



Fermilab

GU0014C

Complete Guide and Reference Manual for UPS, UPD and UPP v4

Part IV: Product Developer's Guide and Part VIII: Developer's Reference

Release 2.0

June 30, 2000

Computing Division
Fermi National Accelerator Laboratory

Compiled by Anne Heavey

ABSTRACT

This manual documents the standard methodology for UNIX product support at Fermilab, which is implemented via the utilities **UPS** (UNIX Product Support), **UPD** (UNIX Product Distribution), and **UPP** (UNIX Product Poll). These utilities were significantly redesigned for version v4, which was initially released in 1998, and have continued to be revised since then. The latest release as of this writing is v4_5_2. This document supersedes GU0014 “UPS and UPD v4 Reference Manual”, released June 5, 1998.

This part of the document (GU0014C) includes a guide and reference manual for product developers.

Revision Record

May 1997	Original Release 1.0 (for UPS v3 and UPD v2)
August 1997	Revisions 1.1 and 1.1a (for UPS v3 and UPD v2)
June 1998	Release 1.0 for UPS and UPD v4
December 1999	Draft release 2.0 for UPS/UPD/UPP v4. Part VI Command Reference only
June 2000	Release 2.0 for UPS, UPD and UPP v4 (current as of v4_5_2)

This document and associated documents and programs, and the material and data contained therein, were developed under the sponsorship of an agency of the United States government, under D.O.E. Contract Number EY-76-C-02-3000 or revision thereof. Neither the United States Government nor the Universities Research Association, Inc. nor Fermilab, nor any of their employees, nor their respective contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights. Mention of any specific commercial product, process, or service by trade name, trademark, manufacturer, supplier, or otherwise, shall not, nor is it intended to, imply fitness for any particular use, or constitute or imply endorsement, recommendation, approval or disapproval by the United States Government or URA or Fermilab. A royalty-free, non-exclusive right to use and disseminate same for any purpose whatsoever is expressly reserved to the U.S. and the U.R.A. Any further distribution of this software or documentation, parts thereof, or other software or documentation based substantially on this software or parts thereof will acknowledge its source as Fermilab, and include verbatim the entire contents of this Disclaimer, including this sentence.

Acknowledgments

The redesign and redevelopment of **UPS** and its companion products in preparation for Fermilab's Run II involved a substantial commitment of resources from the Computing Division in 1997-98. Special thanks to Don Petravick (HPPC), Ruth Pordes (OLS), and Dane Skow (OSS) for providing talented and motivated members of their groups to accomplish this task. Since the initial release of **UPS/UPD v4** in 1998, development has been continuing, and we are at version v4_5_2 as of this writing.

The redevelopment effort was led by Eileen Berman. With her, the principal designers and developers of **UPS/UPD v4** included David Fagan, Marc Mengel, Lars Rasmussen and Margaret Votava. Other contributors to the new design included Lauri Loebel Carpenter, Rob Harris, Alan Jonckheere, Art Kreymer, Liz Sexton-Kennedy. Other contributors to the coding effort included Chuck Debaun, Paul Russo and Don Walsh.

Contributors in the areas of code review, testing, documentation review and deployment included Lauri Loebel Carpenter, Chuck Debaun, Lisa Giacchetti, Alan Jonckheere, Art Kreymer, Liz Sexton-Kennedy, Mike Stolz, Don Walsh and Gordon Watts, in addition to the development team. Special thanks go to Marc Mengel and Margaret Votava for contributing all the updated **UPD** and **UPP** information included in the first release of this manual for **UPS/UPD v4**.

Wayne Baisley and Marc Mengel are currently responsible for on-going support and development of **UPS/UPD**, and thanks go to them for providing quite a bit of updated information for this release of the manual. Thanks are also due to Wayne and Marc as well as to Joy Hathaway, Lauri Loebel Carpenter and Cindy Wike for reviewing portions of the documentation and providing feedback.

Table of Contents for Parts IV and VIII

About this Manual	INT-1
Document Structure, Purpose and Intended Audiences.....	INT-1
Availability	INT-3
Updates.....	INT-3
Conventions	INT-3
Your Comments are Welcome!	INT-5

Part IV: Product Developer's Guide

Chapter 15: UPS Product Development: General Considerations	15-1
15.1 Product Development Considerations and Recommendations	15-1
15.1.1 All Products (Locally Developed and Third Party)	15-1
15.1.2 Products that You Develop	15-2
15.1.3 Third-Party Products Requiring a Hard-Coded Path	15-3
15.2 Tools for Developing and/or Packaging Products	15-5
15.2.1 Buildmanager	15-5
15.2.2 CVS	15-5
15.2.3 Template_product	15-6
15.3 Directory Structure for a UPS Product Instance	15-6
Chapter 16: Building UPS Products	16-1
16.1 Basic Steps for Making a UPS Product	16-1
16.1.1 Build the Directory Hierarchy	16-2
16.1.2 Create the Table File	16-2
16.1.3 Declare the Product to your Development UPS Database	16-2
16.1.4 Copy the Product Executable to the bin Directory	16-3
16.1.5 Provide Product man Pages	16-3
16.1.6 Test the Product	16-4
16.2 Specifics for Different Categories of Products	16-4
16.2.1 Unflavored Scripts	16-4
16.2.2 Pre-built Binaries	16-5
16.2.3 Products Requiring Build (In-House and Third-Party)	16-6
16.2.4 Overlaid Products	16-7

16.3	Sample Auxiliary Files	16-8
16.3.1	README	16-8
16.3.2	INSTALL_NOTE	16-9
16.3.3	RELEASE_NOTES	16-9
Chapter 17: Making Products Available For Distribution		17-1
17.1	Product Distribution Overview	17-1
17.2	Creating Product Tar Files	17-2
17.3	Adding a Product	17-3
17.3.1	Product Categories Defined for KITS	17-3
17.3.2	Examples	17-4
17.4	Adding an Independent Table File	17-5
17.5	Replacing a Component (Table File or ups Directory)	17-6
17.6	Adding/Changing a Chain	17-7
17.7	Deleting a Product or Component	17-8
17.8	Cloning a Product	17-8
17.9	Including Source in one of Fermilab's CVS Repositories	17-9
17.10	Product Announcement Policies	17-10
Chapter 18: Using template_product to Build and Distribute UPS Products		18-1
.....		18-1
18.1	Overview	18-1
18.2	Accessing template_product	18-2
18.3	Cloning template_product	18-2
18.4	The Top-Level Makefile	18-3
18.5	Inserting your Product into the Template	18-4
18.6	Building the Product	18-4
18.6.1	Add Build Instructions	18-4
18.6.2	Run the Initial Build	18-4
18.6.3	Add Build Instructions to Top-Level Makefile	18-4
18.6.4	Rebuild Instructions	18-5
18.7	Testing your Product	18-5
18.8	Customizing your Tar File	18-5
18.9	Adding your Product to a Distribution Node	18-6
18.9.1	Add Product to fnkits	18-7
18.9.2	Specify Multiple Flavors	18-7
18.10	Adding your Product Source to a CVS Repository	18-8
18.11	Removing your Product from a Distribution Node	18-8
Chapter 19: Checklist for Building and Distributing Products		19-1
19.1	Pre-build Checklist	19-1
19.2	Build the Product	19-2
19.3	Test the Product	19-2
19.4	Distribute to fnkits as "test"	19-3
19.5	Announce the Product	19-3
19.6	Distribute to fnkits as "current"	19-4

Part VIII: Developer's Reference

Chapter 33: Actions and ACTION Keyword Values	33-1
33.1 Overview of Actions	33-1
33.2 UPS Command Actions	33-1
33.2.1 UPS Commands as Keyword Values	33-1
33.2.2 “Uncommands” as Keyword Values	33-2
33.3 Chain Actions	33-3
33.3.1 Chains as Keyword Values	33-3
33.3.2 “Unchains” as Keyword Values	33-3
33.4 The “Unknown Command” Handler	33-3
33.5 Actions Called by Other Actions	33-4
Chapter 34: Functions used in Actions	34-1
34.1 Overview of Functions	34-1
34.2 Reversible Functions	34-1
34.3 Function Descriptions	34-2
34.3.1 addAlias	34-2
34.3.2 doDefaults	34-3
34.3.3 envAppend	34-3
34.3.4 envPrepend	34-4
34.3.5 envRemove	34-4
34.3.6 envSet	34-5
34.3.7 envSetIfNotSet	34-5
34.3.8 envUnset	34-5
34.3.9 exeAccess	34-6
34.3.10 exeActionOptional	34-6
34.3.11 exeActionRequired	34-6
34.3.12 execute	34-7
34.3.13 fileTest	34-7
34.3.14 pathAppend	34-8
34.3.15 pathPrepend	34-8
34.3.16 pathRemove	34-9
34.3.17 pathSet	34-9
34.3.18 prodDir	34-9
34.3.19 setupEnv	34-10
34.3.20 setupOptional	34-10
34.3.21 setupRequired	34-10
34.3.22 sourceCompileOpt	34-11
34.3.23 sourceCompileReq	34-11
34.3.24 sourceOptCheck	34-12
34.3.25 sourceOptional	34-13
34.3.26 sourceReqCheck	34-13
34.3.27 sourceRequired	34-14
34.3.28 unAlias	34-14
34.3.29 unProdDir	34-14

34.3.30	unsetupEnv	34-15
34.3.31	unsetupOptional	34-15
34.3.32	unsetupRequired	34-16
34.3.33	writeCompileScript	34-16
34.4	Functions under Consideration for Future Implementation	34-17
34.5	Examples of Functions within Actions	34-18
34.5.1	A setup Action	34-18
34.5.2	A “declare as current” Action	34-18
34.6	Local Read-Only Variables Available to Functions	34-18
34.6.1	List of Current Read-Only Variables	34-19
34.6.2	Read-Only Variables under Consideration for the Future	34-21
Chapter 35:	Table Files	35-1
35.1	About Table Files	35-1
35.2	When Do You Need to Provide a Table File?	35-1
35.3	Recommendations for Creating Table Files	35-2
35.4	Table File Structure and Contents	35-2
35.4.1	Basic Structure	35-2
35.4.2	Grouping Information	35-3
35.4.3	The Order of Elements	35-3
35.5	Product Dependencies	35-4
35.5.1	Defining Dependencies	35-4
35.5.2	Product Dependency Conflicts	35-4
35.6	Table File Examples	35-6
35.6.1	Example Illustrating Use of FLAVOR=ANY	35-6
35.6.2	Example Showing Grouping	35-6
35.6.3	Example with User-Defined Keywords	35-7
35.6.4	Examples Illustrating ExeActionOpt Function	35-8
Chapter 36:	Scripts You May Need to Provide with a Product	36-1
36.1	configure and unconfigure	36-1
36.2	tailor	36-3
36.3	current and uncurrent	36-3
36.4	start and stop	36-3
Chapter 37:	Use of Compile Scripts in Table Files	37-1
37.1	Overview	37-1
37.2	Usage Information	37-1
Chapter 38:	Creating and Formatting Man Pages	38-1
38.1	Creating the Source Document (Unformatted)	38-2
38.1.1	Source File Format	38-2
38.1.2	Man Page Information Categories	38-3
38.1.3	Example Source File	38-4
38.2	Formatting the Source File	38-5
38.2.1	nroff	38-5
38.2.2	groff	38-6
38.3	Converting your Man Page to html Format	38-6

Glossary **GLO-1**
Index **IDX-1**

Table of Contents for Complete Guide

About this Manual	INT-1
--------------------------------	--------------

(This introductory chapter is listed in the front section of the table of contents.)

Part I: Overview and End User's Guide

Chapter 1: Overview of UPS, UPD and UPP v4	1-1
1.1 Introduction to UPS, UPD and UPP	1-1
1.2 Motivation for the UPS Methodology	1-2
1.3 UPS Products	1-3
1.3.1 Versions	1-3
1.3.2 Flavors	1-3
1.3.3 Qualifiers	1-4
1.3.4 Product Instances	1-4
1.3.5 Chains	1-4
1.3.6 Product Dependencies	1-5
1.3.7 Product Overlays	1-6
1.4 UPS Database Overview	1-6
1.4.1 UPS Database Files	1-6
1.4.2 UPS Database Structure	1-7
1.5 Using UPS Without a Database	1-7
1.6 UPS and UPD Commands	1-8
1.6.1 Syntax	1-8
1.6.2 Defaults	1-8
1.7 The UPS Environment	1-9
1.7.1 Initializing the UPS Environment	1-9
1.7.2 Changes UPS Makes to your Environment	1-10
Chapter 2: UPS Operations for the End User	2-1
2.1 Determining your Machine's Flavor	2-1
2.2 Listing Product Information in a Database	2-2
2.2.1 Formatted Output Style	2-3
2.2.2 Condensed Output Style	2-3
2.2.3 Examples	2-4
2.3 Finding a Product's Dependencies	2-7

2.4	Setting up a Product	2-8
2.4.1	The setup Command for the Typical Case	2-9
2.4.2	When You Need to Specify Other Options	2-9
2.5	Running Unsetup on a Product	2-10

Part II: Product Installer's Guide

Chapter 3:	General Product Installation Information	3-1
3.1	Installation Methods for UPS Products	3-1
3.1.1	UPD	3-1
3.1.2	UPP	3-2
3.1.3	FTP	3-2
3.2	User Node Registration for KITS	3-2
3.3	What You Need to Know about Your System's UPD Configuration ..	3-3
3.3.1	Location of UPD Configuration File	3-3
3.3.2	Where Products Get Declared	3-4
3.3.3	Where Products Get Installed	3-4
3.4	Declaring an Instance Manually	3-5
3.4.1	The ups declare Command	3-5
3.4.2	Examples	3-6
3.5	Installation FAQ	3-7
3.5.1	What File Permissions Get Set?	3-7
3.5.2	You're Ready to Install: Should you Declare Qualifiers?	3-8
3.5.3	What if an Install Gets Interrupted?	3-8
3.5.4	What if a Product was Installed under a Different Name?	3-8
3.6	Post-Installation Procedures	3-9
3.6.1	Configuring a Product	3-9
3.6.2	Tailoring a Product	3-9
3.7	Networking Restrictions at your Site	3-9
3.7.1	Proxying Webserver	3-9
3.7.2	Firewall for Incoming TCP Connections	3-10
Chapter 4:	Finding Information about Products on a Distribution Node .	4-1
4.1	Listing Products on a Distribution Node	4-1
4.1.1	Using UPD	4-1
4.1.2	Using UPP	4-3
4.2	Listing Product Dependencies on a Distribution Node	4-5
4.3	Information about Products in KITS	4-6
4.3.1	Access Restrictions and Product Categories	4-6
4.3.2	Product Pathnames for FTP Access	4-7
4.4	Special Instructions for Proprietary Products	4-8

Chapter 5: Installing Products Using UPD	5-1
5.1 The upd install Command	5-1
5.1.1 Command Syntax	5-1
5.1.2 Passing Options to the Local ups declare Command	5-2
5.2 How UPD Selects the Database	5-2
5.2.1 Database Selection Algorithm	5-2
5.2.2 Database Selection for Dependencies	5-3
5.2.3 Selecting a Database for Development or Testing	5-3
5.3 Checklist for Installing a Product using UPD	5-3
5.4 Examples	5-4
5.4.1 Install a Product Using Default Database	5-4
5.4.2 Install a Product, Specifying Database	5-5
5.4.3 Install a Product and All Dependencies	5-5
5.4.4 Install a Product and No Dependencies	5-7
5.4.5 Install a Product and Required Dependencies Only	5-7
Chapter 6: Installing Products Using UPP	6-1
6.1 Overview of Using UPP to Install Products	6-1
6.2 Creating a UPP Subscription File	6-1
6.2.1 Create the Header	6-2
6.2.2 Identify the Product	6-2
6.2.3 Trigger the Product Installation	6-2
6.2.4 Provide Instructions to UPP	6-3
6.3 Sample Subscription File for Installing a Product	6-3
6.4 The UPP Command	6-4
6.5 Automating UPP via cron	6-4
Chapter 7: Installing Products using FTP	7-1
7.1 UPS Product Components to Download	7-1
7.2 Installing Products from fnkits.fnal.gov	7-2
7.2.1 Download the Files from fnkits	7-2
7.2.2 Unwind the Files into your Products Area	7-3
7.2.3 Declare the Product to your Database	7-4
7.3 Installing Products from Other Product Distribution Nodes	7-4
7.3.1 Locate the Product Files on the Server	7-4
7.3.2 Download the Files from the Server	7-5
7.3.3 Unwind the Files into your Products Area	7-5
7.3.4 Declare the Product to your Database	7-5
Chapter 8: Product Installation: Special Cases	8-1
8.1 Installing Products that Require Special Privileges	8-1
8.2 Installing Locally Using UPD from AFS-Space	8-2
8.3 Installing Products into AFS Space	8-3
8.3.1 Overview	8-3
8.3.2 Request a Product Volume	8-4
8.3.3 Install your Product	8-4
8.3.4 Post-Installation Steps	8-5

Chapter 9: Troubleshooting UPS Product Installations 9-1

Part III: System Administrator's Guide

Chapter 10: Maintaining a UPS Database 10-1

10.1	Declare an Instance	10-1
10.1.1	The ups declare Command	10-2
10.1.2	Examples	10-2
10.2	Declare a Chain	10-4
10.2.1	The ups declare Command with Chain Specification	10-4
10.2.2	Examples	10-5
10.3	Remove a Chain	10-6
10.4	Change a Chain	10-7
10.5	Undeclare and Remove an Instance	10-7
10.5.1	Using ups undeclare to Remove a Product	10-8
10.5.2	Undoing Configuration Steps	10-9
10.5.3	Using UPP to Remove a Product	10-10
10.6	Verify Integrity of an Instance	10-10
10.7	Modify Information in a Database File	10-11
10.8	Determine If a Product Needs to be Updated	10-13
10.8.1	Using UPP	10-13
10.8.2	Using UPD	10-13
10.9	Update a Table File or ups Directory	10-14
10.10	Retrieve an Individual File	10-15
10.11	Check Product Accessibility	10-16
10.12	Troubleshooting	10-17

Chapter 11: UPS and UPD Pre-install Issues and General Administration

. 11-1

11.1	Choosing Installer Accounts	11-1
11.1.1	Single Installer Account	11-1
11.1.2	Multiple Installer Accounts	11-1
11.1.3	Separate Installer Accounts for Different Product Categories	11-2
11.2	Setting gids for Multiple Installer Accounts	11-2
11.3	File Ownership, Permissions and Access Restrictions	11-3
11.3.1	Product Files	11-3
11.3.2	Database Files	11-3
11.4	Product File Location and Organization	11-4
11.4.1	Considerations	11-4
11.4.2	Single Flavor or Single Node Systems	11-4
11.4.3	Multi-Flavor and/or Multi-Node Systems	11-5
11.5	Database File Location and Organization	11-6
11.5.1	Choosing Single or Multiple UPS Databases	11-6
11.5.2	UPS Database File Pointers	11-6

11.6	Installing UPS for Use Without a Database	11-7
11.7	CYGWIN (Windows NT) Issues	11-7
11.7.1	Using Correct Perl Version	11-7
11.7.2	Mounting the CYGWIN bin Directory	11-8
11.7.3	Setting Environment Variables	11-8
11.8	General Administration Issues	11-8
11.8.1	Upgrading an Older System	11-8
11.8.2	Adding a New Database and/or Products Area	11-9
11.8.3	Collecting Statistics on Product Usage	11-10
Chapter 12: Providing Access to AFS Products		12-1
12.1	Overview	12-1
12.2	Configuring a Local Database to Work With AFS	12-2
12.2.1	Steps to Create and Configure the Database	12-2
12.2.2	Post-Configuration: Reinitialize FUE Environment	12-4
12.2.3	A Note about Product Installation for this Configuration	12-4
12.3	Installing a Local Copy of CoreFUE	12-4
12.4	Additional Steps for Unfamiliar Naming Conventions	12-5
12.5	Updating /usr/local/bin to Access AFS Products	12-6
Chapter 13: Bootstrapping CoreFUE		13-1
13.1	Downloading the Bootstrap and Configuration Files	13-1
13.1.1	Predefined Configurations for UNIX	13-1
13.1.2	User-defined Configuration for UNIX	13-2
13.1.3	Predefined Configurations for NT	13-2
13.2	Customizing a Bootstrap Configuration	13-3
13.2.1	Bootstrap Configuration File Statement Definitions	13-3
13.2.2	Sample Customization	13-4
13.3	Running the Bootstrap Procedure	13-5
13.3.1	UNIX	13-5
13.3.2	NT	13-5
Chapter 14: Automatic UPS Product Startup and Shutdown		14-1
14.1	Configuring Your Machine to Allow Automatic Startup/Shutdown ..	14-1
14.2	Installing a UPS Product to Start and/or Stop Automatically	14-2
14.2.1	Determine if Auto Start/Stop Feature is Enabled	14-2
14.2.2	Determine if Product is Appropriate for Autostart	14-3
14.2.3	Edit Control File(s)	14-3
14.2.4	Summary	14-4
14.3	Disabling UPS Automatic Start/Stop of Processes	14-4
14.4	A Summary of the UPS Automatic Start-up Process	14-5

Part IV: Product Developer's Guide

(Part IV is listed is listed in the front section of the table of contents.)

Chapter 15: UPS Product Development: General Considerations	15-1
Chapter 16: Building UPS Products	16-1
Chapter 17: Making Products Available For Distribution	17-1
Chapter 18: Using template_product to Build and Distribute UPS Products	18-1
Chapter 19: Checklist for Building and Distributing Products	19-1

Part V: Distribution Node Maintainer's Guide

Chapter 20: Product Distribution Server Configuration	20-1
20.1 How A Server Responds to a UPD Client Command	20-1
20.1.1 The Process for upd addproduct	20-2
20.1.2 The Process for upd install	20-2
20.2 Accounts Required for Distribution Server	20-3
20.2.1 The updadmin Account	20-3
20.2.2 The ftp Account	20-3
20.2.3 The wwwadm Account	20-4
20.3 Web Server Configuration	20-5
20.3.1 The cgi Scripts Used to Access Distribution Database	20-5
20.3.2 Restricting Access to Distribution Database	20-6
20.3.3 Prerequisites for Modifying the Distribution Database	20-7
20.3.4 Permissions on Files Created in the Distribution Database	20-7
20.4 FTP Server Configuration	20-7
20.5 UPD Configuration Items	20-9
20.5.1 Archive File Keywords and \${SUFFIX}	20-9
20.5.2 Pre- and Postdeclare ACTIONS	20-10
20.6 Administrative Tasks and Utilities	20-10
20.6.1 Reporting FTP and Web Server Activity Using Ftpweblog ..	20-10
20.6.2 Restricting Access for Uploads to Distribution Database	20-11
20.6.3 Restricting Access for Downloads from Distribution Database	20-11
20.6.4 Restricting Distribution of Particular Products	20-11
20.6.5 Flagging Special Category Products Using Optionlist	20-12
20.6.6 Searching FTP Server Logfiles Using Searchlog	20-13
20.7 Product Distribution via CD-ROM	20-14
Chapter 21: Configuration of the fnkits Product Distribution Node	21-1
21.1 UPS Configuration for KITS Database	21-1
21.2 UPS Configuration for local Product Database	21-1

21.3 UPD Configuration	21-2
21.3.1 updconfig File Organization	21-2
21.3.2 The Recognized Product Categories	21-3
21.3.3 Matching Product Categories to updconfig Stanzas	21-3
21.3.4 Location and File Name Definitions	21-4
21.3.5 Pre- and Postdeclare ACTIONS	21-4
21.4 fnkits Server Maintenance	21-6
21.4.1 User Accounts and Group Ids	21-6
21.4.2 Database and Configuration File Locations	21-6
21.4.3 Web Server and FTP Log File Information	21-7

Part VI: UPS and UPD Command Reference

Chapter 22: UPS Command Reference	22-1
22.1 setup	22-3
22.1.1 Command Syntax	22-3
22.1.2 Commonly Used Options	22-3
22.1.3 All Valid Options	22-3
22.1.4 More Detailed Description	22-5
22.1.5 setup Examples	22-6
22.2 unsetup	22-9
22.2.1 Command Syntax	22-9
22.2.2 All Valid Options	22-9
22.2.3 More Detailed Description	22-11
22.2.4 unsetup Examples	22-12
22.3 ups configure	22-13
22.3.1 Command Syntax	22-13
22.3.2 Commonly Used Options	22-13
22.3.3 All Valid Options	22-13
22.3.4 More Detailed Description	22-15
22.3.5 ups configure Examples	22-15
22.4 ups copy	22-17
22.4.1 Command Syntax	22-17
22.4.2 Commonly Used Options	22-17
22.4.3 All Valid Options	22-17
22.4.4 Options Valid with -G	22-19
22.4.5 More Detailed Description	22-19
22.4.6 ups copy Examples	22-20
22.5 ups declare	22-21
22.5.1 Command Syntax	22-21
22.5.2 Commonly Used Options	22-21
22.5.3 All Valid Options	22-22
22.5.4 More Detailed Description	22-24
22.5.5 ups declare Examples	22-26

22.6	ups depend	22-27
22.6.1	Command Syntax	22-27
22.6.2	Commonly Used Options	22-27
22.6.3	All Valid Options	22-27
22.6.4	ups depend Examples	22-29
22.7	ups exist	22-31
22.7.1	Command Syntax	22-31
22.7.2	Commonly Used Options	22-31
22.7.3	All Valid Options	22-31
22.7.4	More Detailed Description	22-33
22.7.5	ups exist Examples	22-33
22.8	ups flavor	22-35
22.8.1	Command Syntax	22-35
22.8.2	Commonly Used Options	22-35
22.8.3	All Valid Options	22-35
22.8.4	More Detailed Description	22-36
22.8.5	ups flavor Examples	22-37
22.9	ups get	22-39
22.9.1	Command Syntax	22-39
22.9.2	All valid options	22-39
22.9.3	ups get Example	22-40
22.10	ups help	22-41
22.10.1	ups help Example	22-41
22.11	ups list	22-43
22.11.1	Command Syntax	22-43
22.11.2	Commonly Used Options	22-43
22.11.3	All Valid Options	22-43
22.11.4	More Detailed Description	22-45
22.11.5	ups list Examples	22-49
22.12	ups modify	22-55
22.12.1	Command Syntax	22-55
22.12.2	Commonly Used Options	22-55
22.12.3	All Valid Options	22-55
22.12.4	More Detailed Description	22-56
22.12.5	ups modify Example	22-57
22.13	ups start	22-59
22.13.1	Command Syntax	22-59
22.13.2	Commonly Used Options	22-59
22.13.3	All Valid Options	22-59
22.13.4	More Detailed Description	22-61
22.13.5	ups start Examples	22-61
22.14	ups stop	22-63
22.14.1	Command Syntax	22-63
22.14.2	Commonly Used Options	22-63
22.14.3	All Valid Options	22-63

22.14.4	More Detailed Description	22-65
22.14.5	ups stop Examples	22-65
22.15	ups tailor	22-67
22.15.1	Command Syntax	22-67
22.15.2	Commonly Used Options	22-67
22.15.3	All Valid Options	22-67
22.15.4	More Detailed Description	22-69
22.15.5	ups tailor Example	22-69
22.16	ups touch	22-71
22.16.1	Command Syntax	22-71
22.16.2	Commonly Used Options	22-71
22.16.3	All Valid Options	22-71
22.16.4	ups touch Example	22-72
22.17	ups unconfigure	22-73
22.17.1	Command Syntax	22-73
22.17.2	Commonly Used Options	22-73
22.17.3	All Valid Options	22-73
22.17.4	More Detailed Description	22-75
22.17.5	ups unconfigure Example	22-75
22.18	ups undeclare	22-77
22.18.1	Command Syntax	22-77
22.18.2	Commonly Used Options	22-77
22.18.3	All Valid Options	22-78
22.18.4	More Detailed Description	22-79
22.18.5	ups undeclare Examples	22-80
22.19	ups verify	22-81
22.19.1	Command Syntax	22-81
22.19.2	Commonly Used Options	22-81
22.19.3	All Valid Options	22-81
22.19.4	ups verify Example	22-83
Chapter 23: UPD/UPP Command Reference		23-1
23.1	upd addproduct	23-3
23.1.1	Command Syntax	23-3
23.1.2	Commonly Used Options	23-4
23.1.3	All Valid Options	23-4
23.1.4	More Detailed Description	23-7
23.1.5	Adding Products to fnkits.fnal.gov	23-8
23.1.6	upd addproduct Examples	23-9
23.2	upd cloneproduct	23-11
23.2.1	Command Syntax	23-11
23.2.2	All Valid Options	23-11
23.2.3	Options Valid with -G	23-12
23.2.4	upd cloneproduct Example	23-12
23.3	upd delproduct	23-13
23.3.1	Command Syntax	23-13
23.3.2	Commonly Used Options	23-13

23.3.3	All Valid Options	23-13
23.3.4	upd delproduct Example	23-14
23.4	upd depend	23-15
23.4.1	Command Syntax	23-15
23.4.2	Options	23-15
23.4.3	upd depend Examples	23-15
23.5	upd exist	23-17
23.5.1	Command Syntax	23-17
23.5.2	Options	23-17
23.5.3	upd exist Examples	23-17
23.6	upd fetch	23-19
23.6.1	Command Syntax	23-19
23.6.2	Commonly Used Options	23-19
23.6.3	All Valid Options	23-19
23.6.4	upd fetch Examples	23-21
23.7	upd get	23-23
23.7.1	Command Syntax	23-23
23.7.2	Options	23-23
23.8	upd install	23-25
23.8.1	Command Syntax	23-25
23.8.2	Commonly Used Options	23-25
23.8.3	All Valid Options	23-25
23.8.4	Options Valid with -G	23-28
23.8.5	More Detailed Description	23-28
23.8.6	upd install Examples	23-29
23.9	upd list	23-31
23.9.1	Command Syntax	23-31
23.9.2	Options	23-31
23.9.3	upd list Examples	23-31
23.10	upd modproduct	23-33
23.10.1	Command Syntax	23-33
23.10.2	Commonly Used Options	23-33
23.10.3	All Valid Options	23-34
23.10.4	More Detailed Description	23-35
23.10.5	upd modproduct Examples	23-36
23.11	upd reproduct	23-39
23.11.1	Command Syntax	23-39
23.11.2	Options	23-40
23.11.3	upd reproduct Examples	23-40
23.12	upd update	23-41
23.12.1	Command Syntax	23-41
23.12.2	Commonly Used Options	23-41
23.12.3	All Valid Options	23-41
23.12.4	upd update Examples	23-43

