

**Bellcore**

© Bell Communications Research

---

BELLCORE PRACTICE  
BR 007-252-007  
ISSUE 1, OCTOBER 1997  
RELEASE 1.0

# Automatic Efficient Testcase Generator (AETG™) System Users Guide

*Bellcore*



**Bellcore**

© Bell Communications Research

---

BELLCORE PRACTICE  
BR 007-252-007  
ISSUE 1, OCTOBER 1997  
RELEASE 1.0

# Automatic Efficient Testcase Generator (AETG™) System Users Guide

For further information, please contact:

MYNAH Customer Service Center

1-(732) 699-2668, option 3

To obtain copies of this document, call (732) 699-5802.

Copyright © 1997 Bellcore.

All rights reserved.

We at Bellcore are constantly striving to meet your need for information. Once you've had a chance to use this document that we've written for you, please let us know if it met your needs. Please complete this form and either FAX it to us at (732) 336-2995 or return it to us at the address below.

Document No. BR 007-252-007	Issue No. Issue 1	Publication Date October 1997	Revision No.	Supplement No.
--------------------------------	----------------------	----------------------------------	--------------	----------------

1. In each of the following areas, how well did this document meet your need for information?

	Nearly				Not Applicable
	Missed	Met	Met	Exceeded	
a. Relevance of the information to your work .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
b. Ease of finding the information that you need .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
c. Clarity of the information.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
d. Accuracy of the information .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
e. Usefulness of the information .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
f. Thoroughness of the information.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
g. Level of detail of the information.....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
h. Availability of this document when you needed it .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
i. Overall quality of this document .....	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>

2. Please comment on any of the areas where this document *did not* meet or exceed your need for information.

\_\_\_\_\_  
\_\_\_\_\_

3. Are there features of this document that you found particularly useful or informative? Please explain.

\_\_\_\_\_  
\_\_\_\_\_

4. Are there other ways that we can improve this document? Please feel free to comment on any aspect of it.

\_\_\_\_\_  
\_\_\_\_\_

5. For what purpose did you use this document?

- As a technical reference                       To use a system                       To learn methods/procedures
- As an administrative reference               To install/administer a system       To be better informed
- Other (please specify) \_\_\_\_\_

6. Please tell us something about yourself.

Your company/employer \_\_\_\_\_ Your title \_\_\_\_\_

Your job responsibilities \_\_\_\_\_

If you would like us to let you know what we're doing in response to your feedback, please write your name and address (or telephone number) below.

Name \_\_\_\_\_ Telephone Number \_\_\_\_\_

Address \_\_\_\_\_

*Thank you for your time and cooperation!*

*To return this form, please FAX it to (732) 336-2995, or mail it to Ken Berczik, Bellcore Learning Support, 444 Hoes Lane, Room RRC 4F-808, Piscataway, NJ 08854.*



## NOTICE OF DISCLAIMER AND LIMITATION OF LIABILITY

This document is intended for use solely by Bellcore customers who have licensed the Bellcore software described herein. The software, this document, and the information contained within this document may be used, copied or communicated only in accordance with the terms of a written license agreement with Bellcore. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording without the prior written permission of Bellcore. While the information contained herein has been prepared from sources deemed reliable, Bellcore reserves the right to revise the information without notice, but has no obligation to do so. Unless the recipient has been expressly granted a license by Bellcore under a separate applicable written agreement with Bellcore, no license, express or implied, is granted under any patents, copyrights or other intellectual property rights. Use of the information contained herein is in your sole determination and shall not be deemed an inducement by Bellcore to infringe any existing or later-issued patent, copyright or other intellectual property rights.

BELLCORE PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS. FURTHER, BELLCORE MAKES NO REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED WITH RESPECT TO THE SUFFICIENCY, ACCURACY, OR UTILITY OF ANY INFORMATION OR OPINION CONTAINED HEREIN. BELLCORE EXPRESSLY ADVISES THE USER THAT ANY USE OF OR RELIANCE UPON SAID INFORMATION OR OPINION IS AT THE SOLE RISK AND LIABILITY, IF ANY, OF THE USER AND THAT BELLCORE SHALL NOT BE LIABLE FOR ANY DAMAGE OR INJURY INCURRED BY ANY PERSON ARISING OUT OF THE SUFFICIENCY, ACCURACY, OR UTILITY OF ANY INFORMATION OR OPINION CONTAINED HEREIN. BELLCORE, ITS OWNERS AND AFFILIATES SHALL NOT BE LIABLE WITH RESPECT TO ANY CLAIM BEYOND THE AMOUNT OF ANY SUM ACTUALLY RECEIVED IN PAYMENT BY BELLCORE FOR THE DOCUMENTATION, AND IN NO EVENT SHALL BELLCORE, ITS OWNERS OR AFFILIATES BE LIABLE FOR LOST PROFITS OR OTHER INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

Bellcore does not recommend or endorse products and nothing contained herein is intended as a recommendation or endorsement of any product.

For further information, please contact:

The MYNAH Customer Service Center 8:00 AM and 7:00 PM ET Monday through Friday,  
(732) 699-2668, Option 3. If outside the 732 area, call (800) 795-3119, Option 3.

You can also contact support (for non critical problems) via e-mail at  
[mynah-support@cc.bellcore.com](mailto:mynah-support@cc.bellcore.com).

Copyright © 1997 Bellcore.  
All Rights Reserved.



## Trademark Acknowledgments

Adobe, Acorbat, and Acrobat Reader are registered trademarks of Adobe Systems Incorporated.

HP and HP-UX are trademarks of the Hewlett-Packard Company.

IBM is a trademark of International Business Machines.

Macintosh is a registered trademark of Apple Computer, Inc.

Microsoft and NT are registered trademarks of Microsoft Corporation.

Windows is a trademark of Microsoft Corporation.

Motif is a trademark of Open Software Foundation, Inc.

MYNAH and AETG are trademarks of Bellcore.

Oracle is a registered trademark of Oracle Corporation.

SunOS, Solaris, SPARC, SPARCstation, and NFS are trademarks of Sun Microsystems, Inc.

Telexel is a trademark of Bellcore.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X Window System is a trademark of the Massachusetts Institute of Technology.



---

# AETG System Users Guide

## Contents

Preface .....Preface-1

### Part A: General Information

---

1. Introduction..... 1-1

    1.1 AETG System Overview..... 1-1

        1.1.1 Starting the AETG System..... 1-1

        1.1.2 Defining the Interface ..... 1-2

        1.1.3 Defining and Associating Tests ..... 1-3

        1.1.4 Defining the Rules of the Interface ..... 1-3

        1.1.5 Generating the Efficient Set of Test Cases ..... 1-4

        1.1.6 Automating the Test Cases ..... 1-5

        1.1.7 Summary ..... 1-5

    1.2 Command Line User Interface (CLUI)..... 1-7

    1.3 Some Basic Terms we Use..... 1-7

        1.3.1 Mouse Terms..... 1-7

        1.3.2 Cascading Menus ..... 1-7

    1.4 On-line Documentation..... 1-8

    1.5 Customer Support..... 1-8

### Part B: User Tasks

---

2. A Quick Tour of the AETG System..... 2-1

    2.1 Define what you want to Test ..... 2-2

        2.1.1 Define the Interface..... 2-2

        2.1.2 Define the Rules of the Test..... 2-2

    2.2 Start the AETG System..... 2-3

    2.3 Registering as an AETG User ..... 2-3

    2.4 Create a Folder ..... 2-5

    2.5 Create a Format Object ..... 2-6

    2.6 Create Field Objects and Values ..... 2-9

    2.7 Create a Test Hierarchy and Test Object ..... 2-16

    2.8 Create an Association..... 2-20

    2.9 Create a Relation Object ..... 2-24

    2.10 Generate a Test Matrix..... 2-26

    2.11 Change the Interaction Degree..... 2-30

    2.12 Create an Excluded Test Case..... 2-33

    2.13 Exiting From the AETG System ..... 2-40

---

2.14	What's Next .....	2-41
3.	AETG Basics.....	3-1
3.1	AETG Folders .....	3-1
3.1.1	What is the AETG Desktop?.....	3-2
3.1.2	Title Bar .....	3-3
3.1.3	Menu Bar.....	3-4
3.1.3.1	AETG Icon.....	3-4
3.1.3.2	Aetg Menu .....	3-5
3.1.3.3	Selected Menu .....	3-6
3.1.3.4	Edit Menu .....	3-6
3.1.3.5	View Menu .....	3-6
3.1.3.6	Tools Menu.....	3-7
3.1.3.7	Windows Menu.....	3-7
3.1.3.8	Help Menu .....	3-7
3.1.4	Tool Bar .....	3-7
3.1.5	Client Area .....	3-9
3.1.6	Status Bar .....	3-9
3.1.7	Scroll Bars.....	3-9
3.2	AETG Dialogs.....	3-11
3.3	Navigating Around the AETG GUI .....	3-11
3.3.1	Using the Mouse .....	3-11
3.3.2	Using Icons .....	3-12
3.3.3	Making Menu Selections .....	3-12
3.3.4	Using Mnemonic Keys.....	3-12
3.3.5	Using Accelerator Keys .....	3-13
3.4	AETG GUI Controls .....	3-14
3.4.1	Using Spin Buttons .....	3-14
3.4.2	Using Toggle Buttons .....	3-14
3.4.3	Using Radio Buttons .....	3-15
3.4.4	Using Pushbuttons.....	3-15
3.4.5	Using Ruler Column Headings .....	3-16
3.5	What Are Views? .....	3-18
3.6	List View .....	3-19
3.6.1	Expanding Items in a List View.....	3-19
3.7	Using the Preferences View .....	3-22
3.7.1	Saving Default Settings.....	3-23
3.7.2	Selecting The Default Fonts.....	3-25
3.8	Changing an AETG View .....	3-26
3.9	Creating Folders.....	3-27
3.9.1	Saving Folders.....	3-28
3.9.2	Deleting Folders.....	3-28
3.9.3	Changing a Folder's Name .....	3-28
3.10	Printing.....	3-30
3.11	Exiting From the AETG System .....	3-34

---

---

4.	Using AETG Objects .....	4-1
4.1	What Are Objects? .....	4-1
4.1.1	How Views and Objects Work Together. ....	4-2
4.2	Test Hierarchies .....	4-2
4.3	Creating Objects .....	4-3
4.3.1	Creating Objects on the AETG Desktop.....	4-3
4.3.2	Creating New Objects in Folders .....	4-4
4.3.2.1	Creating New Objects In Open Folders.....	4-4
4.3.2.2	Creating New Objects In Closed Folders .....	4-5
4.4	Opening Objects for Editing .....	4-6
4.5	Copying Objects.....	4-6
4.6	Duplicating Objects.....	4-7
4.6.1	The Differences Between Copying and Duplicating Objects .....	4-7
4.7	Deleting Objects.....	4-8
4.8	Removing Objects from the AETG GUI Display .....	4-8
4.9	Objects' Status and Default View .....	4-9
4.10	Modifying Another Person's Objects.....	4-9
4.11	Multiselecting Objects .....	4-10
5.	Using the Database Browser .....	5-1
5.1	How the Database Browser Helps You.....	5-1
5.2	Accessing the Database Browser .....	5-2
5.2.1	Displaying Object Types with the Database Browser .....	5-3
5.2.2	Using the Include Dialog .....	5-5
5.2.2.1	Object Attributes and Default Queries .....	5-6
5.2.2.2	Conditions Between Attributes and Values.....	5-7
5.2.2.3	Relations between Rows.....	5-7
5.2.2.4	Using the Include Dialog Example.....	5-7
5.2.2.5	Deleting Rows From the Include Dialog .....	5-11
5.2.3	Opening Objects With the Database Browser.....	5-11
5.3	Using Person Objects .....	5-12
5.3.1	Viewing Person Objects.....	5-12
5.3.2	Changing Data in Your Person Object.....	5-14
5.3.3	Using the Person Object Information View .....	5-15
6.	Format Object .....	6-1
6.1	Why use a Format Object.....	6-1
6.2	Working a Format Object.....	6-1
6.2.1	Properties View .....	6-2
6.2.2	Fields View .....	6-4
6.2.2.1	Working with Fields .....	6-5
6.2.2.2	Working with Compound Objects .....	6-9
6.2.2.3	Field Sequence Numbers and Rearranging Field Objects .....	6-12
6.2.3	Associations Views .....	6-14
6.2.4	Duplicating a Format Object.....	6-15

---

---

6.2.4.1	Creating a New Version of Existing a Format Object.....	6-16
6.2.4.2	Creating a New Format Object.....	6-16
7.	Field Object.....	7-1
7.1	Why use a Field Object.....	7-1
7.2	Working with Field Objects.....	7-1
7.2.1	Values View.....	7-2
7.2.1.1	Specifying Valid Values.....	7-3
7.2.1.2	Specifying Parameters.....	7-9
7.2.2	Properties View.....	7-10
7.2.3	Printing a Field Object.....	7-10
8.	Compound Object.....	8-1
8.1	Why use a Compound Object.....	8-1
8.2	Working with Compound Objects.....	8-1
8.2.1	Properties View.....	8-2
8.2.2	Fields View.....	8-4
8.2.2.1	Selecting Values to Participate in a Compound.....	8-5
8.2.2.2	Selecting an Interaction Degree.....	8-6
8.2.3	Tuples View.....	8-7
8.2.3.1	Generating Tuples.....	8-8
8.2.3.2	Changing a Tuple's Name.....	8-12
8.2.3.3	Regenerating System Generated Tuple Names.....	8-13
8.2.3.4	Deleting Tuples.....	8-14
8.3	Saving a Compound.....	8-14
9.	Test Object.....	9-1
9.1	Why use a Test Object.....	9-1
9.2	Using Test Hierarchies and Creating Test Objects.....	9-1
9.2.1	Creating and Naming Test Hierarchies.....	9-2
9.2.2	Deleting Hierarchies.....	9-4
9.2.3	Creating Test Objects in Hierarchies.....	9-4
9.2.4	Duplicating a Test Object.....	9-6
9.3	Working with Test Objects.....	9-8
9.3.1	Properties View.....	9-9
9.3.2	Relations View.....	9-11
9.3.2.1	Associating a Format Object.....	9-12
9.3.2.2	Creating a New Relation.....	9-13
9.3.2.3	Duplicating a Relation.....	9-14
9.3.2.4	Deleting a Relation.....	9-16
9.3.3	Matrix View.....	9-17
9.3.3.1	Generating a Matrix.....	9-19
9.3.3.2	Deleting Test Cases.....	9-20
9.3.3.3	Saving a Matrix.....	9-22
9.3.3.4	Loading an Existing Matrix.....	9-24
9.3.3.5	Deleting a Matrix.....	9-25

---

---

9.3.3.6	Specifying the Maximum Number of Test Cases.....	9-25
9.3.4	Test Save Implications.....	9-26
9.4	Obtaining Test Case Matrix Output.....	9-27
9.4.1	Printing a Matrix.....	9-27
9.4.2	Matrix File Format.....	9-28
9.4.3	Converting a Test Case Matrix File's Format.....	9-29
10.	Relation Object.....	10-1
10.1	Why use a Relation Object.....	10-1
10.2	Working with Relation Objects.....	10-1
10.2.1	Properties View.....	10-3
10.2.2	Fields View.....	10-5
10.2.2.1	Changing Participating Fields.....	10-7
10.2.2.2	Changing Participating Values.....	10-9
10.2.2.3	Rearranging Field Columns.....	10-10
10.2.3	Includes View.....	10-11
10.2.3.1	Adding New Includes.....	10-13
10.2.3.2	Viewing/Modifying Existing Includes.....	10-16
10.2.3.3	Duplicating Include Test Cases.....	10-17
10.2.3.4	Deleting Existing Includes.....	10-18
10.2.4	Excludes View.....	10-19
10.2.4.1	Examples of Field Constraints.....	10-21
10.2.4.2	Adding New Excludes.....	10-22
10.2.4.3	Viewing/Modifying Existing Excludes.....	10-24
10.2.4.4	Duplicating Excluded Test Cases.....	10-25
10.2.4.5	Deleting Existing Excludes.....	10-26
10.2.5	Invalids View.....	10-27
10.2.5.1	Adding New Invalids.....	10-28
10.2.5.2	Viewing/Modifying Existing Invalids.....	10-31
10.2.5.3	Deleting Existing Invalids.....	10-31
11.	Command Line Tools.....	11-1
11.1	xmyConvMatrix.....	11-1
11.2	xmyCreateFormats.....	11-2
11.3	xmyPrintFormatFields.....	11-7
12.	Input Modeling with the AETG System.....	12-1
12.1	Modeling GUIs (Menus).....	12-1
12.2	Command Line Interface (Asynchronous Command).....	12-3
12.3	Configurations (Config).....	12-5

## Part C: Administrator Tasks

---

13.	Installing the AETG System.....	13-1
13.1	Introduction.....	13-1
13.2	Hardware and System Requirements.....	13-1

---

---

13.3	Networking.....	13-2
13.4	Preliminary Background Information .....	13-3
13.4.1	Assumptions and Recommendations .....	13-3
13.4.2	Environment Settings and xmyProfile and xmyLogin.....	13-4
13.4.3	Requirements .....	13-6
13.5	Preliminary Actions .....	13-6
13.5.1	Obtaining License Keys .....	13-6
13.5.2	Creating the mynah Group and AETG Administrator logid (madmin) ... 13-7	
13.5.3	Changes to /etc/services .....	13-7
13.5.4	BAIST Considerations .....	13-8
13.6	Installing the AETG System .....	13-11
13.6.1	AETG System Preinstallation Steps .....	13-11
13.6.2	AETG Software Installation.....	13-11
13.6.2.1	AETG Software CD-ROM Installation .....	13-11
13.6.2.2	AETG Software File Archive Installation .....	13-12
13.6.2.3	AETG Post Installation Steps .....	13-13
13.7	Installing the Telexel System .....	13-14
13.7.1	Telexel System Preinstallation Steps .....	13-14
13.7.2	Telexel Installation.....	13-15
13.7.2.1	Telexel Software CD-ROM Installation .....	13-15
13.7.2.2	Telexel Software File Archive Installation.....	13-15
13.7.2.3	Telexel Post Installation Steps.....	13-16
13.7.3	Configuring the Telexel System .....	13-16
13.7.4	Verifying the Telexel System .....	13-17
13.8	Installing Oracle .....	13-19
13.8.1	Oracle Preinstallation Steps .....	13-19
13.8.2	Oracle Installation .....	13-23
13.8.2.1	Verifying an Installation .....	13-25
13.8.2.2	Oracle Error Messages.....	13-25
13.8.3	TNS Configuration.....	13-26
13.8.4	Configuring the AETG System Oracle Database .....	13-26
13.8.5	Dropping the Oracle Database .....	13-30
13.8.6	Verifying Oracle .....	13-30
13.9	Installing the License Keys .....	13-32
13.10	Installing the On-line Documents .....	13-33
13.10.1	Installing the PDF Files.....	13-33
13.10.2	Obtaining the Acrobat Reader.....	13-33
13.10.2.1	Obtaining the Acrobat Reader from the CD-ROM.....	13-34
13.10.2.2	Obtaining the Acrobat Reader as a File Archive .....	13-34
13.10.2.3	Obtaining the Acrobat Reader from Adobe.....	13-34
13.10.3	Accessing the PDF Files .....	13-34
14.	Configuring the AETG System.....	14-1
14.1	Introduction.....	14-1

---



---

14.2	The xmyConfig.General File .....	14-1
14.3	The xmyConfigOP File Syntax .....	14-3
14.3.1	General Entry .....	14-3
14.3.2	Process Entries .....	14-4
14.3.3	License Server Start, Stop, and Status Commands .....	14-5
14.3.4	OperabilityAgent.....	14-5
14.3.5	Example xmyConfigOP File.....	14-7
14.4	The .xmyMYNAHrc File .....	14-8
15.	Operability Management — Starting and Stopping AETG Processes .....	15-1
15.1	Basic Steps .....	15-1
15.2	Overview .....	15-2
15.3	Operability Design .....	15-2
15.3.1	Operability Manager (OM) .....	15-3
15.3.2	Operability Agent (OA) .....	15-3
15.3.3	Operability xmyOM Subcommands .....	15-4
15.4	Licensing .....	15-5
15.4.1	AETG Licensing Commands .....	15-5
15.4.2	Starting the License Server .....	15-5
15.4.3	Stopping the License Server .....	15-5
15.4.4	License Utilities .....	15-6
15.4.4.1	Monitoring .....	15-6
15.4.4.2	Decoding.....	15-7
15.5	Starting Processes.....	15-8
15.5.1	Solaris Start-up Mechanism .....	15-8
15.5.2	Starting at Boot Time.....	15-8
15.5.2.1	.xmyRemovePips .....	15-9
15.5.2.2	xmyStartUp.....	15-9
15.5.3	Starting a Specific Process.....	15-10
15.5.4	Starting Autostart Processes.....	15-10
15.6	Stopping Processes.....	15-11
15.6.1	Stopping a Specific Process .....	15-11
15.6.2	Stopping Autostart Processes.....	15-11
15.6.3	Stopping an OA and all Autostart Processes on the Local Host .....	15-12
15.6.4	Stopping an OA and all Autostart Processes on a Specific Host .....	15-12
15.6.5	Stopping an OA.....	15-12
15.6.6	Stopping and Restarting a Host.....	15-13
15.7	Obtaining Information about Processes .....	15-13
15.7.1	Obtaining the Status of Processes .....	15-13
15.7.2	Displaying Operability Configuration Settings.....	15-14
15.8	Starting and Stopping AETG Software Packages.....	15-16
15.8.1	Automatically Starting the AETG System.....	15-16
15.8.2	Automatically Starting Oracle .....	15-16
15.9	User Defined Processes.....	15-17
15.10	Operability Summary .....	15-18

---

---

16. Administrative CLUI Commands .....	16-1
16.1 CLUI Command Help Messages .....	16-1
16.2 xmyOM .....	16-2
16.2.1 autostart .....	16-2
16.2.2 autostop .....	16-3
16.2.3 query .....	16-4
16.2.4 readconfig .....	16-5
16.2.5 recycle .....	16-6
16.2.6 shutdown .....	16-7
16.2.7 start/stop/status .....	16-8
17. Person Object .....	17-1
17.1 Creating a Person Object .....	17-2
17.1.1 Editing Properties Attributes .....	17-2
17.1.2 Editing Information Attributes .....	17-6
17.2 Editing an Existing Person Object .....	17-6
Appendix A: Using the AETG System with the MYNAH System .....	A-1
A.1 Related Documentation .....	A-1
A.2 Changes to Format Objects .....	A-2
A.3 Changes to SUT Objects .....	A-3
A.4 Changes to Keyword Objects .....	A-4
Appendix B: Generate Tcl Scripts for AETG Testcases .....	B-1
B.1 Test Object Script Generation View .....	B-1
B.2 Example .....	B-4
B.2.1 Effects of Code Compares .....	B-7
B.2.2 Effects of Pass Parameters .....	B-7
B.3 Generate Script Processing .....	B-7
B.4 Using Script Generation .....	B-8
B.4.1 Automation Considerations .....	B-8
B.4.2 Debugging Procedure Code .....	B-9
B.4.3 Analyzing Failures .....	B-9
Appendix C: Example Installation Files .....	C-1
C.1 Example AETG System Files .....	C-1
C.1.1 Example AETG Installation Session .....	C-1
C.1.2 Example System Changes File .....	C-4
C.1.3 Example AETG xmyProfile File .....	C-5
C.1.4 Example AETG xmyLogin File .....	C-9
C.1.5 Example Solaris AETG Start-up File (S99mynah.eg) .....	C-12
C.2 Example Telexel Installation Session .....	C-14
C.3 Delivered Example Oracle Files .....	C-16
C.3.1 Example Solaris Oracle Start-up File (S96oracle) .....	C-16
C.3.2 Example Oracle Configuration File .....	C-18
C.3.3 Example Oracle Initialization Script (initmynah5.ora) .....	C-19

