width	Sets line width for the counter.
Style	Selects line style for the counter.

## Processes Options

The CPV **Processes Options** settings depends on the target operating system. The Cross Process Viewer supports two operating systems on the target machine and, correspondingly, displays two different dialog boxes:

- [LynxOS Processes Options](#)
- [BlueCat Linux Processes Options](#)

## LynxOS Processes Options

The LynxOS CPV **Options** dialog box is shown below:



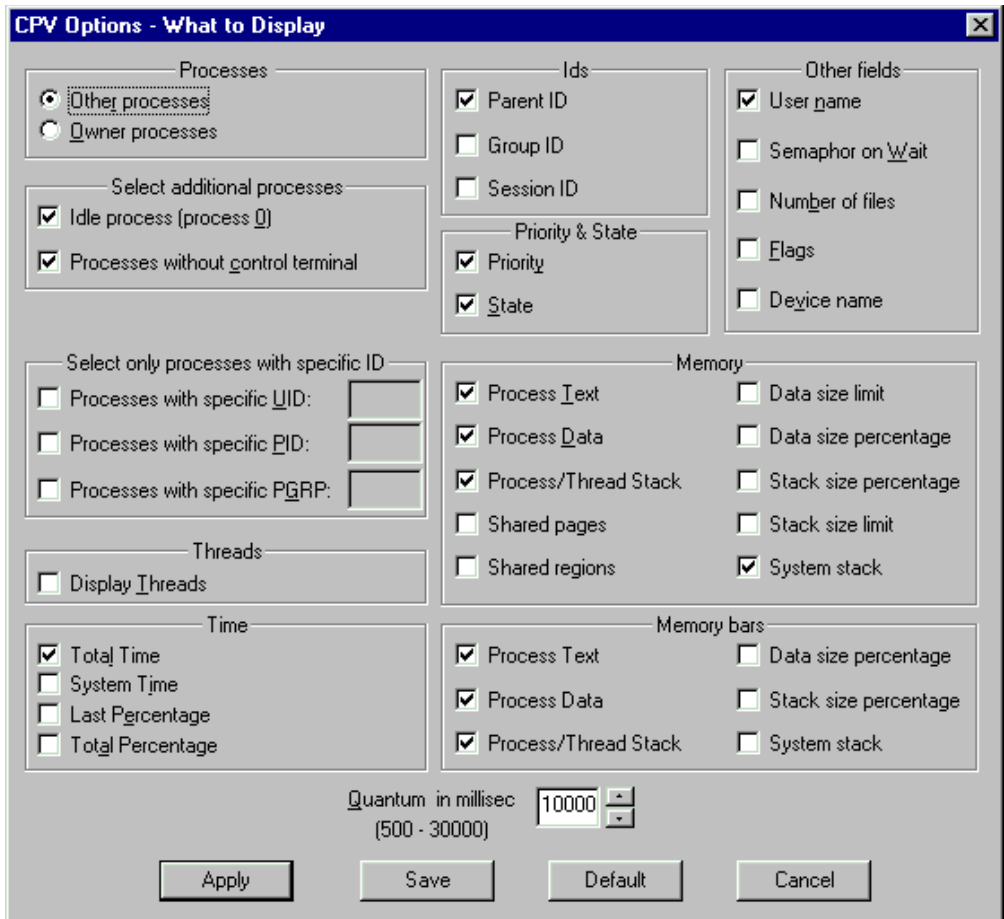


Figure 11-7: LynxOS Processes Options

**NOTE:** Memory bars section (bottom right) toggles settings to display bars for some of the memory parameters.

The controls in the LynxOS CPV Options dialog box are described below:

**Table 11-7: LynxOS Processes Options**

Control	Description
<b>Other processes</b>	Displays information for other processes, not just processes owned by the user.
<b>Owner processes</b>	Displays information only for processes owned by the user.
<b>Idle process (process 0)</b>	Displays information about the idle ( <code>init</code> ) process, idle system time.
<b>Processes without control terminal</b>	Displays information about processes without a control terminal.
<b>Processes with specific UID</b>	Displays only processes owned by a specific user. Check the box and enter the user name.
<b>Processes with specific PID</b>	Displays only processes with a specific process ID.
<b>Processes with specific PGRP</b>	Displays only processes with a specific process group ID.
<b>Display Threads</b>	Displays processes as well as their threads.
<b>Total Time</b>	Includes process total time information.
<b>System Time</b>	Includes process system and user total time information.
<b>Last Percentage</b>	Includes percentage of CPU time for the latest quantum.
<b>Total Percentage</b>	Includes percentage of CPU time for the entire CPU active period.
<b>Parent ID</b>	Includes parent process id information ( <code>ppid</code> ).
<b>Group ID</b>	Includes process group id information ( <code>pgrp</code> ).
<b>Session ID</b>	Includes session id information ( <code>sid</code> ).
<b>Priority</b>	Includes priority information ( <code>pr i</code> ).
<b>State</b>	Includes process state information ( <code>S</code> ).
<b>User name</b>	Includes user name.
<b>Semaphore on Wait</b>	Includes semaphore hexadecimal value.
<b>Number of Files</b>	Includes information about open files.
<b>Flags</b>	Includes processes (and threads) and flags information.

**Table 11-7: LynxOS Processes Options (Continued)**

<b>Control</b>	<b>Description</b>
<b>Device name</b>	Includes the device name if the process owns the control terminal.
<b>Process Text</b>	Displays the size of the text segment in KB.
<b>Process Data</b>	Displays the size of the data and segment in KB.
<b>Process/ Thread Stack</b>	Displays the size of the user stack in KB.
<b>Shared pages</b>	Displays the number of shared pages.
<b>Shared regions</b>	Displays the number of shared regions.
<b>Data size limit</b>	Displays data size limit in KB.
<b>Data size percentage</b>	Displays data size percentage.
<b>Stack size percentage</b>	Displays stack size percentage.
<b>Stack size limit</b>	Displays stack size limit in KB.
<b>System stack</b>	Displays system stack size.
<b>Quantum (in millisec)</b>	Sets the duration of time quantum, that is, the time interval between consecutive snapshots. Varies from 500 milliseconds to 30 seconds.