23.13 upd verify	23-45
23.13.1 Command Syntax	23-45
23.13.2 Options	23-45
23.14 upp	23-47
23.14.1 Command Syntax	23-47
23.14.2 All Valid Options	23-47
23.14.3 upp Examples	23-47
Chapter 24: Generic Command Option Descriptions	24-1
24.1 Alphabetical Option Listing	24-1
24.2 More Information on Selected Options	24-7
24.2.1 -e	24-7
24.2.2 -H	24-7
24.2.3 -K	24-7
24.2.4 -q	24-8
24.2.5 -V	24-9
Chapter 25: UPS/UPD Command Usage	25-1
25.1 Syntax	25-1
25.1.1 Order of Command Line Elements	25-1
25.1.2 Specifying Version/Chain	25-1
25.1.3 Grouping Option Flags	25-2
25.1.4 Specifying Arguments to Options	25-2
25.1.5 Embedded Spaces in Option Arguments	25-2
25.1.6 Invalid Option Arguments	25-3
25.1.7 Specifying Multiple Products in a Single Command	25-3
25.1.8 Multiple Occurrences of Same Option Flag	25-3
25.1.9 Use of Wildcards	25-4
25.2 Options	25-4
Chapter 26: Product Instance Matching in UPS/UPD Commands	26-1
26.1 Database Selection Algorithm	26-1
26.1.1 UPS	26-1
26.1.2 UPD	26-2
26.2 Instance Matching within Selected Database	26-3
26.2.1 Where Does Instance Matching Take Place?	26-3
26.2.2 Flavor Selection	26-3
26.2.3 Qualifiers: Use in Instance Matching	26-4
26.2.4 Flavor and Qualifier Matching Algorithm	26-4

Part VII: Administrator's Reference

Chapter 27: Information Storage Format in Database and Configuration Files

	27-1
27.1 Overview of File Types	27-1
27.2 Keywords: Information Storage Format	27-2
27.2.1 What is a Keyword?	27-2
27.2.2 Keyword Syntax	27-2
27.2.3 User-Defined Keywords	27-2
27.2.4 How UPS/UPD Sets Keyword Values	27-3
27.3 Flexibility of File Syntax	27-3
27.4 List of Supported Keywords	27-3
27.5 Syntax for Assigning Keyword Values	27-8
27.6 Usage Notes on Particular Keywords	27-9
27.6.1 COMPILE_DIR, COMPILE_FILE and @COMPILE_FILE ..	27-9
27.6.2 PROD_DIR_PREFIX, PROD_DIR and @PROD_DIR	27-9
27.6.3 STATISTICS	27-9
27.6.4 TABLE_FILE and @TABLE_FILE	27-10
27.6.5 UPS_DIR and @UPS_DIR	27-11
27.6.6 _UPD_OVERLAY	27-11
Chapter 28: Version Files	28-1
28.1 About Version Files	28-1
28.2 Keywords used in Version Files	28-2
28.3 Version File Examples	28-3
28.3.1 Sample Version File for exmh v1_6_6	28-3
28.3.2 Sample version file for foo v2_0	28-4
28.4 Determination of ups Directory and Table File Locations	28-5
Chapter 29: Chain Files	29-1
29.1 About Chain Files	29-1
29.2 Keywords Used in Chain Files	29-2
29.3 Chain File Examples	29-3
29.3.1 Sample chain file for exmh v1_6_6	29-3
29.3.2 Sample chain file for foo v2_0	29-3
Chapter 30: The UPS Configuration File	30-1
30.1 dbconfig File Organization	30-1
30.2 Keywords Used in dbconfig	30-1
30.3 Sample dbconfig File	30-2
Chapter 31: The UPD Configuration File	31-1
31.1 updconfig File Organization	31-1
31.2 Product Instance Identification and Matching	31-2

31.3	Defining Locations for Product Files	31-3
31.3.1	Required Locations	31-3
31.3.2	Read-Only Variables Usable in Location Definitions	31-4
31.3.3	Sample Location Definitions	31-5
31.4	Pre- and Postdeclare Actions	31-5
31.4.1	ACTION Keyword Values	31-6
31.4.2	The execute Function	31-6
31.5	Examples	31-7
31.5.1	Generic Template updconfig File	31-7
31.5.2	Distribution from the fnkits Node Only	31-8
31.5.3	Customized Treatment of ups Directory and Table Files	31-8
31.5.4	Implementing Multiple Configurations	31-9
31.5.5	Sample Configuration for AFS Space Using ACTIONS	31-10
31.5.6	Distribution Node Configuration	31-10
Chapter 32:	The UPP Subscription File	32-1
32.1	UPP Subscription File Header	32-1
32.2	Stanzas	32-2
32.2.1	Product Instance Identification	32-2
32.2.2	Conditions and Instructions	32-2
32.3	Examples	32-3
32.3.1	Sample UPP Subscription File	32-3
32.3.2	A Longer Annotated Example	32-4

Part VIII: Developer's Reference

(Part VIII is listed is listed in the front section of the table of contents.)

Chapter 33:	Actions and ACTION Keyword Values	33-1
Chapter 34:	Functions used in Actions	34-1
Chapter 35:	Table Files	35-1
Chapter 36:	Scripts You May Need to Provide with a Product	36-1
Chapter 37:	Use of Compile Scripts in Table Files	37-1
Chapter 38:	Creating and Formatting Man Pages	38-1
Glossary	GLO-1	
Index	IDX-1	

About this Manual

This chapter provides an introduction to the *Complete Guide and Reference Manual for UPS, UPD and UPP v4*. In particular you will find:

- the overall structure, the purpose and the intended audience of the manual
- what parts of the manual you need
- where to obtain this manual and where to look for updates
- the typeface conventions and symbols used throughout the document
- an invitation to readers to send us comments

This manual is published in three submanuals: GU0014A, GU0014B, and GU0014C. The structure of the document and its division into these sections is discussed in the following sections.

1. Document Structure, Purpose and Intended Audiences

The *UPS and UPD v4 Reference Manual* is intended for several different user groups as listed on the next page. To best accommodate the different types of users, the manual is divided into five user guides (Parts I-V):

- Part I *Overview and End User's Guide*
- Part II *Product Installer's Guide*
- Part III *System Administrator's Guide*
- Part IV *Product Developer's Guide*
- Part V *Distribution Node Maintainer's Guide*

and three reference manuals (Parts VI-VIII)

- Part VI *UPS and UPD Command Reference*
- Part VII *Administrator's Reference*
- Part VIII *Developer's Reference*

The user guides explain and illustrate the **UPS/UPD/UPP** tasks associated with each user group. The reference guides provide detailed information on commands, concepts, file structure/contents, and so on. On the following page is a guide to which parts of the manual you are likely to need, according to your job functions. Notice that we recommend Parts I and VI for all users:

Parts	User Functions
A: For All Users	
Part I <i>Overview and End User's Guide</i>	<i>End Users:</i> List product information in a UPS database on a user system; Access installed software products Access FermiTools ^a software products (<i>Other user groups' functions described later in table</i>)
Part VI <i>UPS and UPD Command Reference</i>	
B: For Product Installers, UPS Database Administrators, System Administrators of User Machines, Distribution Node Maintainers	
Part II <i>Product Installer's Guide</i>	<i>Product Installers:</i> Install software products from a UPS product distribution node into a UPS database on a user system; Install products into the AFS-space UPS database
Part III <i>System Administrator's Guide</i> and Part VII <i>Administrator's Reference</i>	<i>System Administrators, UPS Database Administrators:</i> Maintain UPS products in a UPS database; Install UPS/UPD/UPP on a user system; Configure UPS on a user system; Configure UPD on a user system; Configure UPP on a user system; Configure an installed product to start/stop automatically at boottime/shutdown
Part V <i>Distribution Node Maintainer's Guide</i>	<i>Distribution Node Maintainers:</i> Install UPS/UPD on a distribution system; Configure UPS and UPD on a distribution system; Configure Web and anonymous FTP servers on a distribution system Maintain UPS database on a distribution system
C: Product Developers	
Part IV <i>Product Developer's Guide</i>	<i>Product Developers and Maintainers:</i> Develop and maintain software products that are intended to be distributed in accordance with UPS standards; Adapt pre-existing or third-party software to conform to UPS standards; Distribute products
Part VIII <i>Developer's Reference</i>	

a. Fermilab-written software products that are made publicly available.



The table above lists rather generally the topics that the manual covers. Note that it is *not* the purpose of this document to provide information on:

- general UNIX system administration
- general UNIX or Fermilab information (see instead *UNIX at Fermilab*, GU0001)
- the use of any particular software product other than **UPS/UPD/UPP**

CDF and D0 collaborators: Also see *A UNIX Based Software Management System* (GU0013) at http://www-cdf.fnal.gov/offline/code_management/run2_cmgt/run2_cmgt.html to find information describing how **UPS** and **UPD** have been implemented in your experiments' code management systems.

2. Availability

Copies of the *UPS and UPD v4 Reference Manual* (GU0014A, B, and C), can be obtained from the following sources:

Web

<http://www.fnal.gov/docs/products/ups/ReferenceManual/>

This can be accessed under **Documentation** on the Computing Division home page. Search using any of the following keywords: afs, develop(ment), distribute(tion), fermitools, GU0014, install(ation), kits, maintain(enance), man page, product, system administration, unix, upd, upp, ups

Paper Copies

Wilson Hall, 8th floor, NE (just across from what used to be the Computing Division library)

3. Updates

Pending subsequent releases of this manual, updates will be maintained on the Web with the on-line version of the manual. To get there from the Computing Division home page, select **Documentation**, request *GU0014* and follow the pointers (see “Web” under section 2. *Availability*).

4. Conventions

The following notational conventions are used in this document:

- | | |
|---------------|--|
| bold | Used for product names (e.g., UPS). |
| <i>italic</i> | Used to emphasize a word or concept in the text. Also used to indicate logon ids and node names. |
| typewriter | Used for filenames, pathnames, contents of files, output of commands. |

typewriter-bold	Used to indicate commands and prompts.
[...]	In commands, square brackets indicate optional command arguments and options.
	When shown in a command example (e.g., x y z), separates a series of options from which one may or must be chosen (depends if enclosed in square brackets). In UNIX commands, used to pipe output of preceding command to the following one.
' ... '	Single vertical quotes indicate apostrophes in commands.
" ... "	Double vertical quotes indicate double quotes in commands
...	In a command, means that a repetition of the preceding parameter or argument is allowed.
%	Prompt for C shell family commands (% is also used throughout this document when a command works for both shell families).
\$	Prompt for Bourne shell family commands; also standard UNIX prefix for environment variables (e.g., \$VAR means “the value to which VAR is set”).
\	UNIX standard quoting character; used in commands throughout the manual to indicate that the command continues to the next line
<...>	In commands, variables, pathnames and filenames, angle brackets indicate strings for which reader must make a context-appropriate substitution. For example, \$<PRODUCT>_DIR becomes \$EMACS_DIR for the product emacs .
{ }	In local read-only variables, e.g., \${UPS_PROD_DIR}, string should be used as shown with the {}.

All command examples are followed by an implicit carriage return key.

Some of the files discussed in this document are shell family-specific, and thus come in pairs. Their filenames carry the extensions `.sh` and `.csh`. We often refer to a pair of these files as `filename.[c]sh`.

The following symbols are used throughout this document to draw your attention to specific items in the text:



A “bomb”; this refers to something important you need to know in order to avoid a pitfall.



This symbol is intended to draw your attention to a useful hint.

5. Your Comments are Welcome!

The *UPS and UPD v4 Reference Manual* may contain some errors, however we endeavor to minimize the error count! We encourage all the readers of this document to report back to us:

- errors or inconsistencies that we have overlooked
- any parts of the manual that are confusing or unhelpful -- please offer *constructive* suggestions!
- other topics to include (keeping in mind the purpose of the manual)
- tricks, hints or ideas that other users might find helpful

Send your comments via email to cdlibrary@fnal.gov.

Part IV Product Developer's Guide

Chapter 15: *UPS Product Development: General Considerations*

This chapter discusses the **UPS** product development methodology and tools that can be used in product development. It also provides recommendations for organizing your local product development area and the individual product root directories you will need to create.

Chapter 16: *Building UPS Products*

In this chapter we describe the steps you need to take in order to prepare a product for inclusion into the **UPS** framework and then to prepare it for distribution. We go through the steps for a simple case, then discuss the additional steps that may be required in more complex situations. Some sample auxiliary files are provided at the end.

Chapter 17: *Making Products Available For Distribution*

This chapter describes the processes of adding, updating, deleting and “cloning” product instances or components on a product distribution system. Information on creating tar files, using Fermilab **CVS** repositories and announcing products is also provided.

Chapter 18: *Using `template_product` to Build and Distribute UPS Products*

In this chapter we describe the **template_product** product, and show how to use it to build and distribute a product.

Chapter 19: *Checklist for Building and Distributing Products*

In this chapter we summarize the steps for preparing to build a product, building it and distributing it. We include information about making the appropriate announcements when a new or upgraded product is available.

Chapter 15: UPS Product Development:

General Considerations

This chapter discusses the UPS product development methodology and tools that can be used in product development. It also provides recommendations for organizing your local product development area and the individual product root directories you will need to create.

15.1 Product Development Considerations and Recommendations

In this section we will provide some guidelines for product development as it affects the product's inclusion in the UPS framework.

Simple scripts which run on any architecture are naturally quite straightforward to implement under UPS. Products which are obtained from the outside world (third party) as executable images with no source code are also generally straightforward. The ones that get complicated are the products which must be compiled and/or otherwise built for each and every supported architecture.

15.1.1 All Products (Locally Developed and Third Party)

Shell Independence

The product should run the same way under both shell families, **sh** and **csh**. If the product requires any actions to take place before it will run (e.g., its `bin` directory added to your `$PATH`, some environment variables set), provide a table file containing these actions. The UPS environment is described in Chapter 1: *Overview of UPS, UPD and UPP v4* and table files in Chapter 35: *Table Files*. The functions supported in table files are designed to work in a shell-independent manner, in general.

Flavor Declaration in UPS

On your development system, we recommend that you declare your products according to the *fully specified flavor* of the machine on which you build them (or on which they were built). We consider this to be very important, especially if your target systems contain or will ever contain mixed OS releases (e.g., IRIX+6.2 and IRIX+6.5). This will help to avoid problems when a new OS release doesn't run images built on an older one, or vice-versa. You don't want to have to go back and comb out which OS release a particular product instance was built for, you want to be able to tell immediately from looking at the database. Installers and users also need this information to facilitate their database maintenance.

Products which have no flavor-dependence at all (shell scripts, for instance), should be declared as NULL to the database (use the “zero” option, `-0`; see Chapter 24: *Generic Command Option Descriptions*). For other products, include the entire flavor string of the build platform in the declaration, or the major portion of that string. For example, if you build on an OSF1 machine running V3.2, declare your products with the flavor OSF1+V3.2 or OSF1+V3 rather than just OSF1 (e.g., use the flavor level corresponding to the options `-3` or `-2` rather than `-1`)

Policy Regarding Use of `/usr/local/bin`



Outside Fermilab, in the UNIX world at large, products typically get put in `/usr/local/bin`. With this in the user's `$PATH`, all the products are accessible. This practice is **inconsistent** with the goal of **UPS** to provide concurrent versions of products. Therefore *only* products specially approved by the FUE working group may write into `/usr/local/bin`. **No other products should write to this area, or to any other area within `/usr/local`.**

15.1.2 Products that You Develop

If you're writing your own product for implementation within **UPS**, you have the luxury (and, we might add, the *responsibility*) of creating it such that it exploits the important features of **UPS**, thus making it easy for the user to install and run, and easy for you or another developer to maintain in the future. We urge you to follow the guidelines we present here.

Self-Containment and Location Determination

First, design the product such that it is self-contained. It should identify its location and the location of any required files *at run time* (as opposed to compile time). You as the product developer have total control over the structure and contents of the product root directory, but no control at all over where the product root directory will reside on a target system.



If you write the product such that it calculates its location at compile time, you'll be putting the hard-coded path to your development environment into the image -- most likely *not* the correct path on the user machine.

You can choose to define the environment variable `$<PRODUCT>_DIR`, which points to the product root directory. In **UPS** v4, this variable is no longer always necessary since much of its usefulness is taken over by the local read-only variable `${UPS_PROD_DIR}`, described in section 34.6 *Local Read-Only Variables Available to Functions*. However, users will still find `$<PRODUCT>_DIR` to be useful since they will have access to it as long as the product is setup.

As an example of the use of `${UPS_PROD_DIR}`, take **myproduct** written in **perl** which requires the file `lib/myprod-headers.pl`. You should refer to this file in the **perl** code as `$ENV:: {MYPROD_PERL_LIB} /myprod-headers.pl` rather than by its full path, e.g., `/path/to/lib/myprod-headers.pl`. In the table file, set `MYPROD_PERL_LIB` to `${UPS_PROD_DIR} /lib`. You should make no assumptions about where users will put the file.



As stated above, products should *not* use or copy files into the areas under `/usr/local`.

Reproducible Build Procedure

All products should be built using a build script in order to ensure that the build procedure is reproducible. If your product is at all complex, we recommend that you use Makefiles for this purpose. We have created a template product for creating **UPS** products, described in Chapter 18: *Using template_product to Build and Distribute UPS Products*. It includes Fermi-standard Makefiles, and automates much of the process. The general UNIX **make** utility and the associated Makefiles are beyond the scope of this document, but the subject is introduced in *UNIX at Fermilab*, and treated in many standard UNIX texts.

System Independence

The various flavors of UNIX have many differences. You will generally have to release separate instances of your (compiled) products for the different flavors. However, the more you are able to insulate your product from flavor/release dependencies, the easier your product will be to maintain, and the less rigid it will appear to installers and users.

15.1.3 Third-Party Products Requiring a Hard-Coded Path

If you're installing a third-party product, downloaded from the Web or elsewhere, you may not have the opportunity to code it such that it identifies its location at run time based on `${UPS_PROD_DIR}` or the `$(PRODUCT)_DIR` environment variable. Whereas many products never need to know their location (they only need to be in your `$PATH`, for example), many other products *do* need to know their location in order to locate auxiliary commands, libraries, utilities, and so on.

Techniques for Implementing these Products

For those that do, the technical note TN0086 *Use of "/usr/local/products" now deprecated*, on-line at <http://www.fnal.gov/docs/TN/TN0086/tn0086.html>, describes recommended techniques for implementing the products. Please refer to it for information. The three approaches it describes are, briefly:

- For a product that is already setup and which contains a script that requires an interpreter, start the script with `#!/usr/bin/env <interpreter>` (e.g., `#!/usr/bin/env perl`). The **env** program will run the first copy of the interpreter it finds on your command search path, and your script is then executable.
- Create a “wrapper” shell script which sets up the **UPS** environment, sets up your product, and then invokes the appropriate commands. (An example is **www v2_6a**.)
- Sometimes getting a product setup before one of its scripts is invoked is not practical, and wrapper scripts may be unacceptably slow to start up. In cases where the product is considered important enough by the FUE working group that it must work properly even in the absence of **UPS**, a “trampoline” executable is provided, usually in `usr/local/bin`. The wrapper script should contain `#!/path/to/trampoline`.

When the product is configured, its **CONFIGURE** action inserts the product path into the trampoline executable. The wrapper script is then executable. Note that these products generally need to be declared as *root*.



In the past for **UPS v3** we used the `/usr/local/products` convention. We include this information for reference purposes only. This convention had serious drawbacks. The old (now deprecated) procedure was standard only on fully FUE-compliant systems (defined in the on-line document DR0009), and required that you:

- configure, build, and/or (re-)code the product so that the hard-coded path it uses is `/usr/local/products/{product}/{version}` (e.g., `/usr/local/products/tk/v4_2a`).
- write a `configure` script which creates the directory `/usr/local/products/{product}`, and creates in it the symbolic link `{version}` back to the *real* product root directory (e.g., `/usr/local/products/tk/v4_2a` is a symbolic link to `/path/to/products/OSF1+V3/tk/v4_2a`).
- write a `current` script that creates a symbolic link called `current` in the same directory, pointing to the link for the instance which is declared as `current` (e.g., `/usr/local/products/tk/current` is a symbolic link to `/usr/local/products/tk/v4_2a`).

Examples of Products Requiring Hard-coded Paths

Here are examples of situations in which hard-coded paths are unavoidable:

- Pre-built products which have hard-coded paths.
- Products that you can rebuild, but which were not coded with the idea of calculating where the files sit at run time. You need to tell them where to look for files at compile time, and this leads to hard-coded paths in the images.¹
- Commands that are not executed in the context of a shell, but rather as a program. An example is the **mh** utility **slocal** (for automatically sorting and foldering your incoming **mh** mail). This command is called via a command line in one of the configuration files (`.forward`).

You can't use the construct `"| ${MH_DIR}/lib/slocal -user joe"` to identify **slocal** because the program running this command will not expand the `${MH_DIR}` environment variable. You also don't want to spell out the whole actual path because you'd have to edit the `.forward` file every time a new version of **mh** is released.

- **cgi** scripts, **rsh** scripts and other situations in which you can't be sure that the product will necessarily have been setup when it is called by another one, and it needs to work anyway. We recommend that you consult with the UAS group (uas-group@fnal.gov) to determine the best course of action. Frequently you can create product configuration scripts that copy or link the product files into the correct location on the target node. In some cases for **cgi** scripts, you can have your Web server setup the product and pass the relevant environment variables.

1. Most vendors (freeware, shareware, and the few paid packages where you get the source code and rebuild it) now make it possible to modify the Makefiles so that you can decide where you want the output files, images, and so on, to go. Unfortunately, these are still frequently hard-coded at compile time, not run time. Therefore, packages that you build in this manner on your development system will not be right when installed on a user system with a different product root directory path.

15.2 Tools for Developing and/or Packaging Products

The tools that we introduce in this section can be used separately or together. They are all available as **UPS** products in **KITS**. See the on-line documentation *Integrating buildmanager, cvs, template_product, and upd* at <http://www.fnal.gov/docs/products/buildmanager/Integrating.html>.

15.2.1 Buildmanager

The **buildmanager** application is a configurable tool which lets you build software on multiple systems simultaneously, in an organized and consistent fashion. It allows you to set up standardized build sequences and define actions to be performed automatically. It can stop if things go wrong, and allows interaction with various build systems to correct problems. It is available as a **UPS** product in **KITS**. Any system to which you can telnet and run commands can be used as a build system with **buildmanager**. See the on-line documentation at <http://www.fnal.gov/docs/products/buildmanager/>.

15.2.2 CVS

It is a common practice to maintain a product's source code as well as its Makefile and **UPS** management files in a **CVS** repository for development and maintenance. **CVS** allows each developer to check out files into a private working directory and to modify them as necessary. With **CVS** you can maintain all the different versions and flavors in a single work area, and you can pull them out to the separate nodes as needed. Developers working with prebuilt binaries (downloaded from the Web or purchased from a vendor) can use **CVS** for just the Makefile and **UPS** management files (e.g., the local **README** and **INSTALL** files, the table file, tests, documentation, and so on) so that they can be properly source-controlled. Documentation for **CVS** can be found on-line at <http://www.fnal.gov/docs/products/cvs/>.

It is useful to be able to use **UPS** to setup these checked-out areas. One way that this can be accomplished is by declaring the checked-out area to either the main or a private **UPS** database, but this is often cumbersome, as these checked-out areas are by nature fairly transient.

A better solution is to exploit the **UPS** capability of setting up a product instance without having it actually declared to any database. To do this, you simply need to supply the **setup** command with all of the necessary information, shown here:

```
% setup <product> -r /your/checked/out/area -M <tableFileDir> \  
-m <tableFile> -q <qualifierList> -f <flavor>
```

15.2.3 Template_product

To simplify and somewhat automate the process of building **UPS** products, we have designed the product **template_product**. Once this product is installed on your system, it can be cloned into a new product area and customized to the new product. **template_product** can be used to build products of all types (shell script, pre-built binary, source code). We discuss this product in detail in Chapter 18: *Using template_product to Build and Distribute UPS Products*.

15.3 Directory Structure for a UPS Product Instance

The top level directory of a **UPS** product instance is called the *product root directory*, and in general it should contain files and subdirectories in which almost everything related to the product instance resides: the executables, the library files, the documentation, and so on. The `ups` directory files (i.e., the **UPS** metadata) and the table file usually reside here, but are not required to do so.

UPS is very lenient in the directory structure it allows. Nothing is required in all situations beyond a product root directory. Normally product instances have a table file containing actions that are run during operations like product installation and setup.

We recommend that you follow a few directory structure guidelines simply to conform to a generally recognized format. This will make it easy for yourself and others to identify each file and directory later on. The following is a relatively complete sample directory structure underneath the product root directory. Most products won't require all of these elements. On the other hand, you may include other directories and/or files not listed here. Elements which we *strongly recommend* that you provide (in addition to the executables) for every product include a `README` file, man pages, a user guide, test scripts and example files.

`README` text file containing information such as origin of the product (by whom, from where, etc.), support level, support group/person, caveats and known bugs (may be contained in the `ups` directory)

`bin` directory containing the executable(s)

`ups` directory containing metadata files and other executable and data files used during implementation and invocation; may also contain `INSTALL_NOTE` (described below) file and the directories `toman`, `toInfo`, `tonews` and `tohtml`. Often the table file resides here. (This directory is no longer a required element of a **UPS** product.)

Default location of the `ups` directory is directly underneath the product root (for compatibility with **UPS** v3), but it may reside anywhere.

`ups / INSTALL_NOTE` text file containing a detailed description of any installation actions that are more easily performed directly by the installer rather than by a script (beyond or instead of running `configure` and/or `tailor` and/or `current`). This should not be a script. This file is not usually needed. If provided, mention it in the `README` file so that product installers know to run it.

<code>lib</code>	directory containing libraries
<code>src</code>	directory containing source code
<code>include</code>	directory containing include files
<code>doc</code>	directory containing a user guide and any other documentation as appropriate; should include the source files (e.g., LaTeX, Word) as well as the printable files (e.g., PostScript)
<code>man</code>	<p>directory containing unformatted man pages. The files get copied into the location specified in <code>\$PRODUCTS/.upsfiles/dbconfig</code> (keyword <code>MAN_TARGET_DIR</code>).</p> <p>Default location (for compatibility with UPS v3): <code>ups/toman/man</code></p>
<code>catman</code>	<p>directory containing formatted man pages. The files get copied into the location specified in <code>\$PRODUCTS/.upsfiles/dbconfig</code> (keyword <code>CATMAN_TARGET_DIR</code>).</p> <p>Default location (for compatibility with UPS v3): <code>ups/toman/catman</code></p>
<code>html</code>	<p>directory containing the html version of the user guide and any other documentation as appropriate (automatic copy of files to standard area defined by <code>HTML_TARGET_DIR</code> not implemented in UPS v4)</p> <p>Default location (for compatibility with UPS v3): <code>ups/tohtml</code></p>
<code>news</code>	<p>directory containing news files to be posted to a newsgroup (automatic copy of files to standard area defined by <code>NEWS_TARGET_DIR</code> not implemented in UPS v4)</p> <p>Default location (for compatibility with UPS v3): <code>ups/tonews</code></p>
<code>Info</code>	<p>directory containing any text files that are to be displayed as a login announcement via the Info feature. The files get copied into the location specified in <code>\$PRODUCTS/.upsfiles/dbconfig</code> (keyword <code>INFO_TARGET_DIR</code>). Info is generally used to communicate to users about the Fermilab computing systems events, (e.g., shutdowns), software upgrades and other systems-related information.</p> <p>Default location (for compatibility with UPS v3): <code>ups/toInfo</code></p>
<code>test</code>	directory containing test scripts and any other test-related files
<code>examples</code>	directory containing example files to help users learn how to use the product

Product Documentation Storage

Each different type of product documentation (e.g., man pages, html files, PostScript files, and so on) must reside in a separate subdirectory. The subdirectories usually reside under the product root directory, but do not have to. In the product's table file, you should use the

keywords `XXX_SOURCE_DIR` as listed in section 27.4 *List of Supported Keywords* (e.g., `MAN_SOURCE_DIR`, `INFO_SOURCE_DIR`) to identify the directory in which each form of documentation is maintained. For example:

```
CATMAN_SOURCE_DIR = ${UPS_PROD_DIR}/catman
MAN_SOURCE_DIR = ${UPS_PROD_DIR}/man
INFO_SOURCE_DIR = ${UPS_PROD_DIR}/Info
```

UPS currently requires that all files in a directory specified by `XXX_SOURCE_DIR` be of the corresponding file type; you cannot mix file types.

The `ups/toman` directory and its subdirectories `man`, `catman` and `toInfo` are used as defaults in **UPS** v4 for backwards compatibility. This structure is not necessarily our recommendation. If a product comes with man pages, Info files, html files and so on, we recommend that you leave them where they are, and simply specify their locations in the table file. If you are writing your own, you can put them in subdirectories directly under the product root directory, which is generally the most convenient place.

In **UPS** v4, `NEWS_SOURCE_DIR` and `HTML_SOURCE_DIR` are not implemented.

Chapter 16: Building UPS Products

In this chapter we describe the steps you need to take in order to prepare a product for inclusion into the **UPS** framework and then to prepare it for distribution. We go through the steps for a simple case, then discuss the additional steps that may be required in more complex situations. Some sample auxiliary files are provided at the end.

16.1 Basic Steps for Making a UPS Product

In this section we will go through the steps of making a simple, unflavored product compatible with **UPS**. The steps we illustrate in this section are also valid for more complicated situations, but additional steps are generally needed in those cases. These will be noted later in the chapter. We'll use the standard "Hello world" example, with a product **hello**, version `v1_0`, of flavor `NULL`. The executable, which is a script in this case, consists of the following text:

```
#!/bin/sh
echo "Hello world"
```

This is a simple case. You don't need any Makefiles or scripts on how to build this product, because it doesn't get built. It runs on all flavors of UNIX without modification, so you should declare it with the flavor `NULL`. It would be nice to have the `$HELLO_DIR/bin` directory added to your `$PATH` to use the product, and that's what the setup action will do. The unsetup action will remove `$HELLO_DIR/bin` from your `$PATH`. No configuration or tailoring is needed, nor are any special actions when the product is declared current.