---

C.3.4	Example Oracle crdbmynah5.sql File .....	C-22
C.3.5	Example Oracle crdb2mynah5.sql File .....	C-23
C.3.6	Example Oracle crdb3mynah5.sql File .....	C-26
C.3.7	Example xmyCreateSequences Execution .....	C-28
C.3.8	Example xmyCreateTemplates Execution .....	C-29
C.3.9	Example root.sh Run .....	C-30
Glossary .....		Glossary-1
Index .....		Index-1

---

## List of Figures

Figure 1-1.	Defining the Interface .....	1-2
Figure 1-2.	Associating a Format with Tests .....	1-3
Figure 1-3.	Defining the Rules .....	1-4
Figure 1-4.	Generating the Test Cases .....	1-5
Figure 1-5.	Recommended Steps for Creating a Test Case.....	1-6
Figure 1-6.	Cascading Menu Example .....	1-7
Figure 2-1.	New User Dialog .....	2-3
Figure 2-2.	MYNAH Desktop.....	2-4
Figure 2-3.	Naming a Folder .....	2-5
Figure 2-4.	New Folder on the AETG Desktop .....	2-5
Figure 2-5.	An Empty new Folder.....	2-6
Figure 2-6.	A new Untitled Format Object .....	2-6
Figure 2-7.	Format Object Properties View .....	2-7
Figure 2-8.	AETG Format Name Dialog.....	2-7
Figure 2-9.	Completed Format Object Properties View.....	2-8
Figure 2-10.	Format Object Fields View.....	2-9
Figure 2-11.	Creating a New Field Object .....	2-10
Figure 2-12.	Entering a Name for a Field Object.....	2-11
Figure 2-13.	Creating a New Value.....	2-12
Figure 2-14.	Entering a New Value.....	2-13
Figure 2-15.	Completed Shape Field Object .....	2-14
Figure 2-16.	Completed Format Object Fields View .....	2-15
Figure 2-17.	A New Untitled Test Hierarchy .....	2-16
Figure 2-18.	An Empty Test Hierarchy .....	2-16
Figure 2-19.	AETG Test Hierarchy Name Dialog .....	2-17
Figure 2-20.	A New Test Object .....	2-17
Figure 2-21.	Test Object Properties View .....	2-18
Figure 2-22.	Edited Test Object Properties View .....	2-19
Figure 2-23.	Test Object Relations View .....	2-21
Figure 2-24.	AETG Database Browser .....	2-22
Figure 2-25.	Associating a Format Object with a Test Object.....	2-23
Figure 2-26.	New Relation Object Properties View.....	2-24
Figure 2-27.	Edited Relation Object Properties View .....	2-25
Figure 2-28.	Test Object Matrix View .....	2-26
Figure 2-29.	Generated Test Case Matrix .....	2-27
Figure 2-30.	Save Matrix Dialog.....	2-28
Figure 2-31.	Entering a Name for the Test Case Matrix .....	2-28
Figure 2-32.	Saved Matrix View.....	2-29
Figure 2-33.	Unlocked Relation Object Properties View.....	2-30
Figure 2-34.	Selecting an Interaction Degree to Change It.....	2-31
Figure 2-35.	Specifying a new Interaction Degree.....	2-31

---

Figure 2-36.	Generated Test Case Matrix for new Interaction Degree .....	2-32
Figure 2-37.	Saving the new Test Case Matrix .....	2-33
Figure 2-38.	Relation Object Excludes View .....	2-34
Figure 2-39.	Creating a new Excluded Test Case .....	2-35
Figure 2-40.	Specifying an Excluded Test Case Name .....	2-36
Figure 2-41.	Selecting Values for an Excluded Test Case .....	2-37
Figure 2-42.	Setting Values for an Excluded Test Case .....	2-38
Figure 2-43.	Excluded Test Case Test Case Matrix .....	2-39
Figure 2-44.	Saving the Excluded Test Case Matrix .....	2-40
Figure 2-45.	Save Changes Dialog .....	2-40
Figure 3-1.	What Are Folders? .....	3-2
Figure 3-2.	AETG Desktop List View .....	3-3
Figure 3-3.	AETG Icon .....	3-4
Figure 3-4.	Using Scroll Bars .....	3-10
Figure 3-5.	AETG Icon .....	3-12
Figure 3-6.	Spin Buttons .....	3-14
Figure 3-7.	Toggle Buttons .....	3-14
Figure 3-8.	Radio Buttons .....	3-15
Figure 3-9.	Pushbuttons .....	3-15
Figure 3-10.	Database Browser Displaying Format Objects .....	3-17
Figure 3-11.	AETG Desktop - List View .....	3-19
Figure 3-12.	List View Expanded .....	3-20
Figure 3-13.	List View Expanded Fully .....	3-21
Figure 3-14.	AETG Desktop Preferences View .....	3-22
Figure 3-15.	Edited AETG Desktop Preferences View .....	3-24
Figure 3-16.	Font Chooser Dialog .....	3-25
Figure 3-17.	Preferences View with Font Change .....	3-26
Figure 3-18.	Requesting Information Dialog .....	3-27
Figure 3-19.	New Folder Icon .....	3-28
Figure 3-20.	Folder Properties View .....	3-29
Figure 3-21.	Print Dialog .....	3-30
Figure 3-22.	Example Print Dialog Box .....	3-31
Figure 3-23.	Printout Using Fixed Width Fonts .....	3-32
Figure 3-24.	Printout Using Variable Width Fonts .....	3-33
Figure 4-1.	New Format Object on AETG Desktop .....	4-3
Figure 4-2.	Creating an Object in an Open Folder .....	4-4
Figure 4-3.	Selecting a Folder .....	4-5
Figure 4-4.	Creating an Object in Closed Folder .....	4-5
Figure 4-5.	New Object Created In a Closed Folder .....	4-5
Figure 4-6.	Object Status Display and Default View Area .....	4-9
Figure 4-7.	Multiselecting Fields Objects on a Format Object .....	4-10
Figure 5-1.	Database Browser Window .....	5-2
Figure 5-2.	AETG Question Dialog .....	5-3
Figure 5-3.	Database Browser Window with Test Objects Displayed .....	5-4

---

---

Figure 5-4.	Include Dialog .....	5-5
Figure 5-5.	Include Query Example .....	5-6
Figure 5-6.	Test Object Include Dialog.....	5-8
Figure 5-7.	Include Dialog with One Condition Specified. ....	5-9
Figure 5-8.	Include Dialog with Two Conditions Specified. ....	5-10
Figure 5-9.	Deleting a Row .....	5-11
Figure 5-10.	Person Object Properties View .....	5-13
Figure 5-11.	Person Object Opened for Editing.....	5-14
Figure 5-12.	Person Object Information View .....	5-15
Figure 6-1.	Format Object Properties View .....	6-2
Figure 6-2.	Format Object Fields View.....	6-4
Figure 6-3.	Creating a New Field Object .....	6-6
Figure 6-4.	Pasted Field Object Dialog .....	6-8
Figure 6-5.	Duplicate Field Object Name Error Dialog.....	6-8
Figure 6-6.	Selecting Field Objects for a Compound.....	6-9
Figure 6-7.	Compound Object Properties View .....	6-10
Figure 6-8.	Field used in Existing Compound Error Dialog .....	6-11
Figure 6-9.	Deleting Used Compound Error Dialog .....	6-12
Figure 6-10.	Selecting a Field for Rearranging.....	6-13
Figure 6-11.	Resequenced Field Objects.....	6-13
Figure 6-12.	Format Object Associations View .....	6-14
Figure 6-13.	AETG Duplicate Format Dialog.....	6-15
Figure 6-14.	Original Format Object.....	6-16
Figure 6-15.	Creating a New Version of an Existing Format Object.....	6-16
Figure 6-16.	Creating a Clone of a Format Object.....	6-17
Figure 7-1.	Field Object Values View.....	7-2
Figure 7-2.	Non-unique Value Error Dialog .....	7-5
Figure 7-3.	Creating a New Value.....	7-5
Figure 7-4.	Entering a New Value.....	7-6
Figure 7-5.	Selecting a Value to Change.....	7-7
Figure 7-6.	Pasting a Copied Value.....	7-8
Figure 7-7.	Deleting a Value Error Dialog.....	7-9
Figure 7-8.	Field Object Properties View .....	7-10
Figure 8-1.	Compound Object Properties View .....	8-2
Figure 8-2.	Compound Object Fields View .....	8-4
Figure 8-3.	Multiselecting Values to be used in a Compound .....	8-5
Figure 8-4.	Compound Object Tuples View .....	8-7
Figure 8-5.	Selecting Fields and Interaction Degree for Generating Tuples.....	8-8
Figure 8-6.	Generated Tuples .....	8-9
Figure 8-7.	Regenerating an Identical Tuple Set.....	8-11
Figure 8-8.	Generating Tuples Confirm Dialog .....	8-12
Figure 8-9.	Maximum Tuples Confirm Dialog .....	8-12
Figure 8-10.	Deleting Used Tuple Error Dialog.....	8-14
Figure 9-1.	Hierarchy Concept.....	9-1

---

---

Figure 9-2.	Test Object Hierarchy.....	9-2
Figure 9-3.	New Test Hierarchy.....	9-3
Figure 9-4.	Test Hierarchy List View .....	9-3
Figure 9-5.	Test Hierarchy Name Dialog .....	9-4
Figure 9-6.	New Test Object Dialog .....	9-5
Figure 9-7.	New Object at Second Level .....	9-5
Figure 9-8.	Another New Object at First Level.....	9-6
Figure 9-9.	Selecting a Test Object to Duplicate .....	9-6
Figure 9-10.	Duplicated Test Object .....	9-7
Figure 9-11.	Test Object Properties View .....	9-9
Figure 9-12.	Test Object Relations View .....	9-11
Figure 9-13.	Selecting a Format to Associate with a Test Object.....	9-12
Figure 9-14.	Associating a Format with a Test Object.....	9-13
Figure 9-15.	Creating a New Relation Object.....	9-14
Figure 9-16.	Duplicate Relation Object .....	9-15
Figure 9-17.	Test Object Matrix View .....	9-17
Figure 9-18.	Generated Test Case Matrix .....	9-19
Figure 9-19.	Selecting Test Case Rows to Delete.....	9-20
Figure 9-20.	Updated Matrix View .....	9-21
Figure 9-21.	Save Matrix Dialog.....	9-22
Figure 9-22.	Entering a Name for a Test Case Matrix File.....	9-22
Figure 9-23.	Saved Matrix.....	9-23
Figure 9-24.	Load Matrix Dialog .....	9-24
Figure 9-25.	Delete Matrix Dialog .....	9-25
Figure 9-26.	Max Testcases Exceeded Dialog .....	9-25
Figure 9-27.	Matrix Print Dialog.....	9-27
Figure 9-28.	Printing a Matrix to a File.....	9-28
Figure 9-29.	Matrix File Content .....	9-28
Figure 9-30.	xmyConvMatrix Output .....	9-29
Figure 10-1.	Relation Object Properties View .....	10-3
Figure 10-2.	Relation Object Fields View.....	10-5
Figure 10-3.	Change Fields Confirm Dialog.....	10-7
Figure 10-4.	Changing Fields.....	10-7
Figure 10-5.	Updated Fields View Showing Affect of Changing Fields.....	10-8
Figure 10-6.	Selecting Values to add to the Don't Use List.....	10-9
Figure 10-7.	Values added to the Don't Use List.....	10-10
Figure 10-8.	Relation Object Includes View .....	10-11
Figure 10-9.	Creating a New Include .....	10-13
Figure 10-10.	Defining a new Included Test Case .....	10-14
Figure 10-11.	Undefined Relation Error Dialog .....	10-15
Figure 10-12.	Non-unique Included Test Case Name Error Dialog.....	10-16
Figure 10-13.	Duplicating an Included Test Case.....	10-17
Figure 10-14.	Relation Object Excludes View.....	10-19
Figure 10-15.	Creating a New Exclude.....	10-22

---

---

Figure 10-16. Defining a new Excluded Test Case.....	10-23
Figure 10-17. Duplicating an Included Test Case.....	10-25
Figure 10-18. Relation Object Invalids View .....	10-27
Figure 10-19. Creating a New Invalid Test Case .....	10-29
Figure 10-20. Specifying a Value for an Invalid Test Case .....	10-30
Figure 11-1. Format File Structure .....	11-3
Figure 11-2. Example Format File .....	11-6
Figure 11-3. Sample Output from xmyPrintFormatFields .....	11-7
Figure 12-1. GUI Modeling Menu Example .....	12-1
Figure 12-2. Test Matrix for Testing the GUI Menus Model .....	12-3
Figure 12-3. Test Matrix for Testing the ls Command Model.....	12-4
Figure 12-4. Test Case Matrix for Testing Configuration Model .....	12-6
Figure 13-1. Networked Services .....	13-2
Figure 13-2. Recommended Oracle Software Directory Structure .....	13-4
Figure 13-3. BAIST Opening Screen .....	13-9
Figure 13-4. BAIST Product Menu .....	13-10
Figure 14-1. xmyConfig.General Entry .....	14-1
Figure 14-2. xmyConfig Process Entry .....	14-4
Figure 14-3. xmyConfigOP OperabilityAgent Entry Structure .....	14-5
Figure 14-4. Example xmyConfigOP OperabilityAgent Entry.....	14-6
Figure 14-5. Example AETG xmyConfigOP File.....	14-8
Figure 15-1. Operability Architecture .....	15-2
Figure 15-2. Operability Feature Summary .....	15-18
Figure 17-1. Creating a New Person Object.....	17-2
Figure 17-2. New Person Object Properties View .....	17-3
Figure 17-3. Edited Person Object Properties View .....	17-4
Figure 17-4. Duplicate UNIX ID Error Box .....	17-5
Figure 17-5. Close Dialog Box.....	17-5
Figure 17-6. Person Object Information View .....	17-6
Figure 17-7. Database Browser - List of Person Objects.....	17-7
Figure 17-8. Edited Person Object — Granting Administrative Privileges .....	17-8
Figure 17-9. Refreshed Database Browser.....	17-9
Figure A-1. Format Object Associations View .....	A-2
Figure A-2. SUT Object Associations View .....	A-3
Figure A-3. Keyword Object Association View .....	A-4
Figure B-1. Test Object Script Generation View.....	B-1
Figure B-2. Edited Script Generation view.....	B-5
Figure B-3. Sample Generated Script Code.....	B-6
Figure B-4. Script Generate Confirm Dialog.....	B-8

---



---

## List of Tables

Table 1.	AETG System Users Guide Sections .....	Preface-1
Table 2-1.	Quick Tour Elements and Values .....	2-2
Table 2-2.	AETG System Users Guide Sections .....	2-41
Table 3-1.	Tool Bar Icons and Functions.....	3-8
Table 3-2.	Mouse Actions.....	3-11
Table 3-3.	Key Accelerators .....	3-13
Table 3-4.	Pushbuttons and Their Functions .....	3-15
Table 4-1.	AETG Objects .....	4-1
Table 4-2.	Attributes Copied by Duplicate .....	4-7
Table 5-1.	Include Attribute and Default Query .....	5-6
Table 7-1.	Field Object Value Type Examples.....	7-4
Table 9-1.	Duplicated/Nonduplicated Test Object Attributes .....	9-7
Table 9-2.	Duplicated/Nonduplicated Relation Object Attributes.....	9-16
Table 12-1.	GUI Modeling Menu Example .....	12-1
Table 12-2.	GUI Menu Fields and Values .....	12-2
Table 12-3.	Menu Relations.....	12-2
Table 12-4.	Modeling ls UNIX Command .....	12-4
Table 12-5.	Modeling Configuration Scenarios.....	12-5
Table 12-6.	System Configuration Restrictions.....	12-5
Table 13-1.	Installation Steps.....	13-1
Table 13-2.	Required Software Packages .....	13-2
Table 13-3.	Oracle Installation Items and Responses .....	13-24
Table 13-4.	Oracle Software Asset Manager Packages .....	13-25
Table 15-1.	xmyOM Sub-commands.....	15-4



---

## Preface

### Document Structure

Table 1 list the sections in this document with a brief description of each section.

**Table 1.** AETG System Users Guide Sections (Sheet 1 of 2)

Section Number	Section Name	Description
Section 1	Introduction	This section contains general information on this guide including an overview of some basic AETG functionalities.
Section 2	A Quick Tour of the AETG System	This section details several basic tasks that will help you become familiar with the AETG System.
Section 3	AETG Basics	This section contains basic information on using the AETG GUI.
Section 4	Using AETG Objects	This section contains information on how to use AETG objects
Section 5	Using the Database Browser	This section contains information on the AETG Database Browser.
Section 6	Format Object	This section contains information on Format objects, which describe a testable interface to an application or a testing situation.
Section 7	Field Object	This section contains information on Field objects, which define the data values that are input to an application.
Section 8	Compound Object	This section contains information on Compound objects, which let you collect several Fields and treat them as one Field.
Section 9	Test Object	This section contains information on Test objects, which define the means by which a requirement is verified in a software application.  This section also contains information on how to generate a Test Case Matrix.
Section 10	Relation Object	This section contains information on Relation objects, which let you collect requirements data to establish how fields and values on the format relate to one another.

**Table 1.** AETG System Users Guide Sections (Sheet 2 of 2)

Section Number	Section Name	Description
Section 11	Command Line Tools	This section contains information on the AETG Command Line tools.
Section 12	Input Modeling with the AETG System	This section provides tips on how the AETG System could be used to model various types of applications.
<b>NOTE</b> — The following sections contain information required by the AETG Administrator.		
Section 13	Installing the AETG System	This section contains information on installation, including required third-party software.
Section 14	Configuring the AETG System	This section contains information on configuring a working installation.
Section 15	Operability Management — Starting and Stopping AETG Processes	This section describes the procedures used to start, stop, and get status of all of the AETG processes.
Section 16	Administrative CLUI Commands	This section describes the tasks used to administer the AETG System.
Section 17	Person Object	This section describes the tasks used to administer Person objects.
Appendix A	Using the AETG System with the MYNAH System	This section contains information on the differences in the GUI when the AETG System is licensed with the MYNAH System.
Appendix B	Generate Tcl Scripts for AETG Testcases	This section contains information on generating test code when the AETG System is licensed with the MYNAH System.
Appendix C	Example Installation Files	This appendix contains examples of the delivered files used to install the AETG System.

## Formatting Conventions

The following formatting conventions are used in this document.

- Substitutable or user supplied items (e.g., option and argument values) appear in ***boldItalics Times***.
- File and directory names appear in *italics Times*.
- The first time a term is used, it will appear in **bold 11-point Helvetica**, such as when we first mention **Format** object.
- Menu names and items will appear in **bold 10-point Helvetica**, such as when we mention the **Edit** menu.
- Cascading menus will appear in **bold Helvetica** with arrows between the menu options to signify the cascades, such as

**Aetg->New->Format**



## **Part A: General Information**

---





## 1. Introduction

This document describes the procedures for using the Automatic Efficient Testcase Generator (AETG™) System.

Software testing is expensive in both time and material resources. By various estimates, testing can cost as much as 20% to 33% of the total development budget of a software project. The AETG System can greatly reduce the cost of testing by reducing time to develop test plans and by reducing the number of maintained test cases. The Bellcore proprietary combinatorial algorithm generates an efficient set of test cases and these test cases provide a high degree of coverage for the application under test. The generated output can be easily introduced into application testing tools to populate screens and/or messages with data for test automation.

### 1.1 AETG System Overview

The AETG System runs on the Solaris™ platform (Release 2.5.1) with any X-windows™ manager.

Using the AETG System entails performing the following steps:

1. Defining the Interface
2. Defining and Associating Tests
3. Defining the Rules of the Interface
4. Generating the Efficient Set of Test Cases
5. Automating the Test Cases.

The following subsections provide general information each of these tasks.

#### 1.1.1 Starting the AETG System

To start the AETG System, simply type

```
xmyRunAetg
```

in a standard UNIX window.

**NOTE** — The AETG System automatically runs in the background.

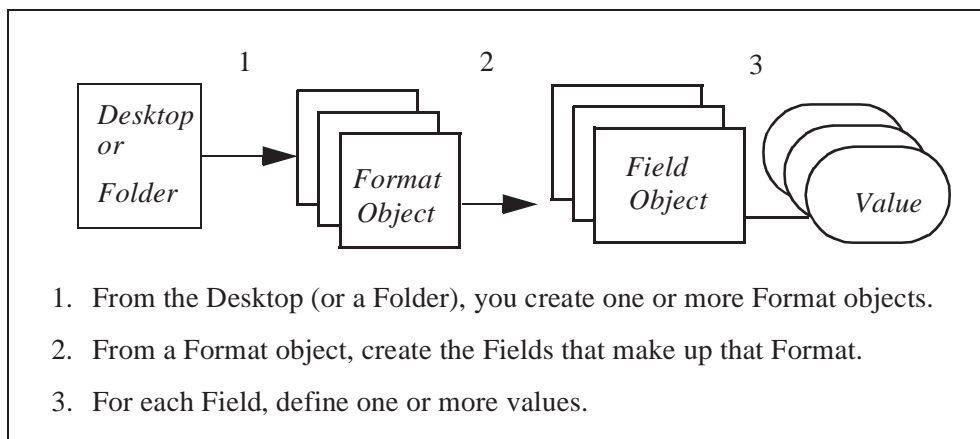
## 1.1.2 Defining the Interface

An interface is the means by which you interact with an application or testing situation, such as a screen or a communications protocol. You define an interface into which data is input to an application using an AETG database object called a **Format** object. A Format object is a collection of **Field** objects, which are database objects that are used to hold information about the data to be applied to an application. You must define one or more Field objects from within the Format and then assign one or more Values to each Field.

A completed Format object, therefore, represents an interface to a System Under Test (SUT) that you wish to validate, i.e., test.

See [Section 6](#) for information on working with Format objects. See [Section 7](#) for information on working with Field objects.

[Figure 1-1](#) illustrates the flow of steps you follow to define an interface and how Formats, Fields and Values are related to each other.



**Figure 1-1.** Defining the Interface

After you have created more than one Field objects, you can optionally create a **Compound** object, which is a database object that lets you collect simple Fields into one set and then treat that set of Fields as a single Field.

Once you have created a Compound object, you must create at least one **Tuple**, which is a reference name given to the specific value set of a Compound.

See [Section 8](#) for information on working with Compound objects and Tuples.

### 1.1.3 Defining and Associating Tests

A **Test** object is a database object that is used to track and verify compliance with specific system requirements. Test objects are created within **Test Hierarchies**, which are equivalent to directories (or folders).

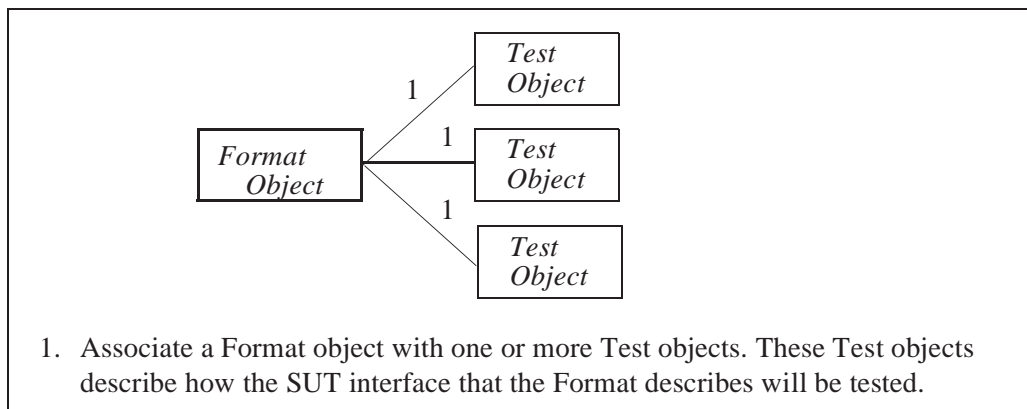
See [Section 9](#) for information on working with Test objects.