## BlueCat Linux Processes Options

The BlueCat Linux CPV Options dialog box is shown in the figure below:

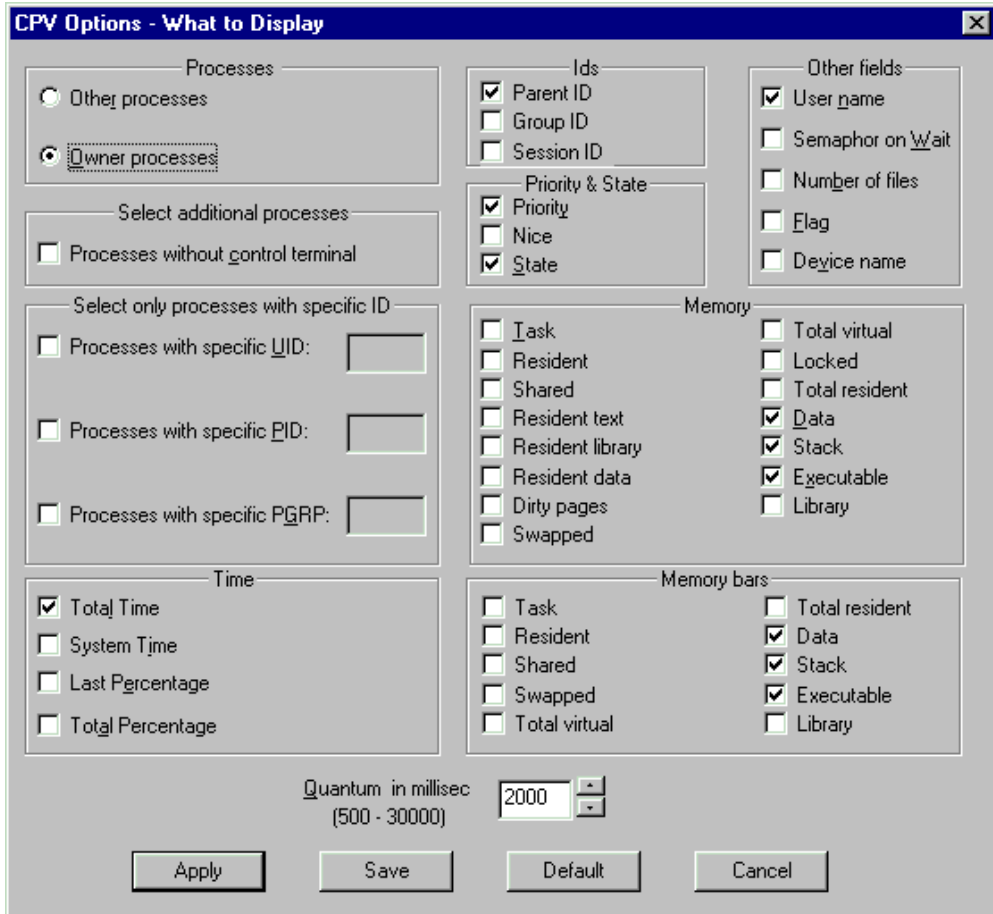


Figure 11-8: BlueCat Linux Processes Options

The controls in the BlueCat Linux **CPV Options** dialog box are described in the following table:

**Table 11-8: BlueCat Linux Processes Options**

<b>Command</b>	<b>Description</b>
<b>Other processes</b>	Displays information for other processes, not just processes owned by the user.
<b>Owner processes</b>	Displays information only for processes owned by the user.
<b>Processes without control terminal</b>	Displays information about processes without a control terminal.
<b>Processes with specific UID</b>	Displays only processes owned by a specific user. Check the box and enter the user name.
<b>Processes with specific PID</b>	Displays only processes with specific process ID.
<b>Processes with specific PGRP</b>	Displays only processes with specific process group ID.
<b>Total Time</b>	Includes process total time information.
<b>System Time</b>	Includes process system and user total time information.
<b>Last Percentage</b>	Includes percentage of CPU time for the latest quant.
<b>Total Percentage</b>	Includes percentage of CPU time for the entire CPU active period.
<b>Parent ID</b>	Includes parent process ID information ( <code>ppid</code> ).
<b>Group ID</b>	Includes process group ID information ( <code>pgrp</code> ).
<b>Session ID</b>	Includes session ID information ( <code>sid</code> ).
<b>Priority</b>	Includes priority information ( <code>pri</code> ).
<b>Nice</b>	Includes nice information ( <code>nice</code> ).
<b>State</b>	Includes process state information ( <code>S</code> ).
<b>User name</b>	Includes user name.
<b>Semaphore on Wait</b>	Includes semaphore hexadecimal value.
<b>Number of files</b>	Includes information about open files.
<b>Flags</b>	Includes process flags information.
<b>Device name</b>	Includes the device name if the process owns the control terminal.

**Table 11-8: BlueCat Linux Processes Options (Continued)**

Command	Description
<b>Task</b>	Displays task size.
<b>Resident</b>	Displays resident size of the task.
<b>Shared</b>	Displays size of shared memory used by task.
<b>Resident text</b>	Displays resident size of the text segment.
<b>Resident library</b>	Displays resident size of library code used by task.
<b>Resident data</b>	Displays resident size of the data segment.
<b>Dirty pages</b>	Displays size of dirty pages.
<b>Swapped</b>	Displays size of the swapped portion of the task.
<b>Total virtual</b>	Displays total virtual size of the task.
<b>Locked</b>	Displays size of locked memory.
<b>Total resident</b>	Displays total resident size of the task (including libraries).
<b>Data</b>	Displays size of the data segment.
<b>Stack</b>	Displays size of the stack.
<b>Executable</b>	Displays size of the executable.
<b>Library</b>	Displays size of memory used by libraries.
<b>Quantum (in millisec)</b>	Sets the duration of time quantum, that is, the time interval between consecutive snapshots. Varies from 500 milliseconds to 30 seconds.