The steps you need to complete are:

- 1) Create a directory hierarchy for the product and its related files.
- 2) Create a `README` file.
- 3) Create a table file in the location you want it to reside (usually either in the product-specific directory directly underneath your **UPS** development database or in the `ups` directory, if your product has one).
- 4) Declare the product to your **UPS** development database with the *development* chain so that it doesn't interfere with other peoples' work. Although the product itself doesn't exist yet, the declaration can be done and we recommend it at this stage for convenience.
- 5) Create the product script in the `bin` directory (or copy it into there).

- 6) Create man pages (a user's guide is recommended also).
- 7) Test the product.

16.1.1 Build the Directory Hierarchy

We will take the product root directory to be `hello/v1_0`. This product root directory can sit anywhere in the file system. An appropriate, simple directory structure underneath the product root directory is as follows:

<code>bin</code>	contains the executable script <code>hello</code>
<code>man</code>	contains the unformatted man page(s)
<code>catman</code>	contains the formatted man page(s)
<code>test</code>	contains the test script(s)

A `README` file should go directly under `hello/v1_0`. We'll put the table file, called `hello.table`, under the database. Remember that most products would have more subdirectories and files than shown here, in particular a `ups` directory as well as `html` and/or `doc` for the user's guide.

16.1.2 Create the Table File

For our example, we'll create the file `hello.table` and put it in the product subdirectory of the development database. A simple table file for this product might look like:

```
FILE=TABLE
PRODUCT=hello
VERSION=v1_0
#
#-----
FLAVOR = ANY
QUALIFIERS = ""

ACTION=SETUP
  pathPrepend(PATH, ${UPS_PROD_DIR}/bin, :)
  setupEnv()
```

16.1.3 Declare the Product to your Development UPS Database

Refer to section 10.1 *Declare an Instance* for instructions on declaring the product to your UPS database, or see the reference section 22.5 *ups declare*. In particular, note two things:

- 1) For an unflavored script like this example, declare the flavor specifically as `NULL` (using either the `-f NULL` or `-0` option).
- 2) Declare it with the chain `development` for your pre-distribution testing (using the `-d` option).

For example:

```
% ups declare -0dz /ups_dev_db -r /ups_dev_prod/hello/v1_0 -m\
hello.table hello v1_0
```

We recommend declaring at this stage for reasons of convenience and organization. It allows you to run **setup [-d]** on the product to make the `$<PRODUCT>_DIR` environment variable available.

16.1.4 Copy the Product Executable to the bin Directory

Create the script in the `bin` directory, or copy or move it to this location.

16.1.5 Provide Product man Pages

See Chapter 38: *Creating and Formatting Man Pages* for more complete instructions on creating man pages.

Create the (unformatted) **nroff** source `$HELLO_DIR/man/hello.1`. It may look similar to this:

```
.TH HELLO 1 LOCAL
.SH NAME
hello - print "Hello world" on stdout
.SH SYNOPSIS
.B hello
.SH DESCRIPTION
.B hello
prints the string "Hello world" on standard output.
```

Use this source to create the formatted man page using the commands¹:

```
% cd $HELLO_DIR/man
% nroff -man hello.1 > ../catman/hello.1
```

Once it is formatted, it will look like this:

```
HELLO(1)                                HELLO(1)

NAME
hello - print "Hello world" on stdout

SYNOPSIS
hello

DESCRIPTION
hello prints the string "Hello world" on standard output.
```

1. If **nroff** is not available, run **setup groff** to get the GNU version.

16.1.6 Test the Product

Now you can setup and test your product. As an example, for our product we might run:

```
% setup hello v1_0
% hello
    Hello world
% unsetup hello v1_0
% hello
```

```
sh: hello: command not found
```

In many cases, writing a good test script can be rather challenging. Include at least a basic test to ensure that the product works properly. For our example, the test script just needs to run our **hello** program and verify its output, e.g.,:

```
#!/bin/sh
hello | grep "Hello world" > /dev/null
```

This will exit with a successful exit code if **hello** prints `Hello world`, and fail otherwise.

16.2 Specifics for Different Categories of Products

This section discusses all the steps you need for turning virtually any product into a **UPS** product. We start with the simpler cases and finish with the more complex ones. For all categories of product, if your product has dependencies, either for building or for execution, you need to have them available to you on your development system when you build and test the product.

16.2.1 Unflavored Scripts

Unflavored scripts, that is scripts with the flavor `NULL`, are the simplest form of **UPS** product. The example in section 16.1 shows how easy it is to create a **UPS** product from an unflavored script. A product like this does not need to be rebuilt on different architectures, and generally does not need `CONFIGURE` and `UNCONFIGURE` actions or scripts. Some, although *very few*, unflavored scripts require `INSTALLASROOT` actions in the table file to copy specific files into `/usr/local/bin`, or to perform similar actions.

We strongly discourage use of `/usr/local/bin` or any other hard-coded path; see section 15.1.1 under 15.1 *Product Development Considerations and Recommendations*.



16.2.2 Pre-built Binaries

Many third-party products obtained from a vendor or downloaded from the Web are binary images without source code. When you go to a vendor's web site, you will often find separate pre-built binaries for several UNIX operating systems/releases. Note that they may use slightly different terminology than we do to refer to the different flavors.

Generally, to run products that consist of executables (as opposed to libraries, for example), you just need to add the executable directories to your `$PATH` after downloading. To make a product compatible with **UPS**, you should provide a table file that modifies the `$PATH`, a `README` file and some documentation. If the vendor provides examples and/or any other user files, include them. Most products distributed in this manner include documentation, either man pages or html files, and sometimes both.

Follow this general procedure:

- Create one master product root directory. Underneath it, create the product directory structure, including at least a `bin` directory.
- Create the appropriate product subdirectories (`html` for Web documents, `doc` for PostScript or other forms of documentation, `man` and/or `catman` for unformatted and/or formatted man pages, respectively) and copy the vendor's documentation into them. You can opt to leave the documentation directory structure the same as the way it is provided.
- Create a `README` file in the product root directory with relevant information such as where this product was obtained, by whom, any licensing restrictions or other notes, and so on.
- Create a table file. It can be modified later as needed, but at least a rudimentary table file must exist in its actual location before declaring the product. In most cases, within the table file, the product instance's `bin` directory should be added to the `$PATH` within a `SETUP` action, e.g.,:

```
ACTION=SETUP
  pathPrepend(PATH, ${UPS_PROD_DIR}/bin, :)
```
- Create other `ups` directory scripts and data files as needed in the `ups` subdirectory. (For most pre-built binaries you shouldn't need any.)
- Declare the product to a **UPS** database with the chain *development* (`-d`) and no flavor (`-f NULL`).

Now it's time to create areas for each flavor of the product that you plan to install.

- Duplicate the product root directory tree once for each flavor of binary you plan to install (using **tar** or other appropriate tool).
- For each flavor, copy the pre-built binary into the appropriate `bin` directory. This usually involves unwinding a tar file.
- Declare the suite of product instances (one per flavor) to your **UPS** development database for testing before you distribute them (strongly recommended!).
- Set permissions for all readable files to `a+r`. Set permissions for all scripts and other executable files to `a+x`.
- Test each one out!

16.2.3 Products Requiring Build (In-House and Third-Party)

Most locally developed products, and many vendor-supplied products, are distributed as source code which must be rebuilt for each OS flavor. We are trying to get away from **UPS**-packaging vendor-supplied products, however, we provide instructions in case you need to do so.

If you are building a product which was obtained from an outside source, you may not have control over the product directory hierarchy. Some outside products include configuration options (via Makefiles) to specify where the resulting libraries and/or images should reside, but in other cases you must give a hard-coded path to the final output file. In the latter cases, when it is absolutely necessary, you may need to use **UPS** as a “bookkeeping” wrapper and common point of distribution. Contact *uas-group@fnal.gov* for assistance.

If you are developing the product yourself, you should follow these guidelines:

- Store the master source code (and all the auxiliary files) in a **CVS** code repository (or other code-version management system) according to your group’s policies.
- Use a sensible product directory hierarchy (`src`, `lib`, `bin`, `html`, `doc`, `ups` subdirectories). See section 15.3 for recommendations.
- If the product needs to know its location (or that of its include files or auxiliary files), use the local read-only variable `${UPS_PROD_DIR}` or the run-time environment variable `$(PRODUCT)_DIR` rather than a hard-coded path. Make sure that your table file sets this variable.

Preparation for Rebuilding Any Product

For any product, you first need to create the infrastructure. Much of the work needs to be done only once, and is reused for each flavor of the product that is built:

- Create the master source product directory hierarchy.
- Create/copy `ups` directory scripts, data files, and auxiliary files as needed in the `ups` subdirectory.
- Create at least a basic table file (include `QUALIFIERS=“BUILD”` or `“build”`, and set `$(PRODUCT)_DIR` under the `SETUP` action)
- Declare the product with the chain *development* and the flavor `NULL+SOURCE-ONLY` to a local **UPS** database. Make sure that all **UPS** product requirements are declared properly.
- Run `setup -d -q “?build?BUILD”` on the product to set `$(PRODUCT)_DIR`.
- Create source code in the `src` directory, or copy it there.
- Create a Makefile in the product root directory, `$(UPS_PROD_DIR)` (or simply write a build script if a Makefile is overkill) to use for building the product binaries. For reproducibility, make sure that you include *all* the steps to go from raw source to the completed product. It is a good idea to have the Makefile or build script run a test suite whenever possible.
- Modify the table file for `SETUP` and `UNSETUP` actions.
- Create documentation in the appropriate directories (`html` for Web documents, `doc` for PostScript or other forms of documentation, `man` and/or `catman` for unformatted and/or formatted man pages). Modify table file to note the locations.

If the documentation came from the vendor in other locations, you don’t need to move it; just indicate the locations in the table file.

- Keep track of any relevant information in a `${UPS_PROD_DIR} /README` file. This information should include where the source code came from, any tweaks that were necessary to make it build, the node names and OS versions that were used to build the binaries, known bugs, and so on.
- Set permissions to `a+x` for scripts and other executables, and to `a+r` for readable files.

Steps for Rebuilding a Product

Once you have created the product structure along with all of the support files, you will need to get down to the business of actually building the product images. If you're planning on redistributing this product to a wider audience than just your machine, you must be careful in selecting a build node. The build nodes should have appropriate levels of compilers, OS, and other products required for building the given product.

We recommend that you create separate build areas, one for each target flavor, so that the different flavors of binary files do not get mixed up. Once you have completed the preparation described above, complete these steps:

- Duplicate this source tree once for each target platform, using the file naming conventions that have been established for your build cluster (use `tar` or other appropriate tool, or you may need to check it out from version control).
- Declare these new directory trees each with its target flavor.

Then for each of the target flavors:

- Declare the product to the database using the flavor, optionally a chain of `-d`, and the case-appropriate qualifier `BUILD` or `build` (e.g., `-q BUILD`). If this is a product which creates links, make sure they were created properly and that each link points to the *correct* parent product root directory!
- Setup the product instance of that flavor in order to set `$<PRODUCT>_DIR` to the right product root directory. Use both the `-d` (for development chain, if declared) and `-q BUILD` (or `-q build`) options (i.e., `setup -dq BUILD <product>`).
- Invoke the product's build procedure or Makefile to rebuild the product from scratch.



If this is a product which is building files in a hard-coded path, check to make sure that these files are being created properly. They should reside under the `${UPS_PROD_DIR}` area, but via the symbolic links, they should *appear* to also reside under the hard-coded directory.

16.2.4 Overlaid Products

An overlaid product gets distributed and maintained in the product root directory of its main product. For example, the overlaid products `cern_bin`, `cern_ups`, `cern_lib`, etc., all reside in the product root directory for the main product `cern`. A patch is another good example of the use of overlaid products. The set of products overlaid on a main product is collectively referred to as *the overlay*.

A special keyword, `_UPD_OVERLAY`, is provided for inclusion in the table file of each overlaid product¹. `_UPD_OVERLAY` takes as its value the main product name in double quotes. Its presence indicates that the product is an overlaid product maintained in the root directory of the main product listed as the keyword's value. For example, the table files for the products `cern_bin`, `cern_ups`, and `cern_lib` would contain the following keyword line:

```
_UPD_OVERLAY = "cern"
```

UPD would then use `cern` as the product name when determining the root directory.



In addition to including all the overlaid products as dependencies of the main product, we recommend including the main product as a dependency of each of the overlaid products. This allows separate installation of each of the pieces. Circular dependency lists are allowed in **UPS**.

16.3 Sample Auxiliary Files

16.3.1 README

Following is the `README` file for the `teledata v1_0` product. It has been edited for brevity, but shows the kinds of information that are important to include:

```
This is the teledata product.
```

```
It contains the HTML files and data files for the Fermilab online
telephone directory.
```

```
The files in $TELEDATA_DIR/data are the data files, read by the
teleserver product. These files are updated daily.
```

```
The files in $TELEDATA_DIR/www are the html files, displayed by
the web server. These files are also updated daily; the A-Z.html
files are rebuilt from raw data, and the index.html, first.html
and last.html are given a new date stamp.
```

```
The HTML files must be visible from the web server's default HTML
area. This is accomplished via links in /usr/local/products
(managed by "ups configure" and "ups declare -c") and links in
the system default HTML directory (handled by the web
administrator). The /usr/local/products links will be created
automatically when the product is declared. The web
administrator must create the link in the top-level "default"
HTML directory, via something similar to
```

```
$ cd /path/to/default/html/area
$ ln -s /usr/local/products/teledata/current/www telephone
```

```
This allows the URL
```

```
http://www-tele.fnal.gov/telephone/
```

```
to map to the file
```

1. **UPS** regards `_UPD_OVERLAY` as a user-defined keyword, but it is defined within **UPD**.

```

$TELEDATA_DIR/www/index.html
...
The structure of the teledata product is:

$TELEDATA_DIR - parent product directory
ups - directory containing ups support files
  configure, unconfigure - manage the /usr/local/products/teledata links
  current, uncurrent - manage the /usr/local/products/teledata/current links
  INSTALL_NOTE - link to this file
data - directory containing data files
  RAWDATA - raw unprocessed data file
  NASTDATA - processed data file
  email - gdbm index file, keyed on email address
...
For further information, see the teleserver product, or please
contact support person name, telephone and email.

```

16.3.2 INSTALL_NOTE

The following is a sample `INSTALL_NOTE` from the **netscape v4_5** product:

```

Fermilab installation of Netscape

The Fermilab ups product imposes certain structure upon
its products. To this end, a wrapper has been provided
which will assist in the downloading and re-structuring
of netscape for use at Fermilab.

To use this tool:

1. Upd install the install_netscape product.

2. setup install_netscape.

3. cd to $INSTALL_NETSCAPE_DIR and execute the
   netscape_install script. The optional
   argument specifies the directory in which
   to install netscape. The default is to
   install and declare netscape in
   $INSTALL_NETSCAPE_DIR.

```

16.3.3 RELEASE_NOTES

The following is a sample `RELEASE_NOTES` file from **UPS v4_3**. Notice that for each release of the product, the new update information gets appended to the previous `RELEASE_NOTES` file contents so as to retain all the update information:

```

UPS v4_3b

Fixed bug in upsact when doing WriteCompileScript for a product already setup.
EnvSetIfNotSet now has no undo.
Better handling of envremove/pathremove, especially for cases where the value
parameter uses backticks.
Better handling of exeAccess, eliminating the use of 'hash' in the Korn shell family,
and printing error messages as appropriate.

UPS v4_3a

Fixed problem with ups verify outputting incorrect information about chains associated
with versions.

```

UPS v4_3

There are new template files in the ups area for the dbconfig file and the upsdb_list file.

Many fixes were made to the configuration script, particularly for NT.

When UPS uses dropit, it will now always use the '-e' switch, for an exact match.

...

Chapter 17: Making Products Available For Distribution

This chapter describes the processes of adding, updating, deleting and “cloning” product instances or components¹ on a product distribution system. Information on creating tar files, using Fermilab CVS repositories and announcing products is also provided.

17.1 Product Distribution Overview

A set of **UPD** commands has been developed for adding, updating and deleting products on a distribution node. They use the central Fermilab product distribution node *fnkits.fnal.gov* as the default distribution node, and declare products in the `KITS` database. The commands can be used to distribute products to any properly configured product server.

These **UPD** commands include:

upd addproduct	adds a product instance to a product distribution database
upd cloneproduct	creates a new product instance on a distribution node by copying one that is already there and changing one or more of its identifying elements
upd delproduct	deletes a product declaration from a distribution database; it also removes any associated tar file, table file and/or <code>ups</code> directory
upd modproduct	modifies a product instance that already exists in a distribution database; it allows you to replace a table file or <code>ups</code> directory, or to add or change chain information for the product
upd reproduct	is equivalent to a upd delproduct followed by a upd addproduct ; it can be used only when the replacement product instance has the same set of identifiers as the one destined for removal

These commands are fully described in Chapter 23: *UPD/UPP Command Reference*.

Before preparing to distribute a product, you should verify that it is complete, tested, and **UPS**-compliant. It is optional to create a tar file of your product prior to running **upd addproduct**, as discussed in section 17.2 *Creating Product Tar Files*. Keep in mind that the **UPD** configuration on your target distribution node determines the locations in which products get installed and declared on that node, and where their auxiliary files/directories get stored. The configuration on the distribution node may bear no resemblance to that of your local

1. “Components” are defined as table files and `ups` directories.

development system, or to that of an end user node. Once your product has been added to a distribution node, you need to make the appropriate announcements regarding product availability (see section 17.10 *Product Announcement Policies*).

17.2 Creating Product Tar Files

You can choose whether to make your own tar file before adding your product to a distribution node or to let **UPD** make it for you. The advantage of making it yourself is that you have control over its contents.



Note that it is not necessary that the product instance's table file or `ups` directory be included in the product root directory or, consequently, in the tar file. Some products may not have one, the other, or both of these components. On the other hand, other products (e.g., bundled products) may consist only of a table file, in which case no tar file is needed. If these components exist and are located outside of the product root directory, their location must be specified in one of two ways when adding the product to the distribution node:

- on the **upd addproduct** command line
- in a **UPS** database declaration on your development machine (database must be listed in `$PRODUCTS`)

When creating a tar file of a product using the **tar** command, perform the operation from the product root directory. This allows you to use simple relative path names to specify the files to include in the tar file. Use an absolute pathname (preferably to a temporary directory) to specify where to put the tar file. Do *not* use absolute pathnames to specify the files to include in the tar file.



Do *not* use the product root directory as the destination for the created tar file; it causes the tar file to try to include itself and to grow infinitely large.

The following steps illustrate the conventions for packing up a tar file for a **UPS** product called **fred** in such a way that (a) the tar file contains a relative path to the product root directory, and (b) the tar file is put in an appropriate temporary directory:

```
% setup [-d] fred
% cd $FRED_DIR
% tar cvf /tmp/fred_IRIX+5_v1_0.tar .
```

This creates a tar file called `/tmp/fred_IRIX+5_v1_0.tar` with all pathnames relative to the current directory (`$FRED_DIR`).



You should *not* replace the trailing dot in the example above with `$FRED_DIR` because that would force the tar file to contain an absolute path to the `$FRED_DIR` as set on *your* system, instead of a relative path to the `$FRED_DIR` on the *target* system where the tar file will be unwound.



Using **template_product** (described in Chapter 18: *Using template_product to Build and Distribute UPS Products*) allows you to customize the contents of your tar file. See section 18.8 *Customizing your Tar File*.

17.3 Adding a Product

Use the `upd addproduct` command to add a new product instance to a distribution server. If no host name is specified with the `-h` option, **UPD** uses the `fnkits.fnal.gov` host as the default. The required command line arguments differ depending on what components the product has and whether it's been declared to a local **UPS** database and/or archived with tar. Refer to the reference section 23.1 *upd addproduct* for the full command syntax and options for these different situations.

A few notes:



- When using `upd addproduct -h <host>`, use the full hostname (i.e., `hostname.fnal.gov` rather than just `hostname`) to prevent problems when people download the product to off-site user nodes.
- The `-P` option is available to prevent **UPS** from searching in a local database for the product instance. If you use it, you must specify sufficient information on the command line so that **UPS/UPD** can identify and locate all the product components.
- Chain information remains identical for the added product instance on the local and distribution nodes under most circumstances. If `-P` is used, local chain information is ignored, but can be set on the distribution node. You can use `upd modproduct` afterwards to change the chain.
- If the product is not declared to a local database, you must include `-m <tableFileName>` on the command line. You must also include `-M <tableFileDir>` if the table file is not in the current directory.

17.3.1 Product Categories Defined for KITS

The central Fermilab Computing Division product distribution database **KITS**, located on the server `fnkits.fnal.gov`, recognizes several different categories of product:

default	regular products added to the KITS database for distribution to any on-site or registered off-site node. ¹
FermiTools	locally-developed and supported software packages that we make available to the public
proprietary	products for which Fermilab has a limited number of licenses
fnalonly	products accessible only to the <code>fnal.gov</code> domain
usonly	US-only (United States only) products are accessible only to U.S. government (<code>.gov</code>) and military (<code>.mil</code>) domains

Most products fall into the default category, and can be added normally. For the other categories, you must first fill out the **Special UPD Product Registration** form (at <http://fnkits.fnal.gov/specialprod.html>) indicating which category of product it is, and submit the form. Then when you receive an email message saying that your product has been registered as a special product, go ahead and add it to `fnkits`. Do not use any

1. See the **Product Distribution Platform Registration Request** form at http://www.fnal.gov/cd/forms/upd_registration.html.

special options (i.e., do not use `-O "<options>"`) with `upd addproduct`; your product will automatically be configured to handle the special requirements according to your selection on the form.

17.3.2 Examples

Example 1

We have a product instance with a table file and a `ups` directory (in addition to all the product files) under the product root directory. The table file is in the `ups` directory. The product (we'll call it `foo` version `v1_0`), was developed for the flavor `SunOS+5`. The tar file has not been made ahead of time. In order for **UPD** to make the tar file for us, the product instance must be declared to a local **UPS** database listed in `$PRODUCTS`.

To add the product to `KITS`, the command can be entered from a `SunOS+5.x` machine as:

```
% upd addproduct foo v1_0 -2
```

Notice we've used the option `-2` which is equivalent in this case to `-f SunOS+5`. All of the other necessary information gets picked up from the local **UPS** declaration.

If we choose to ignore the local declaration via the `-P` option, we must supply the necessary information in the command:

```
% upd addproduct foo v1_0 -2 -P -r /path/to/prod/root/dir \  
-m v1_0.table -M /path/to/prod/root/dir/ups \  
-U /path/to/prod/root/dir/ups
```

Example 2

Let's use the same product as in Example 1, but assume that a tar file already exists. The pre-made tar file includes the entire structure under the product root directory. The tar file is located on our local machine in `/tmp/foo_v1_0_SunOS+5.tar`. We want to add it to `fnkits` and declare it to the `KITS` database with the full development machine flavor specification, no qualifiers, and no chain.

Assuming this product instance was declared to a local **UPS** database before the tar file was created, we use the command:

```
% upd addproduct foo v1_0 -2 -T /tmp/foo_v1_0_SunOS+5.tar
```

UPD can determine where to find the table file and `ups` directory on the local node by querying the local **UPS** declaration. However, if the product instance had *not* been declared to any local **UPS** database, we would need to specify the table file name and location. We would also need to specify the `ups` directory if it were other than `${UPS_PROD_DIR}/ups1`, which is the default location. A sample command that would work for this case is:

```
% upd addproduct foo v1_0 -2 -T /tmp/foo_v1_0_SunOS+5.tar \  
-m foo.table -M ups
```

1. `${UPS_PROD_DIR}` is one of a set of local **UPS** read-only variables listed in section 34.6 *Local Read-Only Variables Available to Functions*. It takes the same value as `$(PRODUCT)_DIR`, the product root directory.

If the command succeeds, **UPD** returns a message indicating that the product was successfully transferred and declared. After the product is added, we can run the **upd list -a** command to see the declaration in **KITS**:

```
% upd list -a foo v1_0

    DATABASE=/ftp/upsdbusr
      Product=foo Version=v1_0 Flavor=SunOS+5
      Qualifiers="" Chain=""
```

If we had wanted to declare the product in **KITS** for several flavors (assuming flavor-independence in the product), we could have specified them in the command as follows:

```
% upd addproduct foo v1_0 -f IRIX+5:SunOS+5:OSF1_v3 \
  -T /tmp/foo_v1_0_ANY.tar -m foo.table -M ups
```

Example 3

This next product, **footwo v1_0**, has no table file (thus no **-m** or **-M** needed), and it has a **ups** directory external to the product root directory. We want to declare it to the (fictional) node *dist_node.fnal.gov* with the test chain (**-t**), the flavor **NULL (-0)**, and the qualifier “debug” (**-q "debug"**):

```
% upd addproduct footwo v1_0 -t0q "debug" \
  -h dist_node.fnal.gov -T /tmp/footwo_v1_0_NULL.tar \
  -U /local/path/to/ups/dir
```

After it is added, we can run the **upd list -a** command to see the declaration on the distribution node:

```
% upd list -a -h dist_node footwo v1_0

    DATABASE=/path/to/dist_db
      Product=footwo Version=v1_0 Flavor=NULL
      Qualifiers="debug" Chain="test"
```

17.4 Adding an Independent Table File

You need to use **upd addproduct** to add a new table file product (i.e., a table file that isn't a component of a product instance). Bundled products are usually table files, for example. To replace a table file that is a component of a product instance already declared to the database on the distribution node, use **upd modproduct** as described in section 17.5 *Replacing a Component (Table File or ups Directory)*.

If the independent table file is declared to a local database, the command syntax is:

```
% upd addproduct [<flavor_option>] [<other_options>] <product>\
  <version>
```

If the table file is not declared, the command syntax becomes:

```
% upd addproduct [-P] <flavor_option> -m <tableFileName> \
  [-M <tableFileDir>] [<other_options>] <product> <version>
```

Example

The product **foothree** v1_0 consists only of a table file (it may be a bundled product); therefore no tar file needs to be specified (no **-T** option). We want to add it and declare it to **KITS** with no chain, no qualifiers, and the flavor **IRIX**. We do not assume that it's been declared to a local **UPS** database:

```
% upd addproduct foothree v1_0 -f IRIX -m foothree.table -M \  
  /local/path/to/table/file
```

The system returns a message saying there is no product root directory. This is correct behavior, and is expected.

After the table file product is added, we can run the **upd list -a** command to see its declaration in **KITS**:

```
% upd list -a foothree v1_0  
  
  DATABASE=/ftp/upsdbusr  
  Product=foothree Version=v1_0 Flavor=IRIX  
  Qualifiers="" Chain=""
```

17.5 Replacing a Component (Table File or ups Directory)

Use **upd modproduct** to update the table file or **ups** directory of a product already existing on the distribution node. This command cannot query the local **UPS** database to find information the way **upd addproduct** can; all necessary information must be specified on the command line. To replace a table file, the command syntax is:

```
% upd modproduct <flavor_option> -m <tableFileName> \  
  [-M <tableFileDir>] [<other_options>] <product> <version>
```

Note: You must include the **-m** option specifying the table file name, as there is no default. You must also include **-M** if the table file is not in the current directory.

For replacing a **ups** directory, the syntax is:

```
% upd modproduct <flavor_option> -U <upsDir> \  
  [-m <tableFileName>] [-M <tableFileDir>] [<other_options>]\ \  
  <product> <version>
```



If the **ups** directory contains a newer table file that should replace the old one on the distribution node, include the **-m** and **-M** options in the command.

Example: Table File

Let's replace the table file in **KITS** for the product **foo** v1_0, from Example 1 of section 17.3. The new table file, **foo.table**, has replaced the old one in the product instance's local **ups** directory. It doesn't matter if the tar file has been remade, since we're not going to send it anyway.

```
% upd modproduct foo v1_0 -2 -m foo.table \  
  -M /local/path/to/ups/dir
```

If you issue the command from the directory specified by **-M**, then you don't need to include it on the command line.

Example: ups Directory

To replace a product instance's `ups` directory, use the `upd modproduct` command with the `-U` option. Specify as much product instance information on the command line as necessary to uniquely identify the instance in the distribution database to which this directory is to belong. Do not make a tar file of the `ups` directory on your local machine. We illustrate with a product called **foofour v1_0**, flavor SunOS, no qualifiers, and use `KITS`.

It doesn't matter whether the product instance is declared to a **UPS** database listed in `$PRODUCTS`, since `upd modproduct` won't query the database anyway. Regardless of its location, the `ups` directory location must be fully specified, for example:

```
% upd modproduct foofour v1_0 -f SunOS \  
-U /local/path/to/ups/dir
```

17.6 Adding/Changing a Chain

A product instance on a distribution node generally has at most one chain associated with it at any time.¹ Whenever you change a chain with `upd modproduct`, you automatically delete any and all previously assigned chains. The command syntax is:

```
% upd modproduct <flavor_option> <chain_option> \  
[<other_options>] <product> <version>
```

Example 1

Product **foo** (of Example 1 in section 17.3) has no chain in its `KITS` declaration. We now wish to declare a test chain for it. We run the `upd modproduct` command with the `-t` option (or `-g test` works too), as follows:

```
% upd modproduct foo v1_0 -f SunOS+5 -t
```

Running `upd list -a` now displays:

```
% upd list -a foo v1_0  
  
DATABASE=/ftp/upsdbusr  
Product=foo Version=v1_0 Flavor=SunOS+5  
Qualifiers="" Chain="test"
```

Example 2

This time we want to change an existing chain. Let's change the test chain for **foo** (declared in Example 1, above) to current. This will remove the test chain.

```
% upd modproduct foo v1_0 -f SunOS+5 -c
```

Running `upd list` now displays:

1. A product instance can have multiple chains if they are declared together in the same command (e.g., `upd modproduct -g test:current ...`).

```
% upd list foo v1_0
```

```
    DATABASE=/ftp/upsdbusr
    Product=foo Version=v1_0 Flavor=SunOS+5
    Qualifiers="" Chain="current"
```

Notice that since we were looking for a current version, we didn't need to specify **-a** in the **upd list** command.

Example 3

To remove a chain on an instance without assigning a new one or assigning the chain to a different instance, you can use:

```
% upd modproduct foo v1_0 -f SunOS+5 -g :
```

This often generates warnings, but it works and causes no database problems.