You may need one or more Test objects to test all the things you want to verify about a Format, such as in the following scenarios:

- You may wish to exercise all the Fields and all combinations of Fields in a Format.
- You may wish to test certain invalid Field values to see how the System Under Test behaves when given invalid input.
- You can test a subset of the Format's Fields with one Test object and the remaining Fields with a different Test object.
- You can test the entire Format with one Test object.

Once you've created a Format object and the desired Test objects, you must **associate** the Format object with each Test object that will validate some aspect of the Format, creating a link between the Format object and the pertinent Test objects.

[Figure 1-2](#) depicts a Format object being associated with multiple Test objects.



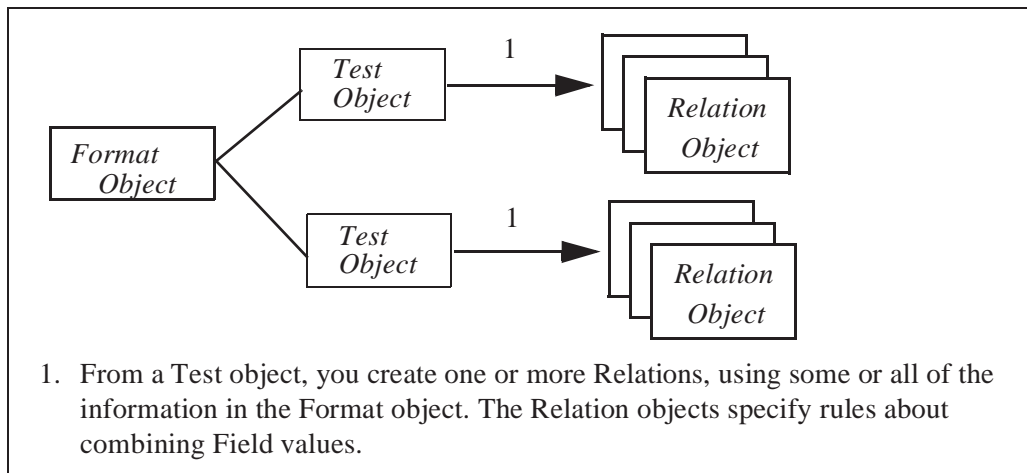
**Figure 1-2.** Associating a Format with Tests

### 1.1.4 Defining the Rules of the Interface

After you define the testing interface and define and associate tests, you create **Relations**, which are database objects that define the rules governing the data (i.e., formal requirements) that will be input to the System Under Test.

You create one or more Relation objects from within a Test object and define, for each one, one or more rules, such as “When Field1 has a value of *blue*, Field2 may not be *pink* or *yellow*.” The Relation uses a set of Fields from a Format.

Figure 1-3 depicts the flow of steps you follow when associating Relation objects with Test objects.



**Figure 1-3.** Defining the Rules

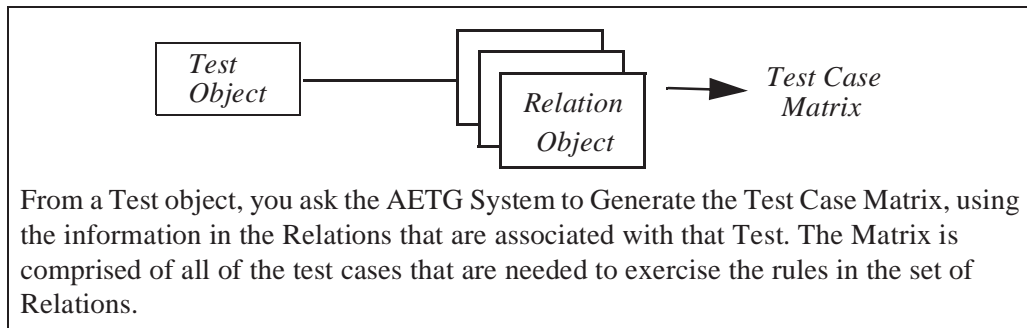
Using a Relation object, you can

- Specify which values must be included in the test case. This does not mean these are the only values that will be included, but that priority will be given that these values are included.
- Specify which values are to be excluded.
- Specify a set of invalid values to do error condition testing.

### 1.1.5 Generating the Efficient Set of Test Cases

When you have finished specifying all the rules that will be used for this Test, you can then ask the AETG System to generate the efficient set of test cases for this Test. This operation is performed from the Test object, and the resulting Test Case Matrix is stored with the Test object.

Figure 1-4 depicts the flow of steps you follow to generate the Test Case Matrix.



**Figure 1-4.** Generating the Test Cases

### 1.1.6 Automating the Test Cases

The generated Test Case Matrix can potentially be read by an automated testing tool, such as the MYNAH System, to create a script to automate test cases.

### 1.1.7 Summary

1. Defining the Interface
  - A. Create a Format object.
  - B. Define one or more Field objects.
  - C. Assign one or more Values to each Field.
2. Defining and Associating Tests
  - A. Create one or more Test objects that will be used to exercise the interface.
  - B. Associate these Tests with the Format.
3. Defining the Rules of the Interface

Create one or more Relation objects from within a Test object and define, for each one, one or more rules.
4. Generating the Efficient Set of Test Cases

Generate a Test Case Matrix using the defined Relation(s).
5. Automating the Test Cases

Figure 1-5 contains a flow chart describing the recommended of the steps required to create a test case.

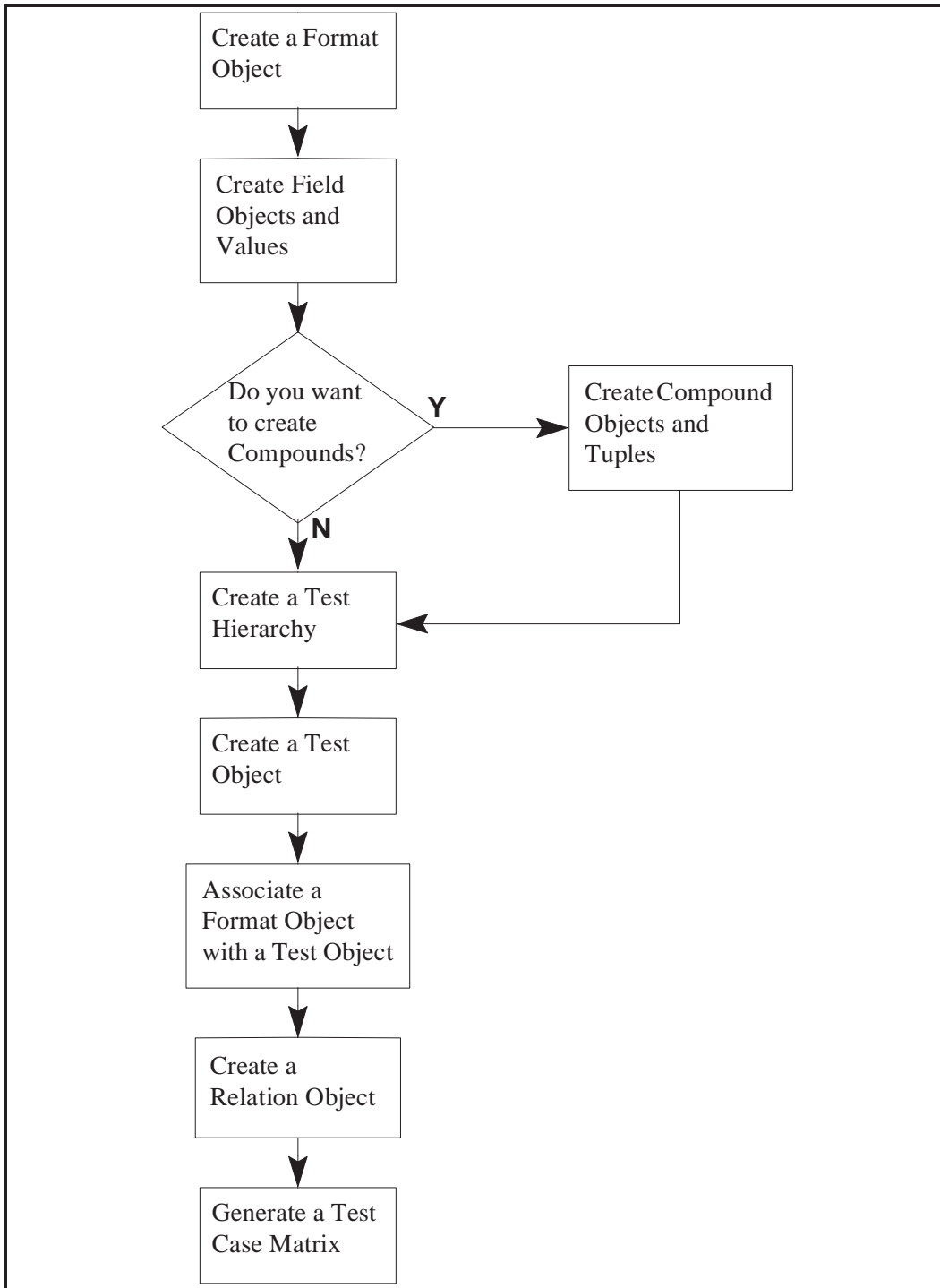


Figure 1-5. Recommended Steps for Creating a Test Case

## 1.2 Command Line User Interface (CLUI)

The Command Line User Interface (CLUI) is a set of commands you can use to perform AETG utility operations from the UNIX<sup>®</sup> command line.

See [Section 11](#) for descriptions of the CLUI commands.

## 1.3 Some Basic Terms we Use

Before we proceed, let us define a few terms and concepts we will use in this document.

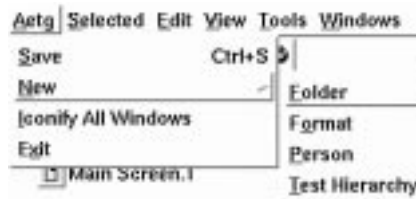
### 1.3.1 Mouse Terms

When referring to how to use a mouse (when working with the AETG GUI), we use the following terms:

- Click** This means that you should place the mouse pointer on an item we identify in an AETG window and press down and release the left mouse button quickly.
- Double click** This means to press down and release the left mouse button twice in rapid succession.
- Pointer** This refers to the arrow that indicates the mouse position.

### 1.3.2 Cascading Menus

Many AETG menu items contain a triangle that indicates that there is a submenu associated with that menu item, such as the one in [Figure 1-6](#).



**Figure 1-6.** Cascading Menu Example

For example, to access the function depicted in [Figure 1-6](#), we would tell you to execute

**Aetg->New->Folder**

This means you would

1. Click on the **Aetg** menu.
2. Click on the **New** menu option.
3. Click on the **Folder** menu option.

## 1.4 On-line Documentation

This document is available on-line in the Adobe<sup>®</sup> Acrobat<sup>®</sup> PDF format. The PDF file is accessible from either the local file system or from an internal URL. See the AETG administrator for the location of the PDF file.

Viewing the PDF file requires that you have installed the Adobe Acrobat Reader<sup>®</sup> (Release 3.0). If you need the Acrobat Reader, contact the AETG administrator. In addition, you can download the Acrobat Reader directly (off the Internet) from Adobe at [www.adobe.com](http://www.adobe.com). Follow the instructions detailed on the web page.

Once you have installed the Acrobat Reader, you can read the file

- Using the Acrobat Reader directly if the AETG System has been installed on a local system.
- Using the Acrobat Reader as plug-in to a browser if the AETG System has not been installed on a local system, such as if the system has been installed on a UNIX Solaris server and you are using an X-windows emulator to access the system on a PC. Consult your browser's documentation for information on how to install plug-ins.

If you access the PDF file via a browser, you may wish to download the file to your local system, which will give you direct access to the file the next time you need to read the file, rather than waiting for the browser to load it.

## 1.5 Customer Support

You can get support for the AETG System from the MYNAH Customer Service Center between 8:00 AM and 7:00 PM ET Monday through Friday by calling (732) 699-2668, Option 3; if outside the 732 area, call (800) 795-3119, Option 3.

During non-standard support hours and holidays, critical problem coverage is provided.

You can also contact support (for non critical problems) via e-mail at [mynah-support@cc.bellcore.com](mailto:mynah-support@cc.bellcore.com).



## **Part B: User Tasks**

---



## 2. A Quick Tour of the AETG System

Before we explain the AETG system in detail we want you to perform some simple tasks that will let you become familiar with how to use the AETG System and what it can do for you. This section provides a Quick Start, touching on the basic features of the AETG System.

**NOTE** — This section does not touch on all of the features of the AETG System nor provide detailed information on the features we do discuss.

In this Quick Tour, you will

1. Create a folder on the AETG desktop to contain the AETG objects.
2. Create a Format object and the required Field objects to define the interface.
3. Create a Test Hierarchy and Test object.
4. Create a Relation object to test all possible combinations of the Fields you create.
5. Generate the Test Case Matrix, which is the the list of the generated efficient set of test cases.
6. Change how the Fields interact with each other.
7. Regenerate the Test Case Matrix.
8. Specify fields to be excluded from a test case.
9. Regenerate the Test Case Matrix.

## 2.1 Define what you want to Test

Before you create the AETG objects, you first have to define what you are using the AETG System to test (or model) and the rules of the test.

### 2.1.1 Define the Interface

This Quick Tour example uses the AETG System to model a very basic display, defining the shape, color, and “attributes” of a display element. [Table 2-1](#) lists the display elements and the possible values for each element. Each element is defined using a Field object.

**Table 2-1.** Quick Tour Elements and Values

Element Field Name	Possible Values
Shape	<ul style="list-style-type: none"><li>• Circle</li><li>• Square</li></ul>
Color	<ul style="list-style-type: none"><li>• Red</li><li>• Blue</li><li>• Yellow</li></ul>
Attribute	<ul style="list-style-type: none"><li>• Blinking</li><li>• Non-blinking</li></ul>

### 2.1.2 Define the Rules of the Test

For the Quick Tour, you will use the AETG System to generate

1. All possible permutations.
2. All permutations excluding Yellow Squares.

## 2.2 Start the AETG System

Start the AETG System by typing

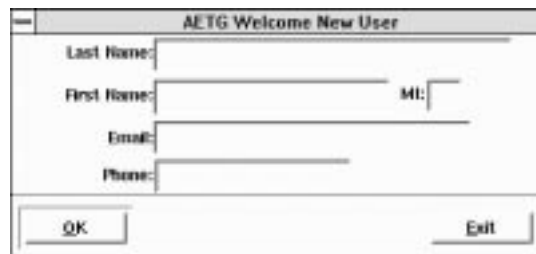
```
xmyRunAetg
```

in a standard UNIX window.

**NOTE** — Before you start the AETG System, your path must be set properly to contain *\$XMYDIR/bin*. Also, your XMYHOME environment variable must be set. If you need to, call your AETG administrator for more information about doing this.

## 2.3 Registering as an AETG User

The first time you start the AETG System as a new user, the system displays a New User dialog (Figure 2-1). This dialog prompts you for basic information about yourself that the system will store in its database.



**Figure 2-1.** New User Dialog

The system is case-insensitive to information you enter here; you can type information in upper case, lower case, or combination of upper and lower case.

To enter information

1. Type in your last name in the **Last Name** field.
2. Position the pointer in the **First Name** field (you can **Tab** over to it or point and click with the mouse) and type in your first name.
3. Position the pointer in the **MI** field and type in your middle initial.
4. Position the pointer in the **Email** field and type in your e-mail address.
5. Position the pointer in the **Phone** field and type in your business phone number.
6. Use the mouse to click the **OK** button in the bottom, left-hand corner of the dialog.

You are now registered with your local system as an AETG user.

The system displays the AETG Desktop (Figure 2-2). This is the first of many AETG System windows and dialog boxes. We will describe the AETG Desktop in Section 3.



**Figure 2-2.** MYNAH Desktop

## 2.4 Create a Folder

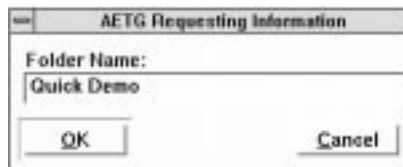
The AETG System lets you organize your work into items called **Folders**, which are the equivalent of UNIX directories or paper folders in which you place a number of documents.

To create a folder

1. Execute

**Aetg->New->Folder**

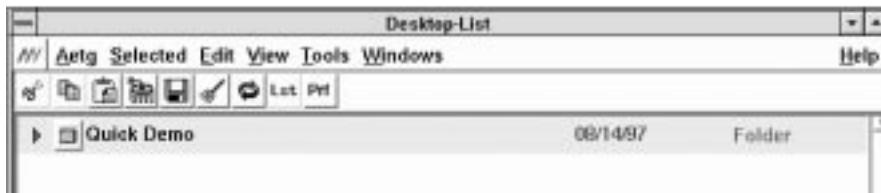
A dialog box (Figure 2-3) appears requesting a name for the folder.



**Figure 2-3.** Naming a Folder

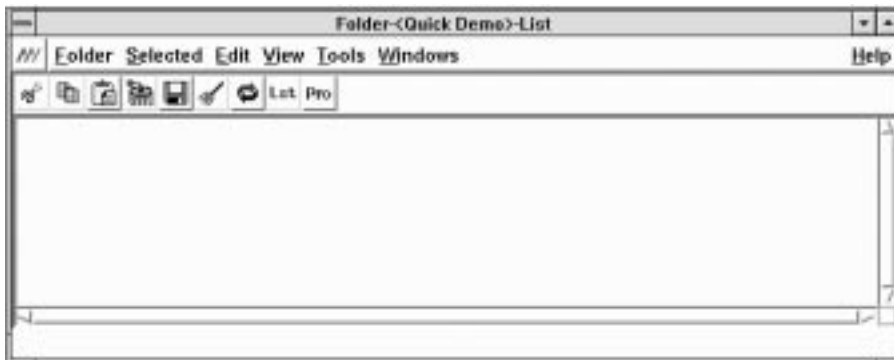
2. Position the pointer in the **Folder Name** text area and type in a name for it, for example, *Quick Demo*.
3. Click on **OK**.

The system creates a new folder and places it in the desktop display as a line item. (See Figure 2-4.)



**Figure 2-4.** New Folder on the AETG Desktop

4. Open the Folder by double clicking on the Folder's icon. An empty Folder appears.  
(See [Figure 2-5](#).)



**Figure 2-5.** An Empty new Folder

## 2.5 Create a Format Object

A Format object lets you

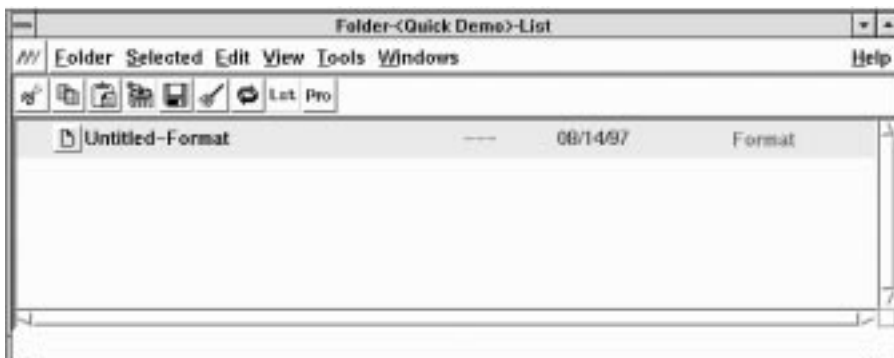
- Document the interface you want to represent.
- Define all of the Fields on the Format.

To create a Format object

1. Execute

**Folder->New->Format**

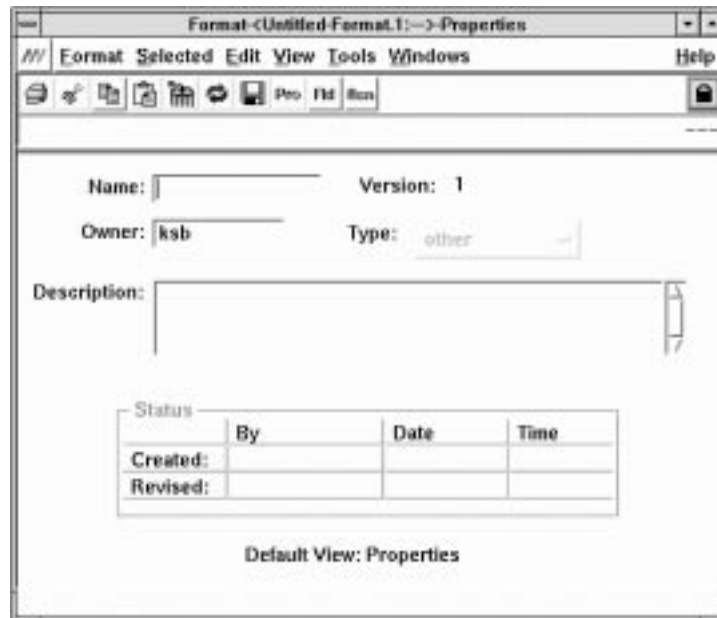
A new untitled Format object item appears in the Folder. (See [Figure 2-6](#).)



**Figure 2-6.** A new Untitled Format Object



2. Open the Format object by double clicking on the Format object icon. By default, the system displays the Format object's Properties view. (See [Figure 2-7](#).)

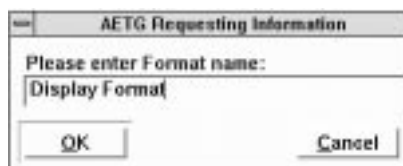


**Figure 2-7.** Format Object Properties View

3. Click on the **Lock** icon (in the upper, right-hand corner of the view) to "Unlock" the Format object.

**NOTE** — See [Table 3-1](#) in [Section 3.1.4](#) for a list of the icons in the AETG Tool Bar.

4. The AETG Format Name Dialog ([Figure 2-8](#)) appears. Enter a name for the Format object, for example, *Display Format* and click on **OK**.



**Figure 2-8.** AETG Format Name Dialog

5. Optionally, you can

- Select the Format type by clicking on the **Type** menu and selecting the type of interface the Format is, for example, **AsyncScreen**.
- Enter a description of the Format object.

The Format object Properties view will look similar to the one in [Figure 2-9](#).



**Figure 2-9.** Completed Format Object Properties View

6. Proceed to [Section 2.6](#) to create the required Field objects and associated Values.

**NOTE** — See [Section 6](#) for detailed information on Format objects.

## 2.6 Create Field Objects and Values

A Field object lets you enter information about the Fields on a Format, such as the names of the Fields and a list of valid values for the Fields.

Creating a Field object is a two step process.

1. You first create a Field object from a Format object's Fields view.
2. You then specify possible values for the Field object.