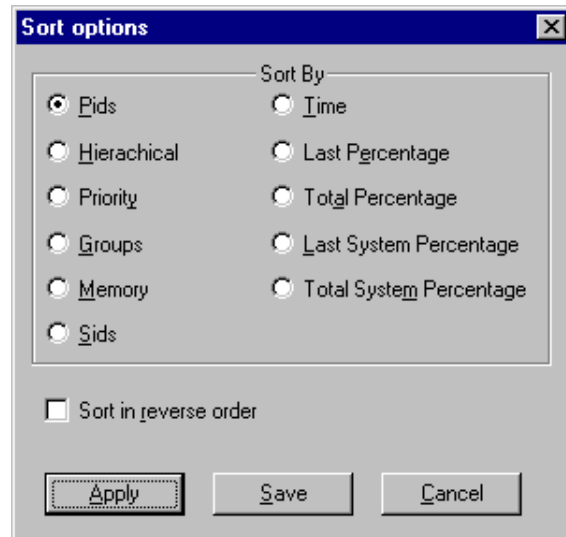
---

**NOTE:** **Memory bars** section (bottom right) toggles settings to display bars for some of the memory parameters.

---

## Sort Options

The CPV **Sort options** dialog box is shown below:



**Figure 11-9: Sort Options Dialog Box**

The **Sort options** controls are described in the table below:

**Table 11-9: CPV Sort Options**

Command	Description
<b>Pids</b>	Sorts by process IDs.
<b>Hierarchical</b>	Sorts processes in the same order as they are represented in the kernel hierarchical tree, that is, first displays a child, then a brother.
<b>Priority</b>	Sorts by process priority.
<b>Groups</b>	Sorts by process group IDs.
<b>Memory</b>	Sorts by total memory the process occupies, that is, text segment, data segment, and stack.
<b>Sids</b>	Sorts by session IDs.
<b>Time</b>	Sorts by time elapsed.
<b>Last Percentage</b>	Sorts by the sum of user and system percentage of CPU time for the last time quantum.

**Table 11-9: CPV Sort Options (Continued)**

Command	Description
<b>Total Percentage</b>	Sorts by the sum of user and system percentage of CPU time for the whole time interval that Cross Process Viewer has been active.
<b>Last System Percentage</b>	Sorts by system percentage of CPU time for the last time quantum.
<b>Total System Percentage</b>	Sorts by system percentage of CPU time for the whole time interval that Cross Process Viewer has been active.
<b>Sort in reverse order</b>	Sorts by options above, but in reverse order.

## Help Menu

The **Help** menu displays the following dialog boxes:

- **Help Topics**
- **About CPV**

## Cross Process Viewer Toolbar Buttons

The figure below shows the CPV toolbar. Buttons are described in the following table.









**Figure 11-10: CPV Toolbar**



The toolbar buttons initiate the following actions:

**Table 11-10: Cross Process Viewer Toolbar Buttons**

Button	Action
	Initiates a new session with a target running LynxOS or BlueCat Linux.
	Loads a snapshot from a file.
	Saves the current state (a snapshot) to the file.
	Displays the <a href="#">Processes Options</a> dialog box.
	Displays the <a href="#">Add to Counters</a> dialog box.
	Displays VisualLynux <b>Help</b> .



---

## CHAPTER 12 *Bootp-Tftp-Pftp Utility*

---

### BTP Overview

The *Bootp-Tftp-Pftp (BTP)* utility provides servers for booting targets remotely.

A *target* is a computer or device for which the user develops LynxOS or BlueCat Linux applications and kernel images (using VisualLynux or a command-line tool on a Windows host). The target is connected to the development/host computer through a network or parallel port connection.

BTP uses the TCP/IP protocol for communicating with targets, and the Parallel Port Protocol as the primary tool to transfer files between the host and the target.

Choose from the following BTP topics:

- [BTP Overview](#)
- [General BTP Page](#)
- [Bootp Page](#)
- [Tftp Page](#)
- [Pftp Page](#)
- [Bootp DB Entry](#)
- [BTP Tray Icon](#)

### BTP Servers

The Bootp-Tftp-Pftp utility provides three servers for booting targets remotely:

- Bootp (Bootstrap Protocol) server
- Tftp (Trivial File Transfer Protocol) server
- Pftp (Parallel Port File Transfer Protocol) server

*Bootp* and *Tftp* servers work together to service target booting requests via a TCP/IP network. This version of BTP includes the Bootp server that supports only single-domain Ethernet local networks. The Tftp server supports TCP/IP networks and uses the UDP protocol to communicate with a target. The target should use Bootp and Tftp protocols to be remotely booted via Ethernet network.

The *Pftp* server uses a special Bidirectional Parallel port driver to communicate with a target via a parallel port interface and implements the Parallel port File Transfer Protocol. The target should also support this protocol and use the appropriate parallel port driver. BTP includes two drivers, (for Windows NT/2000 and Windows 98) both of which support the conventional LPT ports of the Windows host system. LynuxWorks Cross Development Kits (Tools) include appropriate parallel port drivers for LynxOS and BlueCat Linux operating systems.

## Starting BTP

BTP can be started from an MS-DOS command-line window, by double-clicking the icon in the Windows Explorer, or by double-clicking the BTP icon on the Desktop (provided the user has created a shortcut).

If started from an MS-DOS command-line window or by double-clicking a shortcut, the command syntax can include optional parameters:

```
btp [ b={start|stop} ] [ t={start|stop} ] \  
[ p={start|stop} ]
```

where the **b**, **t**, and **p** denote Bootp, Tftp, and Pftp servers, respectively,

**start** forces the server to be started automatically,

and **stop** prevents the server from being started automatically.

When a keyword is omitted, a default value is used. Set the default by checking or unchecking the **Launch server at startup** box and clicking **Apply** in the corresponding page of the **BTP** dialog box.

You can start and stop a server dynamically by using the **Start/Stop** buttons in the corresponding page of the **BTP** dialog box.

The **BTP** dialog box does not appear if BTP is started in hidden mode. The [General BTP Page](#) contains the **Hide Window at startup** box to set the appropriate mode.

When started, the [BTP Tray Icon](#) appears in the Windows tray bar (in the right part of the task bar). The icon shows the status of BTP servers and can be used to display or hide the **BTP** dialog box.

## BTP Dialog Boxes

The main BTP dialog box contains four pages:

- [General BTP Page](#)
- [Bootp Page](#)
- [Tftp Page](#)
- [Pftp Page](#)

The General BTP Page displays copyright and version information and is used to set the BTP startup mode. Other pages are used to dynamically start or stop the appropriate server or to modify its options.