17.7 Deleting a Product or Component

The **upd delproduct** command lets you delete a product declaration plus the product itself and its associated files and directories. The product subdirectory itself does not get deleted. You do not have the choice of leaving an undeclared product in the products area on the distribution node. The command syntax is:

```
% upd delproduct -f <flavor_option> [<other_options>] \  
    <product> <version>
```

Example 1

Let's delete the product **foo v1_0** (from Example 2 in section 17.6):

```
% upd delproduct foo v1_0 -cf SunOS+5
```

Example 2

Let's delete the product **foothree v1_0** from section 17.4. It's just a table file.¹

```
% upd delproduct foothree v1_0 -f IRIX
```

17.8 Cloning a Product

Use **upd cloneproduct** to create a new product instance on a distribution node by copying one that is already there and changing one or more of its identifying elements.

1. If there were a product that consisted only of a **ups** directory (unlikely), **upd delproduct** would work for that too.

The command syntax is:

```
% upd cloneproduct <flavor_option> [<source_options>] \  
  <product> [<version>] -G "<target_options>"
```

where *source* refers to the original instance, and *target* to the cloned one.

To clone a product, you specify the usual **UPS/UPD** options to identify the product, and then use the **-G** option to specify which attributes of the clone should be different from the original.

Why would you want to do this? For example, say that an existing product for the flavor IRIX+5 is found to be appropriate for IRIX+6, too. In this case, you might want the product to appear on the distribution server listed under both flavors. You could install the product on your local system, redeclare it, and add it back to the distribution server, but a much quicker and more efficient way is to use **upd cloneproduct** to *clone* the product instance right on the distribution server. Here is a sample command for doing this:

```
% upd cloneproduct myproduct v1_0 -f IRIX+5 -G "-f IRIX+6"
```

You can put all sorts of options in the **-G** quoted argument list, including product and version (with caveats); so you can even use **upd cloneproduct** to make a clone with a different name, provided the product's table file doesn't specify the product name. For example, to make a clone of **myprod** called **newprod** in **KITS**, you'd issue a command like this:

```
% upd cloneproduct myproduct v1_0 -f IRIX+5 -G "newprod"
```



A few caveats:

- Within the **-G** option structure on the **upd cloneproduct** command line, only include options such that a stanza of the source product's table file can be matched. A failure to match sometimes creates a database inconsistency on the distribution node. In particular, be careful about including qualifiers, e.g., **-G "-q <qualifierList>"**, if there is no stanza for `Qualifiers = qualifierList`.
- If you want product instance clones, one without qualifiers and the other with, add the first instance without qualifiers, and clone it to a new instance with qualifiers. Going the other way is error-prone.
- You can only make a clone with a different product name if the source product's table file doesn't specify the product name.

17.9 Including Source in one of Fermilab's CVS Repositories

Different groups at Fermilab often depend upon each other's software, and people need to be able to rebuild products on occasion. The **CVS Product Repositories** have been created to provide a structure allowing access to source code with revision tracking. The product eligibility standards are described in the document *Using Fermilab CVS Product Source Repositories*, on-line at http://www.fnal.gov/docs/products/template_product/FermiRepository/FermiRepository.html.

17.10 Product Announcement Policies

The separate groups within the Computing Division have differing policies for informing the group members and the user community about product availability. Here we present a checklist of the kinds of things you will be expected to do when you're ready to make a product available. We refer you to your group leader for information specific to your group.

Events which require notification actions on your part are:

- initially placing a product on *fnkits*, declared as "test" (recommended)
- declaring the product as current on *fnkits*
- installing the product in AFS space
- upgrading the product
- modifying or removing the product on/from *fnkits*

The general types of required actions are:

- Inform your group leader.
- Announce product according to group's policy (newsgroups, product user mailing lists)
- Send email to helpdesk@fnal.gov to inform them about the new product or version. Include information on the kinds of questions to expect, if possible, and where to direct users for help.
- Install the product on *fnalu* for the general Fermilab community, if appropriate.
- Check all the chains on *fnkits* (and *fnalu*) to make sure that older versions, flavors, etc. are no longer chained to current.
- Include source code for eligible product in a CVS Repository.
- Make documentation available on-line under http://www.fnal.gov/docs/products/<product_name>. Include html documentation.
- Fill out the on-line **Computing Division Product Input Form** at <http://cddocs.fnal.gov/cfdocs/productsDB/productinput.html> to inform the products database maintainer about your product arriving on *fnkits*.

Chapter 18: Using `template_product` to Build and Distribute UPS Products

In this chapter we describe the `template_product` product, and show how to use it to build and distribute a product.

18.1 Overview

To simplify and somewhat automate the process of building **UPS** products, we have designed the product `template_product`. Once this product is installed on your system, it can be cloned into a new product area and “turned into” the new product. `template_product` can be used to build products of all types (shell script, pre-built binary, source code).

The following is a summary of the steps involved when using `template_product` to build a **UPS**-compatible product. Each step is described in detail later in this chapter:

- 1) Make sure `template_product` is installed on your system; install it if necessary
- 2) Setup `template_product`
- 3) Create a directory for your product
- 4) Clone `template_product` to create a template for your product in the new directory
- 5) Insert the product into the template
- 6) Setup and test the product
- 7) Distribute the product (using the Makefile provided with `template_product`)

Also discussed in this chapter are:

- customizing a tar file
- adding a product to a **CVS** repository
- removing a product from a distribution node using the provided Makefile

18.2 Accessing template_product

The **template_product** product may already be installed on your system. If not, download it from the distribution node and install it into the main products area on your system by using the usual installation commands:

```
% setup upd
% upd install template_product
```

18.3 Cloning template_product

Next you need to setup **template_product**, make a directory to hold your new product, and clone **template_product** into this new area using a script that comes with it called `CloneTemplate`. You need to provide the name and version of your product to this script (we use **newprod** v1_0 in this example). Enter this sequence of commands:

```
% setup template_product
% mkdir /tmp/newprod
% cd /tmp/newprod
% CloneTemplate

Product name? newprod
Product version? 1.0
Platform specific product [yN]? y
Dependant products [list as fred:joe:harry]?
installing template product files in /tmp/newprod
/newprod
/tmp/newprod/.
/tmp/newprod/.header
/tmp/newprod/.manifest.template_product
/tmp/newprod/ups
/tmp/newprod/ups/Version
/tmp/newprod/ups/INSTALL_NOTE.template
/tmp/newprod/ups/template_product.table
/tmp/newprod/ups/.manifest.template_product
/tmp/newprod/Makefile
/tmp/newprod/test
/tmp/newprod/test/TestScript
/tmp/newprod/README.template
42 blocks
Customizing product as newprod...
16955
# for Flavored products
?
for NULL products
for NULL products
# QUALS is added qualifiers, like: "QUALS=mips3:debug"
#
    UPS_SUBDIR=ups
```

```

# for Flavored products
    FLAVOR=$(DEFAULT_FLAVOR)
    QUALS=""
# for NULL products
#     FLAVOR=$(DEFAULT_NULL_FLAVOR)
#     QUALS=""
##-----
## Files to include in Distribution
16957

```

The files listed in the command output have now been copied into the new product directory, and `Makefile` and `ups/template_product.table` have been customized/renamed for the product. Note that the output shows the full pathname to the created files even though you are working from within this new product directory.

18.4 The Top-Level Makefile

The cloning of `template_product` creates a Makefile in the new product's root directory, e.g., `/tmp/newprod/Makefile`. In order for this Makefile to know what it needs to about the new product, you generally need to make a few changes to the top page or so, e.g., change the flavor, add build instructions, and so on. Changes of this type are discussed in section 18.6.3 *Add Build Instructions to Top-Level Makefile*. You can also add commands to other targets.

The first part of the file is reproduced here for reference (comments not shown):

```

SHELL=/bin/sh
DIR=$(DEFAULT_DIR)
PROD=newprod
PRODUCT_DIR=MYPROD_DIR
VERS=v1_0
TABLE_FILE_DIR=ups
TABLE_FILE=newprod.table
CHAIN=development
UPS_SUBDIR=ups
ADDPRODUCT_HOST=fnkits.fnal.gov
DISTRIBUTIONFILE=$(DEFAULT_DISTRIBFILE)
FLAVOR=$(DEFAULT_FLAVOR)
OS=GENERIC_UNIX
QUALS=
CUST=none
...
#-----
all: proddir_is_set build_prefix

clean:
    rm -f $(PREFIX)

spotless:

test: proddir_is_set clean FORCE
    sh test/TestScript
...

```

18.5 Inserting your Product into the Template

Now you need to add your actual program into the `template_product` clone, and run build instructions, if any. For shell scripts and pre-built binaries, all you need to do is create a `bin` directory under the product root, and put the executable in it. For source code, you need to first create a `src` directory under the product root, put the source file in it, and then build the product as described in the next section, 18.6 *Building the Product*.

18.6 Building the Product

18.6.1 Add Build Instructions

We recommend that you create a Makefile (separate from the one provided) to ensure reproducibility of the build procedure. Create or copy the Makefile in the `src` directory, and include a build target, e.g., `install`, as shown (again, we use `echo` to create the file since it's very simple for this example):

```
% echo "install:; cp hello ../bin" > Makefile
```

18.6.2 Run the Initial Build

Now create the `bin` directory under the product root, and run `make` to complete the build:

```
% mkdir ../bin
% make hello install
cc -o hello hello.c
cp hello ../bin
```

18.6.3 Add Build Instructions to Top-Level Makefile

Now it's time to customize the top-level Makefile created by `CloneTemplate` (refer to section 18.4 *The Top-Level Makefile* for a partial file listing). Typical macro definitions that need to be changed for a compiled program are:

```
FLAVOR=$(DEFAULT_FLAVOR)
OS=$(DEFAULT_OS)
QUALS=
CUST=$(DEFAULT_CUST)
```

Next, add the build instructions under the `all` target. For this example, they are the two commands that were just run (`mkdir` and `make`).

```
all: proddir_is_set build_prefix
    -mkdir bin
    cd src; make hello install
```

18.6.4 Rebuild Instructions

The next time this product requires a build, you would just run the command:

```
% make [all]
```

from the product root directory.

18.7 Testing your Product

Now you can setup and test your product. As an example, for our product we might run:

```
% setup newprod v1_0 -r $cwd -M ups -m newprod.table
```

or, for Bourne shell,

```
$ setup newprod v1_0 -r `pwd` -M ups -m newprod.table
```

followed by:

```
% hello
```

```
hello world
```

```
% unsetup newprod v1_0
```

```
% hello
```

```
sh: hello: command not found
```

After testing, edit the `test/TestScript` file so that it tests your software. In many cases, writing a good test script can be rather challenging. Include at least a basic test to ensure that the product works properly. For our example, the test script just needs to run our **hello** program and verify its output, e.g.,:

```
#!/bin/sh
hello | grep "hello world" > /dev/null
```

This will exit with a successful exit code if **hello** prints `hello world`, and fail otherwise.

18.8 Customizing your Tar File

Products generally get distributed as tar files. The **template_product** top-level Makefile can be used to make a product tar file and add it to the distribution node in one step. There are several variables in the Makefile that control what **template_product** includes in the tar file it makes of a product:

```
ADDDIRS="<dir1> <dir2> <dir3>..."
```

lists directories whose non-**CVS**-bookkeeping-files should be added. The default is for this to be set to “.”, the current directory, and the other variables left blank. If you only wanted to include the `bin` and `lib` directories of your product build area, you would specify `ADDIRS=bin lib`.

```
ADDFILES= "<'find' command options>"
```

lists file wildcards to include or exclude with `find(1)` options. E.g., to exclude files ending in tilde (i.e., `emacs` backup files), specify `ADDFILES= ! -name '*~'`.

```
ADDEEMPTY="<dir1> <dir2> <dir3>..."
```

lists empty directories to include in the product tar file. By default the `tar` command does not include empty directories in a tar file. Listing empty directories here causes them to be added.

```
ADDCMD= "<command>"
```

specifies a command that generates a list of files on standard output. These files will then be included in the tar file. This could be used, for example, to use an explicit file inclusion list like `ADDCMD="cat my_file_list"`.

Or it could be used to specify a find command with filtering, sorting, and so on, e.g.,

```
ADDCMD= "find . ! -name '*.o' | egrep -v \  
'/foo/|/bar/' | sort -u"
```

These values are all combined by running the following sequence of commands in the Makefile:

```
(  
  for d in .manifest.$(PROD) $(ADDEEMPTY); do echo $d; done  
  test -z "$(ADDDIRS)" || find $(ADDDIRS) $(PRUNECVS) ! -type d -print  
  test -z "$(ADDFILES)" || find . $(PRUNECVS) $(ADDFILES) ! -type d -print  
  test -z "$(ADDCMD)" || sh -c "$(ADDCMD)"  
)
```

(where `PRUNECVS` holds `find` options to prevent `find` from going into `CVS` directories). This generates a long list of files that get added to the tar file.

18.9 Adding your Product to a Distribution Node

The Makefile for `template_product` is set up to allow distribution to `fnkits` by default:

- The macro `ADDPRODUCT_HOST`, which indicates the distribution node to which products get added, is set to the default value `fnkits.fnal.gov`.
- Under the section called *Standard Product Distribution/Declaration Targets* the target `kits` is configured to add a product to `fnkits` and declare it to the `KITS` database.

To add a product to a different distribution node (e.g., `distnode.fnal.gov`):

- change the value of the macro `ADDPRODUCT_HOST` to `distnode.fnal.gov`

- add the target `distnode: addproduct` to the distribution section
- and run the `make` command with the new target, e.g., `make distnode`

18.9.1 Add Product to fnkits

Keeping the defaults in place, simply change to the directory of your product and run `make kits`:

```
% cd /tmp/newprod
% make kits

rm -f /tmp/build-newprod-v1_0
creating .manifest...
creating /tmp/newprod/./newprodSunOS+5v1_0.tar...
/tmp/newprod/./newprodSunOS+5v1_0.tar:
-rw-rw-r-- mengel/oss      0 Apr  1 11:19 1998 .header
-rw-rw-r-- mengel/oss    381 Apr  1 11:18 1998 .manifest
-rwxrwxr-x mengel/oss     5 Apr  1 11:07 1998 ./ups/Version
-rwxr-xr-x mengel/oss    55 Apr  1 11:07 1998 ./ups/INSTALL_NOTE
-rwxr-xr-x mengel/oss    43 Apr  1 11:07 1998 ./ups/setup.csh
-rwxr-xr-x mengel/oss    49 Apr  1 11:07 1998 ./ups/setup.sh
-rwxr-xr-x mengel/oss    43 Apr  1 11:07 1998 ./ups/unsetup.csh
-rwxr-xr-x mengel/oss    49 Apr  1 11:07 1998 ./ups/unsetup.sh
-rwxr-xr-x mengel/oss    15 Apr  1 11:07 1998 ./ups/current
-rwxr-xr-x mengel/oss    15 Apr  1 11:07 1998 ./ups/uncurrent
-rwxr-xr-x mengel/oss    15 Apr  1 11:07 1998 ./ups/configure
-rwxr-xr-x mengel/oss    15 Apr  1 11:07 1998 ./ups/unconfigure
-rwxr-xr-x mengel/oss   462 Apr  1 11:07 1998 ./ups/action.table
-rw-r--r-- mengel/oss  19858 Apr  1 11:14 1998 ./Makefile
-rw-r--r-- mengel/oss    190 Mar 30 17:21 1998 ./README
-rwxr-xr-x mengel/oss     87 Feb  5 16:32 1998 ./test/TestScript
-rw-rw-r-- mengel/oss     36 Apr  1 11:08 1998 ./src/hello.c
-rw-rw-r-- mengel/oss     26 Apr  1 11:09 1998 ./src/Makefile
-rwxrwxr-x mengel/oss   5380 Apr  1 11:09 1998 ./src/hello
-rwxrwxr-x mengel/oss   5380 Apr  1 11:09 1998 ./bin/hello

upd addproduct -h fnkits -T "/tmp/newprod/./newprodSunOS+5v1_0.tar" \
-M ups -m action.table -U ups -f SunOS+5
upderr::upderr_syslog - successful ups declare newprod v1_0 \
-T ftp://fnkits/ftp/products/newprod/v1_0/SunOS+5.tar -f SunOS+5 \
-r /ftp/products/newprod/v1_0/SunOS+5 -z /ftp/upsdb -q "" \
-M /ftp/upsdb/newprod -m v1_0.table
rm -f "/tmp/newprod/./newprodSunOS+5v1_0.tar"
```

After adding your product, use `upd list` to check that it arrived properly:

```
% upd list -a newprod

DATABASE=/ftp/upsdb
Product=newprod Version=v1_0 Flavor=SunOS+5
Qualifiers="" Chain=""
```

18.9.2 Specify Multiple Flavors

To add different flavors of the same product without having to modify the Makefile, you may find it convenient to specify the flavor on the `make` command line, e.g.,

```
% make "FLAVOR=SunOS+5" kits
```

or, more generally,

```
% make "FLAVOR=${UPS_FLAVOR}" kits
```

18.10 Adding your Product Source to a CVS Repository

At this point, your product is eligible for inclusion in one of Fermilab's CVS repositories. This allows tracking of the software revisions, and allows other people to find it, get a particular version, and build it if they need to. The eligibility standards are described in the document *Using Fermilab CVS Product Source Repositories*, at http://www.fnal.gov/docs/products/template_product/FermiRepository/FermiRepository.html.

First set up CVS appropriately for the repository you're going to use (the example shows fermilab), then import your product:

```
% cvs import newprod v1_0 fermilab
```

18.11 Removing your Product from a Distribution Node

A special target is provided in the top-level Makefile to remove a product from KITS, namely:

```
unkits: delproduct
```

To remove your product from the KITS database on the *fnkits* node, just run the command:

```
% make unkits
```

```
upd delproduct -h fnkits -f SunOS+5 newprod v1_0
upderr::upderr_syslog - successful ups undeclare newprod v1_0 -f SunOS+5
```

If your product is on a distribution node other than *fnkits*, the Makefile has probably already been edited to recognize that node (see section 18.9 *Adding your Product to a Distribution Node*). Add a target analogous to the *unkits* target. For example if you have:

```
distnode: addproduct
```

then add the target:

```
undistnode: delproduct
```

To remove the product, run the command:

```
% make undistnode
```

Chapter 19: Checklist for Building and Distributing Products

In this chapter we summarize the steps for preparing to build a product, building it and distributing it. We include information about making the appropriate announcements when a new or upgraded product is available.

19.1 Pre-build Checklist

1) Create product root directory structure. Here is a comprehensive list of product elements and their suggested subdirectories (most products don't require all of them):

- README (top-level) and RELEASE_NOTES files (top-level or `ups`)
- INSTALL_NOTE file (`ups`)
- top-level Makefile
- executables (`bin`)
- table file and other installation-independent files/scripts (`ups`)
- source code and build instructions (`src`)
- Makefile for build (`src`)
- html user documentation (`html`)
- PostScript or text user documentation (`doc`)
- unformatted man pages (`ups/toman/man`)
- formatted man pages (`ups/toman/catman`)
- test scripts (`test`)
- examples (`examples`)
- libraries (`lib`)
- include files (`include`)

If you use **template_product**, the operation of cloning it creates the product root directory, the top-level file templates and Makefile, several of the listed subdirectories, and a basic table file.

2) For shell script or pre-built binary products, put the executable file(s) in the `${UPS_PROD_DIR}/bin` directory.

For products requiring build, create the file `${UPS_PROD_DIR}/src/Makefile`. (Include instructions for compiling, linking, testing and all other necessary operations, as well as for copying the final binaries into `${UPS_PROD_DIR}/bin`.) Insert the product source code into `${UPS_PROD_DIR}/src`.

- 3) Include documentation (html, man pages, user guide).
- 4) Create/edit README (and INSTALL_NOTE and RELEASE_NOTES as needed). See samples in sections 16.3.1 *README*, 16.3.2 *INSTALL_NOTE* and 16.3.3 *RELEASE_NOTES*. **template_product** creates template files that you need to edit.
- 5) Create/edit the table file (usually under `${UPS_PROD_DIR}/ups`). See section 35.6 *Table File Examples*. **template_product** creates a basic one that you need to edit.
- 6) Create any extra scripts your product needs in `${UPS_PROD_DIR}/ups`. See Chapter 36: *Scripts You May Need to Provide with a Product* for examples.
- 7) Create/edit the top-level Makefile (include targets for building the product, setting permissions, testing, distributing, and so on). Section 18.4 *The Top-Level Makefile* lists the first part of the Makefile that comes with **template_product**, for reference.
- 8) (Optional) Declare the product to a local database (use the `-d` flag).
- 9) Store the master source code and all the auxiliary files in a CVS code repository (or other code-version management system) according to your group's policies.
For OSS group: `CVSROOT=cvsuser@cvcvs.fnal.gov:/cvs/cd`.

19.2 Build the Product

- 10) Verify that dependencies required for build are present.
- 11) Build the product using `${UPS_PROD_DIR}/src/Makefile` (should get called by top-level Makefile).
- 12) Set permissions to `a+x` for scripts and other executables, and to `a+r` for readable files (should get done by top-level Makefile).
- 13) If using **template_product**, modify the top-level Makefile to include build instructions and other targets, as needed, and use the top-level Makefile for subsequent builds.

19.3 Test the Product

- 14) Declare the product to a local database, if you haven't already.
- 15) Verify that dependencies are present.
- 16) Run **ups verify** on the product to check the integrity of the database files (this command is described in section 22.19 *ups verify*).
- 17) Setup and test the product (test scripts should get run by top-level Makefile).

19.6 Distribute to fnkits as “current”

- 27) Wait suitable time (amount of time depends on product).
- 28) Fix problems found during test phase.
- 29) Rebuild product.
- 30) Commit changes to **CVS**.
- 31) Put final release into *fnkits* as “current”.
- 32) Reinstall as “current” on *fnalu*, as appropriate.
- 33) Check all the chains on *fnkits* (and *fnalu*) to make sure that older versions, flavors, etc. are no longer chained to “current”.
- 34) Post news to *fnal.announce.products*, *fnal.announce.unix* (if it is a UNIX product), and *fnal.sys.fnalu.announce* (if installed on *fnalu*).
- 35) Send email to *<product>-users@fnal.gov* announcing current phase.
- 36) Send email to *helpdesk@fnal.gov* to inform them about the new product or version. Include information on the kinds of questions to expect, if possible, and where to direct users for help.

Part VIII Developer's Reference

Chapter 33: *Actions and ACTION Keyword Values*

Table files and **UPD** configuration files often include stanzas which we call *actions*. We describe actions in this chapter.

Chapter 34: *Functions used in Actions*

There is a set of supported functions that can be used in action stanzas. In this chapter we give a general overview of functions, list and describe all the supported functions, provide a couple of examples of functions within actions, and list all the read-only variables available to the supported functions.

Chapter 35: *Table Files*

This chapter describes table files. Table files contain product-specific, installation-independent information. Most, but not all, products require a table file. **UPS** product developers are responsible for providing the table files associated with their products.

Chapter 36: *Scripts You May Need to Provide with a Product*

In **UPS** v4, the functions supported for use in table file actions will not always suffice for completing certain tasks, for instance configuration and tailoring. You may still need to provide executable scripts, and include appropriate functions in your table file to execute them. In this chapter we discuss some scripts you may need to provide with your product.

Chapter 37: *Use of Compile Scripts in Table Files*

Compile scripts can be used in table files to preprocess actions, thus speeding up considerably the time it takes users to execute the actions. We describe the use of compile scripts in this chapter.

Chapter 38: *Creating and Formatting Man Pages*

In this chapter we show you how to create man pages, format them, and even create html documents from them. This is not a comprehensive man page reference, but it contains sufficient information for most purposes.

Chapter 33: Actions and ACTION Keyword

Values

Table files and **UPD** configuration files often include stanzas which we call *actions*. We describe actions in this chapter.

33.1 Overview of Actions

An action is a construction that identifies a **UPS** or user-defined operation via the **ACTION** keyword (defined in section 27.4 *List of Supported Keywords*), and lists functions to perform, in addition to any internal processes, when the operation is executed. An action can be called by a **UPS** command, a user-defined **UPS**-style command, or by another action. An action stanza has the format:

```
ACTION=<VALUE>
  <function_1>([<argument_1>] [, <argument_2>] ...)
  <function_2>([<argument_1>] [, <argument_2>] ...)
  ...
```

As for all keyword values, the **VALUE** is not case-sensitive. Nor are the functions, although some arguments are. The supported **ACTION** keyword values include:

- strings that correspond to **UPS** commands
- chains and “unchains” (explained in section 33.3.2 “*Unchains*” as Keyword Values)
- user-defined strings handled by the *Unknown Command Handler*

The supported functions are listed in section 34.3 *Function Descriptions*.

33.2 UPS Command Actions

33.2.1 UPS Commands as Keyword Values

Most commonly, the **ACTION** keyword value is a string that corresponds to a **UPS** command. The string is usually the command itself (minus the **ups** at the front, if it is part of the command), e.g., **SETUP**, **CONFIGURE**, **DECLARE**. The supported strings in this category include:

```
CONFIGURE and UNCONFIGURE
COPY
DECLARE and UNDECLARE
```

GET
MODIFY
SETUP and UNSETUP
START
STOP
TAILOR

The **UPS** commands that cannot have a corresponding action in a table file are: **ups flavor** and **ups help** (because no table file can be associated with them); **ups depend**, **ups list**, and **ups verify** (because they can operate on more than one database); and **ups exist**, **ups modify** and **ups touch**.

33.2.2 “Uncommands” as Keyword Values

Several of the **UPS** commands have “uncommand” counterparts, namely **setup/unsetup**, **ups declare/undeclare**, **ups configure/unconfigure**. Generally, if the “uncommand” is expected to undo everything that the original command did, and only that, then including an **ACTION=<UNCOMMAND>** action in the table file is unnecessary.

Uncommands and Reversible Functions

If an “unaction” is not present, **UPS** will look for the corresponding **ACTION=<COMMAND>**, and undo all the reversible functions that were performed. In section 34.2 *Reversible Functions* we discuss reversible functions. If the “uncommand” needs to do something other than the exact reversal of the command, include an “unaction” for it (i.e., **ACTION=<UNCOMMAND>**) and specify the functions to execute.



This works both ways. Say the original command is “uncommand” (e.g., **ups undeclare**), and you have included **ACTION=<UNCOMMAND>** but not **ACTION=<COMMAND>** in the table file. Then when you run “command”, **UPS** will attempt to reverse all the functions listed under **ACTION=<UNCOMMAND>**.

Uncommands and Script Execution

For the functions **sourceOptCheck**, **sourceOptional**, **sourceReqCheck**, and **sourceRequired**, the “uncommand” will execute an “unscript” in a similar way. You do not have to specify an “unaction” in the table file as long as the scripts to source are in the same directory and have matching script and “unscript” filenames (i.e., **<scriptname>** and **un<scriptname>**). This also works both ways, as discussed above.

Here is an example. Say a **CONFIGURE** action specifies:

```
ACTION=CONFIGURE
    sourceOptional(${UPS_UPS_DIR}/configure.${UPS_SHELL},UPS_ENV)
```

When you run the **ups unconfigure** command, **UPS** first looks for **ACTION=UNCONFIGURE**, as usual. Failing to find it, **UPS** next looks for **ACTION=CONFIGURE**. Upon encountering the **sourceOptional** function, it searches for the file **unconfigure.\${UPS_SHELL}** in the same directory (**\${UPS_UPS_DIR}**), and sources it.

33.3 Chain Actions

33.3.1 Chains as Keyword Values

Chain names are allowable as ACTION keyword values. This includes any predefined chain name (as listed in section 1.3.5 *Chains*: CURRENT, TEST, DEVELOPMENT, OLD, NEW) or any user-defined chain name (e.g., MY_CHAIN). Chain actions are executed when a chain of the corresponding name is declared to a product instance via the **ups declare** command. For example, if you declare an instance as current, **ups declare -c** looks for ACTION=CURRENT.



Sometimes a **UPS** command executes more than one action. For example, the **ups declare -c** command executes both the CURRENT and DECLARE actions, if they are present.

33.3.2 “Unchains” as Keyword Values

Similarly, when a chain is removed from an instance (which can happen with either **ups declare** or **ups undeclare**), **UPS** looks for the corresponding chain name preceded by the “UN” prefix (e.g., UNCURRENT, UNTEST, UNMY_CHAIN).

The relationship between a chain action and its corresponding “unchain” action (e.g., CURRENT and UNCURRENT) is the same as between commands and “uncommands”, as described in section 33.2.2 “*Uncommands*” as Keyword Values. For example, if an “unchain” action is sought but not found, **UPS** will then look for the corresponding ACTION=<CHAIN> and undo all the reversible functions listed there.

33.4 The “Unknown Command” Handler

The unknown command handler effectively allows you to define a **UPS**-like “unknown” command for use with a product. To define one, include in the product’s table file an ACTION with a unique value of your choosing, e.g., ACTION=XYZ. The corresponding command will be **ups xyz**. The action should contain one or more supported functions (listed in section 34.3 *Function Descriptions*), as usual. Here is an example of what the action may look like:

```
ACTION=XYZ
  envSet(VARIABLE, value)
  sourceRequired(SCRIPT.csh, UPS_ENV_FLAG)
```

The command **ups xyz** is now available for you to use. Enough information must of course be provided on the command line to locate the table file containing the action, e.g.,:

```
% ups xyz [<options>] <product> [<version>]
```

When it is executed, the unknown command handler locates ACTION=XYZ in the table file and executes the functions listed under it.



User-defined ACTION keyword values (e.g., XYZ) do not need to start with underscore (_), as contrasted with user-defined *keywords* (see section 27.2 *Keywords: Information Storage Format*).

Example

An example of the use of the unknown command handler can be found in the table file for the product **xemacs v20_4**.

```
ACTION=CONFIGURE
Execute(echo "Do a 'ups blessmail xemacs' as root to make mail work.",NO_UPS_ENV)
ACTION=BLESSMAIL
Execute(chgrp mail ${UPS_PROD_DIR}/lib/**/movemail, NO_UPS_ENV)
Execute(chmod 2755 ${UPS_PROD_DIR}/lib/**/movemail, NO_UPS_ENV)
```

When the product instance is configured (via the first **ups declare**, or manually via the **ups configure** command), an **echo** command prints to screen an instruction to run the user-defined (“unknown”) command **ups blessmail**. This command is handled by the unknown command handler. It finds ACTION=BLESSMAIL and executes the functions associated with it.

33.5 Actions Called by Other Actions

As mentioned in section 33.1 *Overview of Actions*, one action can execute another in the same file. The called action must be assigned a unique value of your choosing, e.g., ACTION=XYZ, and the calling action (or actions) must include one of the following functions (shown for ACTION=XYZ):

```
exeActionRequired("xyz")
```

or

```
exeActionOptional("xyz")
```

These functions are described in sections 34.3.11 *exeActionRequired* and 34.3.10 *exeActionOptional*, respectively.