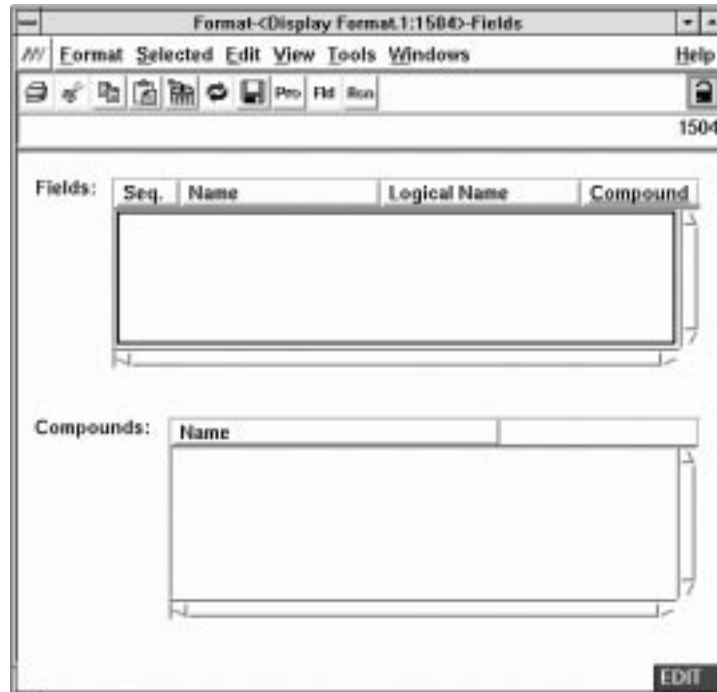
You will create the Field object and values for the *Shape* display element listed in the first row of [Table 2-1](#).

To create the *Shape* Field object

1. On the unlocked Format object you created in [Section 2.5](#), display the Format object Fields view. You can do this in one of two ways:
  - Click on the **Fld** button on the Format object Tool Bar.
  - Execute

**View->Change View->Fields**

The Fields view ([Figure 2-10](#)) appears.



**Figure 2-10.** Format Object Fields View

2. Execute

**Format->New->Field**

An untitled Field row, marked **<new field>**, appears in the **Fields** list on the Format object, and a new unlocked Field object appears.

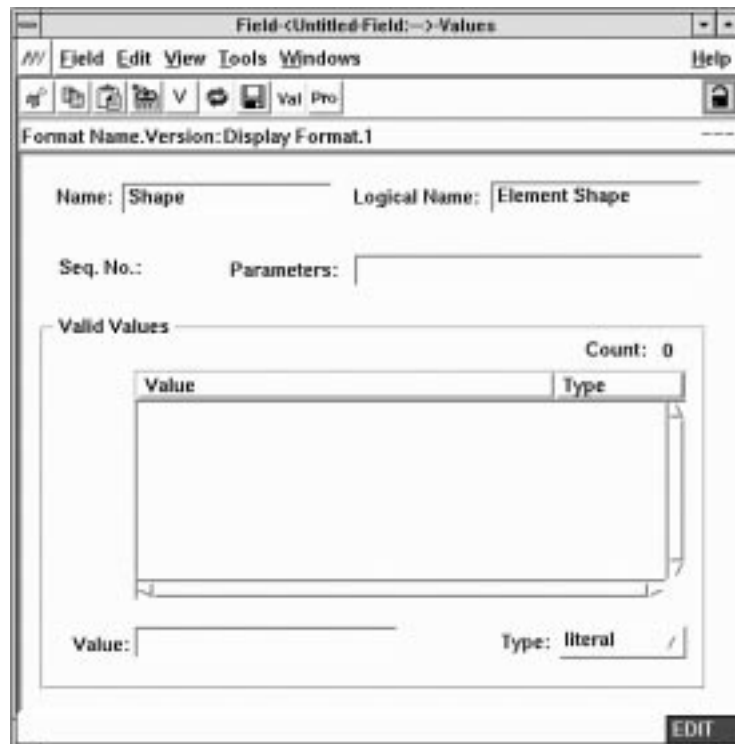


Figure 2-11. Creating a New Field Object

3. Enter, at the minimum, a name for the Field object in the **Name** text area, for example, *Shape*.

**NOTE** — You can not include embedded spaces in a Field object name. However, you can include embedded spaces in a name you enter in the **Logical Name** text area, for example, *Element Shape*.

The edited Field object will be similar to the one in [Figure 2-12](#).

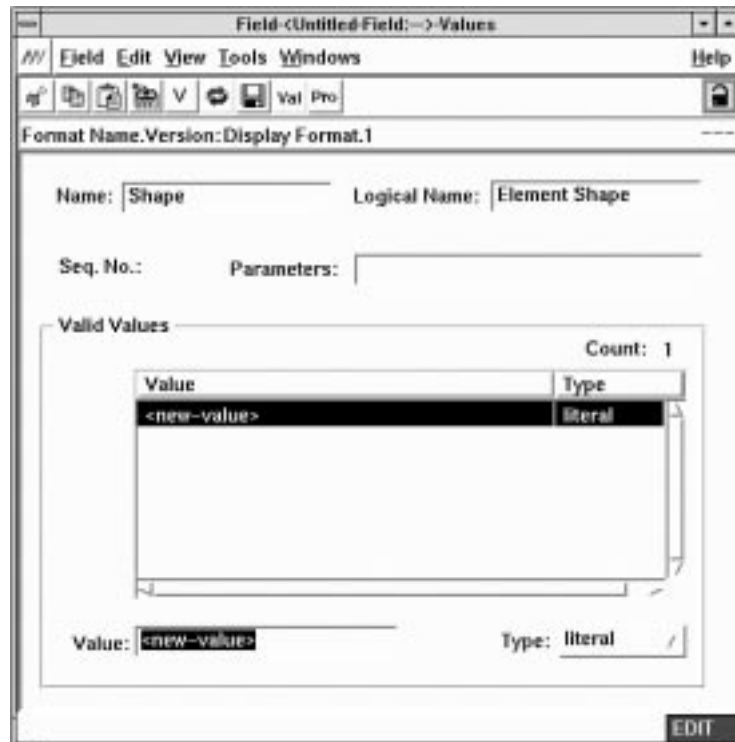


**Figure 2-12.** Entering a Name for a Field Object

4. Execute

**Field->New Value**

An untitled Value row, marked **<new-value>**, appears in the **Valid Values** list on the Field object. In addition, **<new-value>** appears in the **Value** text area and is highlighted, signifying that it is selected and you can type over this entry.



**Figure 2-13.** Creating a New Value

**NOTE** — You can also create a new Values by executing **Ctrl+N** (pressing and holding the **Control** key and then the **N** key) or by clicking on the **V** button on the Tool Bar.

5. Enter the first value in the **Value** text area, for example, *Circle*, and press either the **Return** or **Tab** key. (See Figure 2-14.) The system accepts the value and enters it in the **Valid Values** list.

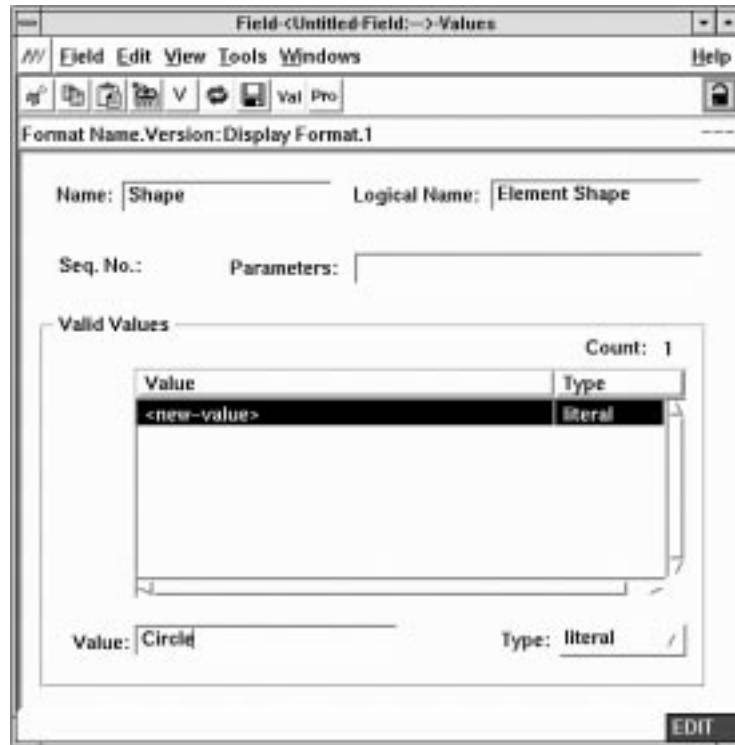


Figure 2-14. Entering a New Value

6. Execute **Field->New Value** to create a new Value row for the second Value.
7. Enter the second Value for example, *Square*, and press the **Return** key.

- To save the Value object, execute

**Field->Save**

Figure 2-15 shows the completed Field object.



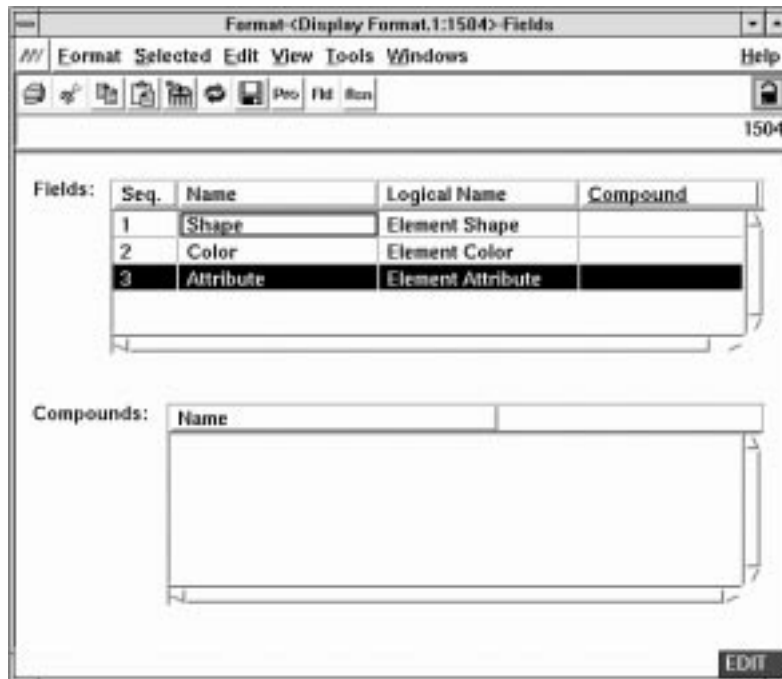
**Figure 2-15.** Completed Shape Field Object

- To close the Value object, execute

**Field->Close**



- Repeat Steps 2 through 9 to create the *Color* and *Attribute* Fields listed in Table 2-1. The completed Format object Fields view will be similar to the one in Figure 2-16.



**Figure 2-16.** Completed Format Object Fields View

- To save the Format object, execute  
**Format->Save**
- Proceed to Section 2.7 to create a Test Hierarchy and Test object for the test case.

**NOTE** — See Section 7 for detailed information on Field objects.

## 2.7 Create a Test Hierarchy and Test Object

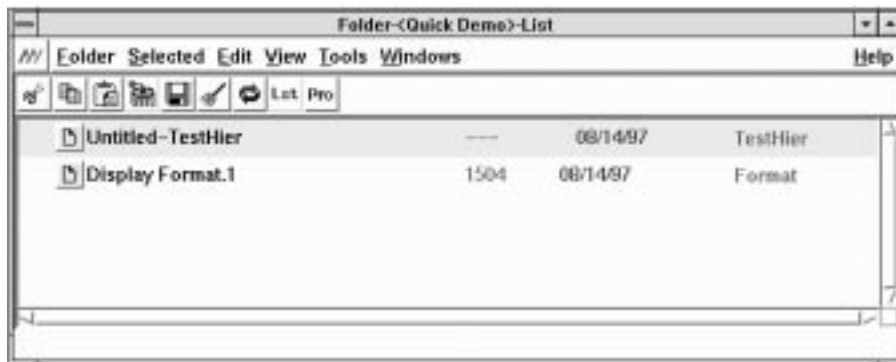
Test objects define the requirements used to verify or test a Format. Test objects always appear in their own **Test Hierarchies**, which are the equivalent of Folders.

To create a Test Hierarchy and Test object

1. In the *Quick Demo* folder, execute

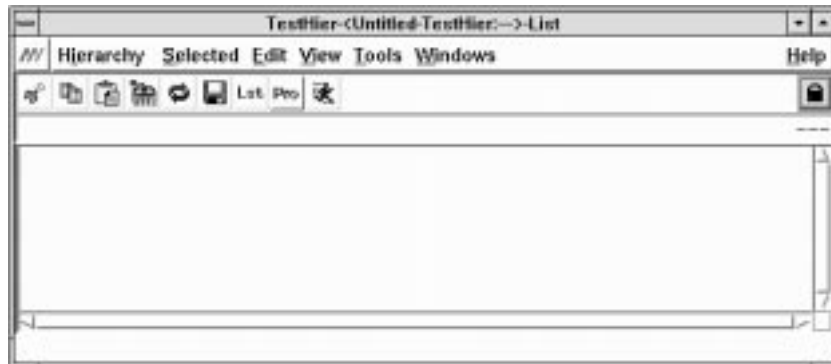
### Folder->New->Test Hierarchy

A new untitled Test Hierarchy item appears in the Folder. (See [Figure 2-17.](#))



**Figure 2-17.** A New Untitled Test Hierarchy

2. Open the Test Hierarchy by double clicking on the Test Hierarchy icon. An empty Test Hierarchy ([Figure 2-18](#)) appears.



**Figure 2-18.** An Empty Test Hierarchy

3. Unlock the Test Hierarchy. A **Test Hierarchy Name** dialog appears (Figure 2-19), prompting you to enter a name for the Test Hierarchy, for example, *Display Test Hierarchy*.

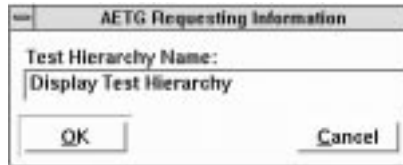


Figure 2-19. AETG Test Hierarchy Name Dialog

4. Click on **OK**.
5. Execute

**Hierarchy->New Test**

A new untitled Test object (Figure 2-20) appears in the Hierarchy.

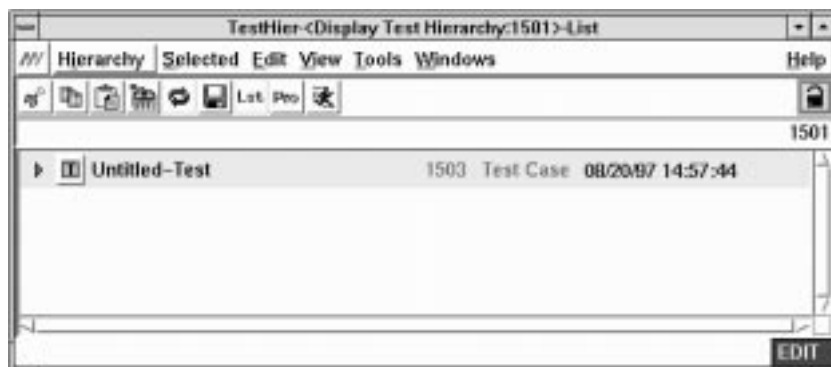


Figure 2-20. A New Test Object

6. Execute

**Hierarchy->Save**

**NOTE** — You must save a Test Hierarchy before you can edit a Test object.

7. Double click on the untitled Test object icon. The Test object opens, displaying the Test object Properties view. (See [Figure 2-21](#).)



**Figure 2-21.** Test Object Properties View

8. Unlock the Test object.

9. Enter, at a minimum, a name for the Test object, for example, *Display Test*.  
[Figure 2-22](#) shows an edited Properties view, including a description for the Test object.



**Figure 2-22.** Edited Test Object Properties View

10. Proceed to [Section 2.8](#) to associate the Test object with the *Display Format* Format object.

**NOTE** — See [Section 9](#) for detailed information on Test Hierarchies and objects.

## 2.8 Create an Association

To create a test case, you must first **associate** a Format object with a Test object, which creates a link between the Format object and the Test object. Associating a Format object with a Test object lets the Test object know what Fields and Values are available for the test case.

**NOTE** — A Test object can only be associated with one Format object at a time. However, a Format object can be associated with more than one Test object at a time.

You will

1. Use the AETG Database Browser to display the Format objects defined in the AETG database.

**NOTE** — The AETG System Database Browser provides a quick and easy way of locating objects in the AETG Database. See [Section 5](#) for information on using the Database Browser.

2. Select and copy the *Display Format* Format object you created in [Section 2.5](#).
3. Associate the Format object with the Test object.

To associate a Format object with a Test object

1. Display the Test object's Relations view. You can do this in one of two ways:
  - Click on the **Rel** button on the Format object Tool Bar.
  - Execute

**View->Change View->Relations**

The Relations view (Figure 2-23) appears.

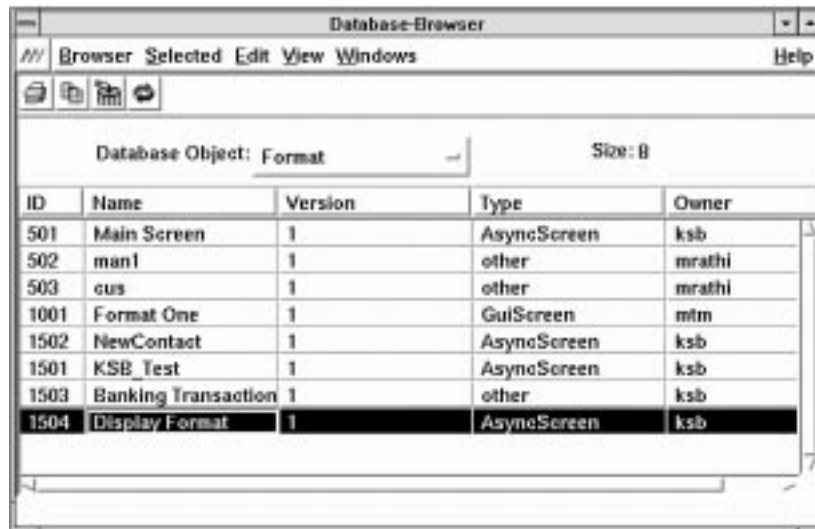


Figure 2-23. Test Object Relations View

2. Execute

**Tools->Database Browser**

The Database Browser (Figure 2-24) appears. By default, the first time you open the Database Browser during an AETG session, the Browser displays the Format objects defined in the AETG database.



**Figure 2-24.** AETG Database Browser

3. Select the *Display Format* entry, and execute

**Edit->Copy**

**NOTE** — You can also select and copy the *Display Format* Format object from on the AETG Desktop.

4. Close the Database Browser by executing

**Browser->Close**

**NOTE** — See [Section 5](#) for detailed information on the Database Browser.



5. On the unlocked *Display Test* Test object Relations view, click in the **Format** box (at the top of the view), and execute

**Edit->Paste**

A listing for the *Display Format* object appears in the **Format** box. (See [Figure 2-25](#).)



**Figure 2-25.** Associating a Format Object with a Test Object

6. Proceed to [Section 2.9](#) to create a Relation object for the *Display Test* Test object.

## 2.9 Create a Relation Object

Relation objects let you collect requirements data to establish how fields and values on the format relate to one and another.

To create a Relation object

1. On the unlocked *Display Test* Test object's Relation view, execute **Test->New->Relation**

A new unlocked Relation object (Figure 2-26) appears.

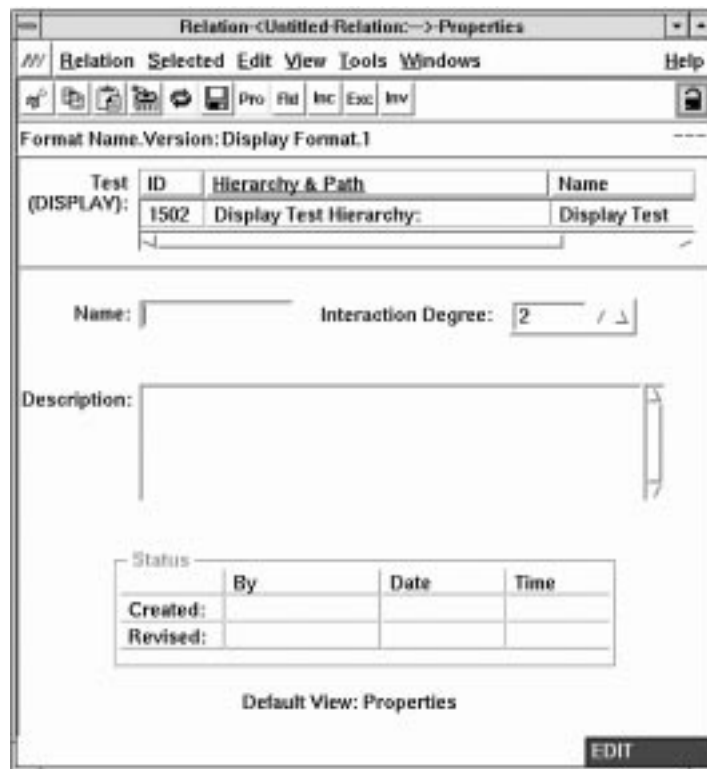


Figure 2-26. New Relation Object Properties View

2. Enter, at the minimum, a name for the Relation object in the **Name** text area, for example, *Display Rel.* Figure 2-27 shows an edited Properties view, including a description for the Relation object.



**Figure 2-27.** Edited Relation Object Properties View

**NOTE** — For now you will accept the default Interaction Degree value, 2. You will change this value in [Section 2.11](#).

3. Execute  
**Relation->Save**
4. Proceed to [Section 2.10](#) to generate the Test Case Matrix for this generation.

**NOTE** — See [Section 10](#) for detailed information on Relation objects.

## 2.10 Generate a Test Matrix

Now that you have created a Relation, you can generate the Test Case Matrix, which is the list of the generated set of efficient test cases.

To generate the Test Case Matrix

1. Display the *Display Test* Test object.

**NOTE** — If you can not see this object, you can display it using the **Windows** menu. (See [Section 3.1.3.7.](#))

2. Display the Test object's Matrix view. You can do this in one of two ways:

- Click on the **Mat** button on the Test object Tool Bar.
- Execute

**View->Change View->Matrix**

The Matrix view ([Figure 2-28](#)) appears.



**Figure 2-28.** Test Object Matrix View

3. Execute

**Test->Matrix->Generate**

The AETG System generates the Test Case Matrix based on the criteria you set on the Relation object. (See [Figure 2-29](#).)

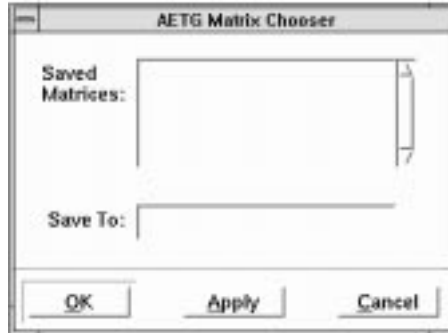


**Figure 2-29.** Generated Test Case Matrix

4. To save the Test Case Matrix, execute

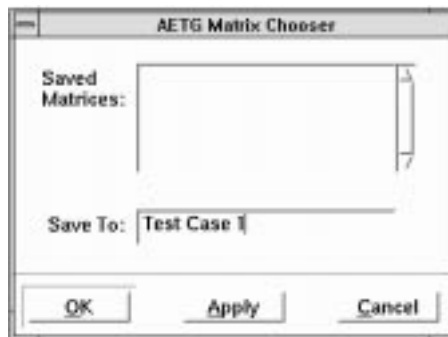
**Test->Matrix->Save**

The Save Matrix Dialog (Figure 2-30) appears.



**Figure 2-30.** Save Matrix Dialog

5. Enter a name for the Test Case Matrix, for example, *Test Case 1*, in the **Save To** text area (Figure 2-31), and click on **OK**.



**Figure 2-31.** Entering a Name for the Test Case Matrix

- The Save Matrix Dialog disappears, and the name is entered on the Matrix view. (See [Figure 2-32](#).)



**Figure 2-32.** Saved Matrix View

- Proceed to [Section 2.11](#) to change the interaction degree.