The Bootp Page displays the [Bootp DB Entry](#) page when you click **New** or **Modify**.

---

## General BTP Page

The General BTP page is used to set the BTP startup mode.

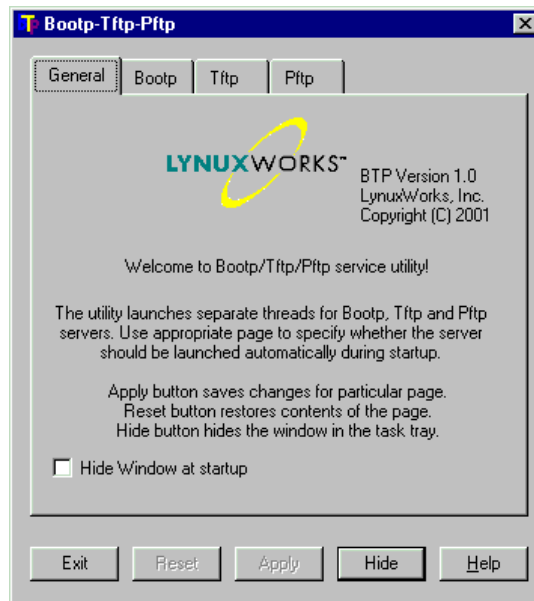


Figure 12-1: General BTP Page

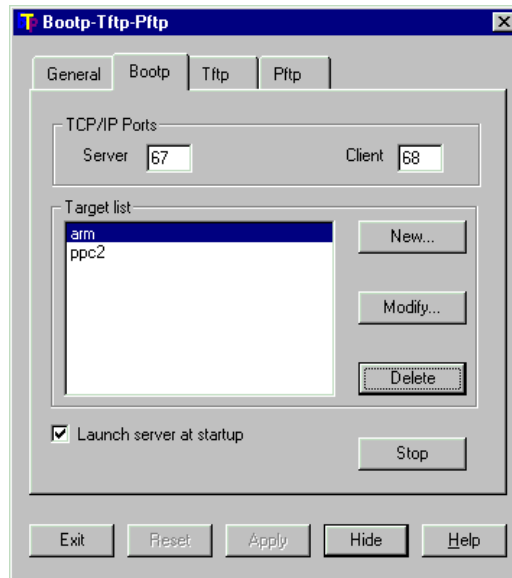
The following control is available on the **General BTP** page:

**Table 12-1: General BTP Page Commands**

Control	Description
<b>Hide Window at startup</b>	If checked, and the <b>Apply</b> button is clicked, BTP starts up in hidden mode the next time it is invoked. The BTP dialog box does not appear. Use the <a href="#">BTP Tray Icon</a> to display the dialog box.

## Bootp Page

The **Bootp** page is used to modify Bootp server options, dynamically start or stop the Bootp server, or display the [Bootp DB Entry](#) dialog box.



**Figure 12-2: Bootp Page**

The following table lists the **Bootp** page controls:

**Table 12-2: Bootp Page Commands**

Control	Description
<b>Server</b>	Specifies the Bootp server port number. Normally, you do not need to modify this number normally because LynuxWorks OS loaders use the standard server port number in Bootp requests.
<b>Client</b>	Specifies the Bootp client port number. Normally, you do not need to modify this number normally because LynuxWorks OS loaders use the standard client port number in Bootp requests.
<b>Target list</b>	Shows entries in the Bootp database. <ul style="list-style-type: none"> <li>• To create a new entry, click the <b>New</b> button.</li> <li>• To modify an entry, select it and click the <b>Modify</b> button.</li> <li>• To delete an entry, select it and click <b>Delete</b>.</li> </ul>
<b>Launch server at startup</b>	Check the box and click <b>Apply</b> to start the Bootp server automatically the next time BTP is invoked.
<b>Start</b>	Click to start the server. The button label changes to <b>Stop</b> . Click again to stop the server.

---

## Tftp Page

The **Tftp** page (see figure below) is used to modify Tftp server options or dynamically start or stop the Tftp server.

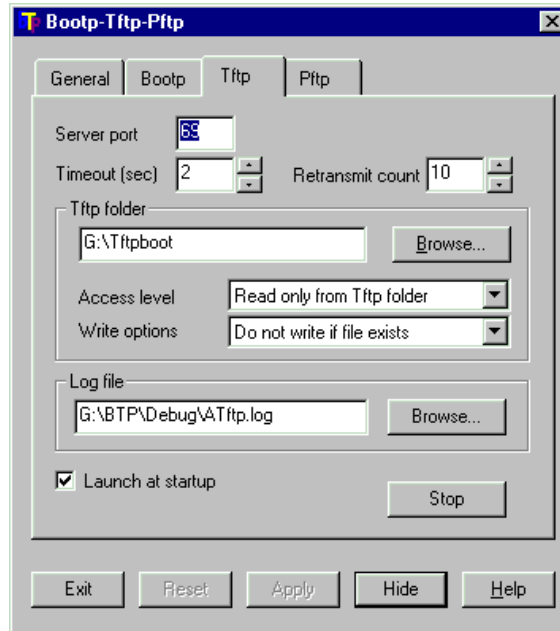


Figure 12-3: Tftp Page

The following table lists the **Tftp** page controls:

Table 12-3: Tftp Controls

Control	Description
<b>Server port</b>	Specifies the Tftp server port number. You need not modify this value normally because LynuxWorks OS loaders use the standard server port number.
<b>Timeout (sec)</b>	Specifies a timeout used in the Tftp protocol.
<b>Retransmit count</b>	Specifies the number of retransmit attempts in the Tftp error recovery protocol.
<b>Tftp folder</b>	Specifies the full path for the Tftp file repository. Specify only a simple file name in the OS loader <code>boot</code> command. Use <b>Browse</b> to find the Tftp folder. Use the <b>Access level</b> and <b>Write options</b> drop-down lists to select a proper access level, and write options (used in case a file with the same name already exists in the Tftp folder).



Table 12-3: Tftp Controls (Continued)

Control	Description
Log file	Specifies a log file for the Tftp server. Use the <b>Browse</b> button to specify a file.
Launch at startup	Check the box and click <b>Apply</b> to start the Tftp server automatically the next time BTP is invoked.
Start	Click to start the server. The button label changes to <b>Stop</b> . Click again to stop the server.