This technique is useful in cases where two different **UPS** operations require overlapping functionality. For example, you may want one or more identical functions to be performed when a product gets configured and when it gets declared as current. The following example shows how to arrange this:

```
action = configure
    <functions for configure>
    exeActionRequired("common")
action = current
    <functions for current>
    exeActionRequired("common")
action = common
    <functions common to both configure and current>
```

Chapter 34: Functions used in Actions

There is a set of supported functions that can be used in action stanzas. Actions are described in Chapter 33: *Actions and ACTION Keyword Values*. In the present chapter we give a general overview of functions, list and describe all the supported functions, provide a couple of examples of functions within actions, and list all the read-only variables available to the supported functions.

34.1 Overview of Functions

Table files and **UPD** configuration files often include actions. An action corresponds to a command, usually a **UPS** command, and lists functions to perform in addition to the command's internal processes, when the command is executed. The supported functions are listed and described in this chapter. A function has the format:

```
<function_name>([<argument_1>] [, <argument_2>] ... [<delimiter>])
```

The default delimiter is the colon (:).

For example, the function:

```
envPrepend(<VARIABLE>, <value>)
```

prepends the specified value to an existing environment variable, using the default delimiter.

Functions are not case-sensitive; e.g., **envPrepend**, **envprepend**, and **ENVPREPEND** are all acceptable and equivalent. A function is specified in a shell-independent manner, but contains enough information to allow it to be transformed into a **sh** or **csh** family command (e.g., **sourceRequired()**, or **execute()**), or to be interpreted directly by **UPS** (e.g., **writeCompileScript()**).

34.2 Reversible Functions

In section 33.2.2 “*Uncommands*” as *Keyword Values* we discussed commands that have corresponding “uncommands”. Usually, when the “uncommand” is run, the desired behavior is to reverse all the functions that were performed when the original command was run. Many of the supported functions are reversible, some are not.

Wherever you plan to default the “uncommand” action (i.e., to specifically *not* include an ACTION=UNCOMMAND stanza) and you want **UPS** to exactly reverse the ACTION=COMMAND functions, make sure that you only include reversible functions under ACTION=COMMAND. Reversible functions are identified as such in the descriptions in section 34.3 *Function Descriptions*.

34.3 Function Descriptions

34.3.1 addAlias

Description

Add an alias (C shell family) or function (Bourne shell family). A **%s** in the **<VALUE>** marks where the argument list should go. Reversible (runs **unAlias**).

Syntax

```
addAlias(<NAME>, <VALUE>)
```

Example 1

```
addAlias(askfor, 'echo May I have some %s, please\?')
```

Defines the alias **askfor**, which when run with an argument like **cake**, e.g.,:

```
% askfor cake
```

produces the response:

```
May I have some cake, please?
```

Example 2

```
addAlias(setup, '${UPS_SOURCE} ` ${UPS_PROD_DIR}/bin/ups setup %s`')
```

\${UPS_SOURCE} is set to “.” or “**source**” depending on the shell, and **%s** is presumed to stand for a product name. This defines the alias **setup**. When issued with a product name, e.g.,

```
% setup upd
```

it sources the executable **\${UPS_PROD_DIR}/bin/ups** with the arguments **setup** and **upd**.

34.3.2 doDefaults

Description

Perform the default functions for the command corresponding to the specified action (only SETUP and UNSETUP have default functions). If no action listed (e.g., `doDefaults()`), then the action under which this function occurs is used. Reversible (runs `doDefaults`).

Note: If an ACTION corresponding to the given command is included in the file, the command's default functions will be executed only if `doDefaults` is specified underneath it. If there is no ACTION for the command, and hence no `doDefaults` function listed, the default functions will be executed when the command is issued.

Syntax

```
doDefaults([<ACTION>])
```

Example

```
doDefaults([SETUP])
```

Specifies that the default functions for the `setup` command will be run when the command is issued. More typically, this is specified in the following manner:

```
ACTION=SETUP
doDefaults()
```

34.3.3 envAppend

Description



Append `<value>` to existing environment variable. Reversible (runs `envRemove`).

It is better to append than prepend if you just want to provide a value in case one is not there. If you want to override any existing value, you should prepend.

Note: Use the function `pathAppend` for \$PATH.

Syntax

```
envAppend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
envAppend(PYTHONPATH, ${UPS_PROD_DIR}/lib)
```

Appends the value of `${UPS_PROD_DIR}/lib` to the variable `PYTHONPATH`, using the default delimiter.

34.3.4 envPrepend

Description



Prepend **<value>** to existing environment variable. Reversible (runs **envRemove**).

It is better to prepend than append if you want to override any existing value. If you just want to provide a value in case one is not there, you should append.

Note: Use the function **pathPrepend** for \$PATH.

Syntax

```
envPrepend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
envPrepend(PYTHONPATH, ${UPS_PROD_DIR}/lib)
```

Prepends the value of **\${UPS_PROD_DIR}/lib** to the variable **PYTHONPATH**, using the default delimiter.

34.3.5 envRemove

Description

Remove the string **<value>** from existing environment variable.

Note: Use the function **pathRemove** for \$PATH.

Syntax

```
envRemove(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
envRemove(PYTHONPATH, ${UPS_PROD_DIR}/lib)
```

Removes the value of **\${UPS_PROD_DIR}/lib** from the variable **PYTHONPATH**; assumes the default delimiter.

34.3.6 envSet

Description

Set a new environment variable. This is particularly useful for representing long strings. Reversible (runs `envUnset`).

Note: Use the function `pathSet` for `$PATH`.

Syntax

```
envSet(<VARIABLE>, <value>)
```

Example

```
envSet(UPD_USERCODE_DIR, ${UPS_THIS_DB})
```

Sets `${UPD_USERCODE_DIR}` (the local database used by `UPD`) to `${UPS_THIS_DB}` (the database in which the product is declared).

34.3.7 envSetIfNotSet

Description

Set a new environment variable, if not already set. This is particularly useful for representing long strings.

Syntax

```
envSetIfNotSet(<VARIABLE>, <value>)
```

Example

```
envSetIfNotSet(HOST, `long_hostname`)
```

If not already set, this sets the variable `HOST` to a long hostname.

34.3.8 envUnset

Description

Unset existing environment variable.

Syntax

```
envUnset(<VARIABLE>)
```

Example

```
envUnset(MYVAR)
```

Unsets the variable `$MYVAR`.

34.3.9 exeAccess

Description

Check for access to specified existing executable through the \$PATH. If executable is found continue. If not found, exit with error.

Syntax

```
exeAccess(<executable>)
```

Example

```
exeAccess(gcc)
```

Ensures that a version of the product `gcc` is in your \$PATH.

34.3.10 exeActionOptional

Description

Process the functions associated with the specified action for the same product instance. Do not fail if the action doesn't exist. Reversible.

Syntax

```
exeActionOptional("<newaction>")
```

Example

```
exeActionOptional("CONFIGURE")
```

Process the functions in CONFIGURE action. If no CONFIGURE action, processing continues.

34.3.11 exeActionRequired

Description

Process the functions associated with the specified action for the same product instance. Fail if it doesn't exist. Reversible.

Syntax

```
exeActionRequired("<newaction>")
```

Example

```
exeActionRequired("CONFIGURE")
```

Process the functions in CONFIGURE action. If no CONFIGURE action, processing fails.

34.3.12 execute

Description

Execute a shell-independent command and (optionally) assign the output to an environment variable, `<VARIABLE>`.

The functions `execute`, `sourceRequired`, `sourceReqCheck`, `sourceOptional`, and `sourceOptCheck` each take a required parameter (`UPS_ENV_FLAG`) which indicates whether to define **UPS** local variables. This parameter can take the following values:

<code>UPS_ENV</code>	define all local UPS environment variables before sourcing (the script or command relies on these being defined)
<code>NO_UPS_ENV</code>	do not define the local UPS environment variables (the script or command doesn't use them)

If the optional third argument, `<VARIABLE>`, is not specified, then the specified command is executed but the output from that command is not saved. This command does not have to be shell-independent.

Syntax

```
execute("<command>", <UPS_ENV_FLAG>, [, <VARIABLE>])
```

Example

```
execute("echo Call final installation script for  
${UPS_PROD_NAME} ${UPS_PROD_VERSION}", NO_UPS_ENV)
```

(All on one line.) **UPS** echoes the given text and sources the `current` script for the product.

34.3.13 fileTest

Description

Run a shell test on `<file>`, fail if `<test>` is not true (see `man test`).

Syntax

```
fileTest(<file>, <test> [, <errormessage>])
```

Example

```
fileTest(/, -w, "You must be root to run this command.")
```

This tests for write access in the root directory and returns the shown error message if the test fails.

34.3.14 pathAppend

Description

Append **<value>** to existing \$PATH-like environment variable. Reversible (runs `pathRemove`).



It is better to append than prepend if you just want to provide a value in case one is not there. If you want to override any existing value, you should prepend.

Syntax

```
pathAppend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
pathAppend(PATH, ${UPS_PROD_DIR}/bin)
```

Appends the value `${UPS_PROD_DIR}/bin` to the \$PATH variable using the default delimiter.

34.3.15 pathPrepend

Description

Prepend **<value>** to existing \$PATH-like environment variable. Reversible (runs `pathRemove`).



It is better to prepend than append if you want to override any existing value. If you just want to provide a value in case one is not there, you should append.

Syntax

```
pathPrepend(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
pathPrepend(PATH, ${UPS_PROD_DIR}/bin)
```

Prepends the value `${UPS_PROD_DIR}/bin` to the \$PATH variable using the default delimiter.

34.3.16 pathRemove

Description

Remove the string `<value>` from existing \$PATH-like environment variable. Reversible (runs `pathAppend`).

Syntax

```
pathRemove(<VARIABLE>, <value> [, <delimiter>])
```

Example

```
pathRemove(PATH, ${UPS_PROD_DIR}/bin)
```

Removes the value `${UPS_PROD_DIR}/bin` from the \$PATH variable.

34.3.17 pathSet

Description

Set a \$PATH-like environment variable (in `csh` family, setting a \$PATH is different than setting other environment variables). No choice of delimiter offered. Reversible (runs `envUnset`).



If this gets set wrong, your \$PATH could get deleted. (To recover from this problem, should it occur, simply run `setup setpath`.)

Syntax

```
pathSet(<VARIABLE>, <value>)
```

Example

```
pathSet(PATH, /afs/fnal.gov/ups/<prod1/v1_0/SunOS+5/bin: ...)
```

Sets the \$PATH to the value given (sample value truncated after first delimiter for brevity).

34.3.18 prodDir

Description

Set the `$<PRODUCT>_DIR` environment variable to the root directory of the product instance. Reversible (runs `unProdDir`).

Syntax

```
prodDir()
```

34.3.19 `setupEnv`

Description

Set the `$SETUP_<PRODUCT>` environment variable so that product can later be unsetup. Reversible (runs `unsetupEnv`).

Syntax

```
setupEnv()
```

34.3.20 `setupOptional`

Description

Setup another **UPS** product as a dependency, do not fail if the product doesn't exist. Reversible (runs `unsetupOptional`).

Syntax

The syntax is similar to the command `setup`:

```
setupOptional("[<options>] <product> [<version>]")
```

Example

```
setupOptional("perl")
```

Setup the default instance of the product **perl**, if available. Do not fail if not found.

34.3.21 `setupRequired`

Description

Setup another **UPS** product as a dependency; fail if product not found. Reversible (runs `unsetupRequired`).

Syntax

The syntax is similar to the command `setup`:

```
setupRequired("[<options>] <product> [<version>]")
```

Example

```
setupRequired("-j Info")
```

Setup the default instance of the product **Info** and no dependencies; fail if not available.

34.3.22 sourceCompileOpt

Description

If **<fileName>** exists, then source it and skip remaining functions; otherwise just complete the remaining functions. This is typically used in conjunction with **writeCompileScript**; see section 34.3.33 *writeCompileScript*.

Syntax

```
sourceCompileOpt("<fileName>")
```

Example

```
sourceCompileOpt("/my/compile/script")
```

This sources the specified script which was created with **writeCompileScript**. If script doesn't exist, process continues.

34.3.23 sourceCompileReq

Description

Source **<fileName>** and skip all remaining functions; fail if file not found. This is typically used in conjunction with **writeCompileScript**; see section 34.3.33 *writeCompileScript*.

Syntax

```
sourceCompileReq("<fileName>")
```

Example

```
sourceCompileReq("/my/compile/script")
```

This sources the specified script which was created with **writeCompileScript**. If script doesn't exist, process fails.

34.3.24 sourceOptCheck

Description

Check if specified script exists and if so, source it and check return status for error. If error, abort script and return. Reversible (runs **sourceOptCheck** on the “un” script, e.g., **current** and **uncurrent**).

The functions **execute**, **sourceOptCheck**, **sourceOptional**, **sourceReqCheck**, and **sourceRequired** each take a required parameter (**UPS_ENV_FLAG**) which indicates whether to define **UPS** local variables. This parameter can take the following values:

UPS_ENV	define all local UPS environment variables before sourcing (the script or command relies on these being defined)
NO_UPS_ENV	do not define the local UPS environment variables (the script or command doesn't use them)

The functions **sourceOptCheck**, **sourceOptional**, **sourceReqCheck**, and **sourceRequired** each take an optional parameter (**EXIT_FLAG**). This parameter can take the following values:

CONTINUE	after sourcing the script, continue with the next function (the default)
EXIT	after sourcing the script, skip the rest of the functions

Syntax

```
sourceOptCheck(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [, EXIT_FLAG])
```

Example

```
sourceOptCheck(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if **\${UPS_UPS_DIR}/current** exists. If so, first define all local **UPS** environment variables, then source the script and check return status for error. If error, abort script and return.

34.3.25 sourceOptional

Description

Check if `<SCRIPT>` exists and if so, source it. If script not found, continue. Reversible (runs `sourceOptional` on the “un” script, e.g., `current` and `uncurrent`).

See section 34.3.24 *sourceOptCheck* for information about the parameters `UPS_ENV_FLAG` and `EXIT_FLAG`.

Syntax

```
sourceOptional(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
EXIT_FLAG])
```

Example

```
sourceOptional(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if `${UPS_UPS_DIR}/current` exists. If so, first define all local `UPS` environment variables, then source the script. If not, continue.

34.3.26 sourceReqCheck

Description

Source `<SCRIPT>` and check return status for error; fail if script not found. If error, abort script and return. Reversible (runs `sourceOptCheck` on the “un” script, e.g., `current` and `uncurrent`).

See section 34.3.24 *sourceOptCheck* for information about the parameters `UPS_ENV_FLAG` and `EXIT_FLAG`.

Syntax

```
sourceReqCheck(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
EXIT_FLAG])
```

Example

```
sourceReqCheck(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if `${UPS_UPS_DIR}/current` exists. If not, it will fail. If script exists, first define all local `UPS` environment variables, then source the script and check return status for error. If error, abort script and return.

34.3.27 sourceRequired

Description

Source `<SCRIPT>`; fail if script not found. Return status not checked. Reversible (runs `sourceOptional` on the “un” script, e.g., `current` and `uncurrent`).

See section 34.3.24 *sourceOptCheck* for information about the parameters `UPS_ENV_FLAG` and `EXIT_FLAG`.

Syntax

```
sourceRequired(<SCRIPT>.${UPS_SHELL}, UPS_ENV_FLAG [,
EXIT_FLAG])
```

Example

```
sourceRequired(${UPS_UPS_DIR}/current.${UPS_SHELL}, UPS_ENV)
```

Check if `${UPS_UPS_DIR}/current` exists. If not, it will fail. If script exists, first define all local `UPS` environment variables, then source the script.

34.3.28 unAlias

Description

Remove alias/function of specified name.

Syntax

```
unAlias(<NAME>)
```

34.3.29 unProdDir

Description

Unsets the `$_PRODUCT>_DIR` environment variable. Reversible (runs `prodDir`).

Syntax

```
unProdDir()
```

34.3.30 `unsetupEnv`

Description

Unsets the `$SETUP_<PRODUCT>` environment variable. Reversible (runs `setupEnv`).

Syntax

```
unsetupEnv()
```

34.3.31 `unsetupOptional`

Description

Runs `unsetup` on a product, does not fail if the product doesn't exist or if it's already unsetup. Reversible (runs `setupOptional`).

Syntax

The syntax is similar to the command `unsetup`:

```
unsetupOptional("[<options>] <product> [<version>]")
```

For previously setup products, the only options that are recognized include `-e`, `-j`, and `-v`.

Example

```
unsetupOptional("perl")
```

Unsets the default instance of the product `perl`, if already setup. Does not fail if product doesn't exist or has already been unsetup.

34.3.32 `unsetupRequired`

Description

Runs `unsetup` on a product; fails if product not found. Reversible (runs `setupRequired`).

Syntax

The syntax is similar to the command `unsetup`:

```
unsetupRequired("<options>] <product> [<version>]")
```

For previously setup products, the only options that are recognized include `-e`, `-j`, and `-v`.

Example

```
unsetupRequired("perl")
```

Unsets the default instance of the product `perl`, if already setup. Fails if product doesn't exist or has already been unsetup.

34.3.33 `writeCompileScript`

Description

Write a file of compiled functions for the given ACTION keyword value. It actually writes four files in total: `<script>.[c]sh` and `un<script>.[c]sh`.

The function `writeCompileScript` takes an optional parameter which can be one of the following:

- OLD if `fileName` exists, move the old one to `fileName.old` before creating the new one.
- DATE if `fileName` exists, move the old one to `fileName.{datestamp}` before creating the new one.

Syntax

```
writeCompileScript("<fileName>", "<ACTION>" [, OLD|DATE])
```

Example

```
writeCompileScript("/my/compile/script", "SETUP", OLD)
```

This executes the SETUP action and writes the output of the functions to the specified script, first saving the pre-existing script to `/my/compile/script.old`. This function knows to ignore the function `sourceCompileReq` or `sourceCompileOpt` if it encounters either at the top of the list of SETUP functions. See sections 34.3.22 *sourceCompileOpt* and 34.3.23 *sourceCompileReq*.

34.4 Functions under Consideration for Future Implementation

copyCatMan	Will copy catman files from source directory specified in table file by CATMAN_SOURCE_DIR to target directory specified in the UPS database configuration file by CATMAN_TARGET_DIR. Reversible (will run uncopyCatMan)
copyHtml	Will copy html files from source directory specified in table file by HTML_SOURCE_DIR to target directory specified in the UPS database configuration file by HTML_TARGET_DIR.
copyInfo	Will copy Info files from source directory specified in table file by INFO_SOURCE_DIR to target directory specified in the UPS database configuration file by INFO_TARGET_DIR.
copyMan	Will copy man files from source directory specified in table file by MAN_SOURCE_DIR to target directory specified in the UPS database configuration file by MAN_TARGET_DIR. Reversible (will run uncopyMan)
copyNews	Will copy news files from source directory specified in table file by NEWS_SOURCE_DIR to target directory specified in the UPS database configuration file by NEWS_TARGET_DIR.
else ()	Will begin an alternative branch
elseif (<condition>)	Will proceed to another condition
endif ()	Will end a conditional branch
if (<condition>)	Will begin a conditional branch
uncopyCatMan	Will remove catman files from target directory specified in the UPS database configuration file by CATMAN_TARGET_DIR. Reversible (will run copyCatMan)
uncopyMan	Will remove man files from target directory specified in the UPS database configuration file by MAN_TARGET_DIR. Reversible (will run copyMan)

34.5 Examples of Functions within Actions

34.5.1 A setup Action

This first example shows a **setup** action:

```
ACTION=SETUP
prodDir()
setupEnv()
pathAppend(PATH, ${UPS_PROD_DIR}/bin)
setupRequired("crow")
setupOptional("gypsy")
```

When the product instance gets setup, **UPS** does five things in addition to **setup**'s internal processes:

- sets the variable `$<PRODUCT>_DIR` to the product root directory
- sets the variable `$SETUP_<PRODUCT>` to identify the product instance for unsetup
- appends the product's `bin` directory to the path
- sets up the product **crow** (and aborts the setup if a suitable current instance of **crow** is not available)
- sets up the product **gypsy**, if found (**setup** proceeds whether or not a suitable current instance of **gypsy** is available).

34.5.2 A “declare as current” Action

A second example illustrates steps for **UPS** to complete when the product instance is declared as current to the database:

```
ACTION=CURRENT
execute("echo Call final install script for ${UPS_PROD_NAME} ${UPS_PROD_VERSION}")
sourceRequired(${UPS_UPS_DIR}/current, UPS_ENV)
```

UPS echoes the given text and sources the `current` script for the product.

34.6 Local Read-Only Variables Available to Functions

The read-only variables listed below are set by **UPS** and available for use with the functions described in section 34.3 *Function Descriptions*. In several functions, the flag `UPS_ENV_FLAG` controls whether these variables get set (see section 34.3.24 *sourceOptCheck*).

These **UPS** variables do not get exported to the environment, but exist only for the duration of, and in the context of, the processing of an action (actions are described in Chapter 33: *Actions and ACTION Keyword Values*). By contrast, the environment variables `$<PRODUCT>_DIR` and `$SETUP_<PRODUCT>` (described in section 22.1 *setup* under *Environment Variables Set by Default During setup*), if defined, remain set and available for use as long as the product is setup.¹

34.6.1 List of Current Read-Only Variables

When you use these variables, always enclose them in curly brackets ({}) as shown in the list.

Local Read-Only Variable	Description of Value
<code>\${PRODUCTS}</code>	Generally has the same value as the environment variable <code>\$PRODUCTS</code> . The difference is that (read-only) <code>\${PRODUCTS}</code> keeps the value set at the time UPS was invoked, whereas (environment) <code>\$PRODUCTS</code> may be reset. You can reset <code>\$PRODUCTS</code> (i.e., using the function <code>envSet (PRODUCTS, "<value>"</code> in the table file) in order to use a new value in the temp file; <code>\$PRODUCTS</code> won't get overwritten by <code>\${PRODUCTS}</code> as the temp file executes. See the example that follows this table. Note that this is not valid for the other read-only variables in this table; if you try to reset them (as environment variables), your values will get overwritten by the read-only values as the temp file executes.
<code>\${UPS_COMPILE}</code>	Location and file name of a file containing compiled functions (see Chapter 37: <i>Use of Compile Scripts in Table Files</i>). It has the value of the combined keywords: <code>COMPILE_FILE_DIR/COMPILE_FILE</code>
<code>\${UPS_EXTENDED}</code>	This set to 1 if the <code>-e</code> (extended) option was specified in the <code>setup</code> command (see section 24.2.1 <i>-e</i>)
<code>\${UPS_OPTIONS}</code>	Option string that was passed with the <code>-O</code> (upper case o) flag (see Chapter 24: <i>Generic Command Option Descriptions</i>)
<code>\${UPS_ORIGIN}</code>	This specifies the location of the master source files.
<code>\${UPS_OS_FLAVOR}</code>	Operating system flavor as obtained from <code>ups flavor</code>
<code>\${UPS_PROD_DIR}</code>	Product instance root directory; same value as the environment variable <code>\$<PRODUCT>_DIR</code>
<code>\${UPS_PROD_FLAVOR}</code>	Product flavor chosen during instance matching
<code>\${UPS_PROD_NAME}</code>	Product name as declared in the UPS database
<code>\${UPS_PROD_QUALIFIERS}</code>	Product qualifiers chosen during instance matching. These are the qualifiers <i>declared with the selected instance</i> . They are not necessarily the same set of qualifiers specified on the command line via the <code>-q</code> option (the UPS matching algorithm chooses the "best fit" based on the specified qualifiers; not necessarily an exact match).
<code>\${UPS_PROD_VERSION}</code>	Product version as declared in the UPS database
<code>\${UPS_THIS_DB}</code>	Database in which this product instance is declared.

-
1. The `setup` command and these variables are described in section 22.1 *setup*.

Local Read-Only Variable	Description of Value
<code>\${UPS_UPS_DIR}</code>	Path to the product instance's <code>ups</code> directory
<code>\${UPS_VERBOSE}</code>	This is set to 1 if the <code>-v</code> (verbose) option was specified (see Chapter 24: <i>Generic Command Option Descriptions</i>).

\$PRODUCTS vs. \${PRODUCTS}: Resetting \$PRODUCTS

This example is intended to illustrate the interaction between the read-only variable `${PRODUCTS}` and the environment variable `$PRODUCTS`. There are a couple of potentially confusing points.

Let `${PRODUCTS}` be set to `/fnal/ups/db`. Say in your table file you set `$PRODUCTS` to `/path/to/mydb` in the `SETUP` action, like this:

```
ACTION=SETUP
  envSet(PRODUCTS, "/path/to/mydb:${PRODUCTS}")
```

Now `${PRODUCTS}` and `$PRODUCTS` are different. The following `execute` functions show the difference in the values. The function:

```
execute("echo $PRODUCTS", NO_UPS_ENV)
```

would produce:

```
/path/to/mydb:/fnal/ups/db
```

whereas the same function using `${PRODUCTS}`, e.g.,

```
execute("echo ${PRODUCTS}", NO_UPS_ENV)
```

would produce only:

```
/fnal/ups/db
```

\$PRODUCTS vs. \${PRODUCTS}: Effects on setup and ups depend

Another issue is the `setup...` functions. Say you have a product `fred v1_0` declared in `/path/to/mydb` (the database not included in `${PRODUCTS}`). If you include a `setupRequired` or `setupOptional` function later in the `SETUP` action, e.g.,:

```
ACTION=SETUP
  envSet(PRODUCTS, "/path/to/mydb:${PRODUCTS}")
  setupRequired(fred v1_0)
```

the setup will fail because these functions only reference the read-only variable `${PRODUCTS}`, which in this case doesn't include your product. You can get around this by using the `execute` function to set the product up:

```
execute("setup fred v1_0", NO_UPS_ENV)
```

This function uses the environment variable `$PRODUCTS`.

Remember though, when you run a `ups depend` on a product, only products identified in `setupRequired` or `setupOptional` functions get listed. You would not see `fred v1_0` listed in the `ups depend` output for the main product in our example.

34.6.2 Read-Only Variables under Consideration for the Future

We plan to make the keyword values, listed in section 27.4 *List of Supported Keywords*, available as read-only variables available to functions. The read-only variable corresponding to a keyword will typically include “UPS_” prepended to it. E.g., the read-only variable corresponding to the keyword DECLARED will be `UPS_DECLARED`. Several of these are already implemented in this way, e.g., `UPS_PROD_DIR` corresponds to the keyword PROD_DIR.

Chapter 35: Table Files

This chapter describes table files. Table files contain product-specific, installation-independent information. Most, but not all, products require a table file. **UPS** product developers are responsible for providing the table files associated with their products.

35.1 About Table Files

Table files are created and maintained by product developers. Table files contain the non-system-specific and non-shell-specific information that **UPS** uses for installing, initializing, and otherwise operating on product instances. For a given product, usually a single table file suffices for several instances, especially of a single version. Sometimes each instance has a separate table file. Table file names are arbitrary; we present recommendations in section 35.3 *Recommendations for Creating Table Files*.

Typically, when a **UPS** command is issued, **UPS** finds the table location from the command line or the version file (see section 28.4 *Determination of ups Directory and Table File Locations*). The command completes its internal processes, and then within the table file, it proceeds to:

- 1) locate the stanza that matches the specified product instance
- 2) find an **ACTION** keyword value that corresponds to the command, if any (see Chapter 33: *Actions and ACTION Keyword Values*)
- 3) execute the functions listed underneath the corresponding **ACTION** keyword, if any (see Chapter 34: *Functions used in Actions*), or
- 4) reverse the functions listed underneath the **ACTION** corresponding to the “uncommand” (see section 33.2.2 *“Uncommands” as Keyword Values*)

35.2 When Do You Need to Provide a Table File?

Not all products require a table file. In particular, if *no* processing besides the internals and defaults needs to be done for *any* **UPS** command run on a particular product, and if its `ups` directory and documentation reside in the default areas, then the product doesn't need a table

file. However, for products that do need a table file (most), at least a rudimentary table file must be in place before any instance is declared to a target **UPS** database. If it's not added right away, users may see incorrect behavior before it is there.

35.3 Recommendations for Creating Table Files

- Although table files can have any file name, we recommend that they be named as `<product>.table` (e.g., `emacs.table`) or `<version>.table` (e.g., `v19_34b.table`) for easy identification. If a table file is unique to a particular version of the product (which is likely because versions of product dependencies often change along with the version of the main product) then the name should be `<product>_<version>.table` (e.g., `emacs_v19_34b.table`).
- Table files should not source any `setup.[c]sh` script unless flow control (if then else, looping, etc.) is needed. For assistance, contact `uas-group@fnal.gov`.
- In most cases, “un” actions (e.g., `UNSETUP`, `UNCURRENT`) are not needed (see section 33.2.2 “*Uncommands*” as *Keyword Values*). If an “un” action is *not* specified in the table file, **UPS** will undo what the corresponding action did (e.g., `SETUP`, `CURRENT`), in reverse order, provided reversible functions were used (see section 34.2 *Reversible Functions*).
- Individual groups or experiments at Fermilab may set standards regarding table files that members should follow; contact your group leader to find out if there are any you need to be aware of. For example, ODS prefers that table files be maintained in the **UPS** database product subdirectory (e.g., `$PRODUCTS/emacs`) rather than in the product's `ups` directory.

35.4 Table File Structure and Contents

35.4.1 Basic Structure

The file starts with a header that identifies the file type and the product:

```
File=Table
Product=<product>
```

The basic structure of table file contents consists of an instance identifier followed by one or more actions (described in Chapter 33: *Actions and ACTION Keyword Values*). By the time **UPS** accesses the table file, it has already determined the database, product name and product version. Therefore `FLAVOR` and `QUALIFIERS` together are sufficient to identify the instance.

Here is a sample table file that illustrates the basic structure:

```
File=Table
Product=exmh

FLAVOR=SunOS+5
QUALIFIERS= " "