**NOTE** — See [Section 9.3.3](#) for detailed information on generating Test Case Matrices.

## 2.11 Change the Interaction Degree

The Interaction Degree, which you set on the Relation object Properties view (Figure 2-26), determines how the fields in the generated test cases interact with each other. The interaction degree is restricted to an integer from  $1$  to  $N$ , where  $N$  is the number of Fields in the Relation.

For example, a degree of  $2$  guarantees that, for any two fields, all possible value combinations are covered in the set of generated test cases.

**NOTE** — See Section 10.2.1 for more information on the Interaction Degree.

For this Quick Tour, you will change the Interaction Degree to  $3$ , the maximum value for this Relation. This creates all possible permutations for the defined Fields and Values.

To change the Interaction Degree

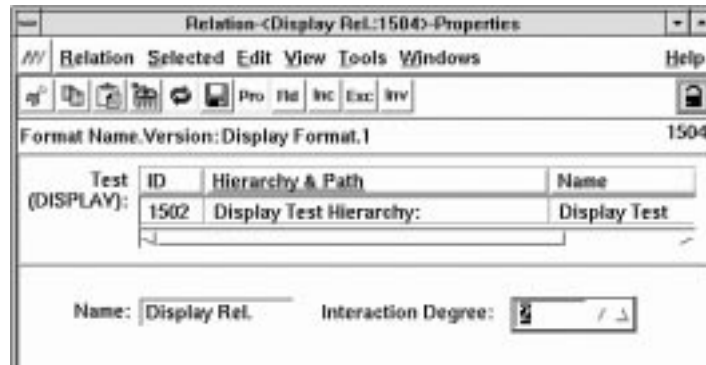
1. Display the *Display Rel.* Relation object's Properties view, and unlock the object. (See Figure 2-33.)



Figure 2-33. Unlocked Relation Object Properties View

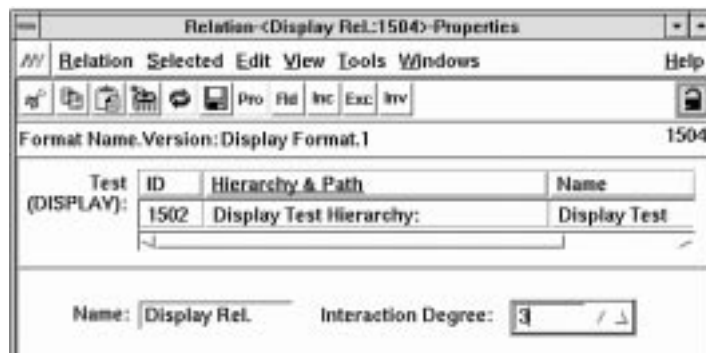


2. Select the current Interaction Degree value, 2, in the **Interaction Degree** text area.  
(See [Figure 2-34](#).)



**Figure 2-34.** Selecting an Interaction Degree to Change It

3. Type 3. (See [Figure 2-35](#).)



**Figure 2-35.** Specifying a new Interaction Degree

4. Execute  
**Relation->Save**
5. Display the *Display Test* Test object's Matrix view.

6. Execute

**Test->Matrix->Generate**

The AETG System regenerates the Test Case Matrix using the new Interaction Degree. (See [Figure 2-36](#).)

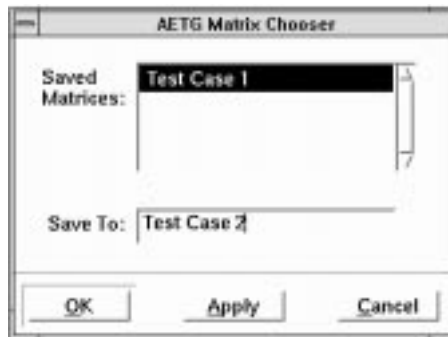


**Figure 2-36.** Generated Test Case Matrix for new Interaction Degree

7. Execute

**Test->Matrix->Save**

8. Enter a new test case name in the Save Matrix Dialog, for example, *Test Case 2*, (Figure 2-37) and click on **OK**.



**Figure 2-37.** Saving the new Test Case Matrix

9. Proceed to [Section 2.12](#) to create an Exclude.

## 2.12 Create an Excluded Test Case

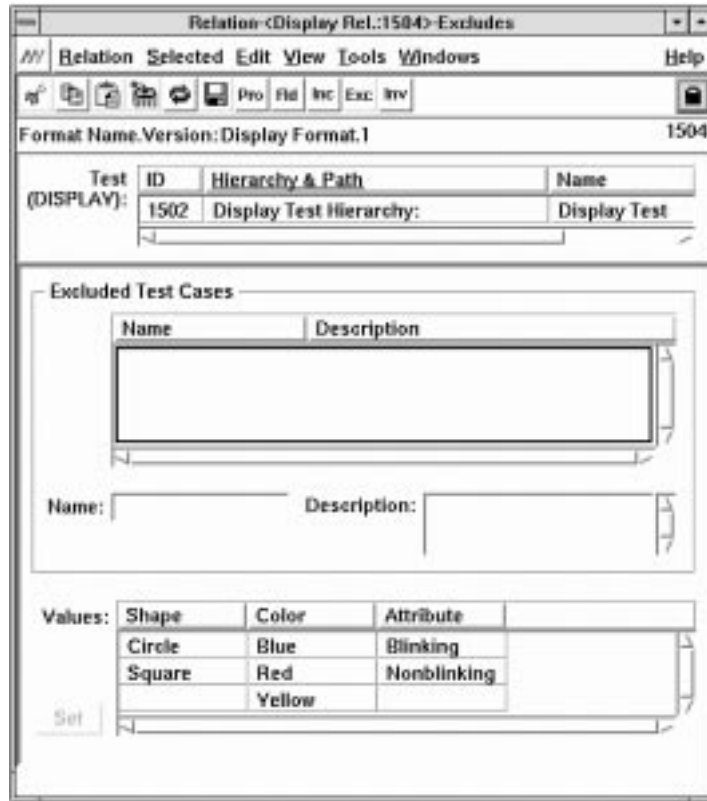
So far, you have used the AETG System to create test cases that generate all permutations for the defined Fields and Values. Often, however, there may be permutations you don't want to include in a Test Case Matrix. Conversely, there are permutations that you want to ensure are included. Further still, you may want generate test cases containing invalid values to do error condition testing. The AETG System lets you define such rules.

For this Quick Tour, you will create an Excluded Test Case that omits all Yellow Squares from the Test Case Matrix.

To create an Excluded Test Case

1. Display the *Display Rel.* Relation object.
2. Display the Relation object Excludes view. You can do this in one of two ways:
  - Click on the **Exc** button on the Relation object Tool Bar.
  - Execute  
**View->Change View->Excludes**

The Excludes view (Figure 2-38) appears.



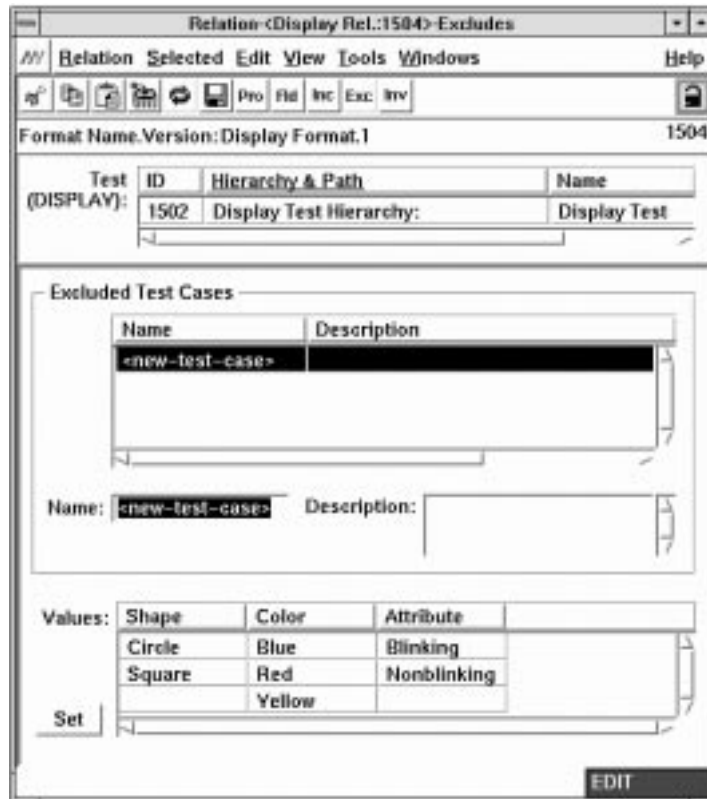
**Figure 2-38.** Relation Object Excludes View

3. Unlock the Relation object Excludes view.

4. Execute

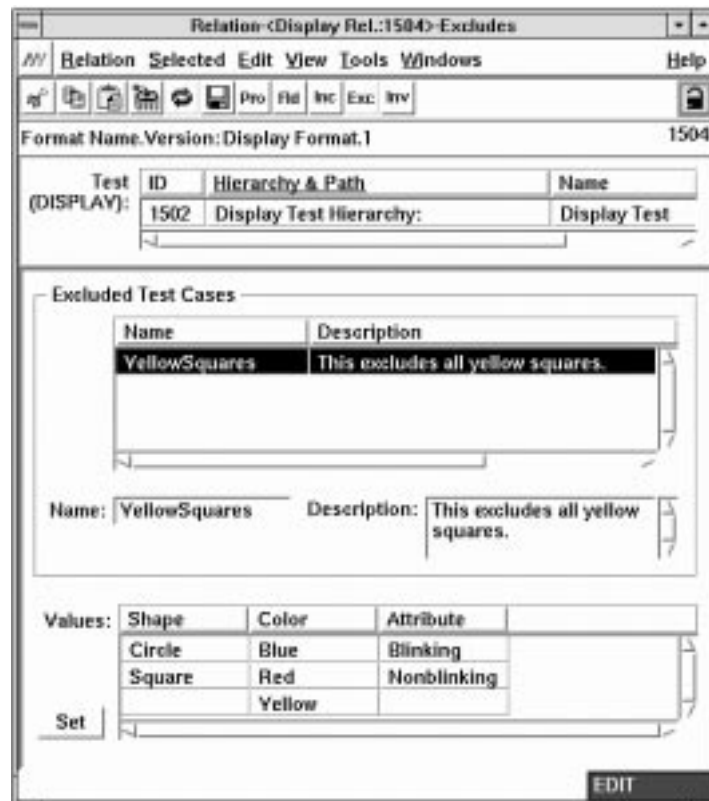
**Relation->New->Exclude**

The **Name** text area is populated with *<new-test-case>*, as is the **Name** field in the **Excluded Test Cases** list. (See [Figure 2-39](#).)



**Figure 2-39.** Creating a new Excluded Test Case

5. Enter a name for the Excluded Test Case, for example, *YellowSquares*, and, optionally, a description of the Excluded Test Case. (See [Figure 2-40](#).)



**Figure 2-40.** Specifying an Excluded Test Case Name

**NOTE** — You cannot include embedded spaces in an Excluded Test Case name.

6. Select the values for the Excluded Test Case in the **Values** list (at the bottom of the view).
  - A. Click on the *Square* entry in the **Shape** column.
  - B. Hold down the **Control** key, and click on the *Yellow* entry in the **Color** column.The Excludes view will be similar to the one in [Figure 2-41](#).

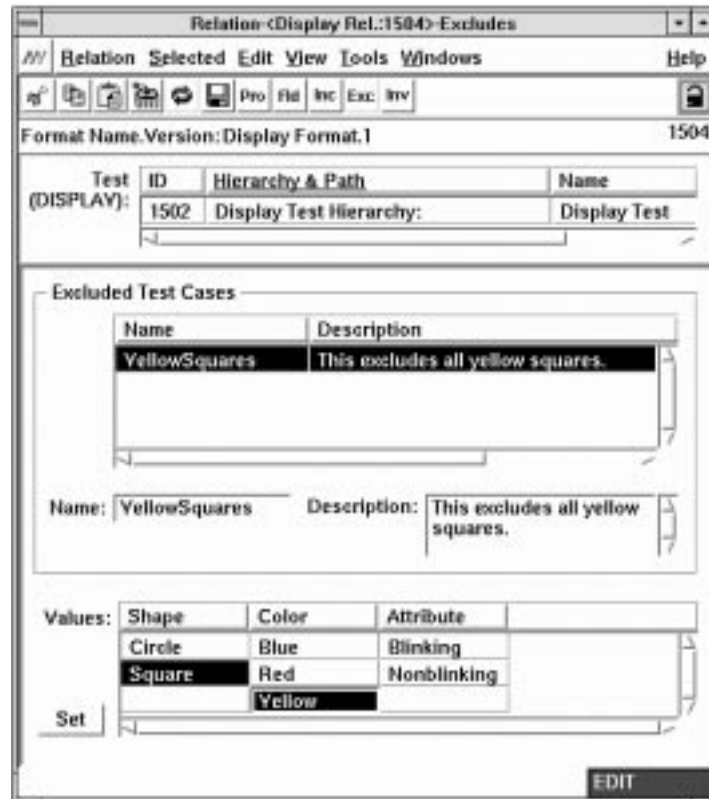
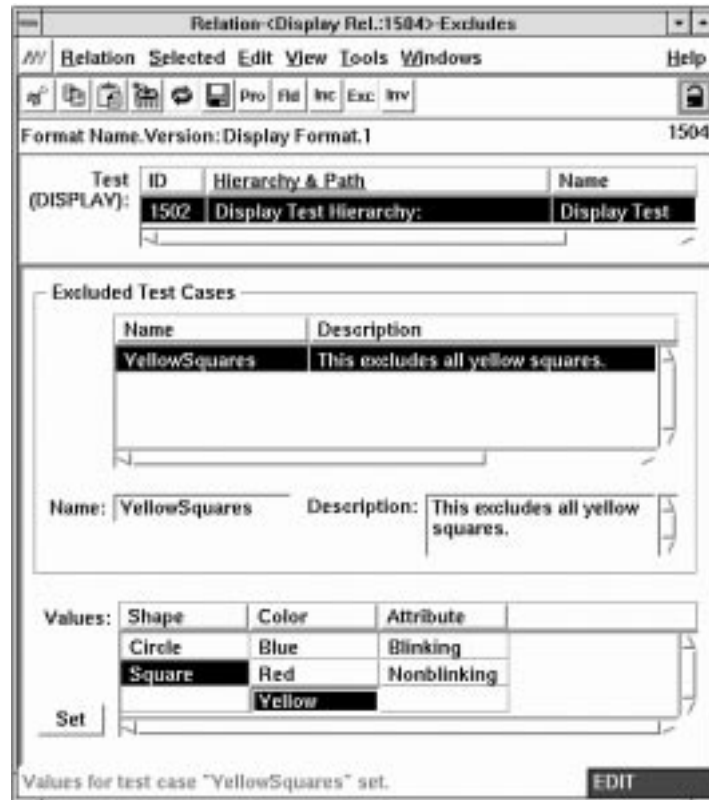


Figure 2-41. Selecting Values for an Excluded Test Case

- Click on the **Set** button. This applies the selected values to the Excluded Test Case. The system displays a message that the values have been set for the test case. (See [Figure 2-42.](#))



**Figure 2-42.** Setting Values for an Excluded Test Case

- Execute  
**Relation->Save**
- Execute  
**Relation->Close**
- Display the *Display Test* Test object's Matrix view.



11. Execute

**Test->Matrix->Generate**

The AETG System regenerates the Test Case Matrix using the Excluded Test Case.  
(See [Figure 2-36](#).)

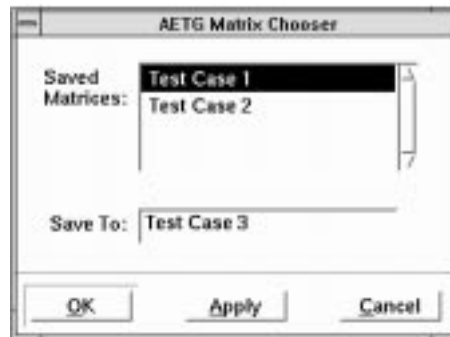


**Figure 2-43.** Excluded Test Case Test Case Matrix

12. Execute

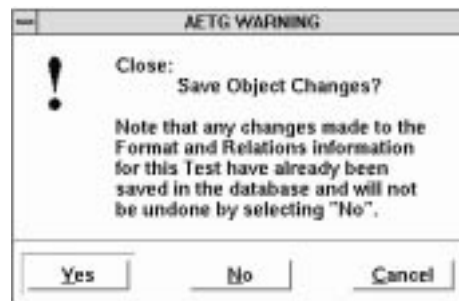
**Test->Matrix->Save**

13. Enter a new test case name in the Save Matrix Dialog, for example, *Test Case 3*, (Figure 2-37) and click on **OK**.



**Figure 2-44.** Saving the Excluded Test Case Matrix

14. Execute  
**Test->Close**
15. The AETG System displays an alert dialog (Figure 2-45) asking you if you want to save your changes. Click on **OK**.



**Figure 2-45.** Save Changes Dialog

**NOTE** — See [Section 10.2.4](#) for information on creating Excluded Test Cases.

## 2.13 Exiting From the AETG System

If you wish to exit from the AETG System, execute

**Aetg->Exit**

## 2.14 What's Next

This Quick Tour should have given you a general feeling for the AETG System and what it can do for you.

The remainder of this Users Guide provides detailed information on how to use the AETG System. [Table 2-2](#) list the sections in this document that contain information on how to use the AETG System.

**Table 2-2.** AETG System Users Guide Sections

<b>Section Number</b>	<b>Section Name</b>	<b>Description</b>
<a href="#">Section 3</a>	<a href="#">AETG Basics</a>	This section contains basic information on using the AETG GUI.
<a href="#">Section 4</a>	<a href="#">Using AETG Objects</a>	This section contains information on how to use AETG objects
<a href="#">Section 5</a>	<a href="#">Using the Database Browser</a>	This section contains information on the AETG Database Browser.
<a href="#">Section 6</a>	<a href="#">Format Object</a>	This section contains information on Format objects, which describe a testable interface to an application or a testing situation.
<a href="#">Section 7</a>	<a href="#">Field Object</a>	This section contains information on Field objects, which define the data values that are input to an application.
<a href="#">Section 8</a>	<a href="#">Compound Object</a>	This section contains information on Compound objects, which let you collect several Fields and treat them as one Field.
<a href="#">Section 9</a>	<a href="#">Test Object</a>	This section contains information on Test objects, which define the means by which a requirement is verified in a software application.  This section also contains information on how to generate a Test Case Matrix.
<a href="#">Section 10</a>	<a href="#">Relation Object</a>	This section contains information on Relation objects, which let you collect requirements data to establish how fields and values on the format relate to one another.
<a href="#">Section 11</a>	<a href="#">Command Line Tools</a>	This section contains information on the AETG Command Line tools.
<a href="#">Section 12</a>	<a href="#">Input Modeling with the AETG System</a>	This section provides tips on how the AETG System could be used to model various types of applications.



### 3. AETG Basics

Although the GUI is easy to use, we will explain the basics of its operation in some detail here to help you enjoy the power of the AETG System.

This section covers:

- What folders are.
- The layout and operation of the system's major windows and dialog boxes.
- Basic actions needed to navigate around the AETG System.
- Controls used in the AETG GUI.
- How to perform some routine tasks.

#### 3.1 AETG Folders

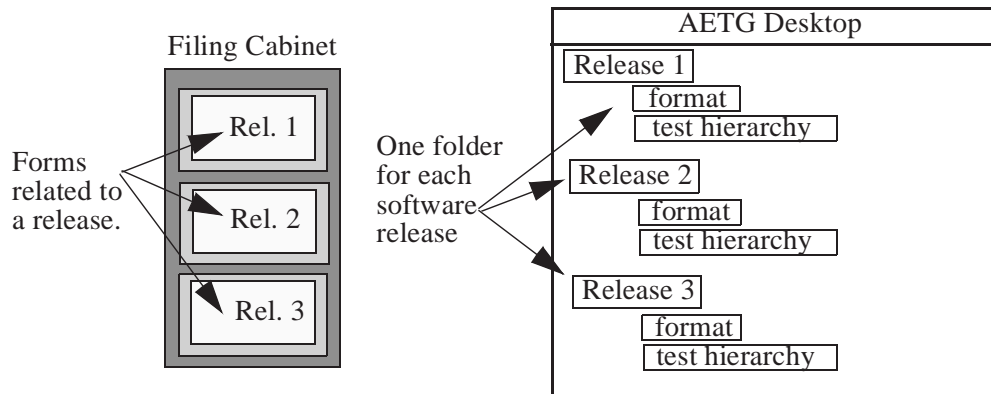
Folders are provided for your convenience so that you can organize your work. You can think of folders as drawers in a filing cabinet. For example, suppose you have a filing cabinet with three drawers.

- The first drawer contained documents related to Release 1 of a software product.
- The second contained documents related to Release 2 of a software product.
- The third drawer contained documents related to Release 3 of a software product.

In the AETG System, the filing cabinet itself would be the AETG Desktop. Within the AETG Desktop there could be other folders that correspond to drawers in the filing cabinet.

- The first folder could contain objects related to Release 1.
- The second could contain objects related to Release 2.
- The third could contain objects related to Release 3.

Figure 3-1 illustrates this idea.



**Figure 3-1.** What Are Folders?

You can create folders when you need them and delete them when you no longer need them. Folders can appear in the AETG desktop. You can move objects in and out of them by copying the object from one folder to another.

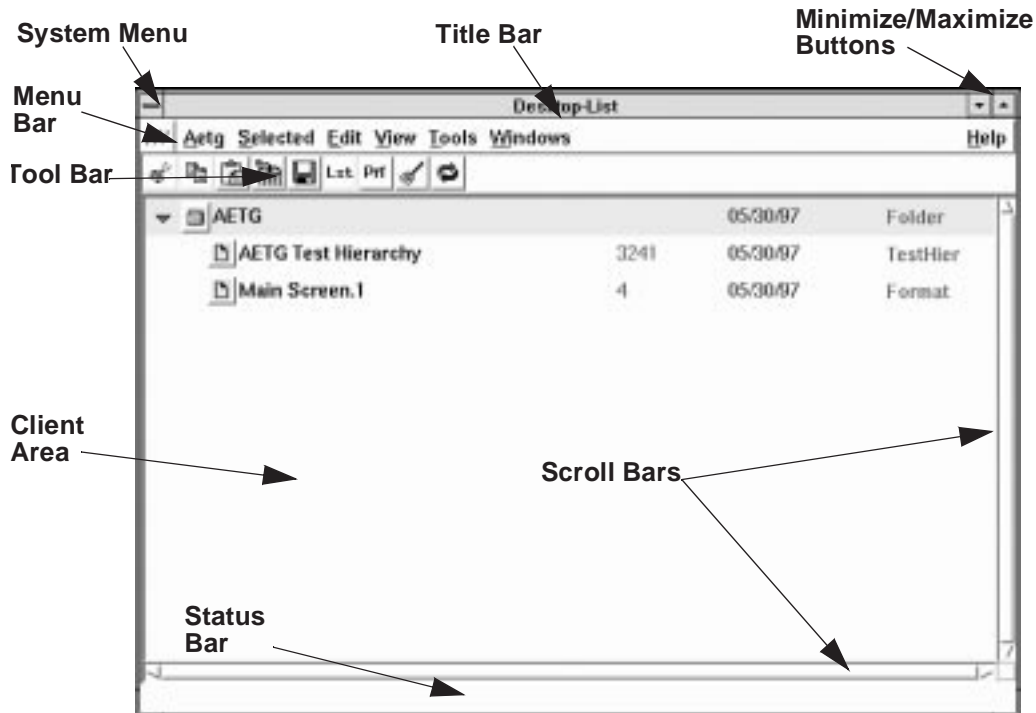
The AETG Desktop is the first folder you encounter when you start the AETG System and serves as the main entry point to the system. (See [Figure 3-2](#).) Unlike all other folders, the AETG Desktop cannot be created or deleted, but you can create and delete other folders within it.