## Pftp Page

The **Pftp** page (see next figure) is used to modify Pftp server options or dynamically start or stop the Pftp server.

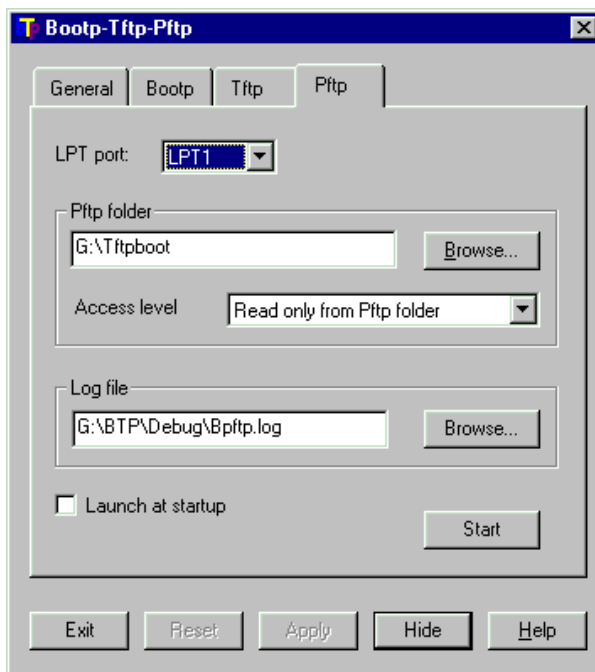


Figure 12-4: Pftp Page

The table below lists the **Pftp** page controls:

**Table 12-4: Pftp Page Commands**

Control	Description
<b>Lpt port</b>	Use this drop-down list to specify the LPT port used to boot the target.
<b>Pftp folder</b>	Specifies the full path name for the Pftp file repository. You need supply only a simple file name in the OS loader <code>boot</code> command. Use the <b>Browse</b> button to find the Pftp folder. Use <b>Access level</b> to specify access level.
<b>Log file</b>	Specifies a log file for the Pftp server. Use <b>Browse</b> to specify a file.
<b>Launch at startup</b>	Check the box and click <b>Apply</b> to start the Pftp server automatically when BTP is next invoked.
<b>Start</b>	Click to start the server. The button label changes to <b>Stop</b> . Click again to stop the server.

---

## Bootp DB Entry

The **Bootp DB Entry** dialog box is used to change Pftp target descriptions for the Bootp server.

**Figure 12-5: Database Entry**

The following table lists the Bootp DB controls:

**Table 12-5: Bootp DB Commands**

Control	Description
<b>Target name</b>	A symbolic name for a target
<b>Hardware address (hexadecimal)</b>	MAC address for the target
<b>Target IP address</b>	Target IP address sent to the target by the Bootp server
<b>Server IP address</b>	Server IP address sent to the target by the Bootp server - If this field is blank, one of the current host addresses is used to serve Bootp requests. Is useful in situations when host gets IP address from a DHCP server.
<b>Name server IP address</b>	Name server IP address sent to the target by the Bootp server. Is optional for this version of BTP.

Table 12-5: Bootp DB Commands (Continued)

Control	Description
Domain name	Domain name sent to the target by the Bootp server. Optional for this version of BTP.
Network type	The current version uses only Ethernet local area networks.

## BTP Tray Icon

The BTP Tray Icon (see figure below) appears in the Windows tray bar (in the right part of the task bar). The icon shows the status of BTP servers and can be used to display or hide the BTP dialog box.

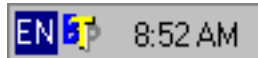


Figure 12-6: BTP Tray Icon

When the mouse cursor enters the icon area, a tip text window appears to show the servers currently active. The icon itself also shows the BTP status. If a server is not active, the appropriate letter in the icon is grayed-out. In the example shown above the Pftp server is not active.

Double-click the icon to display the hidden BTP dialog box or to set it as a foreground window. When clicking with the right mouse button, a pop-up menu appears; it can be used to switch the BTP mode or to terminate the program.

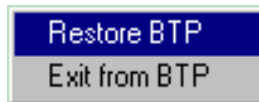


Figure 12-7: BTP Tray Icon Context Menu

---

# Index

---

## Symbols

.a file 136  
.bsc file, browser database 134  
.cfg file 182  
.CPP 28  
.d file 116, 148  
.dsp file 4, 31, 105  
.dsw file 4, 31, 105  
.vlm file 5

---

## A

a.out file, kernel executable 79  
Add to Counters dialog box, CPV 249  
AddIn Tools 101  
adding nodetab entry 186  
adding or removing source and header files, static  
device driver 77  
always use custom build, file processing 28  
ANSI 152, 155  
Application Wizard, launching 43  
ar.exe, static library building 29  
arbitrary batch commands 30  
arbitrary commands 30  
archive file 171  
authentication data 224  
Authentication step 224  
automatic probing, BlueCat Linux device  
driver 71

---

## B

Basic Null Character and Simple Streams Echo  
drivers 63  
bidirectional parallel port driver, Pftp 262  
block driver 184  
BlueCat Linux 66  
kernel configuration 182  
Block Ramdisk drivers 63  
BlueCat Linux  
CPV options settings, dialog box 254  
man pages xi, 10  
online documentation 104  
User's Guide 9  
Board Support Package 2, 181  
selecting for kernel build 81  
Bootp  
client port number 265  
database 265  
server definition 262  
server port number 265  
Bootp DB Entry 268  
Bootp Page 263, 264  
Bootp server, Pftp target descriptions 268  
break points, debugging 119  
Browse Info file 135  
browser database file 134  
Browser page  
Commands 134  
project settings 134  
browsing  
class definitions 134  
FTP Folders 199  
BTP  
command syntax 262  
General page 263

- starting 262
- BTP servers
  - launching automatically 262
  - launching dynamically 262
- BTP Tray Icon 261, 262, 270
- build configuration name 183
- build configuration, uparam.h file 193
- build environment 144
- Build macros 138
- Build Menu 39
- build tools, CYGWIN programs 11
- Building project targets
  - Batch Build 27
  - Rebuild All 27