ACTION=SETUP
    setupRequired(expect)
    setupRequired(mh)
    ...
ACTION=UNSETUP
    ...
```

User-defined keywords, described in section 27.2 *Keywords: Information Storage Format*, can also be included after an instance identifier for use within actions.

35.4.2 Grouping Information

When a single table file represents multiple instances, a grouping structure can be superimposed on this basic structure to organize the information. To avoid having to repeat identical actions for a series of FLAVOR/QUALIFIER identifiers, the keyword FLAVOR can take the value ANY in table files. FLAVOR=ANY is taken as a best match, assuming all other instance identifiers match (see Chapter 26: *Product Instance Matching in UPS/UPD Commands* for more information on instance selection).

Grouping information within table files is supported via the use of the following three markers:

GROUP:	Groups together multiple flavor/qualifier pairs. All entries subsequent to GROUP: are part of this group until an END: marker is found.
COMMON:	Groups together actions that apply to all instances represented in GROUP:. COMMON: is only valid within a GROUP:.
END:	Marks the end of a GROUP: or COMMON:. One END: marker is used to jointly end a GROUP: and an included COMMON:.

UPS does not *require* grouping in table files; these markers are available for convenience and for organizing information clearly. However, if GROUP: or COMMON: is used, END: must appear at the end of it, even if that is the very end of the file.

35.4.3 The Order of Elements

Blank lines are ignored, and therefore can be placed anywhere.

- The first keywords after GROUP: must always be FLAVOR followed by QUALIFIERS (i.e., the instance identifiers).
- FLAVOR and QUALIFIERS *cannot* be included within a COMMON: grouping.
- User-defined keywords can be defined anywhere except between GROUP: and the instance identifiers.
- Actions (described in Chapter 33: *Actions and ACTION Keyword Values*) for each instance are located after the instance-identifying keywords, and often between a COMMON: and END:.

- All actions after **COMMON:** apply to all the **FLAVOR-QUALIFIERS** pairs listed above it within the current **GROUP:**.
- All statements apply to the most recently defined **FLAVOR/QUALIFIER** keywords except for the statements between **COMMON:** and **END:** (which apply to all the flavors in the current **GROUP:**)
- **GROUP:**s cannot be nested.

35.5 Product Dependencies

35.5.1 Defining Dependencies

UPS product dependencies get listed in the **SETUP** action for the product instance in question. The **setupRequired** and **setupOptional** functions, described in section 34.3 *Function Descriptions*, can be used within the **SETUP** action to setup the dependencies along with the main product. These two functions take the same set of options and arguments as a normal **setup** command (see section 22.1 *setup*) in order to clearly specify the desired instance of the dependent product. We discourage specification of particular versions of products, and recommend using chains instead, e.g.:

```
ACTION=SETUP
  setupRequired("perl")
```

This example sets up the default instance of **perl**, chained to current. Using chains, it is easier to keep the dependencies and the main product in sync.

Products that are not maintained in the **UPS** framework can also be designated as dependencies. You would need to use the function **exeAccess** to locate and access a non-**UPS** executable through your **\$PATH**. For example, the action:

```
ACTION=SETUP
  setupOptional(gcc)
  exeAccess(gcc)
```

tells **UPS** to setup the current instance of **gcc** if there is one declared; the **exeAccess** function checks for a version of **gcc** in your **\$PATH**, even if it's not one that is managed by **UPS**, and exits with an error if one is not found.

35.5.2 Product Dependency Conflicts

When different dependencies include the same product via different dependency trees (and therefore may require different instances of the same product), rules have been established to determine which instance of the dependent product is selected and in which order the required products are setup.

Selection Algorithm for Conflicting Dependencies

The rules are as follows:

- 1) First level product dependencies, defined as those products listed as dependencies in the table file of the main product instance, take precedence over lower level dependencies when selecting which instance of the required product to set up.
- 2) Dependencies listed later in the table file take precedence over those listed earlier.

Example of Dependency Selection and Order of Setup

We'll take you through an example that illustrates how the dependencies are selected and in what order they are setup. Our sample dependency structure starts with the product **A** as the parent product. It has two dependencies, which in turn have dependencies of their own. **B b1** refers to product **B**, version **b1**, and so on. (We recommend that developers avoid using specific version dependencies in general; we use them in our example for illustrative purposes.) Some of the dependencies are conflicting:

In **A**'s table file:

```
product A
setupRequired(B b1)
setupRequired(C c1)
```

In **B b1**'s table file:

```
product B b1
setupRequired(C c2)
setupRequired(D d1)
```

In **C c2**'s table file:

```
product C c2
setupRequired(D d3)
```

In **C c1**'s table file:

```
product C c1
setupRequired(D d2)
```

The tree is traversed starting at **A**, then going down each dependency branch. So the order in which the products are encountered is:

- 1) **A** (no conflict)
- 2) **A**'s dependencies **B b1** and **C c1** are selected since they are the highest level dependencies.
- 3) Start down **B b1** branch: find **C c2** (version **c1** already selected by rule 1; **C c2** ignored)
- 4) Completing the **B b1** branch, find **D d1**. It is examined, and ultimately passed over (by rule 2) because **D d2**, a dependency of **C c1** and therefore also a second-level dependency of **A**, is encountered later.

35.6 Table File Examples

35.6.1 Example Illustrating Use of FLAVOR=ANY

Below is a sample table file for the product **exmh** version v1_6_6 which uses FLAVOR=ANY. For the **exmh** instances whose version files point to this table file, all except those with qualifiers share the same stanza:

```
File=Table
Product=exmh
#*****
# Starting Group definition
Group:
Flavor=ANY
Qualifiers=""

Common:
  Action=setup
    setupRequired(expect)
    setupRequired(mh)
    setupOptional(glimpse)
    setupOptional(www)
    setupOptional(mimetools)
    setupOptional(ispell)
    setupOptional(popclient)
    prodDir()
    setupEnv()
    pathPrepend(PATH, ${UPS_PROD_DIR}/bin)
  Action=configure
    execute(${UPS_PROD_DIR}/ups/configure, UPS_ENV)
End:
```

Actions, functions and variables as used in this example are described in Chapter 33: *Actions and ACTION Keyword Values*, section 34.3 *Function Descriptions* and section 34.6 *Local Read-Only Variables Available to Functions*, respectively.

You'll notice that there are no functions specified for **unsetup** in this table file. Due to the defaults that **UPS** has in place, when **unsetup** is run all of the **setup** functions will be reversed (the required products will get unsetup, the defined environment variables will get undefined, and the product's `bin` directory will be dropped from `$PATH`). See sections 33.2.2 "*Uncommands*" as *Keyword Values* and 34.2 *Reversible Functions*.

35.6.2 Example Showing Grouping

Grouping is illustrated in the following example:

```
FILE=Table
PRODUCT=exmh
#*****
# Starting Group definition
GROUP:
FLAVOR=IRIX+5
QUALIFIERS=""

FLAVOR=IRIX+5
QUALIFIERS="mips2"
```

```

COMMON:
  ACTION=SETUP
    setupOptional(expect)
    ...
  ACTION=CONFIGURE
    execute(${UPS_PROD_DIR}/ups/configure,UPS_ENV)
    ...
END:
#*****
# Starting Group definition
GROUP:
FLAVOR=ANY
QUALIFIERS=""

COMMON:
  ACTION=SETUP
    setupRequired(expect)
    ...
  ACTION=CONFIGURE
    execute(${UPS_PROD_DIR}/ups/configure,UPS_ENV)
    ...
END:

```

The second group (defined by `FLAVOR=ANY`) matches all the instances not matched in the first group, except those with qualifiers.

35.6.3 Example with User-Defined Keywords

User-defined keywords are described in section 27.2 *Keywords: Information Storage Format*. All user-defined keywords must have underscore (`_`) as the initial character (e.g., `_dest_arch`). The following example illustrates their use in a table file:

```

File=Table
Product=vxboot
#*****
# Starting Group definition
Group:
Flavor=NULL
Qualifiers="narrow29"
  _dest_arch=ppc
  _dest_env=VxWorks-5.3
  _dest_type=MVME2301
...
Common:
  Action=setup
    setupEnv()
    envSet (VXB_DEST_ARCH,${_dest_arch})
    envSet (VXB_DEST_ENV,${_dest_env})
    envSet (VXB_DEST_TYPE,${_dest_type})
...

```

35.6.4 Examples Illustrating ExeActionOpt Function

Example 1

In this example, there are actions for the first two instance identifiers, but not for the third. We want to execute the XYZ action at setup time if it's there, but continue processing if it's not. To do this, we must call the action using the `exeActionOpt` function.

```
FILE=Table
PRODUCT=fred
#*****
# Starting Group definition
GROUP:
FLAVOR=SunOS+6
QUALIFIERS=""
ACTION=XYZ
    fileTest(/, -w, "You must be root to run this command.")

FLAVOR=IRIX+6
QUALIFIERS=""
ACTION=XYZ
    fileTest(/, -w, "You must be root to run this command.")

FLAVOR=IRIX+6
QUALIFIERS="mips2"
# No XYZ action

COMMON:
ACTION=SETUP
    exeActionOpt(XYZ)

END:
...
```

Example 2

In this example, we use the `exeActionOpt` function to instruct **UPS** to execute one action or another, depending on whether the user supplies an option on the `setup` command line.

```
FILE=Table
PRODUCT=fred
#*****
# Starting Group definition
GROUP:
FLAVOR=ANY
QUALIFIERS=""

ACTION=SETUP
    exeActionOpt(XYZ_${UPS_OPTIONS})

ACTION=XYZ_
    function_1()
ACTION=XYZ_FULL_LICENSE
    function_2()
...
```

If you run:

```
% setup fred
```

you'll execute ACTION XYZ_. To execute ACTION XYZ_FULL_LICENSE, you need to run:

```
% setup fred -O FULL_LICENSE
```


Chapter 36: Scripts You May Need to Provide with a Product

In **UPS** v4, the functions supported for use in table file actions will not always suffice for completing certain tasks, for instance configuration and tailoring. You may still need to provide executable scripts, and include appropriate functions in your table file to execute them. In this chapter we discuss some scripts you may need to provide with your product.

Since these types of scripts generally get executed only once, speed isn't critical. We plan to provide more functions in later **UPS** releases so that scripts will no longer be necessary for this purpose.



Note that these files can be binaries, but scripts are recommended.

36.1 `configure` and `unconfigure`

The `configure` executable must perform whatever steps are necessary to install the product on a system, minus anything that requires direct interactive input from the installer. In cases where the installer must supply some information, you can choose to use a `tailor` script to collect data, and pass the values to the `configure` script to use.

The `unconfigure` executable must undo everything that `configure` does. **UPS** is “smart” enough that if one of the functions `sourceOptCheck`, `sourceOptional`, `sourceReqCheck`, or `sourceRequired` is used in the `CONFIGURE` action, when **ups unconfigure** is run, **UPS** can find and source the `unconfigure. ${UPS_SHELL}` script.

Here is an example. Say a `CONFIGURE` action specifies:

```
ACTION=CONFIGURE
    sourceOptional( ${UPS_UPS_DIR}/configure. ${UPS_SHELL} , UPS_ENV)
```

When you run the **ups unconfigure** command, **UPS** first looks for `ACTION=UNCONFIGURE`, as usual. Failing to find it, **UPS** next looks for `ACTION=CONFIGURE`. Upon encountering the `sourceOptional` function, it searches for the file `unconfigure. ${UPS_SHELL}` in the same directory (`${UPS_UPS_DIR}`), and sources it.

Sample Configure Script

The `tex_files` product has a good example of a `configure/current` script (they are identical in this case):

```
#!/bin/sh
# "current" and "configure" for $TEX_FILES_DIR/ups/current

case "$TEX_FILES_DIR" in
/afs*)
    find $TEX_FILES_DIR/texmf/fonts/tmp -type d \
-exec fs setacl {} system:anyuser rlidkw \;
    ;;
*)
    chmod -R 1777 $TEX_FILES_DIR/texmf/fonts/tmp
    ;;
esac
```

The directory `$TEX_FILES_DIR/texmf/fonts/tmp` must be writable by anybody using `tex_files`, in order that **TeX** can create the requested fonts on-the-fly from font metadata files. (This way, rarely-used fonts can be generated as the document is created, and they don't need to be stored.) The script evaluates `$TEX_FILES_DIR`. If it begins with `/afs`, it runs the appropriate AFS command to make the `tmp` area world-writable. If not, then it uses the standard UNIX `chmod`. **UPS** does not yet have "if-then-else" capability within table files, so we can't write these things into actions. The table file calls the scripts via the actions:

```
action=configure
  prodDir()
  execute(${UPS_UPS_DIR}/configure,UPS_ENV)
  unprodDir()
action=current
  prodDir()
  execute(${UPS_UPS_DIR}/current,UPS_ENV)
  unprodDir()
```

As described in section 33.5 *Actions Called by Other Actions*, one of the identical scripts could have been eliminated and a common action could have been used in this way:

```
action=configure
  exeActionRequired("common")
action=current
  exeActionRequired("common")
action=common
  proddir()
  execute(${UPS_UPS_DIR}/configure,UPS_ENV)
  unprodDir()
```

Configure Scripts for Products with Hard-Coded Paths

As discussed in section 15.1.3 *Third-Party Products Requiring a Hard-Coded Path*, many third-party products require a hard-coded path assigned when the product is built. Most of these products come with configurable Makefiles thereby allowing you to choose the path. The technical note TN0086 *Use of "/usr/local/products" now deprecated*, on-line at <http://www.fnal.gov/docs/TN/TN0086/tn0086.html>, describes recommended techniques for implementing these products. The third approach that it discusses involves using the `configure` script to modify a trampoline executable. Please refer to TN0086 for information.

36.2 tailor

As discussed in section 3.6.2 *Tailoring a Product*, tailoring is the aspect of the product implementation that requires input from the product installer (e.g., the location of hardware devices for a software driver package, a specific area for log files, which node should run the server, etc.). If your product requires any interactive input from the installer, you will need to furnish a `tailor` executable for this purpose. Generally `tailor` files are scripts that ask the installer a series of questions, and write the answers to a `<node>.dat` file which in turn gets read by the `configure`, `current`, and/or `start` scripts.

Usually undoing the steps done via `tailor` require interactive input. However, if your `tailor` steps are such that they can be undone via a script, go ahead and provide an `untailor` script. When you run the `ups untailor` command (available via the unknown command handler discussed in section 33.4 *The “Unknown Command” Handler*), **UPS** will execute `untailor`, the same way as described for `unconfigure` in section 36.1 *configure and unconfigure*.

It still may be best to avoid including anything in `tailor` that needs to be undone when the product is removed and that requires input from a person. If `tailor` is used to collect information and pass it to the `configure` script (recommended), then anything that needs to be undone can be addressed in `unconfigure`.

For a sample `tailor` script, see `$JUKE_DIR/ups/tailor`.

36.3 current and uncurrent

Most things that need to be done when a product instance is declared `current` can be done directly via functions in the table file in a `CURRENT` action. However, if the available functions prove to be insufficient for your product, create a `current` script to perform the function(s).

Likewise, when a `current` chain is removed from a product instance, the `uncurrent` script (if it exists) should undo all the things that were done in `current`. It works the same way as `UNCONFIGURE`, described in section 36.1 *configure and unconfigure*.

A sample `current` script is shown in section 36.1 *configure and unconfigure*.

36.4 start and stop

The `start` and `stop` files may be needed if your product needs to startup automatically at boot time and run until system shutdown. Refer to Chapter 14: *Automatic UPS Product Startup and Shutdown* for information on this topic. In the table file for this type of product you must include the actions `ACTION=START` and `ACTION=STOP`. These actions must include all the steps necessary to startup the product and shut it down. You may need to put these steps in scripts and execute them from the table file. You can call the scripts whatever you like, but we recommend `start` and `stop` for easy recognition.

Sample start and stop Scripts

We'll use scripts for **juke** v5_2 as examples.

The start script

```
#!/bin/sh

case "$0" in
/*)    JUKE_DIR=`echo $0 | sed -e 's:/ups/start:;'`
        export JUKE_DIR
        PATH=$JUKE_DIR/bin:$PATH
        ;;
*)     ;;
esac

cd $JUKE_DIR/log

host=`hostname`
local=`$JUKE_DIR/bin/juke show jukebox | grep $host | sed -e 's/@.*//'`

if [ "$local" != "" ]
then
    if [ -f $JUKE_DIR/log/jukerpcd.$host.pid ]
    then
        # it looks like one is running
        if kill -0 `cat $JUKE_DIR/log/jukerpcd.$host.pid`
        then
            #daemon is already running, we're done
            exit 0
        fi
    fi
    nohup $JUKE_DIR/bin/jukerpcd >> jukerpcd.$host.out 2>&1 </dev/null &
    echo $! > $JUKE_DIR/log/jukerpcd.$host.pid

    sleep 10          # wait for jukerpcd to wake up

    for i in $local
    do
        if [ "`uname -s`" = "AIX" ]
        then
            # AIX driver doesnt autoconfigure, so configure it
            dev=`$JUKE_DIR/bin/juke show jukebox |
                grep $local |
                sed -e 's:.*dev/;;' -e 's/[ ].*//'`
            mkdev -l $dev
        fi
        $JUKE_DIR/bin/juke online -j $i &
    done
fi
```

The stop Script

```
#!/bin/sh

if [ "" = "$JUKE_DIR" ]
then
    JUKE_DIR=`echo $0 | sed -e `s;/ups/stop;`
    export JUKE_DIR
    PATH=$JUKE_DIR/bin:$PATH
fi

host=`hostname`

if [ -f $JUKE_DIR/log/jukerpcd.$host.pid ]
then
    kill -15 `cat $JUKE_DIR/log/jukerpcd.$host.pid`
    rm $JUKE_DIR/log/jukerpcd.$host.pid
fi
```


Chapter 37: Use of Compile Scripts in Table Files

Compile scripts can be used in table files to preprocess actions, thus speeding up considerably the time it takes users to execute the actions. We describe the use of compile scripts in this chapter.

37.1 Overview

Generally, when a **UPS** command is issued, if **UPS** finds a corresponding action in the product instance's table file, the listed functions get executed. If this function list is lengthy, the command may take a long time to execute. To speed up execution in these cases, **UPS** v4 supports the preprocessing of actions in compile scripts. When you preprocess, you run the list of functions once, store the output in a script, and then when the command is later executed, the script is run instead of the functions.

This mechanism can be used for any **UPS** command, but it was developed with the **setup** command in mind. If a **setup** command must setup many, many required products, reading all the files for instance matching can be slow. By use of a compile script, the files can be read once, instead of each time a user runs **setup** on the product.

37.2 Usage Information

The use of compile scripts is most easily explained using an example. The (partial) table file below creates a compile script for the **setup** command when the product instance gets configured. Alternatively, since **ACTION=COMPILE** is defined, you could manually run the command **ups compile** to create the script. The functions listed are described in section 34.3 *Function Descriptions*:

```
ACTION=CONFIGURE
    exeActionRequired("COMPILE")

ACTION=COMPILE
    writeCompileScript("SETUP", "/my/compile/script")

ACTION=SETUP
    sourceCompileReq("/my/compile/script")
    doDefaults()
    setupRequired("dog v2_0")
    setupRequired("cat v1_1")
    ...long list...
    setupRequired("mouse v3_9")
```

This table file performs the following actions:

- 1) When the product instance is configured (via ACTION=CONFIGURE, which is usually run as part of **ups declare**), the function **exeActionRequired("COMPILE")** runs the functions under ACTION=COMPILE.
- 2) The function **writeCompileScript("SETUP", "/my/compile/script")** under ACTION=COMPILE executes a single operation: run **setup**, and write the output of the setup actions to the script `/my/compile/script`. This **writeCompileScript** function executes all the functions under ACTION=SETUP except the first one (**writeCompileScript** knows to ignore **sourceCompileReq**), and outputs the results to the script `my/compile/script`. For example, for each **setupRequired** line, it completes all the instance matching, and outputs the matched instance to the script.

Later, when **setup** is run by a user, the first function under ACTION=SETUP is executed (**sourceCompileReq ("my/compile/script")**), and the remaining functions are ignored. Therefore, none of the file reads have to occur during normal product setup.



The compiled script will contain hard-coded paths to the instances that were in effect when the script was created. If any product version, root directory or table file changes, the script must be recompiled for **setup** to work properly. Use **ups depend** to determine what the current dependencies are.

Chapter 38: Creating and Formatting Man

Pages

In this chapter we show you how to create man pages, format them, and even create html documents from them. This is not a comprehensive man page reference, but it contains sufficient information for most purposes.

For further information, from the **UNIX Resources** Web page, see “How to Create Man Pages” under *Software Development*.



First, a few notes:

- The man pages for a **UPS** product can go anywhere, as long as the location is specified in the table file. A recommended location is `${UPS_PROD_DIR}/man` for the formatted pages and `${UPS_PROD_DIR}/catman` for the unformatted pages. The **UPS** backwards-compatible default, however, is `${UPS_UPS_DIR}/toman/man` for the formatted pages and `${UPS_UPS_DIR}/toman/catman` for the unformatted pages.
- Man page file names should consist of the product name, a period, and the section number as described in the following note. This applies to both formatted and unformatted files, which are distinguished by residing in separate directories.
- Man pages for commands are generally maintained as section 1, and library and system calls as section 3. The section number should appear as an extension of the man page file name (e.g., `hello.1` for the command **hello**). Here is a full listing of categories by section:

1	user commands
2	system calls
3	C library functions (on some platforms 3c for C, 3f for FORTRAN, etc.)
4	devices and network interfaces
5	file formats
6	games and demos
7	environments, tables, and troff macros
8	maintenance commands
9	x window system
l	local commands
n	new commands (tcl and tk use this)
- We recommend using either of the utilities **nroff** or **groff** with the `-man` option to format your man pages in a standard way. These utilities are documented in many standard UNIX texts, and you can also find man pages for them.

38.1 Creating the Source Document (Unformatted)

38.1.1 Source File Format

We recommend writing man pages in the source form using simple macros from the **nroff** macro package **-man**. Most of these macros require a dot (.) in the first column. The following list of macros is sufficient for writing standard man pages:

.TH <name> <section> <date>

Title Heading; specify product name, man page section (usually 1), and date, in this order, to produce a man page format of this type:

name (section)	name (section)
...man page text...	
date	page number

- .SH "<text>"** Section Heading; if no blanks in text, quotes are not needed.
- .SS "<text>"** Subsection Heading; if no blanks in text, quotes are not needed.
- .P** Paragraph break
- .IP "<item>"** Starts an indented paragraph where "item" is put to the left of it; if no blanks in "item", quotes are not needed.
- .HP** Starts a paragraph with a hanging indent; i.e. lines after the first are indented
- .RE** Defines an indented region
- .B "<text>"** Bold; if no blanks in text, quotes are not needed.
- .I "<text>"** Italic; this shows up as underlined on most terminals. If no blanks in text, quotes are not needed.
- .TP <columns>** Term/paragraph format; columns specify how many columns to allocate to the term column. As an example, this input:

<pre>.TP 5 f1 is one option .TP f2 is another option</pre>
--

produces this output under **nroff -man**:

```
f1  is one option

f2  is another option
```

where “is” starts in column 6. Notice that the first **.TP** sets the column value of the term, and the second one picks it up.

.P	New paragraph
.br	Break line
.nf	Nofill (used to suppress normal line filling; used for preformatted text)
.fi	Fill (used to resume normal line filling, usually after a .nf)
./"	Comment line

38.1.2 Man Page Information Categories

Categories of information that you may want to include as section headings (**.SH**) are:

NAME

This should be the product name followed by a short description. The text on this line is also used as the keyword list for **man -k** and **apropos**.

SYNOPSIS or SYNTAX

Document here the complete syntax of the command used to invoke the product.

AVAILABILITY

Document here the OS flavors for which the program is available.

DESCRIPTION

Document here a full but succinct description of the use of the product.

OPTIONS

Document here all the options available for the invoking command.

EXAMPLES

Document here situations in which the program can be used, if there are uses that are not obvious.

NOTES

Document here any information the user should be aware of when using the command.

MESSAGES AND EXIT CALLS

Document here all errors and other messages returned to the user. Include the cause and the recovery actions whenever appropriate and possible.

AUTHOR

Document here the product coordinator and/or the major developers and contributors, along with their particular areas of expertise, as appropriate.

HISTORY

Document here the significant changes in each release of the product.

RESOURCES

If your product is designed to work under X windows, document here any X resources that affect the product's behavior.

FILES

Document here all files, or at least their directories if there are too many files. Also mention here any files in the user's home area that are needed/accessed (e.g., `$HOME/.mh_profile`, `$HOME/Mail/components` for the **mh** and **exmh** products).

BUGS

Document here things that do not (yet!) work as designed. Provide work-arounds whenever possible.

CAVEATS

Document here things that work *as designed* but which may be unclear or surprising to the user. (This is the System V replacement for the **BUGS** category; you too can pretend your product has no bugs!)

SEE ALSO

Document here other related commands and manual sections, especially if not obvious.

38.1.3 Example Source File

In section 16.1.5 we presented a simple example for the product **hello** showing how to create a formatted man page from a simple unformatted **nroff** input file. We will expand upon it here to illustrate the macros listed above. The **nroff** source is created in `$HELLO_DIR/man/hello.1`. Sample contents:

```
.TH HELLO 1 LOCAL
.SH NAME
hello - print "Hello world" on stdout
.SH SYNOPSIS
.B hello [options]
.I option option
.B ["
.I -yy -zz
.B ..."]
.SH AVAILABILITY
All UNIX flavors
.SH DESCRIPTION
hello prints the string "Hello world" on standard output.
.SH OPTIONS
There are no options, but we'll make some up.
.TP 5
-YY
is one option
.TP
-ZZ
is another option
.SH AUTHOR
U. R. Friendly
```

38.2 Formatting the Source File

38.2.1 nroff

To create an ascii-formatted man page, you can run the utility **nroff** with the **-man** macro package as follows:

```
% nroff -man <input_file> > <output_file>
```

We recommend following the prescription for unformatted and formatted man page locations as stated above and in section 15.3. This ensures that the source file always gets run through the formatter and the formatted file is never run through it again, which would produce odd results. First, **cd** to the source file directory:

```
% cd $HELLO_DIR/man
```

The following command creates the formatted man page for our **hello** example in the correct directory:

```
% nroff -man hello.1 > ../catman/hello.1
```

Once it is formatted, the example above will look like this:

```
HELLO(1)                                HELLO(1)

NAME
    hello - print "Hello world" on stdout

SYNOPSIS
    hello [options] option option [" -yy -zz ..."]

AVAILABILITY
    All UNIX flavors

DESCRIPTION
    hello prints the string "Hello world" on standard output.

OPTIONS
    There are no options, but we'll make some up.