### 3.1.1 What is the AETG Desktop?

You can think of the AETG Desktop as you would the top of your desk. It is a work space where you can collect all the objects and folders you will need for testing.

The top of your desk contains folders and tools, such as pens, a telephone, and a computer, which help you do your job. The AETG desktop also has folders and tools that help you create your test cases.

In this chapter we will discuss some of these tools. In [Section 4](#) we will discuss more of them.



**Figure 3-2.** AETG Desktop List View

The AETG Desktop we have depicted above shows the List view. We will explain more about views in [Section 3.5](#).

The basic window layout is the same for all folders as it is for the AETG Desktop. The layout we describe here applies to all other windows.

### 3.1.2 Title Bar

The top line of the window, called the Title Bar, displays the name of the GUI window. In our example, the name of the folder, **Desktop-List**, appears here.

**NOTE** — The AETG System was designed to use the Motif™ window manager, but will work with any X-windows manager. What elements you see on the Title bar will depend on which window manager you use.

When using the Motif window manager, the left-hand side of the **Title Bar** contains the **System Menu**, represented by a small bar, which operates the window itself. When you

click on the bar, the window manager displays the **System** Menu that, depending on window manager, has the following menu selections:

<b>Restore</b>	Restores the window to its original size after you have re-sized it. This is not the maximized size, but the size the window was originally displayed by the window manger.
<b>Move</b>	Lets you move the window using the arrow keys or by dragging it.
<b>Size</b>	Lets you re-size the window using the arrow keys or mouse.
<b>Maximize</b>	Increases the window size to the maximum allowed by the window manager.
<b>Minimize</b>	Reduces the window to an icon.
<b>Lower</b>	Moves the window to the bottom of the window hierarchy.
<b>Close</b>	Closes the window.

On the right-hand side of the **Title Bar** are two buttons. The smaller one on the left minimizes the window, i.e., turns it into an icon; the larger one on the right maximizes the window, i.e., increases the window size to the maximum allowed by the window manager.

### 3.1.3 Menu Bar

The next line in the window is called the **Menu Bar**. The menu bar contains all the menus for the window you currently have open. Many of the activities you will do to perform operations will begin from the menu bar.

The **Aetg**, **Selected**, **Edit**, **View**, **Tools**, **Windows**, and **Help** menus appear on the AETG Desktop. The name of the first menu on a folder varies with the name of the window. So, for example, the AETG Desktop has **Aetg** as the first menu. The **Selected**, **Edit**, **View**, **Tools**, **Windows**, and **Help** menus appear on all windows although the menu selections available on these menus sometimes vary.

#### 3.1.3.1 AETG Icon

An AETG System icon ([Figure 3-3](#)) appears to the left of the menus.



**Figure 3-3.** AETG Icon

If you click on the AETG icon, the system displays the AETG Banner window.



We describe the AETG Desktop menus and their selections in some detail here. The following sections of this Users Guide discuss variations in menu selections where appropriate.

### 3.1.3.2 Aetg Menu

The menu selections for the **Aetg** Menu are

<b>Save</b>	Saves the contents of the AETG Desktop, any preferences you set with the AETG Preferences view and any queries defined for the Database Browser.
<b>New</b>	Lets you create a new AETG object.
<b>Iconify All Windows</b>	Reduces all open windows to icons.
<b>Exit</b>	Closes the main folder, any open folders, and exits to the UNIX prompt.

#### 3.1.3.2.1 *What Happens When You Save a Folder*

To save a folder, click on the **Save** option of the first menu that appears on the left-hand side of the Menu bar. The system saves the contents of the desktop or folder.

**NOTE** — Remember, the AETG Desktop is a specialized folder.

The system saves the organization of folders and any preferences you selected to the *.xmyMYNAHrc* file. This file is located in your home directory and is read by the system when you start-up the system. Every time you **Save**, the system copies the *.xmyMYNAHrc* file to a *.xmyMYNAHrc.bak* file and then saves to the *.xmyMYNAHrc* file.

While you are running the AETG System, it saves the contents of folders, the desktop, and the preferences you selected every 60 seconds to a file named *.xmyMYNAH.rc<hostname>.<process.id>*. The system uses this file as a backup in the unlikely event of a system crash. This ensures that you will never lose more than sixty seconds of desktop work. This file is removed during normal exits, but if there is a crash, the system will ask you if you want to open it as the crash recovery file.

### 3.1.3.3 Selected Menu

The **Selected** Menu offers you actions that apply only to AETG objects you have selected and to create new objects. Selections include:

- New** Displays a cascade menu that lets you create new AETG objects on the AETG Desktop or in a folder. (See [Section 4](#) for a detailed explanation of AETG objects.)
- Open** Lets you open a selected object.
- Delete** Deletes a selected object from the desktop and database.
- Expand Fully** Displays the complete contents (objects) of folder or hierarchies, i.e., displays all parent and child objects contained in the desktop or a folder.

### 3.1.3.4 Edit Menu

The **Edit** menu provides you with a number of editing features. Selections include:

- Undo** Cancels the previous change to a text field.
- Cut** Removes the selected item from the display and temporarily places it on a clipboard.
- Copy** Places a copy of a selected item on a clipboard.
- Paste** Retrieves an item from the clipboard and places it in the client area of folder.
- Clear** Removes the selected item from the display, but does not place it on the clipboard.
- Select All** Places all visible items in a selected state.
- Deselect All** Deselects all visible items.

### 3.1.3.5 View Menu

The **View** Menu lets you change to another view or refresh the AETG Desktop display. Selections are:

- Change View** Displays a cascade menu that lets you change the view for the currently opened object.
- Refresh** For the Desktop, refreshes the current view by removing objects from the display that are no longer in the database; for Folders and objects, gets the

latest copy of object from the database; and for the **Database Browser**, reissues any defined or default queries.

#### 3.1.3.6 Tools Menu

The **Tools** menu lets you display the **Database Browser**, which lets you search for objects in the AETG database. You can open or run objects directly from the Database Browser or copy them to a folder. See [Section 4](#) for information on the Database Browser.

#### 3.1.3.7 Windows Menu

The **Windows** menu helps you to manage the windows associated with the AETG System. Selections include all the currently opened windows even if they are iconified. Selecting one of the window names from the list brings that window to the top of your display.

#### 3.1.3.8 Help Menu

The final menu on the AETG Desktop window is the **Help** menu. The selections are:











<b>Contents</b>	Displays a list of help topics. (Available in future releases.)
<b>Procedures</b>	Displays a list of hypertext links to help on tasks and activities. (Available in future releases.)
<b>Keyboard</b>	Lists key accelerators and their uses.
<b>On Help</b>	Displays help messages on the help system. (Available in future releases.)
<b>On Icons</b>	Displays the icons that appear on the AETG Desktop, folders and Tool Bar, and gives a brief description of each.
<b>On Version</b>	Displays product information.

### 3.1.4 Tool Bar

As you can see in [Figure 3-2](#), icons appear directly below the Menu Bar in what we call the **Tool Bar**. These icons represent often used functions that correspond to menu selections. We placed them here for your convenience. The features available on the Tool Bar change depending on the window. [Table 3-1](#) lists the tools available with the AETG Desktop along

with a brief description of each tool. We will describe other tools when we discuss the object view on which they appear.

**Table 3-1. Tool Bar Icons and Functions**

Icon	Function	What It Does
	print	Causes the system to display the Print dialog with which you can print the contents of objects or folders.
	cut	Removes the selected item and temporarily places it on a clipboard. This does not remove items from the database.
	copy	Places a copy of a selected item on the clipboard.
	paste	Retrieves an item from the clipboard and places it in the client area of folder.
	delete	Deletes a selected item from the database and clears it from the display.
	Save	Saves the contents of the AETG Desktop, any preferences you set with the AETG Preferences view and any queries defined for the Database Browser.
	change view	Changes the current view to the view represented by the icon, e.g. <b>Lst</b> changes view to the List view. Icons vary with the views available for an item.
	clear	Removes the selected item from the desktop, but does not place it on the clipboard.
	refresh	Refreshes the AETG Desktop view.
	lock	Unlocks the AETG item, making it ready for input or editing.

You use the Tool Bar by clicking on the icon that corresponds to the function you want. Depending on the icon you selected, the AETG System will either perform the action (e.g., **Cut, Copy, Paste, Change View**) or display a dialog with which to perform the function (e.g., **Print**).

### 3.1.5 Client Area

The central part of the AETG Desktop window between the Toolbar and Status box is called the client area. The client area is where you will perform most of the operations needed for test and script development and execution. The AETG System displays the contents of GUI objects in this area.

### 3.1.6 Status Bar

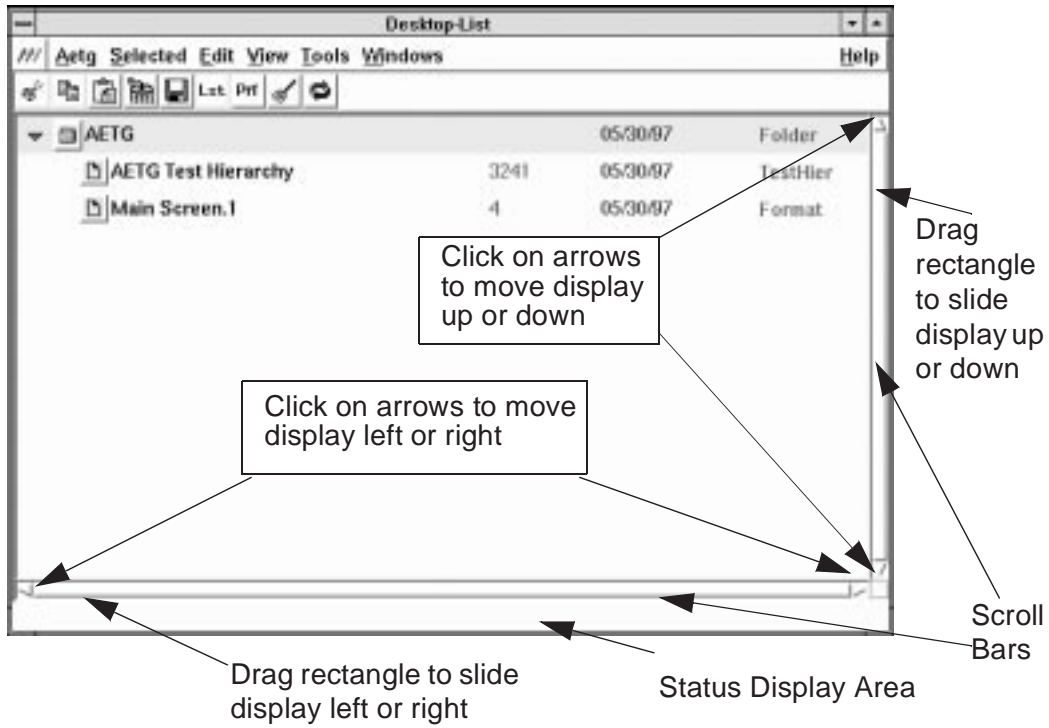
The bottom line of the AETG Desktop is the Status Bar that displays text messages on the left-hand side. Two areas appear on the right-hand side of the Status Bar: one on objects that displays the cursor location and the other that displays the current mode the AETG System is in. Mode refers to whether the object view is locked and inaccessible for editing or unlocked and ready for editing. When you unlock an object, the word EDIT will appear in the Status Bar.

### 3.1.7 Scroll Bars

The AETG Desktop has two scroll bars. One appears along the right-hand side of the window while the other appears directly below the client area. The scroll bar on the right moves the client area display up or down. The scroll bar at the bottom moves the client area display left or right.

These scroll bars are particularly useful since the AETG display can have a virtual display area, i.e., text is present that appears beyond the visible display. Scroll bars let you see this text.

Figure 3-4 shows how to use scroll bars.



**Figure 3-4.** Using Scroll Bars

## 3.2 AETG Dialogs

The AETG System uses a number of Dialogs (often called dialog boxes) to help you perform test and task-automation activities. In general, dialogs are windows accessed from a folder that presents you with a number of controls with which to perform actions supporting the function of the folder from which they were called.

The AETG System uses three basic types of dialogs:

- Dialogs that let you complete or control operations you initiated from a folder or object view.
- Dialogs that let you set the properties of objects.
- Dialogs that present system information to you such as error messages.

We will describe other dialogs where appropriate in the sections that follow.

## 3.3 Navigating Around the AETG GUI

The AETG GUI provides you with a number of controls to help you move through windows and dialog boxes and perform operations. Generally, the way you get from one folder to another is by making a menu selection or activating an icon.

### 3.3.1 Using the Mouse

Table 3-2 defines several terms used when describing mouse actions throughout this manual. All of these terms refer to action you take with the **left mouse button**. You do not use the **right mouse button** for any AETG specific operations.

**Table 3-2. Mouse Actions**

Action	Description
Click	Press and release the mouse button.
Double Click	Press and release the mouse button twice.
Drag	Place the pointer on an object or menu. Press the mouse button and hold it while moving the pointer. Release the mouse button when the action you want is accomplished.

### 3.3.2 Using Icons

Icons are graphical representations of objects and folders that appear in the client area. They also represent tools in the Toolbar. We have already discussed the icons in the tool ribbon and how to use them. (See [Section 3.1.4.](#)) Here we will describe the icons that appear in the client area. Folders and other objects can be reduced to an icon when you are not using them. [Figure 3-5](#) shows a folder reduced to an icon.



**Figure 3-5.** AETG Icon

An arrow appears next to an icon if the icon represents a folder that contains other objects. For example, if we had a folder that contained a number of other objects, an arrow would appear next to it as we have illustrated. The peak of the arrow points at the object when the object has not been expanded. We will explain how to use the arrow when we describe the List view ([Section 3.6](#)).

To transform an open object into an icon:

1. Click on the first menu
2. Click on the **C**lose menu selection.

To activate a window that has been iconified, double click on the icon that represents the object.

### 3.3.3 Making Menu Selections

You can select an item from a menu by clicking on the menu and then clicking on the menu selection you want. You can also use Mnemonic keys.

### 3.3.4 Using Mnemonic Keys

Mnemonic keys are individual keys you can use rather than the mouse to make menu selections. Letters that correspond to mnemonic keys appear underlined in the menu selection name. The **alt** key combined with a single keystroke can initiate menu selections. For example, you can open a selected object by typing **alt** and **S** and then **O**, e.g., S for Selected menu and **O** for Open.



### 3.3.5 Using Accelerator Keys

The AETG GUI provides accelerator keys. Accelerator keys are key combinations that you can use instead of mouse actions to make menu selections and perform other common actions. [Table 3-3](#) list the common accelerator keys for the AETG Desktop.

**Table 3-3.** Key Accelerators

Keys	Function
Ctrl+P	Print
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Ctrl+/	Select all items
Ctrl+\	Deselect all items
Ctrl+F4	Close
Del	Delete selected item
F5	Refresh
Esc	Deselect a menu.
Tab	Moves the cursor focus from one data entry field to the next.

There are other accelerator keys. These are listed next to the menu selections they perform.

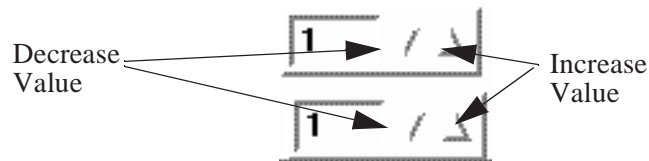
## 3.4 AETG GUI Controls

If you are familiar with common GUI controls, you can skip most of this section and go on to [Section 3.4.5](#).

There are a number of common controls you will have to use to accomplish tasks associated with testing. You may be familiar with many of these, but we describe them here for you in case you need a quick refresher course on GUIs.

### 3.4.1 Using Spin Buttons

Spin buttons are GUI control that helps you to make settings. You use spin buttons to set values when the values are an ordered exclusive set. [Figure 3-6](#) shows some typical spin buttons.



**Figure 3-6.** Spin Buttons

To set a value with a spin button, perform one of the following:

- Click on the up or down arrows that appear next to the Value display. The up arrow increases the value in the display area; the down arrow decreases the value.
- Position the pointer in the value display area and type in a value.

### 3.4.2 Using Toggle Buttons

Toggle buttons are simple on/off buttons. Toggle buttons can appear in a group or individually. [Figure 3-7](#) shows a toggle button.

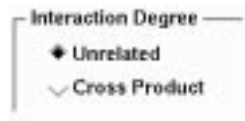


**Figure 3-7.** Toggle Buttons

To set a toggle button, click on it to set it on or off. When it is ON the button will be highlighted.

### 3.4.3 Using Radio Buttons

Radio buttons let you select one option from a group of options. They are exclusive settings; you can only select one from a group. [Figure 3-8](#) shows a group of radio buttons with **All Code** selected.



**Figure 3-8.** Radio Buttons

### 3.4.4 Using Pushbuttons

Pushbuttons are GUI controls used to execute actions that affect the entire window. They generally appear at the bottom of an AETG window or dialog. You activate a pushbutton by clicking on it. [Figure 3-9](#) shows typical pushbuttons arranged at the bottom of a window.



**Figure 3-9.** Pushbuttons

[Table 3-4](#) lists the pushbuttons used with the AETG System.

**Table 3-4.** Pushbuttons and Their Functions (Sheet 1 of 2)

PushButton	Function
<b>OK</b>	Acknowledges messages from the system and approves changes to settings. Usually, this button also closes the window.
<b>Apply</b>	Commits the current setting on a window without closing the window.
<b>Reset</b>	Cancels changes made to the window content and returns the window to the last approved settings.
<b>Stop</b>	Stops any ongoing processes.
<b>Continue</b>	Continues a process that has been interrupted.
<b>Retry</b>	Lets you retry a process interrupted by the operating environment.
<b>Pause</b>	Suspends an ongoing process without terminating it.

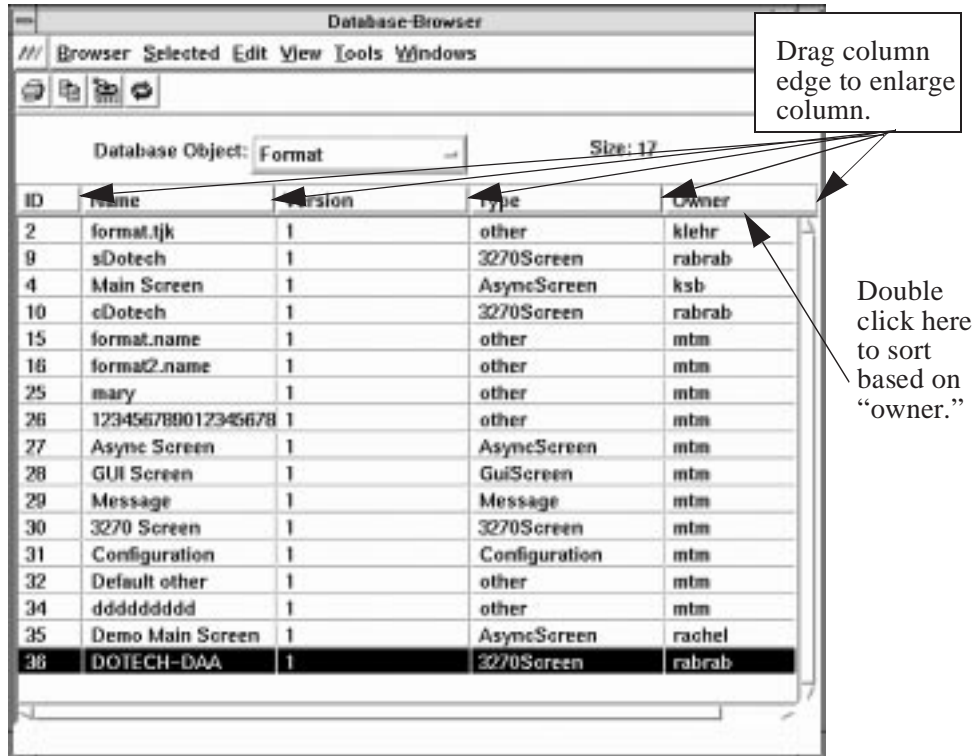
**Table 3-4.** Pushbuttons and Their Functions (Sheet 2 of 2)

<b>PushButton</b>	<b>Function</b>
<b>Resume</b>	Continues a process that was paused (See <b>Pause</b> above.)
<b>Close</b>	Closes a window without affecting a process.
<b>Cancel</b>	Closes a window without applying any changes to a window's content that have not been applied (See <b>Apply</b> above.)
<b>Yes</b>	Indicates a positive response to information contained in a message.
<b>No</b>	Indicates a negative response to information contained in a message.
<b>Defaults</b>	Use default values.
<b>Help</b>	Displays contextual help information for a window.

### 3.4.5 Using Ruler Column Headings

In many AETG views and dialogs we use what are called Ruler Column Headings. [Figure 3-10](#) shows a window from the **Database Browser** (See [Section 4](#)) in which all the column headings are rulers. If you position the pointer on a column edge and drag it, you

can make the column wider or narrower. This will be useful when the information displayed in the column exceeds the default column size.



**Figure 3-10.** Database Browser Displaying Format Objects

If you double click on a column heading in the ruler, the system will sort objects based on the attributes in that column. For example, if you wanted to arrange objects based on the owner of the object, you would double click on the **Owner** column heading. Figure 3-10 shows Script objects sorted in alphabetical order based on the "Owners" name.

### 3.5 What Are Views?

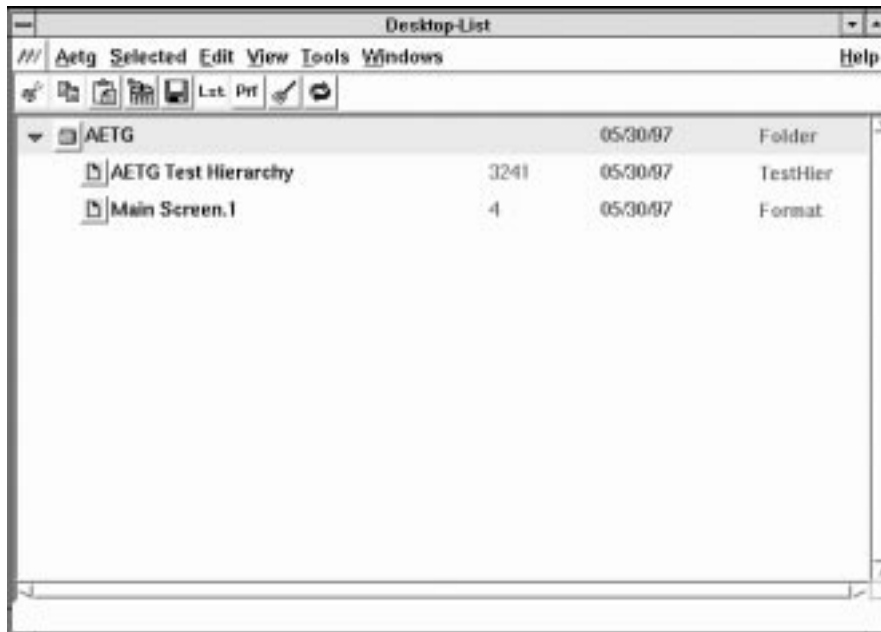
The AETG System uses what we call views. Views appear as a windows on your terminal screen when you open an AETG object. Views organize information about objects. Each view displays a certain type of information about an object.

Objects can have up to six different views. You can think of views as pages of information about an object or as screens in a traditional character based application. Views also let you enter information about objects and associate objects with other objects.

The AETG Desktop we displayed in [Figure 3-2](#) uses a List view to represent objects as line items with icons and text descriptions. This is one of three AETG Desktop views. The other two are Preferences and Default View views.