---

## C

- C expressions, kernel parameter values 193
- C User Application Types 46
- C User Application Wizard 45
- C++ User Application Types 53
- C++ User Application Wizard 52
- C/C++
  - Listings 154
  - Optimization 159
    - Custom Option Commands 160
    - Customize 159
    - Disable 159
    - Extended Optimization 159
    - Minimal Size 159
  - Preprocessor 164
  - Preprocessor Options 165
  - Profiling 158
    - Commands 158
  - Project Options 165
  - Special Options 165
  - Warnings 155
    - Warning level drop-down list 155
- C/C++ Page 145
  - Category drop-down list 147
  - Code Generation 149
    - Commands 150
    - Position Independent Code 150
  - Debug level 146
  - General 148
  - General Settings 148
  - Language 152
    - ANSI 152
    - Project Options 145
    - Special options 146
- CDK
  - configuring 13
  - directory 84
  - mountpoints 11
- CDK tree, LynxOS 14
- CDT
  - configuring 15
- character driver 184
  - BlueCat Linux 65
  - kernel configuration 182
- checking Telnet connection to the target 227
- Cloned device node, kernel configuration 182
- command line tools, bash shell 124
- communication protocols with target 226
- Compiler
  - assigning 130
  - C 28
  - C++ 28
- Compiler Option Categories 147
  - C/C++ Language 147
  - Code Generation 147
  - General 147
  - Listings 147
  - Optimization 147
  - Preprocessor 147
  - Profiling 147
  - Special Options 147
  - Warnings 147
- Compiler Options
  - kernel executable project 83
  - project settings 145
- Component Functions 47, 49
  - Priority Alteration 51
- config.h file 182, 187
  - creating line entry 187
- config.h line, editing 188
- CONFIG.TBL file 182
  - configuring kernel with 182
  - kernel configuration data 181
- configuration data 11
- directory 185
  - subdirectory 98
- configuration entry 184
- configuring
  - BlueCat Linux CDTs 15
  - KDI 87
    - file system 88
  - LynxOS CDKs 13

- target 216
- contacting LynuxWorks xiii
- copy errors, kernel wizard 98
- copying files from CDK directory, kernel wizard 96
- CPV 241
  - About CPV 258
  - Add to Counters 249
  - Cross Process Viewer 103
  - File menu 245
  - Help Menu 258
  - Memory Usage Pane 242
  - Options menu 248
  - Performance Pane 242
  - Processes Options 250
    - BlueCat 254
    - LynxOS 250
  - Processes Pane 242
  - Sort Options 256
  - Sort options dialog box 256
  - System Tasks Pane 242
  - Toolbar Buttons 258
  - transactions with target 242
  - using after development cycle 241
  - View menu 247
  - Window 242
- cpv.exe 241
- creating
  - an FTP folder 203
  - config.h line 187
  - device node 62, 75
  - device node entry, driver configuration file 190
  - driver entry 188
  - new device entry, driver configuration file 190
  - new module entry, driver configuration file 189
  - system device entry, driver configuration file 191
- cross development environment 111
- Cross Development Kit 2, 10, 11, 181
- Cross Development Tools 15, 106, 144
- cross development, definition 1
- cross linker 166
- Cross Process Viewer 215, 241
- CSE 162, 163
- curses 52
- Custom Build 27, 28, 30
  - File 142

- Directory menu 144
- File menu 145
- Project
  - Build Environment 141
  - Build macros 139
  - Commands 139
  - Cross Development Tools 141
  - Directory menu 140
  - File menu 141
  - Steps 138
- Custom C LynuxWorks Application 46
- CYGWIN 7
  - binaries 17
  - conventions 108, 176
  - Installation property page 17

---

## D

- DDD
  - Block-based driver 64
  - Character-based driver 64
  - creating empty project 59, 64
  - Network driver 64
  - Null Character-based driver 59
  - RAMdisk driver 59
  - Streams Loopback/Echo driver 59
- Debug page
  - Commands 133
  - project settings 133
- debuggers 118, 120
- debugging
  - project 26
- Debugging step 231
  - GDBSERVER directory 232
- dependency file 148
  - .d file 116
- dependency, generating 116, 146
- device driver project selection 58
- Device Driver Wizard 58
- device node 184
  - creation 62, 75
  - kernel configuration 182
  - properties 184
  - type 191
- device specification entry 184
- Device, kernel configuration 182
- directory
  - msdev.exe 31

- platform independent tools 31
    - Visual Studio build tools 31
    - VisualLynux installation 31
  - Directory macros 140, 144
  - displaying
    - Contents of FTP Folders 202
    - File Properties 205
    - FTP Folder Properties 204
  - documentation 104
    - BlueCat Linux xi, 10
    - GNU tools 10
    - LynxOS xi, 10
    - VisualLynux xi, 10
  - Driver Basics 1, 9
  - driver configuration file 181, 182, 184, 188, 189
    - creating device node entry 190
    - creating entry 188
    - creating new device entry 190
    - creating new module entry 189
    - creating system device entry 191
    - editing device entry 190
    - editing device node properties 191
    - editing module properties 189
    - editing system device entry 192
    - importing 185
    - inserting 185
  - Driver Ports and IRQs, BlueCat Linux 69
  - driver specification line 184
  - dynamic device driver 58
    - BlueCat Linux 64
    - LynxOS 59
    - testing 60, 74
- 
- E**
- editing
    - a config.h line 188
    - a driver entry, driver configuration file 189
    - device entry, driver configuration file 190
    - device node properties, driver configuration file 191
    - environment variables, Make page 176
    - module properties, driver configuration file 189
    - system device entry, driver configuration file 192
  - Empty C Project, C user application 46
  - Empty C++ Project 53
  - entropy pool 73
  - entry point names, driver configuration file 189
  - enum type 150
  - Export
    - Directories for New Source Tree 114
    - directories, setting up 113
    - Initial Screen 111
    - Makefile names 115
    - Output directories 113
    - Project Files to Copy 116
    - Setting Configurations 112
    - Subdirectories tree control 114 to Makefile 111
  - Export Directories page 113
  - Export Wizard 111
  - Extended Optimization 159
  - external dependencies 116
- 
- F**
- F7 key 25, 27
  - File Custom Build 142
  - File macros 141, 145
  - FTP
    - Browsing Local Folders 205
    - Changing the View Mode 206
    - Command line 198
    - Connect as 198
    - Creating a New Folder 207
    - Directory tree 196
    - Disconnect 197
    - Displaying File or Folder Properties 208
    - Displaying the Parent Folder 206
    - Down Arrow 200
    - Downloading Files 210
    - Downloading folders 210
    - End 201
    - Folder Hierarchy 200
    - FTP host name 198
    - Home 201
    - home directory 228
    - Keyboard 200
    - Left Arrow 201
    - Main window 196
    - Manipulating Files on a Local Computer 207
    - Menu and Toolbar Commands 201
    - Mouse 201