    -yy is one option
    -zz is another option

AUTHOR
    U. R. Friendly

                                LOCAL                                1
```

38.2.2 groff

You can also use **groff** to format your man page source file. You must setup **groff** before use (not necessary for **nroff**). The command:

```
% groff -man -Tascii <input_file> > <ascii_output_file>
```

produces ascii-formatted man pages (the same output as the **nroff** command above). If you want to produce a PostScript output file, enter:

```
% groff -man <input_file> > <ps_output_file>
```

38.3 Converting your Man Page to html Format

An ascii-formatted man page can be run through the utility **man2html** and then accessed via a Web browser. First setup **conv2html**, then run the command:

```
% man2html -title '<manpage_title>' < <ascii_output_file> >\
  <html_file>
```

Glossary

This glossary defines terminology as it is used in the context of **UPS** and **UPD** v4.

action

Also called a **UPS** action. Actions are used in table files to group together functions that **UPS** must perform when a particular command is issued. An action consists of an ACTION=VALUE keyword (e.g., ACTION=SETUP) plus any functions listed underneath it.



active product instance

The product instance that is currently setup. The *active instance* may be different than the *current instance*.

archive UPS database

A **UPS** database on a product distribution node in which the **UPS** product instances are stored in archive format (e.g., tar, gzip), available for downloading to a user node. Also called a *distribution database*.

bootstrap

(In this manual, we discuss bootstrapping the **CoreFUE** product, which includes **UPS**, **UPD** and **perl**.) Install **UPS/UPD** on a machine on which no prior versions of these products are installed.

build

The process by which a distributable instance of a software product is constructed. The build procedure results in a unique combination of product name, version, flavor, and qualifiers. The actual process varies by product and by developer. It can simply consist of a set of copy commands, or be as sophisticated as generation of executables from a master source library of the software.

chain

A chain is a **UPS** database entry (in a chain file) that points to a declared product instance, tagging the product instance according to its release status (e.g., current, test). Chains allow users to specify the version of a product according to its status, rather than by its version number. The defined chain names are: *current*, *test*, *development*, *new*, and *old*. Their corresponding options (or flags) used in commands are: **-c**, **-t**, **-d**, **-n**, **-o**. The **-g <chainName>** option allows definition of an arbitrary chain name.

Chains are set by the **ups declare** command; hence the term *declare a product instance as current*.

chain file

Chain files reside in the product-specific directory under the **UPS** database directory, and maintain the chain information. Chain files are named according to the chain name, and end with *.chain*, e.g., *current.chain*. A chain file's contents is simply the list of the product instances (specified via sets of keyword/value pairs) that have been declared with that chain.

cluster

For the purposes of this document, a cluster is set of CPU nodes which share one or more **UPS** databases and product areas. Generally the nodes of a cluster also share (at least) login areas.

configure a product instance

For any product instance that requires configuration, an **ACTION=CONFIGURE** line is provided in its table file, with functions listed beneath it. In **UPS** *configuring a product instance* means executing these functions by issuing the **ups configure** command with appropriate options. This happens by default when a product is declared, otherwise it can be run manually. The functions perform all the configuration needed for the product to run, minus that which requires input from the installer (see *tailor a product instance* and *INSTALL_NOTE* for that portion).

coreFUE

A bundle of **UPS**, **UPD** and **perl**, the core pieces of the Fermi UNIX Environment.

current instance (of a product)

A product instance that is declared as current in the database (i.e., to which the chain “current” points). The current instance of a product is the default for **UPS** and **UPD** commands when no version or chain is specified. For a given product, there may be one current instance each for several flavor/qualifier pairs.

daemon process

A background process that is configured to start up automatically on a system at boot time and to stop at shutdown.

database

See *UPS database*.

database configuration file

The **UPS** database configuration file contains system-specific information that customizes the **UPS** installation on a node or cluster. If it exists, it must reside under the database directory in the file `/path/to/ups_database/.upsfiles/dbconfig`.

declare a product instance to UPS

The **ups declare** command makes a product instance known to the **UPS** database and accessible by **UPS**. Declaration does not by itself make the product instance usable since any product requirements (and often other conditions) must also be satisfied, but declaring the product instance is a prerequisite for use (unless you’re using **UPS** products without a database).

declare a product instance current

Declaring a product instance as “current” essentially tags it as the default instance (when its flavor/qualifiers are matched). The declaration creates a current chain file or chain file entry that points to the version file for the instance. Product instances can also be declared as test, development, new or old, or as a user-defined chain for easy access.

declared product instance

An instance of a product which has been declared to a **UPS** database.

default function

The functions (as listed in section 34.3 *Function Descriptions*) that a **UPS** command completes (in addition to its internal processes) if no corresponding **ACTION=COMMAND** keyword line is found in the matched table file, or if the function **doDefaultFaults([<ACTION>])** is listed under the corresponding **ACTION=COMMAND** keyword line. Only the commands **setup** and **unsetup** actually have default functions.

dependencies

Additional products that must be installed, declared, and setup to ensure the successful operation of a given product or to enable special features within it. When a product instance is setup, its dependencies also get setup by default.

distribution database

A **UPS** database in which **UPS** product instances are available for distribution to user nodes. A distribution database may be in archive or live format. The default distribution database at Fermilab is **KITS** which is maintained by the Computing Division on the node *fnkits.fnal.gov*.

distribution node

This term is used in **UPD** to refer to the node on which **UPS** products are stored and available for distribution to user nodes. A distribution node contains a distribution **UPS** database (can be live or archive) and a distribution products area, and runs **UPS**, **UPD**, a Web server and an **FTP** server (preferably **WU-FTP**). It is sometimes called a *server node*.

It is possible to maintain a distribution database on one machine running **UPS** and **UPD** and a Web server, and maintain the corresponding distribution products area(s) on a different one running an **FTP** server, if the machines share a file system.

end user

Anyone who uses **UPS** products, but does not install, update, maintain, or develop them.

FermiTools

FermiTools are Fermilab-developed software packages that are believed to have general value to other application domains, and thus have been made publicly available in a special subdirectory of **KITS** via anonymous **FTP** and **www**. They do not require **UPS**. Installation and use instructions come with each product.

Fermi UNIX Environment (FUE)

FUE started as a project for providing a cross-department, cross-division structure for the proposal, discussion, design and implementation of all things that affect the user when operating in a UNIX environment at Fermilab. Currently it consists of scripts and programs that form a uniform UNIX environment, standards documents, and the **UPS** suite of tools (see <http://www.fnal.gov/cd/FUE/>).

flavor

To indicate the operating system (OS) dependency of a product instance, we use the term *flavor*. This extra term allows us to differentiate by operating system, and optionally OS version, while maintaining the same product name and version number for separate instances. Some products do not require customizing for the different operating systems (typically those without compiled code), but most do and therefore come in several flavors.

flavor table

A list of a machine's flavor including every level of specificity that you could use to find or declare a product instance. For example, on a SunOS+5.6 machine, the complete flavor table reads:

```
SunOS+5.6
SunOS+5
SunOS
NULL
ANY
```

FTP server node

As regards **UPD**, this node contains **UPS** product instances (and files associated with them) that may be downloaded to a user node, and it runs an **FTP** server. Usually it is the same node as the Web server node, and called simply the *server node* or the *distribution node*.

FUE

See *Fermi UNIX Environment*.

fullFUE

A bundle of **coreFUE** plus the pieces which are strongly recommended for on-site systems: **systools**, **shells** and **futil**.

function

A **UPS**-defined entity used in table files that executes an operation within an action. The supported functions are listed in section 34.3 *Function Descriptions*. One or more functions always follow an ACTION=VALUE keyword line.

A function is specified in a shell-independent manner, but contains enough information to allow it to be transformed into a **sh** or **csh** family command (e.g., **sourceRequired()**, or **execute()**), or to be interpreted directly by **UPS** (e.g., **writeCompileScript()**).

install a product instance

Copy a product instance to a local system from another location (usually from a distribution node) and perform the necessary steps to make it work.

INSTALL_NOTE

A file that describes procedures that the installer must perform manually to complete the installation of a product. This file is provided by the product developer as needed.

instance

See *product instance*.

internal processes (or internals)

The set of processes that a **UPS** command completes, regardless of the contents of the product instance's table file. The internal processes are driven by the command line parameters and options, and relevant environment variables.

keyword

Keywords are used in the **UPS** database files. They are essentially parameters to which values must be assigned. The supported set of keywords listed in section 27.4 *List of Supported Keywords* collectively contains the information **UPS** requires for managing a **UPS** installation and all its **UPS** products. Some of the keywords can be used in all the **UPS** product management file types, others are restricted to certain file types.

keyword value

The value assigned to a keyword in one of the **UPS** database files.

KITS

The name of the **UPS** product distribution database on the central product distribution node at Fermilab, *fnkits.fnal.gov*. The location of the **KITS** database is `/ftp/upsdb`. **UPS** products are stored in the corresponding product area, `/ftp/products` (symlinked to `/ftp/KITS`), as tar files, generally. **UPD** commands access the **KITS** database and products area by default.

live UPS database

A **UPS** database in which the **UPS** product instances are unwound, i.e., not stored in archived format (e.g., tar, gzip).

local UPS database

A live **UPS** database on a local node. For user nodes, a database in which **UPS** product instances are declared and available to be accessed and used.

local user node

See *user node*.

make

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. See *UNIX at Fermilab* or a standard UNIX reference text for more information.

Makefile

First, see *make* above. A Makefile is a blueprint that you design and that **make** uses to create or update one or more target files (usually executables) based on the most recent modify dates of the constituent files. See *UNIX at Fermilab* or a standard UNIX reference text for more information.

operating system (OS)

A control program for a computer that allocates computer resources, schedules tasks and provides the user with a way to access the resources. See document *DR0010* in the Computing Division Web pages for the latest information on supported UNIX operating systems at Fermilab.

operating system version (OS version)

Like other software, an operating system gets fixed and enhanced periodically, and is released by the vendor with a new version number (e.g., IRIX 5.1, IRIX 5.2). Sometimes **UPS** products must be changed to continue to work properly under a new operating system version.

operating system type (OS type)

The name of the basic operating system, without release number, as returned by the command **ups flavor -2** (for example IRIX or SunOS).

overlay

An overlaid product gets distributed and maintained in the product root directory of its main product. The set of products overlaid on a main product is collectively referred to as *the overlay*.

parent product

A dependency's *parent product* is that for which it is a dependency. A product may have multiple parent products.

platform

Platform technically refers to the machine type (hardware) of a computer system. However, since until quite recently in the UNIX world there has been a near-perfect correspondence between hardware platform and OS type (e.g., Digital Alphastations run OSF1), sometimes platform is used loosely to refer to the OS type. This correspondence is changing as Linux can be run on PC, Digital, Sun and IBM hardware.

process an action

UPS converts the shell-independent functions listed underneath an ACTION keyword line in a table file into code appropriate to the shell, and writes the output to a temporary file. This is call processing an action.

product

See *UPS product*

product developer

A person who develops and maintains software products, and makes them available for distribution by installing and declaring them to the **KITS** or other distribution database. Sometimes called a product maintainer.

product installer

A person who downloads **UPS** products from a distribution node (through **UPD**, **UPP** or **FTP**), installs them on a local system, and declares them to a local **UPS** database (often the local system administrator acts as the product installer).

product instance

The term *product instance*, or just *instance*, is used to represent a copy of a product, namely a unique combination of product name, version, flavor and qualifiers within a **UPS** database. For a given product, multiple instances may exist in the database to allow users a choice of version and/or flavor/qualifier pair. A product instance may be chained; hence the term “the current instance of a product”.

product name

The name of a **UPS** product as it appears in its **UPS** database files.

product root directory

The directory in which a product instance (i.e. its executables) and (optionally) its associated files reside. The product instance generally has a directory structure of its own, starting at this root directory. Each instance of a product has a separate product root directory.

product user

See *end user*.

product version

The net result of any change to an existing product is that a new *version* of the product is created; it is still the same product, but it will usually run a little differently. The versions of a product are tracked by version numbers, e.g., v1_0, v1_1, etc. **UPS** allows for multiple versions of a given product to be accessible concurrently to end users.

PRODUCTS (or \$PRODUCTS)

The environment variable that points to the **UPS** database(s) on your system. If multiple **UPS** databases exist, \$PRODUCTS can be reset in your login files to a colon-separated list of databases.

<PRODUCT>_DIR (or \$<PRODUCT>_DIR)

PRODUCT here is the name of a product in upper case (e.g., EMACS_DIR). This is the environment variable that points to the product root directory of the active instance of a particular product; it gets set when the **setup** command is run.

qualifier

The product developer may include information about options used at compilation time (e.g., *debug* or *optimized*) or other qualifying information for easy identification of special compilations. This information is declared in the form of *qualifiers*. Qualifiers, when present, are part of the unique instance identification along with product name, version and flavor.

read-only variable

UPS sets several read-only variables that can be used in functions in table files. Many of them correspond to keywords set in the **UPS** configuration file. There is another set of read-only variables available for use in setting location definitions in the **UPD** configuration file.

root directory for product

See *product root directory*.

setup

Each installed, declared **UPS** product instance requires that the **setup** command be issued prior to use (unless it is a dependency of one that is already setup). **setup** performs the necessary operations in your login environment to make an installed, declared product accessible to you. Typically, the operations include modifying environment variables or adding to your \$PATH. Any dependencies defined for the product get setup by default at the same time.

table file

Table files contain non-system-specific and non-shell-specific information that **UPS** uses for installing, initializing, and otherwise operating on product instances. That is, information pertinent to one or more product instances, independent of the installation machine. Table files are provided by the product developer as needed.

tailor a product instance

Tailoring is the aspect of the product implementation that requires input from the product installer (e.g., specifying the location of hardware devices for a software driver package). If the product requires tailoring, a file is usually supplied in the format of an interactive executable (script or compiled binary), and it is run by issuing the **ups tailor** command with appropriate options. To *tailor a product instance* means to run this action, and hence, run the file.

tar

The **tar** (tape archive) utility can create, add to, list, and retrieve files from an archive file.

tar file

A tar file is in archived format, and must be unwound for use. **UPS** products are generally stored in `KITS` as tar files.

unknown command handler

A **UPS** feature that allows user-defined actions (e.g., `ACTION=XYZ` followed by **UPS**-supported functions) in table files that can be run via a corresponding **UPS**-style command (e.g., **ups xyz [<options>] <product> [<version>]**)

unsetup

unsetup generally undoes the changes to the user's software environment made by **setup** in order to make the product no longer available for use. Any dependencies get deactivated automatically at the same time by default.

UPD - Unix Product Distribution

A companion product to **UPS** which provides the functionality for uploading/downloading products between local systems and product distribution servers.

UPD commands

Any of the commands supported by **UPD**. They are listed and described in Chapter 23: *UPD/UPP Command Reference*. These include commands to retrieve **UPS** products or certain individual files or directories from a distribution database, and commands to manage products within a distribution database.

UPP - Unix Product Poll

A layer on top of **UPD** that allows a client to request notification of changes in a distribution node database and to download pre-specified products. **UPP** can be automated. This is a useful tool for keeping abreast of changes/enhancements to your favorite products.

UPS - Unix Product Support

UNIX Product Support (**UPS**) is a software support toolkit which provides a methodology for creating/managing all the UNIX products provided and/or supported by the Computing Division, and a uniform interface for accessing these products. **UPS** is itself a product that must be installed on any machine that will be used to run other **UPS** products.

UPS has two parts: one or more databases which function as a central repository of information about the products, and a set of procedures/programs to manipulate the database(s).

UPS action

See action.

UPS commands

Any of the commands supported by **UPS** to manage products in a **UPS** environment. They are listed and described in section Chapter 22: *UPS Command Reference*.

UPS database

A directory that functions as a repository of information about all the installed, accessible **UPS** product instances on a system. **UPS** allows multiple installed and declared instances of each product. The database contains files for each product which store pointers to and information about the declared instances of the product.

ups directory (or ups subdirectory)

A directory that may contain miscellaneous important files for a product instance; e.g., its table file, scripts that the table file needs to execute, and so on. This directory may reside anywhere; it often resides directly under the product instance's root directory. Not all products have `ups` directories.

UPS product

Software products distributed and managed by the **UPS** system are called **UPS** products. **UPS** products include Fermilab-written programs, a wide range of public domain software, and a host of third party licensed (proprietary) products. **UPS** products are available for distribution in the `KITS` database on *fnkits.fnal.gov*.

user node

A node from which users can run **UPS** products; usually contains a live local **UPS** database and locally-installed products.

version

For a product see *product version*; for an operating system see *operating system version*.

version file

A version file contains system-specific information for each instance of a **UPS** product. One *version file* must exist in the product-specific directory under the **UPS** database directory for each version of a product that is declared to the **UPS** database. The name of the version file is the version number followed by `.version`, e.g., `v2_2.version`.

Web server node

As regards **UPD**, this node contains one or more distribution databases and runs a Web server, and **coreFUE**. Usually it is the same node as the **FTP** server node, and called simply the *server node* or the *distribution node*.

Index

Symbols

"-?" option 2-1, 10-1
+ argument for -K option 2-3
.upfiles directory 1-6
.upsfiles directory 1-6
/etc/init.d directory 14-5
/etc/rc*.d directories 14-5
/usr/local/ area
 Fermilab policy regarding use of 15-2, 15-3
@ symbol 27-8
 use with keywords 22-46
_UPD_OVERLAY keyword 16-7, 27-11
 description 27-8

Variables

\$_<PRODUCT>_DIR variable 34-18
 as set during setup 22-5
 description 22-5
\${DASH_PROD_FLAVOR} read-only variable 31-4
\${DASH_PROD_QUALIFIERS} read-only variable 31-5
\${PROD_DIR_PREFIX} read-only variable 31-5
\${PRODUCTS} read-only variable
 comparison to PRODUCTS env variable 34-19
 description 34-19
\${SUFFIX} read-only variable 31-5
\${UPD_USERCODE_DB} read-only variable 3-4
\${UPD_USERCODE_DIR} read-only variable 3-4
\${UPS_BASE_FLAVOR} read-only variable 31-4
\${UPS_COMPILE} read-only variable
 description 34-19
\${UPS_EXTENDED} read-only variable
 description 34-19
\${UPS_OPTIONS} read-only variable
 description 34-19
\${UPS_ORIGIN} read-only variable
 description 34-19
\${UPS_OS_FLAVOR} read-only variable
 description 34-19
\${UPS_PROD_DIR} read-only variable
 description 34-19
\${UPS_PROD_FLAVOR} read-only variable 31-4
 description 34-19
\${UPS_PROD_NAME} read-only variable 31-4
 description 34-19
\${UPS_PROD_QUALIFIERS} read-only variable 31-4
 description 34-19

\${UPS_PROD_VERSION} read-only variable
 description 34-19
\${UPS_THIS_DB} read-only variable
 description 34-19
\${UPS_UPS_DIR} read-only variable
 description 34-20
\${UPS_USERCODE_DB} read-only variable 31-4
\${UPS_USERCODE_DIR} read-only variable 31-4
\${UPS_VERBOSE} read-only variable
 description 34-20
\$PATH variable 1-10, 2-10, 22-11
\$PRODUCTS variable 1-6, 1-10, 25-4
 as used in UPD commands 26-1
 as used in upd install 5-2
 comparison to read-only \${PRODUCTS} 34-19
 for multiple databases 25-2
 use in database selection 26-1
 use with private database 11-9
 with AFS database 12-4
\$SETUP_<DIR> variable 34-18
 as set during setup 22-5
\$SETUP_<PRODUCT> variable
 description 22-5
 use with unsetup command 22-6, 22-11
\$SETUP_UPS variable 1-10
\$TEMPDIR variable
 use with upd addproduct 17-1, 23-7
\$UPS_DIR variable 1-10
\$UPS_EXTENDED variable
 as set by -e option 24-2
\$UPS_EXTRA_DIR variable 12-5
\$UPS_OPTIONS variable
 as set by -O option 24-4
\$UPS_SHELL variable 1-10

"@" Keywords

@COMPILE_FILE keyword 22-47, 27-9
@PROD_DIR keyword 22-48, 27-9
@TABLE_FILE keyword 22-48, 27-10
@UPS_DIR keyword 22-48, 27-11

A

access.conf file 20-11
accessing a UPS product 2-8, 22-5

- accounts
 - for managing distrib node 20-3
 - for product installation 11-1, 11-2
 - ftp 20-3, 20-8
 - separate by product category 11-2
 - the products account 11-1
 - upadmin 20-3, 20-5, 20-12, 21-6
 - wwwadm 20-3, 20-4, 20-7, 20-8
- ACTION keyword
 - "unchain" names as values 33-3
 - chain names as values 33-3
 - description 27-4
 - detailed 31-5, 33-1
 - UPS command as keyword value 33-1
 - use in table files 34-1
 - user-defined values 33-3
- actions
 - "unchain" name as keyword value 33-3
 - and "unactions" 33-2
 - called by other actions 33-4
 - chain name as keyword value 33-3
 - examples 34-18
 - functions used in 34-1
 - overview 31-5, 33-1
 - processing of 24-9
 - reference 33-1
 - undoing chains in table files 33-3
 - undoing reversible functions 33-2
 - UPS commands used as 33-1
 - use in table files 33-1
 - use in updconfig 31-5
 - use with "unknown" commands 33-3
- add chain to product on distrib node 17-7, 23-33
- add product to distrib node 17-3, 23-3, 23-8
 - using template_product 18-6
- add product to KITS 17-3, 23-3, 23-8
 - special product registration 17-3
- add table file to distrib node 17-5
 - update for existing product 17-6
- add ups directory to distrib node
 - update to existing product 17-6
- addAlias function
 - description 34-2
- AFS
 - \$PRODUCTS variable 12-4
 - \$UPS_EXTRA_DIR variable 12-5
 - configuring local database 12-2
 - installing into local database 12-5
 - installing into local products area 12-4
 - installing product into AFS product area 8-3
 - local configuration options 12-1
 - local FUE initialization files 12-3
 - products requiring special privileges 12-6
 - providing access to AFS products 12-1
 - updating /usr/local/bin 12-6
 - upsdb_list file 12-2
 - using AFS UPD and installing locally 8-2
 - using local database with 12-1, 12-2
- AFS database
 - use with local database 5-3
- aliases defined by UPS 1-10
- announcement of new/updated product 17-10
- anonymous FTP 7-5
 - download files from fnkits 7-2

- apache product
 - for distrib node web server 20-5, 20-10
- apropos command 38-3
- ARCHIVE_FILE keyword 22-47
 - as set by -T option 24-4
 - description 27-4, 28-2
- AUTHORIZED_NODES keyword 22-47, 30-1
 - as set by -A option 24-1
 - description 27-4, 28-2
- autostart
 - configuring UPS to allow 14-1
 - control files 14-3
 - permissions 14-4
 - disabling 14-5
 - installing product for 14-2
 - START action 14-3
 - start script example 36-4
 - STOP action 14-3
 - stop script example 36-5
 - TAILOR action 14-3
 - ups script 14-1
 - ups_shutdown script 14-1, 14-2
 - ups_startup script 14-1, 14-2

B

- bin directory of product 16-1, 16-3, 16-5, 18-4, 19-1
 - description 15-6
- bootstrapping CoreFUE
 - bootstrap script 13-1, 13-5
 - config.custom file 13-2
 - configurator script 13-2
 - customizing configuration 13-3
 - log file 13-5
 - predefined configurations
 - for NT 13-2
 - for UNIX 13-1
 - running the procedure 13-5
 - sample customization 13-4
 - space requirements 13-1
 - stage1.sh file 13-1, 13-5
 - stage2.sh file 13-5
 - user defined configurations 13-2
 - user-customized configuration 13-2

C

- catman directory 15-7
- CATMAN_SOURCE_DIR keyword 22-47
 - description 27-4
- CATMAN_TARGET_DIR keyword 22-47, 30-1
 - description 27-4
- CD-ROM
 - product distribution 20-14
 - setup product directly from 22-7
- chain
 - adding product to distrib node 17-3, 23-7
 - as action in table files 33-3
 - change (on declared instance) 10-7
 - current 1-4
 - declare at product declaration 3-6, 10-2
 - declare to installed instance 10-4

- definition 1-4
- development 1-4
 - new 1-4
 - old 1-4
 - remove and add new 10-7
 - remove from instance 10-6
 - specification in command 25-1
 - test 1-4
 - usage 1-5
 - use in instance matching 26-3
 - user-defined 1-4
- chain files 1-6, 22-79, 29-1
 - and product removal 10-7
 - creating 29-1
 - description 29-1
 - examples 29-3
 - information storage format 29-1
 - instance matching within 26-3
 - keywords 29-1
 - overview 27-1
- CHAIN keyword 22-47
 - description 27-4, 29-2
- chain names 1-5
- chain options 1-5
- change a chain 10-7
- change product chain on distrib node 17-7
- command defaults 1-8
- command output formats for ups list 24-7
- command syntax 1-8
 - description 25-1
- comment solicitation INT-5
- COMMON: keyword 35-3
 - description 27-4
 - use in table files 35-3
 - use in updconfig file 31-2
- COMPILE action 37-1
- compile script 37-1
- COMPILE_DIR keyword 22-47, 27-9
 - description 27-4, 28-2
- COMPILE_FILE keyword 22-47, 27-9
 - as set by -b option 24-1
 - description 27-4, 28-2
- config.custom file 13-2
- configurator script 13-2
- configure a product instance 3-9, 22-13
 - in AFS space 8-5
- CONFIGURE action 10-8, 22-80, 36-1
- configure script 36-1
 - for prebuilt binaries 16-5
- configuring distribution node 20-1
- conventions, notational INT-3
- copy a product declaration 22-19
- CoreFUE
 - and AFS 12-1
 - bootstrapping 13-1
 - components 12-4, 12-5, 13-1
 - customizing configuration 13-3
 - local installation on AFS machine 12-4
 - predefined configurations
 - for NT 13-2
 - for UNIX 13-1
 - running the bootstrap procedure 13-5
 - sample bootstrap customization 13-4
 - space requirements 13-1
 - user defined configurations 13-2

- courtesy links to initialization files 1-9
- create a database
 - checklist for preparation 11-9
 - on machine running AFS 12-2
- cron
 - use to automate UPP 4-3, 6-4
- CURRENT action 36-3
- current chain 1-4
 - as default 1-8
- current script 36-3
- CVS 17-9
 - use with template_product 18-8
- CYGWIN
 - bin directory 11-8
 - perl version 11-7
 - UPS/UPD installation issues 11-7

D

- database (See UPS database)
- database configuration file (See UPS configuration file)
- database files
 - chain files 29-1
 - included comments 27-3
 - keywords 27-1
 - location 11-6
 - ownership 11-3
 - permissions 11-3
 - pointers to directories 11-6
 - syntax 27-3
 - UPD configuration file 31-1
 - UPP subscription file 32-1
 - UPS configuration file 30-1
 - version files 28-1
- database on distrib node
 - file permissions 20-7
 - host-based access restriction 20-6
 - user-based access restriction 20-6
- database selection algorithm 5-2, 26-1
- database specification in commands 25-4
- dbconfig file (See UPS configuration file)
- dbconfig.template file 30-1
 - listing 30-2
- declare a chain to an instance 3-6, 10-2, 22-21
- declare a product 3-5, 10-1, 22-21
 - after download via FTP 3-5, 10-1
 - as part of installation 5-1
 - declare chain at same time 3-6, 10-2
 - node/flavor-specific functions present 10-4
 - specifying ups dir and table dir 3-5, 10-2
 - to local database 7-4
- DECLARED keyword 10-6, 22-47
 - description 27-4, 28-2, 29-2
- DECLARER keyword 10-6, 22-47
 - description 27-4, 28-2, 29-2
- defaults for UPS/UPD commands 1-8
 - Also see command reference chapters
- delete product component from distrib node 17-8
- delete product from distrib node 17-8
 - using template_product 18-8
- dependencies
 - and unsetup command 22-11
 - conflict resolution 35-4

- cross-database support for 1-5
- database selection for install 5-3
- definition 1-5
- finding them for a product 2-7
- list using ups depend 2-7
- multiple levels of 1-5
- non-UPS products 35-4
- on distribution node, list using upd depend 4-5
- order of product setups 35-5
- setupOptional function in table file 35-4
- setupRequired function in table file 35-4
- dependency matching 26-2
- DESCRIPTION keyword 22-47
 - description 27-4, 28-2, 29-2
- determine if product update needed
 - using upd install -s 10-13
 - using upd update -s 10-13
 - using upp 10-13
- development chain 1-4
 - use during product development 16-2
- distributing UPS products
 - announcement policies for new products 17-10
 - overview 17-1
 - to KITS (checklist) 19-3
 - to KITS (using template_product) 19-3
- distribution node
 - ~ftp area 20-4
 - access restrictions on database
 - host-based 20-6
 - user-based 20-6
 - configuration and management 20-1
 - configure and manage 20-1
 - fnkits.fnal.gov 3-2, 7-2
 - FTP server 20-1
 - configuration 20-7
 - KITS database (on fnkits.fnal.gov) 3-2
 - limiting product distribution 20-11
 - nodes other than fnkits 7-4
 - option_list product description 20-12
 - reporting on FTP and Web accesses 20-10
 - response to upd addproduct command 20-2
 - response to UPD commands 20-1
 - response to upd install command 20-2
 - response to upd modproduct command 20-2
 - restrict downloads from database 20-11
 - restrict uploads to database 20-11
 - updconfig pre and postdeclare actions 20-10
 - user accounts 20-3
 - web server 20-1
 - configuration 20-5
- doc directory 15-7
- documentation for products 15-7
- doDefaults function
 - description 34-3

E

- editing database files 10-11
- END: keyword 35-3
 - description 27-4
 - use in table files 35-3
 - use in updconfig file 31-2

- envAppend function
 - description 34-3
- environment
 - and usage of command options 25-4
 - changes made by UPS 1-10
 - initializing for UPS 1-9
- envPrepend function
 - description 34-4
- envRemove function
 - description 34-4
- envSet function
 - description 34-5
- envSetIfNotSet function
 - description 34-5
- envUnset function
 - description 34-5
- examples directory 15-7
- exeAccess function
 - description 34-6
- exeActionOptional function
 - description 34-6
 - use to call another action 33-4
- exeActionRequired function
 - description 34-6
 - use to call another action 33-4
- execute function
 - description 31-6, 34-7
 - use in dbconfig 31-6

F

- Fermi UNIX Environment
 - initializing 1-9
- FermiTools INT-2, 4-6, 7-1, 7-2, 21-3
- FILE keyword 30-1
 - description 27-5, 28-2, 29-2
- file ownership
 - considerations 11-3
 - database files 11-3
 - product files 11-3
- file permissions
 - configuring UPD to set (product files) 11-2
 - database files 11-3
 - extra security 11-3
 - unwound tar files 11-2
- file system semantics
 - and group ids 11-2
 - Berkeley 11-2
 - setting 11-2
 - System V 11-2
- fileTest function
 - description 34-7
- flavor
 - ANY, as used in flavor matching 26-4
 - definition 1-3
 - NULL 1-3
 - specification in KITS 1-3
- FLAVOR keyword 22-47
 - description 27-5, 28-2, 29-2
 - value ANY 35-3
- flavor levels 2-2, 24-7
- flavor of machine, determining 2-1, 22-35

- flavor specification
 - (-f, -H and number options) 1-3
 - use in instance matching 26-3
- flavor table 24-7
 - definition 2-2, 22-36
- flavor.products file 14-3, 14-5
 - permissions 14-4
- fnalonly products 21-3
- fnkits.fnal.gov distribution node 4-1
 - adding products to 23-8
 - anonymous FTP for downloading products 7-1, 7-2
 - config file locations 21-6
 - database location 21-6
 - directory hierarchy 4-6
 - FermiTools 4-6, 7-1, 7-2
 - FTP server log file 21-7
 - ftpgroups file 21-6
 - KITS database 3-2
 - KITS product categories 21-3
 - product pathnames for FTP access 4-7, 4-8
 - product permissions 4-6
 - proprietary products 4-8
 - registration for downloading products 3-2, 7-2
 - server maintenance 21-6
 - using FTP to download products 7-1
 - web server log file 21-7
- formatted ups list output 22-45
- FTP
 - declare product after download 3-5, 10-1
 - downloading product components 7-1
 - product installation 7-1, 7-2, 7-5
- FTP server
 - access file 20-11
 - log file on fnkits 21-7
 - log searching 20-13
 - on distrib node 20-1
- ftpaccess file 20-7, 20-11
- ftpgroups file 21-6
- ftpweblog product 20-10
- FUE initialization files
 - courtesy links to 12-3, 12-5, 12-6
 - for use with AFS 12-3
- functions
 - addAlias 34-2
 - case (in)sensitivity of 34-1
 - doDefaults 34-3
 - envAppend 34-3
 - envPrepend 34-4
 - envRemove 34-4
 - envSet 34-5
 - envSetIfNotSet 34-5
 - envUnset 34-5
 - examples 34-18
 - exeAccess 34-6
 - exeActionOptional 34-6
 - exeActionRequired 34-6
 - execute 31-6, 34-7
 - fileTest 34-7
 - overview 34-1
 - pathAppend 34-8
 - pathPrepend 34-8
 - pathRemove 34-9
 - pathSet 34-9
 - preprocessing via compile script 37-1
 - prodDir 34-9
 - reference 34-1
 - reversible 33-2, 34-1
 - setupEnv 34-10
 - setupOptional 34-10
 - setupRequired 34-10
 - sourceCompileOpt 34-11
 - sourceCompileReq 34-11
 - sourceOptCheck 34-12
 - sourceOptional 34-13
 - sourceReqCheck 34-13
 - sourceRequired 34-14
 - to be added in future 34-17
 - translation into shell commands 24-9
 - unAlias 34-14
 - unProdDir 34-14
 - unsetupEnv 34-15
 - unsetupOptional 34-15
 - unsetupRequired 34-16
 - use with ACTION keyword 34-1
 - writeCompileScript 34-16

G

- g option for user-defined chain 1-5
- groff command
 - ascii output 38-6
 - man option 38-1
 - PostScript output 38-6
- GROUP: keyword 35-3
 - description 27-5
 - use in table files 35-3
 - use in updconfig file 31-2

H

- hardcoded paths problem 15-4
- help on UPS/UPD commands 2-1, 10-1
- help online
 - ups help command 22-41
- html directory 15-7
- HTML_SOURCE_DIR keyword 22-47
 - description 27-5
- HTML_TARGET_DIR keyword 22-47, 30-2
 - description 27-5

I

- include directory 15-7
- independent table file 17-5
- Info directory 15-7
- INFO_SOURCE_DIR keyword 22-47
 - description 27-5
- INFO_TARGET_DIR keyword 22-47, 30-2
 - description 27-5
- init.d directory
 - location 14-1
- initializing UPS environment 1-9
 - courtesy links to files 1-9

- INSTALL_NOTE file 7-1, 15-6, 19-1
 - configuring product 22-15
 - mention of node/flavor-specific functions 10-4
 - mention of unconfigure actions 10-8
 - sample 16-9
- installation methods for UPS products, summary 3-1
- installer accounts
 - choosing 11-1
 - file system semantics 11-2
 - multiple 11-1, 11-2
 - products account 11-1
 - separate by product category 11-2
 - setting gid 11-1, 11-2
 - single 11-1
 - UPD configuration issues 11-2
- installing a product
 - choose whether to declare qualifiers 3-8
 - components to download (using FTP) 7-1
 - configuring 3-9
 - declare manually after FTP download 7-4
 - for development/testing 5-3
 - interruption during install 3-8
 - into AFS space 8-3
 - into private database 11-9
 - KITS product categories 17-3
 - KITS special product registration 17-3
 - local install using AFS UPD 8-2
 - onto distrib node 17-3
 - pass options to local declare 5-2
 - procedural checklist when using UPD 5-3
 - products requiring special privileges 8-1, 12-6
 - root privileges 12-6
 - table file product 17-5
 - tailoring 3-9, 22-67, 22-69
 - troubleshooting 9-1, 10-17
 - ups installasroot command 12-6
 - using FTP 7-1, 7-2, 7-4
 - using UPD 5-1
 - using UPP 6-1
 - with all dependencies (using UPD) 5-5
 - with different name than on server 3-8
 - with no dependencies (using UPD) 5-7
 - with required dependencies (using UPD) 5-7
- instance
 - declare a chain for 10-4
 - definition 1-4
 - determine if update needed 10-13
 - determine instance to act upon 26-1
 - install and declare 5-1
 - specification via chain or version 25-4
 - specify multiple ones in command 25-3
 - verify integrity of 10-10
- instance matching 26-1
 - in chain file 26-3
 - in table file 26-3
 - in updconfig file 31-2
 - in version file 26-3
 - use of flavor and qualifiers 26-4
- instance selection by chain 1-4
- instance specification on command line 25-4
- internal command processes 24-9

K

- K option
 - description for use with ups list 22-46
 - keyword arguments 2-3, 22-46
 - with upd list 4-2
 - with ups depend or upd depend 2-8, 22-30
- keywords 27-1, 28-1
 - case (in)sensitivity of 27-2
 - DECLARED 10-6
 - DECLARER 10-6
 - definition 27-2
 - in ups list output 2-3
 - list with descriptions 22-47, 27-3
 - list with file types 27-3
 - MODIFIED 10-6, 10-13
 - MODIFIER 10-6
 - overriding values 27-3
 - syntax 27-2, 27-8
 - use of @ symbol 22-46
 - used with -K option in ups list 2-3
 - user-defined 27-2
- KITS 4-1
 - adding products to 23-8
 - dbconfig file 21-1
 - FermiTools 7-1, 21-3
 - fnalonly products 21-3
 - product categories 21-3, 23-8
 - product registration for special categories 21-3
 - proprietary products 21-3
 - registration 4-6, 7-2
 - updconfig file 21-2
 - updconfig pre and postdeclare actions 21-4
 - using FTP to download products 7-1
 - US-only products 21-3
- KITS distribution database 17-3

L

- lib directory 15-7
- licensed products
 - permissions 11-3
- link for hard-coded paths 36-2
- links to initialization files 1-9
- list all current products 22-49
- list all fields for a product 2-6, 22-51
- list dependencies on distribution node 4-5, 23-15
- list product dependencies 2-7, 22-27
- list products in database 2-4, 22-49
- list products on distribution node 23-31
 - use in troubleshooting product installs 9-1
- location of database files 11-6
- location of product files, considerations 11-4, 11-5

M

- man directory 15-7
- man -k command 38-3

- man page
 - ascii output 38-6
 - convert to html 38-6
 - determine directory for 11-6
 - file names 38-1
 - groff 38-1
 - information categories 38-3
 - location of files 38-1
 - nroff 38-1
 - nroff output file 38-5
 - nroff source file 16-3, 38-4
 - PostScript output 38-6
 - section numbers 38-1
- MAN_SOURCE_DIR keyword 22-47
 - description 27-5
- MAN_TARGET_DIR keyword 22-47, 30-2
 - description 27-5
- man2html command 38-6
- managing distribution node 20-1
- matching product instance
 - in chain file 26-3
 - in table file 26-3
 - in updconfig file 31-2
 - in version file 26-3
 - use of flavor and qualifiers 26-4
- MODIFIED keyword 10-6, 22-47
 - description 27-5, 28-2, 29-2
 - updating 22-71
 - used to determine if update needed 10-13
- MODIFIER keyword 10-6, 22-47
 - description 27-5, 28-2, 29-2
 - updating 22-71
- multiple databases
 - adding a private database 11-9
 - AFS and local 8-2
 - and your UPD configuration 3-4
 - configuring UPD for 31-9
 - database selection algorithm 26-1
 - default database 1-8
 - how UPD selects a database 5-2, 26-1
 - reasons for using 11-6
 - specifying \$PRODUCTS 1-8, 25-2
 - support for 1-6
 - z option for specifying database 24-5

N

- new chain 1-4
- news directory 15-7
- NEWS_SOURCE_DIR keyword 22-48
 - description 27-6
- NEWS_TARGET_DIR keyword 22-48, 30-2
 - description 27-6
- NFS-mounted database
 - using local database with 12-1
- NIS cluster 12-1
- node.products file 14-3, 14-5
 - permissions 14-4
- notational conventions INT-3
- nroff command 38-4
 - for man page 16-3
 - man option 38-1, 38-5
- NULL flavor 1-3

- number options (-0 through -3) 2-2, 22-36
 - usage information 25-4

O

- old chain 1-4
- online help
 - ups help command 22-41
- option flags
 - command-specific info in reference chapters
 - embedded spaces in arguments 25-2
 - grouping in commands 25-2
 - invalid arguments 25-3
 - multiple arguments 25-2
 - multiple occurrences 25-3
 - wildcards 25-4
- option usage in commands 25-4
- option_list product
 - description 20-12
- order of command line elements 25-1
- ORIGIN keyword 22-48
 - description 27-6, 28-2
- OS determination using ups flavor 2-1, 22-36
- overlaid products 1-6, 16-7, 27-11
- overlays 1-6, 16-7, 27-11

P

- parent product determination 10-8, 22-79
- parse ups list output
 - in perl 22-52
 - in sh script 22-53
- pathAppend function
 - description 34-8
- pathPrepend function
 - description 34-8
- pathRemove function
 - description 34-9
- pathSet function
 - description 34-9
- perl
 - parse ups list output in 22-52
 - version for use with CYGWIN 11-7
- permissions
 - configuring UPD to set for product files 11-2
 - database files 11-3
 - extra security 11-3
 - on downloaded products 3-7
 - on files created in distrib database 20-7
 - unwound tar files 11-2
- pointers in database files 11-6
- pre-built binary products 16-5
 - inserting into template_product 18-4
 - pre-build checklist 19-1
- PROD_DIR keyword 22-48, 27-9
 - as set by -r option 24-4
 - description 27-6, 28-2
- PROD_DIR_PREFIX keyword 3-4, 22-48, 27-9, 30-2
 - description 27-6
- prodDir function
 - description 34-9
- product announcement checklist 19-3

- product categories in KITS 17-3
 - default 21-3
 - FermiTools 21-3
 - FNAL only 21-3
 - proprietary 21-3
 - registration for special categories 17-3
 - U.S. only 21-3
- product dependencies (See dependencies)
- product dependency matching 26-2
- product development 16-7
 - announcement policies for new products 17-10
 - checklist for building product 19-2
 - checklist for distributing to KITS 19-3
 - checklist for pre-build 19-1
 - checklist for product announcements 19-3
 - checklist for testing 19-2
 - code management system 16-6
 - compile script 37-1
 - configure script 36-1
 - configure third-party product 16-6
 - current script 36-3
 - declaring product during development 16-2
 - distributing the product 17-1
 - documentation location 15-7
 - example procedure for simple product 16-1
 - man page creation 16-3
 - overlaid products 16-7
 - pre-build checklist with template_product 19-1
 - pre-built binaries 16-5
 - prep for rebuilding 16-6
 - read-only variables 34-18
 - recommendations
 - fully-specified flavor 15-1
 - location determination 15-2
 - nonuse of /usr/local/bin 15-2
 - nonuse of /usr/local/products 15-3
 - reproducible build procedure 15-3
 - self-containment 15-2
 - shell-independence 15-1
 - system-independence 15-3
 - sample directory hierarchy 16-2
 - selection of build node 16-7
 - simple build procedure 16-1
 - start script 36-3
 - stop script 36-3
 - table files 35-1
 - sample 16-2
 - tailor script 36-3
 - testing product 16-4, 18-5
 - third-party products 15-3
 - uncurrent script 36-3
 - unflavored scripts 16-4
 - using template_product 18-1
 - vendor-supplied products, rebuilding 16-6
- product development tools
 - buildmanager 15-5
 - CVS 15-5
 - template_product 15-6
- product distribution
 - announcement policies for new products 17-10
 - overview 17-1
 - using template_product 18-1, 18-6
 - via CD-ROM 20-14
- product distribution node (See distribution node)
- product documentation 15-7
- product files
 - configure UPD to set location 11-4, 11-5
 - location 11-4, 11-5
 - ownership 11-3
 - permissions 11-3
- product flavor 1-3
- product installation (See installing a product)
- product instance (see instance)
- product instance matching (See instance matching)
- PRODUCT keyword 22-48
 - description 27-6, 28-2, 29-2
- product registration for KITS 21-3
- product removal (See remove a product)
- product root directory 15-6
 - definition 1-3
 - locate using ups list -K 22-52
 - simple example of structure 16-2
- product use statistics 27-9
- product version 1-3
- products account 11-1
- products area 3-4
 - adding a new one 11-9
 - as set in UPD config 3-4
 - choosing location 11-4
 - defining during UPS bootstrap 13-2
 - for development/testing 11-9
 - for KITS 21-1
 - PROD_DIR keyword 27-6, 28-2
 - PROD_DIR_PREFIX keyword 27-6
 - structure of product root directory 15-6
 - unwind product tar files into 7-3
- products for use only at FNAL 21-3
- products for use only in U.S. 21-3
- products requiring build 16-6
 - build script recommendations 15-3
 - inserting into template_product 18-4
 - pre-build checklist 19-1
- proprietary products 21-3
 - on fnkits 4-8

Q

- qualifiers
 - choosing whether to declare them 3-8
 - description 24-8
 - mixing required and optional 24-9
 - optional 24-9
 - overview 1-4
 - required 24-8
 - use in instance matching 26-4
- QUALIFIERS keyword 22-48
 - description 27-6, 28-3, 29-2

R

- reader comment solicitation INT-5
- README file 7-1, 15-6, 19-1
 - sample 16-8
- read-only variables 34-18
 - PRODUCTS 34-19
 - to be added in future 34-21
 - UPS_COMPILE 34-19

- UPS_EXTENDED 34-19
- UPS_OPTIONS 34-19
- UPS_ORIGIN 34-19
- UPS_OS_FLAVOR 34-19
- UPS_PROD_DIR 34-19
- UPS_PROD_FLAVOR 34-19
- UPS_PROD_NAME 34-19
- UPS_PROD_QUALIFIERS 34-19
- UPS_PROD_VERSION 34-19
- UPS_THIS_DB 34-19
- UPS_UPS_DIR 34-20
- UPS_VERBOSE 34-20
- rebuilding product 16-7
- registering products for KITS 21-3
- RELEASE_NOTES file 19-1
 - sample 16-9
- remove a product 10-7, 22-79
 - unconfiguring 10-9
 - using UPP 10-8, 10-10
 - using ups undeclare command 10-8, 22-77
- remove a product component
 - from distrib node 17-8
- remove access to product 2-10, 22-11
- remove product from distrib node 17-8
 - using template_product 18-8
- retrieve file or dir from distribution node 10-15
- retrieve product from distribution node 5-1
- reversible functions 33-2
 - definition 34-1

S

- searchlog.cgi script 20-13
- selecting database for dependency install using UPD 5-3
- selecting database for product install using UPD 5-2
- setup command 1-1, 2-8, 22-5
 - associated environment variables 22-5
 - for chained instance 2-9
 - for current instance 2-9
 - for unchained instance 2-9
 - reference 22-3
 - special options 2-9
 - test if setup would succeed 10-16, 22-33
 - use in troubleshooting problem installations 9-1, 10-17
 - v option for use in troubleshooting 9-1, 10-17
- setupEnv function
 - description 34-10
- setupOptional function
 - description 34-10
 - use to define dependencies 35-4
- setupRequired function
 - description 34-10
 - use to define dependencies 35-4
- setups.[c]sh files 1-9
 - courtesy links to 12-3
 - determine directory for 11-6
 - pointers to 11-6
- SETUPS_DIR keyword 22-48, 30-2
 - description 27-6
- sh
 - parse ups list output in a script 22-53

- shell script products
 - inserting into template_product 18-4
 - pre-build checklist 19-1
- simulate command 9-1, 10-17
- source code
 - revision tracking 17-9
 - storage in CVS 17-9, 18-8
- sourceCompileOpt function
 - description 34-11
- sourceCompileReq function
 - description 34-11
- sourceOptCheck function
 - description 34-12
- sourceOptional function
 - description 34-13
- sourceReqCheck function
 - description 34-13
- sourceRequired function
 - description 34-14
- src directory 15-7
- stage1.sh file 13-1, 13-5
- stage2.sh file 13-5
- stanzas
 - table file 35-1
 - UPD config file 31-1
 - UPP subscription file 6-1, 32-2
- START action 36-3
- start script 14-3, 36-3
- statistics
 - how to gather 11-10, 27-9
 - output 27-10
- STATISTICS keyword 22-48, 30-2
 - as set by -L option 24-3
 - description 27-6, 28-3
 - detailed description of use 27-9
 - output from 27-10
- STOP action 36-3
- stop script 14-3, 36-3
- subscription file for UPP
 - creating 6-1
 - reference 32-1
 - sample for product installation 6-3
- SUFFIX keyword 20-9, 20-10
- syntax of UPS/UPD commands 1-8, 25-1

T

- table files 1-6
 - compile script used with 37-1
 - detailed description 35-1
 - examples
 - action present for some instances only 35-8
 - execute one action or another 35-8
 - grouping 35-6
 - use of FLAVOR=ANY 35-6
 - with user-defined keywords 35-7
 - grouping information in 35-3
 - information storage format 27-2
 - instance matching within 26-3
 - keywords 27-2
 - locate using ups list -K 22-52
 - location specification 28-5
 - naming 35-1

- ordering elements in 35-3
- overwrite 10-14
- read-only variables available for use in 34-18
- recommendations to developers 35-2
- sample for simple product 16-2, 16-4
- stanzas 35-1
- structure and contents 35-2
- test if needs update 10-13
- undoing reversible functions 33-2
- V option for debugging 24-9
- TABLE_DIR keyword 22-48
 - description 27-6, 28-3
- TABLE_FILE keyword 22-48, 27-10
 - description 27-6, 28-3
- tailor a product instance 3-9, 22-69
- TAILOR action 3-9, 22-67, 22-69, 36-3
- tailor script 36-3
- tar file creation
 - by upd addproduct 17-1, 23-7
 - using template_product 18-5
- template_product 15-6, 17-2
 - adding build instructions 18-4
 - to top-level Makefile 18-4
 - checklist for building product 19-2
 - checklist for distributing to KITS 19-3
 - checklist for pre-build 19-1
 - cloning 18-2
 - customizing product tar file 18-5
 - downloading 18-2
 - editing top-level Makefile 18-3
 - inserting pre-built binaries 18-4
 - inserting product requiring build 18-4
 - inserting shell scripts 18-4
 - inserting your product 18-4
 - Makefile (top-level) 18-3
 - overview 18-1
 - removing product from distrib node 18-8
 - running a build procedure 18-4
- temporary script
 - prevent deletion 24-9
- test chain 1-4
- test directory 15-7
- testing products 18-5
 - checklist 19-2
- third-party products 15-3
- toInfo directory 15-6
- toman directory 15-6

U

- umask 3-7
- unAlias function
 - description 34-14
- unchain
 - as action in table files 33-3
 - replace chain on distrib node using upd modproduct 17-7
 - use ups undeclare to remove chain 10-6, 22-77
- UNCONFIGURE action 10-9, 22-75, 36-1
- unconfigure script 36-1
- UNCURRENT action 36-3
- uncurrent script 36-3
- undeclare a chain 10-6, 22-77
 - undeclare a product instance 10-7, 22-79
 - using UPP 10-8
 - using ups undeclare command 10-8
- undoing chains in table files 33-3
- unflavored scripts 16-4
- UNIX Product Distribution
 - overview 1-1
- UNIX Product Poll 32-1
 - overview 1-1
- UNIX Product Support
 - overview 1-1
- unknown command handler
 - description 33-3
- unProdDir function
 - description 34-14
- unsetup command 2-10, 22-11
 - \$SETUP_UPS variable 1-10
 - behavior with dependencies 22-11
 - reference 22-9
 - use of \$SETUP_<PRODUCT> variable 22-6, 22-11
- unsetupEnv function
 - description 34-15
- unsetupOptional function
 - description 34-15
- unsetupRequired function
 - description 34-16
- UNWIND_ARCHIVE_FILE keyword 20-9, 20-10
 - description 27-6
 - use in updconfig 31-4
- UNWIND_PROD_DIR keyword 3-4
 - description 27-7
 - use in updconfig 31-3
- UNWIND_TABLE_DIR keyword
 - description 27-7
 - use in updconfig 31-4
- UNWIND_UPS_DIR keyword
 - description 27-7
 - use in updconfig 31-3
- UPD
 - command syntax 1-8
 - configuration file
 - info for installers 3-3
 - overriding default 3-4
 - reference 31-1
 - overview 1-1
 - procedural checklist for installation 5-3
- upd addproduct command
 - adding table file product 17-5
 - adding typical product 17-3
 - chains 17-3, 23-7
 - detailed functions 20-2
 - internal processes 23-8
 - reference 23-3
 - response of distrib node 20-2
 - tar file creation 17-1, 23-7
- upd cloneproduct command
 - reference 23-11
- UPD commands
 - defaults 1-8
 - dependency matching 26-2
 - instance matching 26-1
 - interaction with distrib node 20-1
 - option flag grouping 25-2
 - option usage 25-4
 - order of command line elements 25-1

- specifying version/chain 25-1
- specifying multiple products 25-3
- UPD configuration file 27-3
 - AFS issues 8-2
 - distrib node 20-9
 - KITS database pre and postdeclare actions 21-4
 - pre and postdeclare actions 20-10
 - examples 31-7
 - AFS 31-10
 - distrib node config 31-10
 - distribution from fnkits 31-8
 - multiple dbs and distrib nodes 31-9
 - for KITS database 21-2
 - info for installers 3-3
 - organization 31-1
 - overriding default 3-4, 31-1
 - overview 27-1
 - pre and postdeclare actions 31-5
 - product matching 31-2
 - reference 31-1
 - required location definitions 31-3
 - sample location definitions 31-5
 - setting file permissions 11-3
 - stanzas 31-1
- upd delproduct command 17-8
 - reference 23-13
- upd depend command 4-5
 - reference 23-15
- upd exist command 10-16
 - reference 23-17
- upd fetch command 10-15
 - reference 23-19
- upd get command
 - reference 23-23
- upd install command 5-1
 - database selection 5-2
 - database selection for dependencies 5-3
 - detailed functions 20-2
 - G (pass options to local declare) 5-2, 23-28
 - internal processes 23-29
 - procedural checklist for installation 5-3
 - reference 23-25
 - response of distrib node 20-2
 - summary of functions it performs 3-1
 - syntax and commonly used options 5-1, 23-25
 - use to determine if product update needed 10-13
- upd list command 4-1
 - reference 23-31
- upd modproduct command 17-6, 17-7
 - reference 23-33
 - response of distrib node 20-2
- upd move_archive_file script 20-2
- upd moved_ups_dir script 20-2
- _UPD_OVERLAY keyword 16-7, 27-11
 - description 27-8
- upd reproduct command
 - reference 23-39
- upd update command 10-13, 10-14
 - reference 23-41
- upd verify command
 - reference 23-45
- upd.cgi script 20-2, 20-11
 - access restrictions 20-6
 - description 20-5
- UPD_USERCODE_DB keyword 22-48
 - description 27-7
- UPD_USERCODE_DIR keyword 3-4, 22-48, 30-2
 - description 27-7
 - on fnkits 21-2
- update product
 - determine if update needed 10-13
 - using UPD 10-13
 - using UPP 10-13
- updconfig file (see UPD configuration file)
- updconfig.template file 31-1, 31-7
- updusr.pm file 31-1
- upgrading UPS installation 11-8
- UPP
 - automate upp command via cron 6-4
 - command syntax 6-4
 - monitor products on distribution node 4-3
 - notification of update needed 4-3, 10-13
 - overview 1-1
 - remove a product 10-8, 10-10
 - subscription file
 - creating 6-1
 - definition 4-3
 - sample for product installation 6-3
 - uses 32-1
- upp command 4-3, 6-1
 - automation via cron 6-4
 - reference 23-47
 - syntax 4-4, 6-4
- UPP subscription file
 - adding instructions 32-2
 - available functions 32-3
 - creating 6-1
 - definition 4-3
 - header description 32-1
 - instance matching 32-2
 - reference 32-1
 - sample 32-3
 - sample for product installation 6-3
 - stanza description 32-2
- UPS
 - aliases defined 1-10
 - benefits of methodology 1-2
 - chains 1-4
 - command syntax and defaults 1-8
 - database 1-1
 - database directory specification 1-10
 - motivation for methodology 1-2
 - multiple database support 1-1
 - multiple product flavor support 1-3
 - multiple product version support 1-2, 1-3
 - overview 1-1
 - pointer to product root directory (\$UPS_DIR) 1-10
 - product instance 1-4
 - product version 1-3
 - products distributed and managed by 1-3
 - upgrading your UPS installation 11-8
 - use without a database 1-7, 11-7
- UPS commands
 - "-?" for usage information 2-1
 - "uncommands" as action keyword values 34-1
 - as ACTION keyword 33-1
 - database selection 26-1
 - defaults 1-8
 - dependency matching 26-2

- instance matching 26-1
- keeping statistics on 11-10, 27-9
- option flag grouping 25-2
- option usage 25-4
- order of command line elements 25-1
- specifying multiple products 25-3
- specifying version/chain 25-1
- UPS configuration file 27-3
 - defining directory locations in 11-6
 - for KITS database 21-1
 - for local database on fnkits 21-1
 - keywords used in 30-1
 - overview 27-1
 - reference 30-1
 - sample 30-2
- ups configure command 3-9
 - reference 22-13
- ups copy command 22-19
 - reference 22-17
- UPS database
 - \$PRODUCTS variable 1-10
 - \$UPS_EXTRA_DIR variable for AFS 12-5
 - .upfiles subdirectory 1-6
 - .upsfiles subdirectory 1-6
 - checklist for creating a database 11-9
 - choosing single or multiple 11-6
 - configuring local to work with AFS 12-2
 - create a private database 11-9
 - create local database to work with AFS 12-2
 - declare a product instance to 3-5, 10-1
 - declaring products into local (not AFS) 12-4
 - definition 1-6
 - for development/testing 11-9
 - installing products into local (not AFS) 12-5
 - list all current products in 2-4
 - list product information 2-2
 - listed in upsdb_list file 12-2
 - multiple (See multiple databases)
 - NFS mounted 12-1
 - permissions for files (distrib node) 20-7
 - providing access to multiple databases 12-2
 - setting up your own 5-3
 - with AFS 12-2
 - standard naming conventions for use with AFS 12-2
 - structure and contents 1-6
 - using AFS and local 5-3
 - using UPS without a database 1-7, 11-7
- UPS database files 1-6
 - chain files 1-6, 29-1
 - check for inconsistencies 10-10
 - editing 10-11
 - keywords 27-1
 - overview 27-1
 - UPD configuration file 31-1
 - UPS configuration file 30-1
 - version files 1-6, 28-1
- ups declare command 3-6, 10-3
 - as used internally by upd install 5-2
 - reference 22-21
 - specifying database 3-5, 10-2
 - specifying table file path 3-5, 10-2
 - specifying ups directory 3-5, 10-2
 - syntax and common options
 - for declaring chain 10-4
 - for declaring instance 3-5, 7-4, 10-2
 - use during development 16-2
 - use to declare chain 10-4
 - use to declare instance 3-5, 10-1
- ups depend command 2-7, 10-8, 22-79
 - reference 22-27
- ups directory 3-5, 7-1, 10-2, 15-6, 27-11
 - description 15-6
 - locate using ups list -K 22-52
 - overwrite 10-14
 - test if needs update 10-13
- UPS environment (See environment)
- ups exist command 10-16, 22-33
 - reference 22-31
- ups flavor command 2-1
 - H option (specifies other flavor) 22-36
 - l option (returns flavor table) 22-36
 - number options (specify OS level) 2-2
 - obtain flavor levels 2-2
 - obtain flavor table 2-2
 - reference 22-35
- ups get command
 - reference 22-39
- ups help command
 - reference 22-41
- UPS initialization file 11-6
- ups installasroot command 12-6
- ups list command 2-2, 3-6, 10-3, 10-5
 - condensed output 2-3, 22-46
 - default output fields 22-45
 - for db managers and product installers 27-1
 - formatted output 2-3, 22-45
 - K option
 - for script-readable format 2-3, 22-46
 - keyword arguments 22-46
 - use to locate product files 22-52
 - keywords for -K option 2-3
 - list all current products 2-4
 - list all output fields 2-6
 - long listing 22-51
 - parse output
 - in perl 22-52
 - in sh script 22-53
 - reference 22-43
- ups modify command
 - editing database files 10-11
 - reference 22-55
- UPS product overlay (See overlays)
- UPS product requirements (See dependencies)
- UPS products
 - accessibility 10-16
 - announcement policies 17-10
 - bin directory 15-6
 - build and distribute using template_product 18-1
 - catman directory 15-7
 - compilation options 1-4
 - definition 1-3
 - directory structure 15-6
 - distribution restrictions 20-11
 - distribution via CD-ROM 20-14
 - doc directory 15-7
 - documentation storage 15-7
 - examples directory 15-7
 - files and directories to include 19-1
 - hardcoded locations 15-3
 - html directory 15-7

- include directory 15-7
- Info directory 15-7
- INSTALL_NOTE file 15-6
- installation methods, summary 3-1
- installed with different name than on server 3-8
- interruption during installation 3-8
- lib directory 15-7
- list on distribution node 4-1
- man directory 15-7
- news directory 15-7
- overlays 16-7
- permissions set at installation 3-7
- proprietary products
 - on fnkits 4-8
- qualifiers 1-4
- README file 15-6
- special categories, flagging 20-12
- src directory 15-7
- support levels 17-10
- test directory 15-7
- third-party 15-3
- toInfo directory 15-6
- toman directory 15-6
- ups directory 7-1, 15-6, 27-11
- ups script 14-1
- ups setup command (for troubleshooting) 9-1, 10-17
- ups start command 14-2, 14-5
 - reference 22-59
 - usage in autostart 14-3
- ups stop command 14-2
 - reference 22-63
 - usage in autostart 14-4
- ups tailor command 3-9, 22-69
 - reference 22-67
- ups touch command
 - reference 22-71
- ups unconfigure command 10-7, 10-9, 22-79
 - reference 22-73
- ups undeclare command
 - reference 22-77
 - remove chain 10-6, 22-77
 - remove product instance 10-7, 10-8, 22-79
 - syntax and common options
 - for chain removal 10-6
 - for product removal 10-8
 - y and -Y options to remove root directory 10-8
- ups verify command 10-10
 - reference 22-81
 - run by ups modify 10-11
 - use in troubleshooting problem installations 9-1, 10-17
- ups.cgi script 20-2
 - description 20-5
- UPS/UPD/UPP installation components 1-1
- UPS_ARCHIVE_FILE keyword 20-9, 20-10
 - description 27-7
- UPS_ARCHIVE_FILES keyword
 - use in updconfig 31-4
- UPS_DB_VERSION keyword 30-2
 - description 27-7, 28-3, 29-2
- UPS_DIR keyword 22-48, 27-11
 - as set by -U option 24-5
 - description 27-7, 28-3
- UPS_EXTENDED variable 24-7

- UPS_PROD_DIR keyword 3-4
 - description 27-7
 - use in updconfig 31-3
- ups_shutdown script 14-1, 14-2, 14-5
- ups_startup script 14-1, 14-2, 14-5
- UPS_TABLE_DIR keyword
 - description 27-7
 - use in updconfig 31-3
- UPS_TABLE_FILE keyword
 - description 27-8
 - use in updconfig 31-4
- UPS_THIS_DB keyword
 - description 27-7
 - use in updconfig 31-3
- UPS_UPS_DIR keyword
 - description 27-8
 - use in updconfig 31-3
- upsdb_list file 12-2
 - for AFS 12-5
- upsdb_list variable 13-3
- ups-decl.cgi script 20-2, 20-11
 - access restrictions 20-6
 - description 20-5
- user comment solicitation INT-5
- USER keyword
 - description 27-8
- user-defined chains 1-4
- user-defined commands 33-3
- user-defined keywords 27-2
- US-only products 21-3

V

- variables (read-only) defined within UPS 34-18
- vendor-supplied products
 - rebuilding 16-6
- verbose command output (-v) 9-1, 10-17
- version files 1-6, 22-79
 - and product removal 10-7
 - creating 28-1
 - description 28-1
 - examples 28-3
 - information included in 28-1
 - information storage format 28-1
 - instance matching within 26-3
 - location 28-1
 - overview 27-1
 - table location specification in 28-5
- VERSION keyword 22-48
 - description 27-8, 28-3, 29-2
- version of product 1-3
- version specification in commands 25-1

W

- web server
 - access file 20-11
 - log file on fnkits 21-7
 - on distrib node 20-1
 - prerequisites for cgi scripts 20-7

writeCompileScript function
description 34-16

www
download products from 16-5