### 3.6 List View

List Views are very important in the sense that List views are where you begin with the AETG System and they are views to which you will return often. List views show you all the objects available in a folder. [Figure 3-11](#) shows a list view for the AETG desktop.



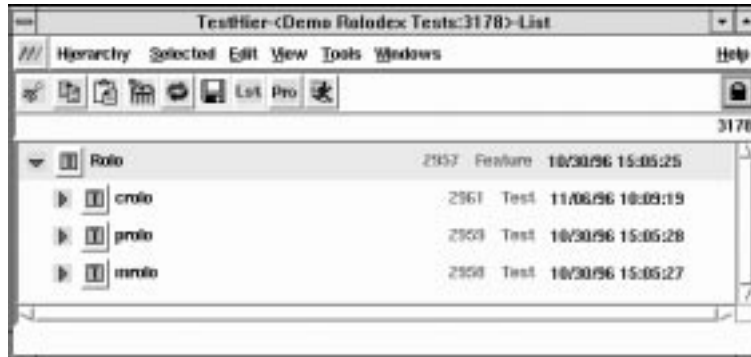
**Figure 3-11.** AETG Desktop - List View

The List view illustrated in [Figure 3-11](#) shows only the first level of objects in your object tree. But an object can be expanded to show all the child objects within it, which we will explain in the next subsection.

#### 3.6.1 Expanding Items in a List View

To expand an object's listing, click on the arrow next to the object's iconic representation. For example, if you clicked on the arrow for icon labelled **AETG** in [Figure 3-11](#), the view

would expand as shown in [Figure 3-12](#). Note that the arrow next to the icon points down to indicate that this is an expanded display of the object.



**Figure 3-12.** List View Expanded

Items that appear on the AETG Desktop may have several levels of items below them in a hierarchical relationship. When you click on the arrow next to a list item's icon, as we did above, the system shows only the first level of the hierarchy.

To see all the levels in a selected item's hierarchy, execute

**Selected->Expand Fully**

The system will display all the levels of a hierarchy, as shown in [Figure 3-13](#).



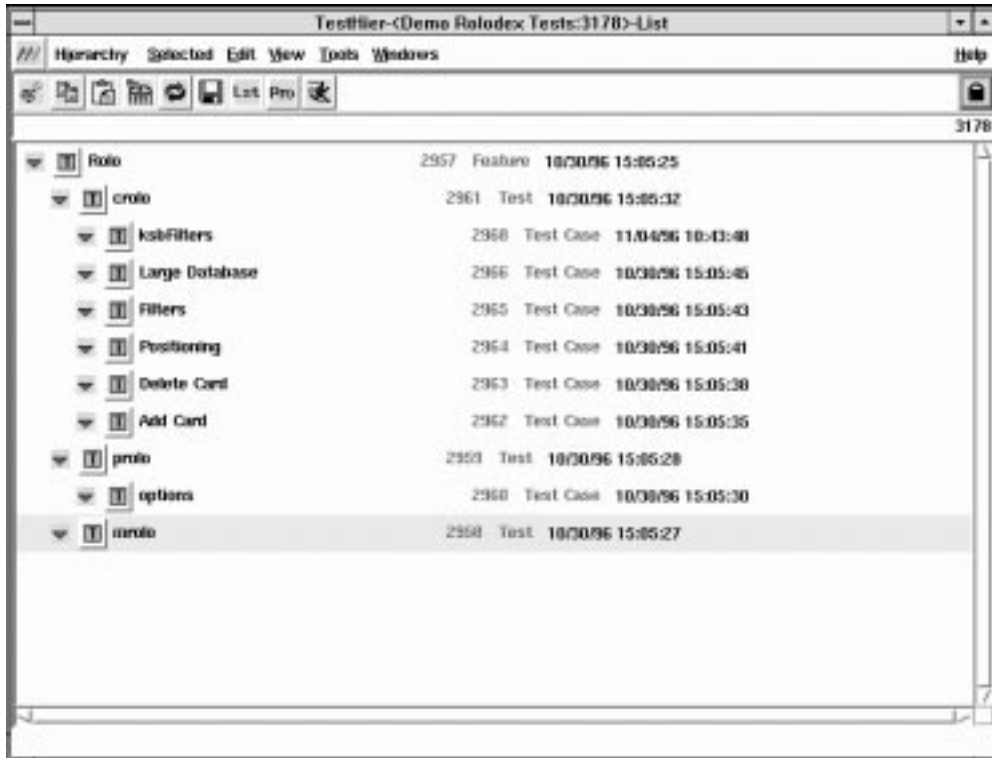


Figure 3-13. List View Expanded Fully

### 3.7 Using the Preferences View

The AETG Desktop Preferences view (Figure 3-14) lets you set items that affect all objects and processing done during the session.



Figure 3-14. AETG Desktop Preferences View

The Default Settings are:

- Printer** Sets the AETG System printer. This selection will appear automatically on all print dialogs.
- Font** Sets the font size for system displays.

<b>Database Max Size</b>	Specifies the number of objects displayed in a Database Browser view with initial queries. For example, if you specified 50 objects here and the system found 100, the system would prompt you to choose either the number you select here (50) or the number found (100). See <a href="#">Section 4</a> for information about the Database Browser.
<b>Max Test Cases</b>	Specifies the number of test cases that the system will automatically load into a Test object Matrix view. See <a href="#">Section 9.3.3.6</a> .
<b>Unlock Objects When Created</b>	Lets you set whether or not newly created objects will automatically be unlocked by the system when opened for the first time.

### 3.7.1 Saving Default Settings

You can change default values. As an example, let's go through the default settings as if we were setting them.

1. Position the pointer in the **Printer** data entry area and type in a printer name, e.g., **printer1**.
2. To learn how to select a font, see [Section 3.7.2](#).
3. Position the pointer in the **Database Max Size** data entry area and type in a number, e.g., **250**.
4. Position the pointer in the **Max Test Cases** data entry area and type in a number, e.g., **600**.
5. Check **Unlock Objects When Created** if you want the system to automatically open newly created objects.

Figure 3-15 shows an edited Preferences view.



**Figure 3-15.** Edited AETG Desktop Preferences View

If you save the Preferences view, the settings will become the default settings for all subsequent AETG sessions until you change them again. If you don't save these settings, they will remain in force for the current session and then revert to the last saved default settings for the next session.

To save these preferences, execute

**AETG->Save**

### 3.7.2 Selecting The Default Fonts

To change the default font:

1. Click on the **Font** pushbutton that appears in the client area of the Preferences view.

The Font Chooser dialog in [Figure 3-16](#) will appear. The top panel of the dialog shows the alphanumeric characters for the current font.



**Figure 3-16.** Font Chooser Dialog

2. Choose a font **Family** by clicking on the one you want, e.g., **Lucida**.  
Notice that the system changes the font display in top panel.
3. Choose a **Face** by clicking on the one you want, e.g., **sans bold**.
4. Choose a **Size** by clicking on the one you want, e.g., **14**.

**NOTE** — Don't set **Size** to anything over **14**.

5. Click **Apply** or **OK** to change the fonts.

[Figure 3-17](#) shows the new font on the Preferences view.

Like the other default preferences, if you change the default font it will remain in force until you change it again with the Font Chooser dialog.



Figure 3-17. Preferences View with Font Change

### 3.8 Changing an AETG View

You can change your view during a session using the **View** menu or by clicking on the Icon in the Tool Bar that corresponds to the view you want. Changing views will be a very common activity as you summon different views to complete test and scripting related tasks.

To change a view perform either of the following:

- Click on the icon in the Tool Bar that corresponds to the view you want, e.g., click **Prf** to change to the Preferences view on the AETG Desktop.

- Execute

**View->Change View**

and select the view you want from the **Change View** drop-down list.

### 3.9 Creating Folders

As we said earlier, you can create folders and use them to organize your work. We want to create a folder where we will place all the items needed to document, develop, and analyze tests for an application. We will name this folder **AETG Demo**.

You create folders from the AETG Desktop. To do this:

1. Execute

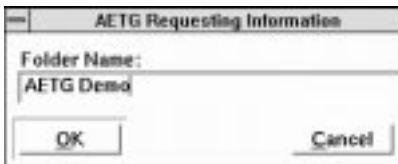
**Edit->Deselect All**

to make sure nothing is selected.

2. Execute

**AETG->New->Folder**

The system will display a dialog requesting a name for the new folder (Figure 3-18.)



**Figure 3-18.** Requesting Information Dialog

3. Enter a name in the **Folder Name** field, e.g., **AETG Demo**.
4. Click **OK**.

The AETG System will display the icon for the new folder. (See [Figure 3-19](#).) The system will display the folder under the new name you gave it. Note that since this is a folder, an arrow appears next to it even though there is nothing in the folder yet.



**Figure 3-19.** New Folder Icon

You can open this folder by double clicking on its icon in the List view. You can create objects from within it, or copy objects into it using the **Database Browser**. We explain how to use the Database Browser in [Section 4](#).

**NOTE** — Test objects do not appear in ordinary folders but in special folders called hierarchies. See [Section 9](#) for information on Test Hierarchies.

### 3.9.1 Saving Folders

To save a folder, execute

**Folder->Save**

### 3.9.2 Deleting Folders

You can delete a folder the same way you delete any other object, which will be explained [Section 4](#). However, you must remember to remove all objects from a folder before you can delete it.

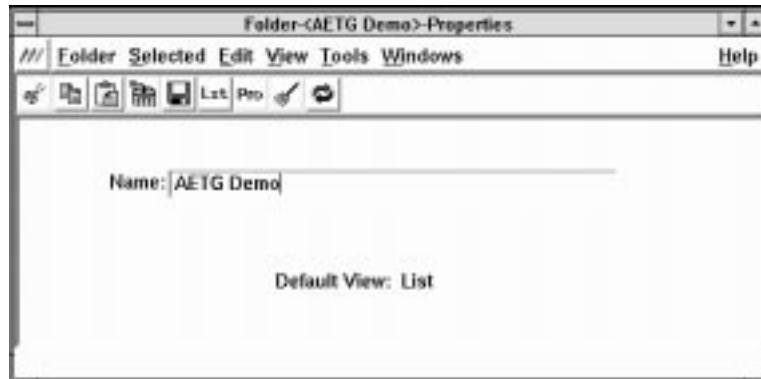
### 3.9.3 Changing a Folder's Name

After you have created a folder you can change its name using the Folder Properties view. To access this view, perform one of the following:

- Execute  
**View->Change View->Properties**
- Click on the **Pro** icon.



The system will change the view to the one shown in [Figure 3-20](#).



**Figure 3-20.** Folder Properties View

We will explain an object's Properties view in detail when we discuss the object in the sections that follow. Here we will explain how to change the name of the folder.

To change the name of a folder:

1. Erase the old name.
2. Type in a new name.
3. Execute

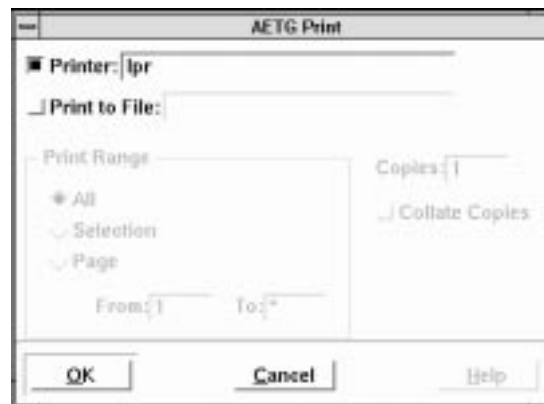
**Folder->Save**

to save the folder under the new name.

## 3.10 Printing

The AETG System provides you with a print feature for most objects. You send information to the default printer, specify another printer, or print to a file.

You access the Print feature from the first menu that appears in the Menu bar. For example, for Format objects this is the **Format** menu. When you do this, the system will display the dialog shown in [Figure 3-21](#).



**Figure 3-21.** Print Dialog

The printer specified as your default printer Preferences view appears in the **Printer** data display area. You can simply click **OK** at this point and the system will print the contents of the current object according to the default settings.

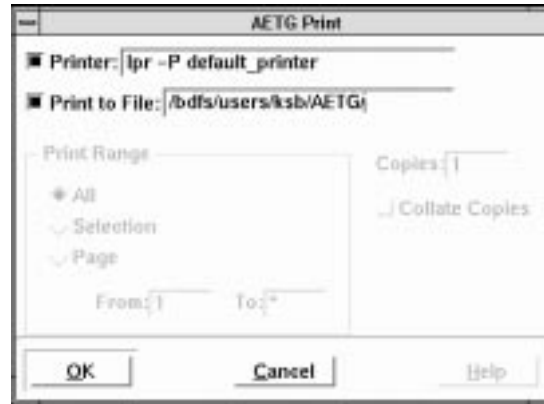
You can choose to change any of the settings by:

1. Positioning the pointer in the **Printer** data entry area and typing in another printer name.

**NOTE — Print Range, Copies, and Collate Copies** will be available in future releases.

2. If you want to print the contents of the object to a file, click on the **Print to File** toggle button and type the output destination in the **File** data entry area.
3. After you make your selections, click **OK** to print.

Figure 3-22 shows an example of a Print Dialog box where we have entered a printer destination *and* the name of a file that will contain the printout.



**Figure 3-22.** Example Print Dialog Box

If you print the output to a file, you must specify, at a minimum, the path to the directory where you want to store the output file, e.g., */bdfs/users/ksb/AETG*.

If you specify a path, the AETG System saves the file using a system generated filename of the form *<Objectname>.n*, where

- *<Objectname>* is a type of AETG object that lets you generate output
- *n* is a system generated number.

For example, if you want to print a Format object and you specify a path only, e.g., */bdfs/users/ksb/AETG*, then the AETG System will generate an output file with a full path and filename such as */bdfs/users/ksb/AETG/Format.3120*.

You can override the system generated filename by specifying your own filename, e.g., */bdfs/users/ksb/AETG/Format\_output*.

When you print information directly from the GUI, your output may not line up correctly due to the internal fonts installed in the printer.

For example, Figures 3-23 and 3-24 show the same excerpt of the printout generated by a Format object. Figure 3-23 shows an example of a printout generated on a printer using fixed width fonts.

```
**FORMAT PROPERTIES VIEW**

Name: Main Screen                      SysID: 4
Version: 1
Owner: ksb
Type: AsyncScreen
Description:

Created By: ksb           Date: 05/30/1997   Time: 11:00:17
Revised By: rachel       Date: 06/05/1997   Time: 14:50:13

**FORMAT FIELDS VIEW**

Fields:
Seq. Name                      Logical Name                      Compound
-----
1   Login ID                    Login Id                          logon
3   Passwd                      Password                          logon
2   Date                        Date                             

Compounds:
Name
-----
logon

**FORMAT ASSOCIATIONS VIEW**

Tests:
ID          Hierarchy & Path                Name
-----
3164       AETG Test Hierarchy:            Login Test
```

**Figure 3-23.** Printout Using Fixed Width Fonts

Figure 3-24 shows an example of the same output generated on a printer using variable width fonts.

```
**FORMAT PROPERTIES VIEW**

Name: Main Screen          SysID: 4
Version: 1
Owner: ksb
Type: AsyncScreen
Description:

Created By: ksb           Date: 05/30/1997  Time: 11:00:17
Revised By: rachel       Date: 06/05/1997  Time: 14:50:13

**FORMAT FIELDS VIEW**

Fields:
Seq. Name          Logical Name          Compound
-----
1  Login ID        Login Id              logon
3  Passwd          Password              logon
2  Date            Date                 

Compounds:
Name
-----
logon

**FORMAT ASSOCIATIONS VIEW**

Tests:
ID    Hierarchy & Path          Name
-----
3164  AETG Test Hierarchy:      Login Test
```

**Figure 3-24.** Printout Using Variable Width Fonts

As you can see, the headings and the data in Figure 3-24 do not line up because of the difference in width for each character.

If you wish to have your the text in your printout line up, save the printout to a file and send the file to a printer using the **lp** command.

### 3.11 Exiting From the AETG System

If you wish to exit from the AETG System, execute

**Aetg->Exit**

---

## 4. Using AETG Objects

We introduced you to the concept of views and what they are in the AETG System in [Section 3](#). Here we will introduce you to another key concept: objects. We will also explain how to organize objects and perform routine activities with them.

In this section we will describe:

- What objects are
- How views are related to them
- Test hierarchies
- Creating new objects
- Opening objects for editing and updating
- Deleting objects.

### 4.1 What Are Objects?

AETG objects represent all of the entities you need to create your test cases. They represent entities such as a format or test. Information contained in objects are called attributes.

**Attributes** are pieces of information that define the object and are used by the AETG System to refine test case generation. For example, a Test object has a name and collection of information that identifies the test and helps you keep track of changes made to the test. [Table 4-1](#) lists all of the AETG objects.

**Table 4-1.** *AETG Objects*

<b>Object</b>	<b>Function</b>
Format	Describes a testable interface to an application or a testing situation. This is the only AETG object that can appear on the AETG Desktop or in a folder.
Field	Defines the data values that are input to an application. Field objects are created and accessed from within Format objects.
Compound	Gives you the option to collect several Fields and treat them as one Field. Compound objects are created and access from within Format objects.
Test	Defines the means by which a requirement is verified in a software application. Test objects reside in Test Hierarchies.
Relation	Lets you collect requirements data to establish how fields and values on the format relate to one another. Relation objects are created and accessed from within Test objects.

The AETG System stores objects along with the object's attributes in its database. All objects have a **Created By** attribute that identifies who created the object and **Revised By** attribute that identifies who made the last changes to the object. Other object properties vary with the function of the object.

You can think of Objects as database records that store information and attributes as fields within a database record.

#### **4.1.1 How Views and Objects Work Together.**

Views display an object's attributes. Views are like screens in a character-based application in that they display information and let you enter information. A single object usually has several views, each of which displays a different set of an object's attributes. Views let you enter information into and retrieve information from the object. The AETG System processes this information in the course of helping you with your activities.

## **4.2 Test Hierarchies**

Test objects always appear in their own Test hierarchies, which are similar to folders. See [Section 9.2](#) for information on working with Test hierarchies.



## 4.3 Creating Objects

It is easy to create new objects, but once you create a new object you still have to define its attributes. This too is fairly simple, but it varies from object type to object type. In this section we explain the basic actions to create an object. We will explain how to specify the attributes for each object in later sections of the manual.

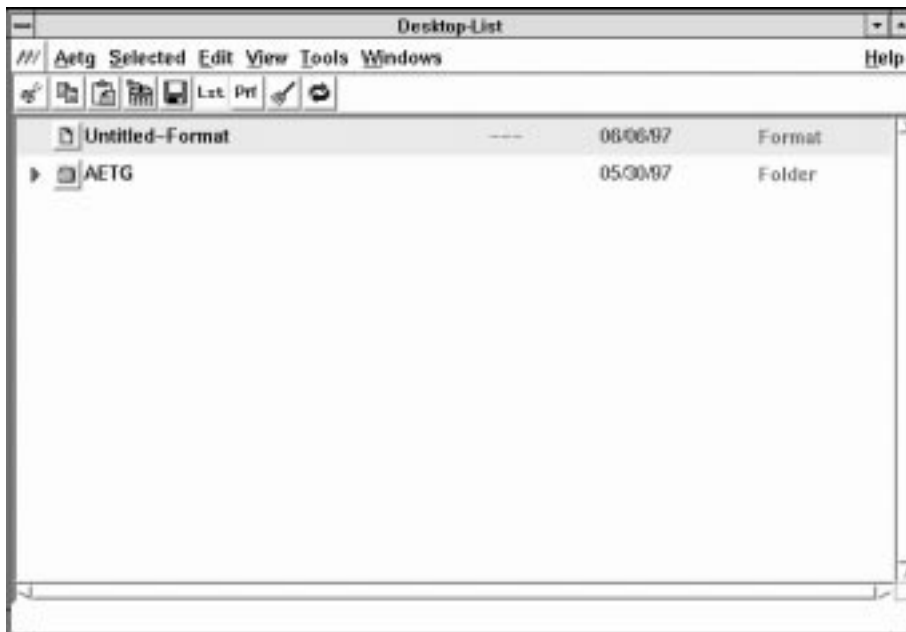
### 4.3.1 Creating Objects on the AETG Desktop

To create an object on the AETG Desktop, click on the **AETG** menu, on the **New** menu selection, and finally on the object type you want.

For example, to create a Format object, execute

**AETG->New->Format**

The system will place the new object in the first position on the AETG Desktop. (See [Figure 4-1.](#))



**Figure 4-1.** New Format Object on AETG Desktop

## 4.3.2 Creating New Objects in Folders

You can create objects in both opened and closed folders.

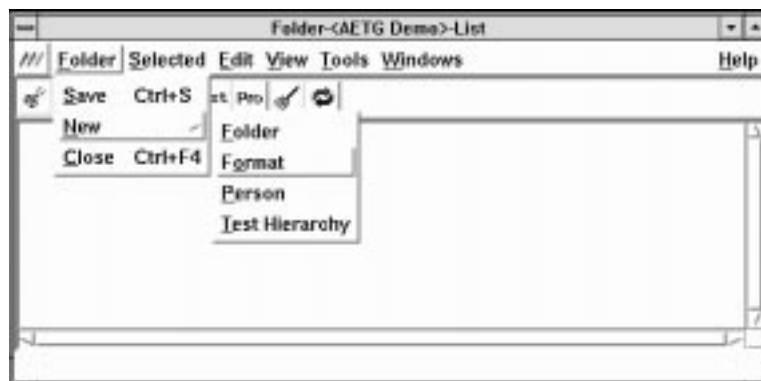
### 4.3.2.1 Creating New Objects In Open Folders

To create objects in open folders

1. Open the folder by double clicking on its icon.
2. In the folder window, click on the **Folder** menu, on the **New** menu selection, and finally on the type of object you want, e.g., execute

**Folder->New->Format**

as shown in [Figure 4-2](#).

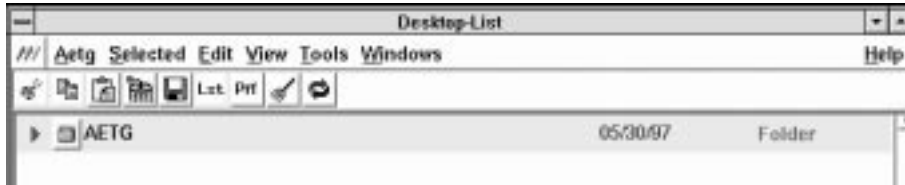


**Figure 4-2.** Creating an Object in an Open Folder

#### 4.3.2.2 Creating New Objects In Closed Folders

To create objects in closed folders

1. Select the folder, as shown in [Figure 4-3](#).

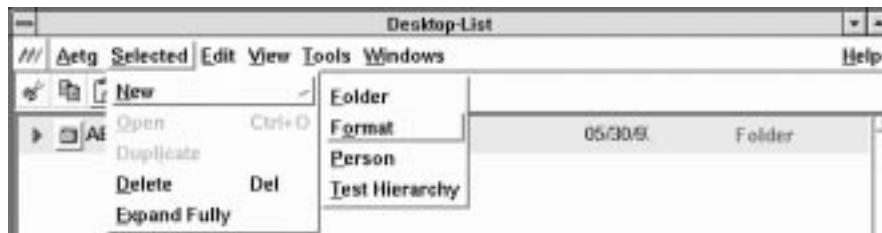


**Figure 4-3.** Selecting a Folder

2. Click on the **Selected** menu, on the **New** menu selection, and finally on the type of object you want, e.g., execute

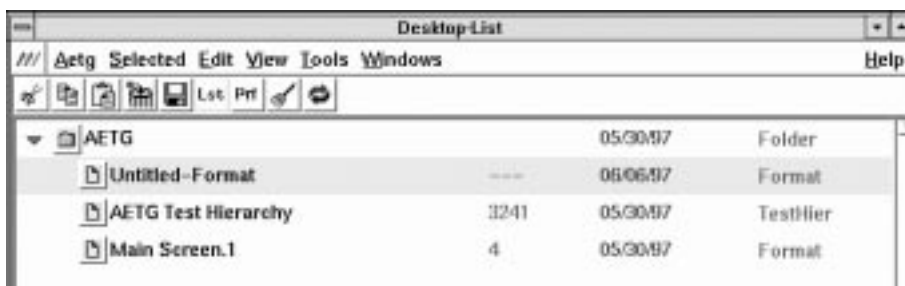
**Selected->New->Format**

as shown in [Figure 4-2](#).