---

- Opening Local Folders 205
- Password 198
- Port number 198
- Removing Files and Folders 209
- Renaming Files or Folders 207
- Right Arrow 201
- Up Arrow 200

FTP server 210

FTP/Telnet configuration 215

---

## G

- g++.exe 28, 166
- gcc.exe 28, 166, 168
- gcov code-coverage utility 159
- GDB 26, 118, 133
- GDB text-mode window 119
- GDBSERVER 102, 118, 120, 216, 219, 231
- GDBSERVER TCP/IP port number 233
- General BTP Page 263
- General information step, target wizard 220
- General page commands, project settings 131
- General page, File 131
- General page, project settings 131
- Generate dependencies 29
- Generic Device Drivers Library project 79
- Generic Devices Library project 79
- GNU 153
- GNU documentation set 10
- GNU GDB and Insight debuggers 26
- GNU Tools Documentation xi, 104, 126
- gprof analysis program 158

---

## H

- Help Topics 258
- Help, resources 3
- HTML help 126

---

## I

- IDE Tools 101
- Import
  - Adding/Editing Environment Variables 108

- Build Environment 107
- Environment variables 107
- Initial Screen 105
- Insert files in project 109
- Project Configurations 106
- Project name 105
- Shell scripts 107
- Specifying build target 110

Import Wizard 104

importing configuration file 185

init 252

input buffer size, device driver 66, 68, 69

inserting configuration file 185

installation procedure, VisualLynux 1

installing

- block device 62
- character device 62

IntelliSense options 36

interrupt handler routine 72

IRQ line number, BlueCat Linux device driver 72

ISA boards 71, 72

---

## J

- jump optimization 162

---

## K

- KDI 86
  - configuring 87
  - configuring file system 88
  - creation 86
  - File System 88
  - source files added to target directory 92
  - Target Directory 92
- kernel build parameters, editing 178
- kernel configuration 84
  - editing item properties 183
- kernel configuration data 179, 181
  - kernel executable project 83
- kernel configuration files 84
- Kernel Configuration Page 181
- Kernel Downloadable Image 86
- kernel executable
  - a.out file 79

- nodetab file 79
- Kernel Executable project 79, 80
  - creating kernel configuration data 83
  - Extracting compiler options 83
  - Extracting linker options 83
  - Extracting source file dependencies 84
- Kernel General Page 180
- kernel library project 79, 80, 94
- Kernel Options 178
- Kernel page 178
- Kernel Parameters 85
  - Page 192
- Kernel Project 178
  - configuration 179
- Kernel Wizard 79

---

## L

- ld 170, 171, 173
- ld.exe 29, 168
- libar.a file, archive file 171
- Library page
  - Commands 136
  - project custom build 140
  - project settings 135
- Linker Options 166
  - Category drop-down list 166
  - kernel executable project 83
- Linker Options Categories
  - Customize 168
  - Debug 170
  - General 167
  - Input 171
  - Output 172
  - Special Options 174
- Linker page, project custom build 140
- linking a project target 136
- low water mark 244
- LynuxWorks FTP 195, 215
- LynuxWorks, contacting xiii
- LynxOS
  - Dynamic Device Driver 29
  - Installation Guide 9
  - Kernel project 178
  - Kernel Wizard 79
  - man pages xi, 10, 104, 126
  - Processes options 250

---

## M

- macros
  - file-level 30, 33
  - global 30, 31
  - project-level 30, 31
- major number, device driver 65, 67, 69
- Make
  - Adding/editing environment variables 176
  - Environment variables 176
  - File 177
  - Project 174
  - Setting up Scripts 176
  - Shell scripts 175
- Make page 174, 177
  - Environment 175
  - General 174
- make utility 11, 107
- Makefile, selecting for import 105
- Makefile-based VisualLynux projects 104
- Makefiles page 115
- man pages
  - BlueCat Linux xi, 10
  - LynxOS xi, 10
- Manipulating FTP Files 204
- Memory Usage pane, CPV 244
- Menu, VL
  - Build 39
  - Edit 36
  - File 36
  - Insert 38
  - Overview 35
  - Project 38
  - Tool 40
  - View 37
  - Window 42
- Microsoft browser builder (bscmake.exe) 135
- Microsoft Class and Symbol browser 40
- Microsoft compiler (cl.exe) 135
- Microsoft Developer 141
- Microsoft Visual C++ 9
  - Enterprise Edition 10
  - Professional Edition 10
  - Standard Edition 10
- module properties 185
- module specification line 184
- Module, kernel configuration 182
- Mount Table, CDK tools 19
- mountpoint

---

- CDK tools 11, 19
- CYGWIN utilities/commands 20
- LynxOS root installation 20
- msdev.exe 31
- mutexes 50

## N

- native-development environment 111
- Network Connection Step 222
- Network driver 76
  - BlueCat Linux 68
- Network Driver wizard 69
- Network File System support 223
- Network Neighborhood window 223
- new project information, kernel wizard 99
- nodetab entry, adding 186
- nodetab file 180, 183, 185
  - kernel executable 79

## O

- OLE/COM objects, viewing 41
- Online Help xi, 9
- OS loader 265, 266, 268
- output buffer size, device driver 66

## P

- PCDSRVR 216, 219
  - TCP/IP port number 233
- PCI boards 72
- Performance Chart Options, CPV 248
- performance counters, CPV 243
- Performance Options dialog box, CPV 248
- Performance pane, CPV 243
  - counter options 249
  - settings 248
- Pftp
  - file repository 268
  - server definition 262
- Pftp Page 267
- port addresses, BlueCat Linux device driver 70
- Position Independent Code 150
- POSIX Signal Handler 49

- POSIX-based programming concepts 46
- Post-Build 30
  - commands grid 137
  - description 137
- Post-Build page, project settings 137
- Pre-Link page, project settings 136
- Pre-Link step 30
  - Build macros 137
  - commands 136
  - description 136
- Process Viewer server (PCDSRVR) 233
- Process Viewer server directory 234
- Process Viewer Step 233
- processes and threads statistics 241
- Processes Options
  - BlueCat Linux 254
  - CPV 250
- Processes pane, CPV 245
- prof analysis program 158
- Project
  - Build Steps 29
  - configuration 6, 183
  - Conversion utility 19
  - creation, overview 22
  - Custom Build 138
  - Debugging overview 26
  - definition 3
  - project directory 4
  - project file 4, 31, 105
  - project goals 3
  - Project Settings 38, 40
    - File level property pages 130
      - Build 130
      - C/C++ 130
      - Custom Build 130
      - General 130
  - Overview 127
  - Project level property pages 129
    - Browse Info 129
    - C/C++ 129
    - Custom Build 129
    - Debug 129
    - General 129
    - Kernel 129
    - Library 129
    - Link 129
    - Make 129
    - Post-Build 129
    - Pre-Link 129

- Project Type 22, 45
  - selecting 44
- project-level tools 27
- Projects Directory Step 228
  - Browse for FTP folders 230
  - Create now 230
  - Home FTP directory 230
  - Root Project Directory 230
- Property Pages 129, 130
- pseudo-driver project 70
- pseudo-target 110, 139, 143
- pthreads, linker options 168

---

## R

- Release mode 124
- release notes, VisualLynux 10
- removing
  - a component, kernel configuration 183
  - an FTP folder 203
  - FTP Files 204
- renaming
  - an FTP File 204
  - an FTP folder 203
- Root Project Directory name 230
- Run command dialog box 239

---

## S

- sample workspace 99
- Settings Dialog Box 27
- shared instead of static libraries, linker options 168
- shared interrupts 73
- Shell scripts 108
- Simple Block driver 75
- Simple C "Hello World" Application 46
- Simple C++ "Hello World" Application 53
- Source Browser Database 129
- source file dependencies, kernel executable project 84
- Source File Options, C/C++ page 146
- source files copied with errors, kernel wizard 98
- source files for project 27
- Specific BSP Device Drivers Library project 79
- Specific BSP Device Library project 79

- starting assembler commands, Total/db 122
- static device driver 58
  - creating 60
  - creating for BlueCat Linux 73
- Static Library 29
- Static Library Application Wizard 77
- Static Library projects 135
- system device 192
- system device entry 184
- System device, kernel configuration 182
- System Requirements 9
- System Tasks pane, CPV 245

---

## T

- Target Administration
  - Cancel 219
  - Default target 236
  - Edit target properties 219
  - Editing target properties 235
  - Help 219
  - Invoking 216
  - Make default 219
  - New Target 235
  - New target 219
  - OK 219
  - Options 219
  - Remove target 219
  - Removing a target 236
  - Renaming a target 238
  - Run command 219
  - Target selection mode 237
  - Tasks 234
- Target Administration Dialog Box 217, 218
- Target Authentication
  - anonymous connection 226
- Target Configurations 22, 45
- Target identification 221
  - Connection mode 222
  - Target name 222
- target platforms supported 2
- Target selection dialog box 237
- Target Wizard 217, 220
- TCP/IP 104, 198
  - connection 222
  - port number 198
  - protocols 215
- Technical Support xiii

---

Telnet 26, 103, 125, 215  
Telnet/FTP configuration 219  
Terminology 7  
Tftp  
    error recovery 266  
    file repository 266  
    server definition 262  
    server port number 266  
timeout, device driver 66, 67, 69  
Toolbar 101  
    About VisualLynux 104  
    Bash 103  
    Copy Executable 103  
    Export to Makefile 102  
    GDB 102  
    Import Makefile 102  
    LynuxWorks Cross Process Viewer 103  
    LynuxWorks FTP 103  
    LynuxWorks Total/db 103  
    Target Administration Wizard 104  
    Telnet 103  
    VisualLynux Help 104  
    VisualLynux Project Settings 102  
toolbar, Addin tools 101  
toolbar, VisualLynux 1  
Total/db 26, 103, 120, 133  
transactions, CPV and target 242  
Transfer dialog box 211  
    Status window 212  
Typographical Conventions xi, xii

---

## U

UDP protocol 262  
uninstalling  
    device driver 63, 76  
unregistering  
    CDKs 12  
    VL 1.x 19  
uparam.h file 85  
    build configuration 193  
    kernel parameters 85  
Upload 124  
Uploading Files and Folders 211  
Utilities Step 226  
    Check FTP connection 228  
    Check Telnet connection 227  
    FTP home directory 228

FTP port number 227  
Telnet port number 227

---

## V

Verifying the TCP/IP connection 224  
Visual component manager tool 40  
Visual Studio 55  
Visual Studio menu 35  
VisualLynux Application Wizard 5, 20  
VisualLynux Configuration Utility 10, 11  
VisualLynux Import Wizard 5  
VisualLynux Integrated Development  
    Environment 241  
VisualLynux Kernel Wizard 20  
VisualLynux LynxOS Kernel Wizard 5  
VisualLynux Online Help 10  
VisualLynux Release Notes 10  
VisualLynux Target Administration  
    Overview 215  
VisualLynux Target Administration Wizard 104  
VisualLynux Toolbar 35  
VisualLynux User's Guide 10, 104, 126  
VisualLynux, definition 1  
VisualLynux 1.x 19  
vlftp.exe 195

---

## W

Warning level drop-down list 155  
Window Menu 42  
Windows 2000 Professional 9  
Windows 98 9  
Windows cross development system 9  
Windows Network browser 223  
Windows NT 9  
Windows Registry 11, 15  
wizards 5  
    launching 20  
Workspace 4  
    Current 22  
workspace directory 4  
workspace file 31, 105

---

## X

X Error Handler 58

X/Motif

    Application 29

    Custom Application 55

    Empty Project 55

    Graphical Application Wizard 54

    Simple application 55

XT Runtime Warning 58