**Figure 4-4.** Creating an Object in Closed Folder

The System will expand the folder's listing on the AETG Desktop and place the object in the first position. (See [Figure 4-5](#).)



**Figure 4-5.** New Object Created In a Closed Folder.

## 4.4 Opening Objects for Editing

You edit objects by opening them and changing the data in them. Here we will describe how to open an existing object so you can edit it. In later sections we explain how to enter information into an object. The way you enter data is the same whether you are entering it for the first time or changing information already there. The only difference is that you will probably have to erase data if you are editing an object.

To open an object for editing, perform one of the following:

- Double click on its icon
- Select the object, and execute  
**Selected->Open**
- Select the object and press **alt** and **S**, then **o**.

The AETG System will open the object to the delivered default view unless you have changed the default view with the AETG Preferences view, in which case the system will open the object to the view you specified. From here you can change an object's attributes or associations.

## 4.5 Copying Objects

When you are copying closed objects from one folder to another, you use the **Copy** selection on the **Edit** menu. However, if you want to copy opened objects you have to use the **Copy Object** menu option.

To copy a closed object:

1. Highlight the object.
2. Execute

**Edit->Copy**

To copy an opened object, execute

**Edit->Copy Object**

Once you have copied the object into the clipboard, you can paste it into a Folder, the AETG Desktop, or a ruler area.

**NOTE** — Paste does not create a new object. It simply makes a link to an existing object.

## 4.6 Duplicating Objects

When you want to copy a Format, Test, or Relation object, you must use the **Duplicate** option on the **Selected** menu. **Duplicate** creates a new object with many of the attributes of the original.

To duplicate a Test or Format object, simply select the object and execute

**Selected->Duplicate**

How you use the **Duplicate** option depends on which object you are duplicating. Therefore, we will discuss the usage of the **Duplicate** option when we discuss each object.

See [Section 6.2.4](#) for information on duplicating a Format object.

See [Section 9.2.4](#) for information on duplicating a Test object.

See [Section 9.3.2.3](#) for information on duplicating a Relation object.

### 4.6.1 The Differences Between Copying and Duplicating Objects

There are differences between copying an object, and duplicating an object.

- **Copy** and **Copy Object** don't create a new object. They create a link between the existing object and a new location (folder or the AETG desktop) so you can access the object from the new location.
- **Duplicate** creates a new object in the AETG database and copies some (but not all) attributes to the new object. The attributes copied over to a duplicated object depend on the object being duplicated. [Table 4-2](#) lists what attributes and associations the system copies to the new object when you use **Duplicate**.

**Table 4-2.** *Attributes Copied by Duplicate*

Object	Attributes
Format	Name <sup>a</sup> , Owner, Type, Fields, Compounds
Test	Priority, Type, Description
Relation	Name, Field Values, Description, Includes, Excludes, Invalids

a. This attribute is only copied when you create a new version of an existing Format object. See [Figure 6.2.4](#).

## 4.7 Deleting Objects

When we speak of deleting objects we mean deleting objects from the AETG database. You can remove objects from the AETG Desktop display or from a folder or Hierarchy display using the **Cut** or **Clear** selection on the **Edit** menu. These actions simply remove an object from the display, but does not remove it from the database.

You must be careful when deleting objects since deleting an object may cause other things to happen.

Here are some things you should keep in mind when deleting objects.

- When you delete hierarchies, all objects in the hierarchy are deleted no matter who owns them.
- When you delete a Test parent object, child objects are deleted.
- All of an object's associations are removed.

Generally, only the owner of an object or the AETG System Administrator can delete an object.

To delete an object from the AETG database,

1. Select the object you want to delete.
2. Execute

### **Selected->Delete**

The system will remind you of the consequences of delete with a pop-up dialog and ask you to confirm the deletion before removing the object from the AETG database and from the display.

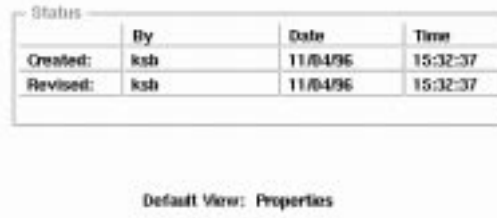
## 4.8 Removing Objects from the AETG GUI Display

As we mentioned earlier, you can remove objects from the GUI display using either the **Cut** or **Clear** selection on the **Edit** menu. **Cut** removes the object from the desktop and copies it to the clipboard; it can then be pasted elsewhere in the GUI. **Clear** completely removes the object from the GUI. Once you use **Clear**, the object can not be retrieved or pasted.

**NOTE** — Untitled objects cannot be copied to the clipboard, therefore, you can not use **Cut** on untitled objects. You can, however, use **Clear** to remove untitled objects from the display.

## 4.9 Objects' Status and Default View

An object's revision history appears on all Properties views for objects that appear in the AETG database. Figure 4-6 shows the Status display area for a Test object.



Status	By	Date	Time
Created:	ksh	11/04/96	15:32:37
Revised:	ksh	11/04/96	15:32:37

Default View: Properties

**Figure 4-6.** Object Status Display and Default View Area

The Status area displays the **Date** and **Time** an object was **Created** or **Revised**, and the name of the user who created or revised it.

The system displays the **Default View** for the object directly below the **Status** area. This is the view specified as the default on the AETG Desktop's Preference view.

## 4.10 Modifying Another Person's Objects

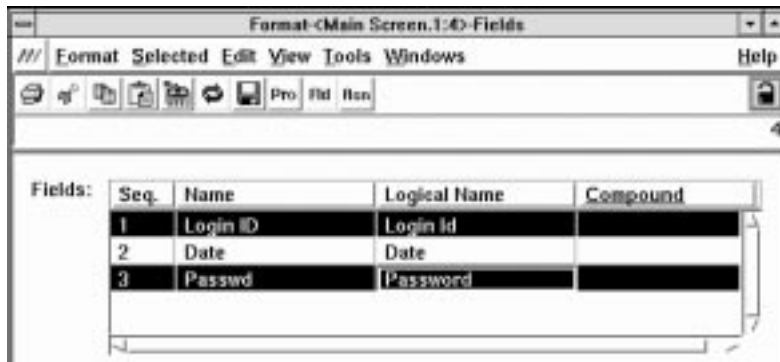
The *xmyConfig.General* file contains a parameter that specifies whether a user has the ability to modify other people's objects in the AETG GUI. If this parameter is set to false, only the owner of an object or an administrator will be able to edit the object.

**NOTE** — All users will still be able to open the object in read-only mode.

The ability to open Test Hierarchies for editing purposes is not affected by the setting of this configuration tag; non-owners and non-administrators can open Test Hierarchies for editing even if **NonOwnerObjectModification** is set to *false* .

## 4.11 Multiselecting Objects

While working with the AETG System, you often have to multiselect objects, that is you select more than one object at a time. For example, if you choose to create a Compound (Section 6.2.2.2.1 and Section 8), you must select more than one Field objects from a Format object, such as shown in Figure 4-7.



**Figure 4-7.** Multiselecting Fields Objects on a Format Object

There are several methods you can use to multiselect objects.

- To multiselect non-adjacent items,
  1. Click on one item you want to multiselect.
  2. Control-click on the other items you want to multiselect, that is, hold down the **Control** key (**Ctrl**) while you click with the **Left** mouse button.
- To multiselect adjacent items,
  1. Click on the first item one item you want to multiselect
  2. Drag through the list of object until you select all of the objects you want to multiselect.
- To deselect a highlighted item, **Ctrl-Click** on the value.



## 5. Using the Database Browser

The AETG System **Database Browser** provides a quick and easy way of locating objects in the AETG Database

In this section we cover:

- Accessing the Database Browser.
- Changing Object Type.
- Defining include queries.
- Changing the owner of an object.
- Associating objects with other objects.
- Using Person objects.

### 5.1 How the Database Browser Helps You

When using the AETG System, you create many objects to support your activities. These objects are stored in the AETG Database. The Database Browser provides quick and easy way of locating objects.

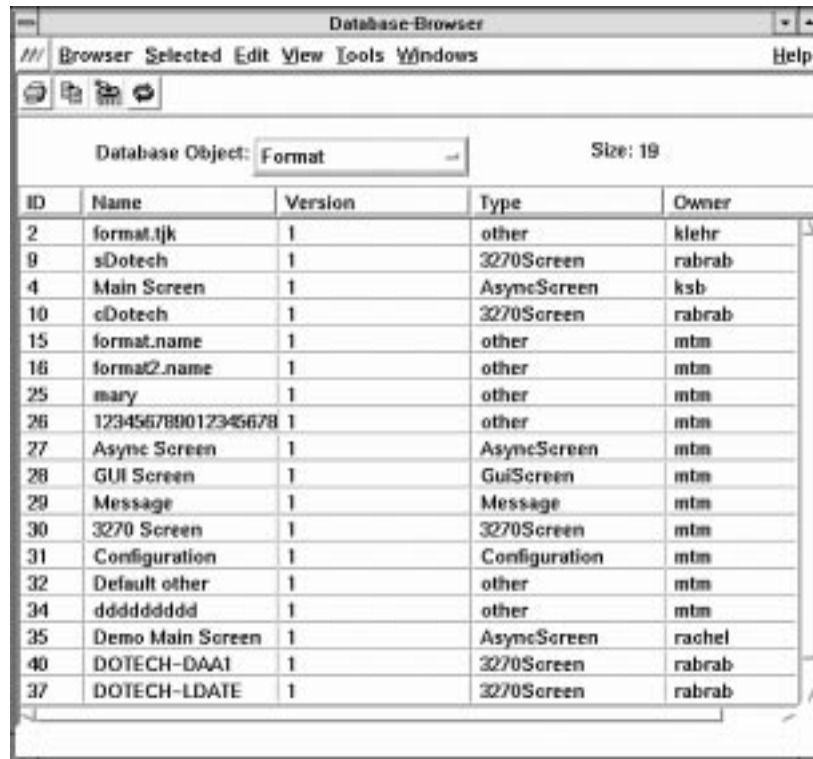
The Database Browser is the chief tool for managing objects, since it allows you to display, open, and edit all existing objects.

## 5.2 Accessing the Database Browser

To access the Database Browser, execute

**Tools->Database Browser**

The system displays a Database Browser main window such as the one shown in [Figure 5-1](#).



The screenshot shows a window titled "Database-Browser" with a menu bar (Browser, Selected, Edit, View, Tools, Windows, Help) and a toolbar. Below the toolbar, there is a "Database Object:" dropdown menu set to "Format" and a "Size: 19" indicator. The main area contains a table with the following data:

ID	Name	Version	Type	Owner
2	format.tjk	1	other	klehr
9	sDotech	1	3270Screen	rabrab
4	Main Screen	1	AsyncScreen	ksb
10	cDotech	1	3270Screen	rabrab
15	format.name	1	other	mtm
16	format2.name	1	other	mtm
25	mary	1	other	mtm
26	123456789012345678	1	other	mtm
27	Async Screen	1	AsyncScreen	mtm
28	GUI Screen	1	GuiScreen	mtm
29	Message	1	Message	mtm
30	3270 Screen	1	3270Screen	mtm
31	Configuration	1	Configuration	mtm
32	Default other	1	other	mtm
34	dddddddd	1	other	mtm
35	Demo Main Screen	1	AsyncScreen	rachel
40	DOTECH-DAA1	1	3270Screen	rabrab
37	DOTECH-LDATE	1	3270Screen	rabrab

Figure 5-1. Database Browser Window

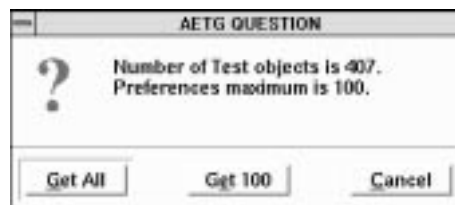
## 5.2.1 Displaying Object Types with the Database Browser

By default the system displays Format object when you first access the browser, but you can select any object you want using the **Database Object** option list.

For example, if you wanted to see Test objects, you would simply execute

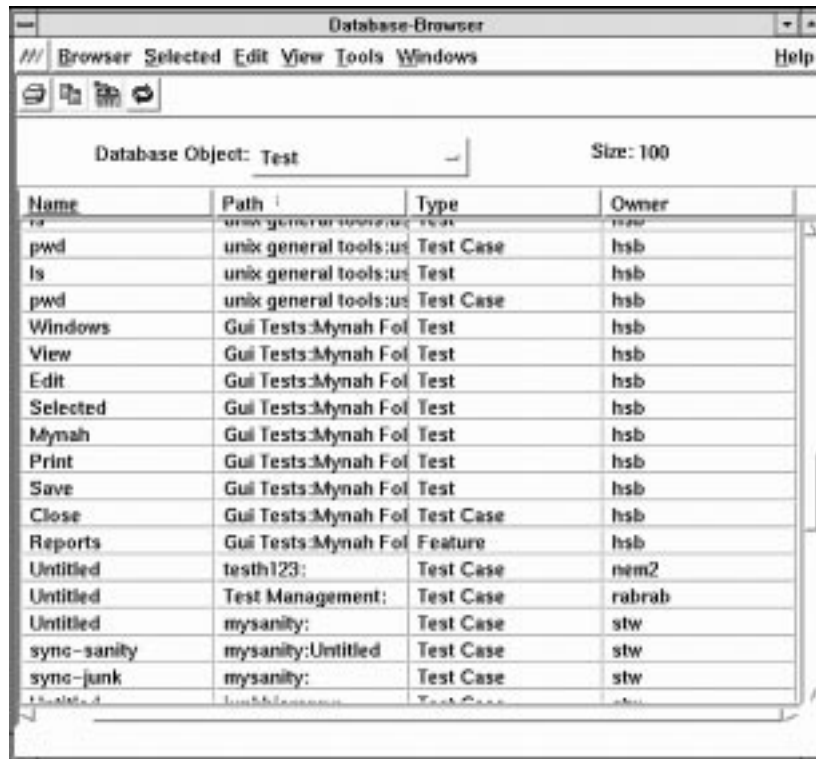
**Database Object->Test**

You can specify what objects will appear based on query criteria you enter. (See [Section 5.2.2.](#)) If more than 100 objects of the selected object match your query in the database, the system will display the dialog shown in [Figure 5-2.](#)



**Figure 5-2.** AETG Question Dialog

After you respond to the AETG Question dialog, the system displays a list of Test objects similar to the one shown in [Figure 5-3.](#) Remember, you can sort objects by an attribute by double clicking on a column heading since this is a “ruler” display. See [Section 3](#) for an example of doing this.



The screenshot shows a window titled "Database-Browser" with a menu bar containing "Browser Selected Edit View Tools Windows" and a "Help" button. Below the menu bar is a toolbar with several icons. The main area of the window displays "Database Object: Test" and "Size: 100". A table with four columns is shown: "Name", "Path", "Type", and "Owner". The table contains the following data:

Name	Path	Type	Owner
pwd	unix:general:tools:us	Test Case	hsb
ls	unix:general:tools:us	Test	hsb
pwd	unix:general:tools:us	Test Case	hsb
Windows	Gui Tests:Mynah Fol	Test	hsb
View	Gui Tests:Mynah Fol	Test	hsb
Edit	Gui Tests:Mynah Fol	Test	hsb
Selected	Gui Tests:Mynah Fol	Test	hsb
Mynah	Gui Tests:Mynah Fol	Test	hsb
Print	Gui Tests:Mynah Fol	Test	hsb
Save	Gui Tests:Mynah Fol	Test	hsb
Close	Gui Tests:Mynah Fol	Test Case	hsb
Reports	Gui Tests:Mynah Fol	Feature	hsb
Untitled	testh123:	Test Case	nem2
Untitled	Test Management:	Test Case	rabrab
Untitled	mysanity:	Test Case	stw
sync-sanity	mysanity:Untitled	Test Case	stw
sync-junk	mysanity:	Test Case	stw

Figure 5-3. Database Browser Window with Test Objects Displayed

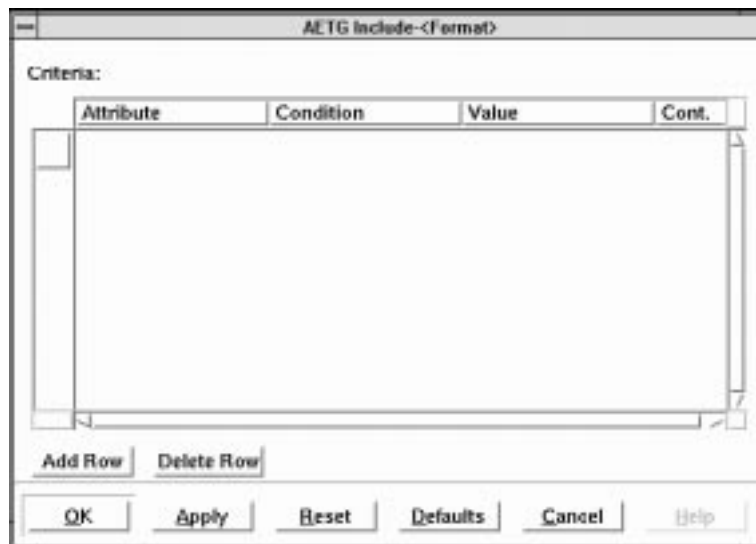
## 5.2.2 Using the Include Dialog

The Include dialog allows you to filter out objects based on criteria you enter. It provides you with a way to control how many objects appear in the Database Browser. This will help you to organize objects in a convenient way. You can also specify attributes for an object and even conditions between attributes to select objects included in the Browser.

You access the Include dialog by executing

**View->Include**

The system will display the Include dialog shown in [Figure 5-4](#).



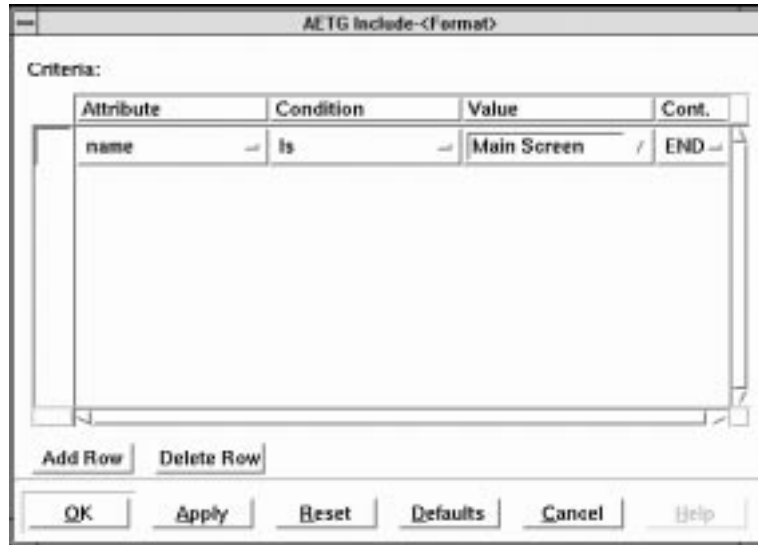
**Figure 5-4.** Include Dialog

An Include query contains an object **Attribute**, **Value** and a **Condition** between them. In addition, there are logical operators that define relations between rows (**Cont.**). We provide default queries for most objects.

You can see the current query for Format objects in [Figure 5-5](#) is

**name is <Main Screen> END**

This will search for a Format named Main Screen.



**Figure 5-5.** Include Query Example

### 5.2.2.1 Object Attributes and Default Queries

[Table 5-1](#) shows the attributes used with objects and the default include queries.

**Table 5-1.** Include Attribute and Default Query

Object Type	Attributes	Default Query	What Query Does
<b>Person</b>	id, authority, firstname, <keyword>, lastname.	<b>Authority NOT EQUAL "Inactive"</b>	Displays all Person objects except Inactive.
<b>Format</b>	id, name, OwnedBy, Type, Version Number	<b>None</b>	Displays all Format objects.
<b>Test</b>	id, measureEffort, measureImport, name, OwnedBy, priority, Type	<b>OwnedBy is &lt;login&gt;</b>	Displays Test objects you own.
<b>Test Hierarchy</b>	id, name, RevisedBy, WhenRevised, CreatedBy	<b>None</b>	Displays all Test Hierarchies.

### 5.2.2.2 Conditions Between Attributes and Values

When you define an **Attribute** for the include statement, you have to define a **Condition** between the attribute and the attribute's **Value**. The **Conditions** available are defined by the **Attribute** you select. The conditions you can use are:

<b>is</b>	The attribute has the specified value.
<b>is not</b>	The attribute does not have the specified value.
<b>Greater Than</b>	The attribute has a value greater than the specified value.
<b>Greater Than or Equal To</b>	The attribute has a value greater than or equal to the specified value.
<b>Less Than</b>	The attribute has a value less than the specified value.
<b>Less Than or Equal To</b>	The attribute has a value less than or equal to the specified value.
<b>Contains</b>	The attribute contains this value.

### 5.2.2.3 Relations between Rows

You must specify a relationship between rows for queries with multiple rows. The possible values for **Cont.** are

<b>END</b>	Indicates that the statement is not continued. There are no other conditions.
<b>and</b>	Indicates that this statement plus the next statement(s) define the query.
<b>or</b>	Indicates that this statement or the next statement but not both define the query.

### 5.2.2.4 Using the Include Dialog Example

To illustrate the Include dialog, we find all the Test objects owned by a user named "ksb" that are the type "Test Case." We use two rows in the Include dialog to do this.

- The first row finds all the scripts owned by **ksb**.
- The second finds all of ksb's Formats that are the type **Test Case**.

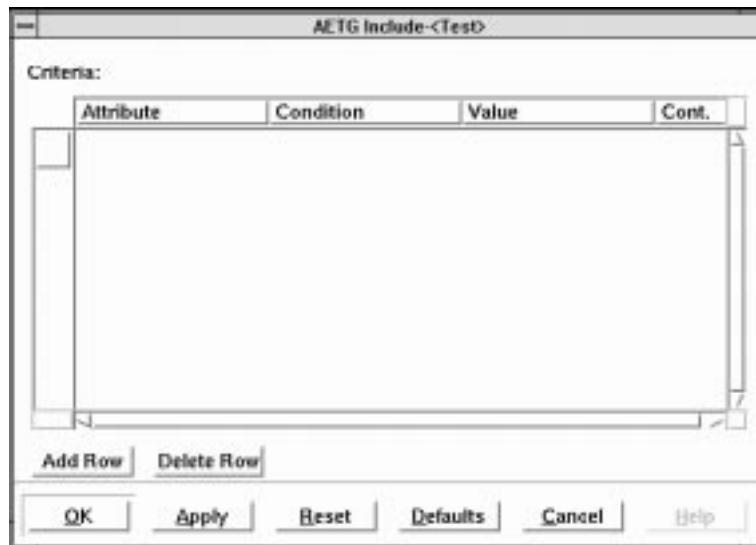
We connect these two rows with **and** so that the system finds Format objects satisfying both conditions simultaneously. If we select **or**, the system finds all the Format objects that satisfies one or the other of the conditions, but not necessarily both. For example, a Format object owned by someone other than ksb but that has the type Test Case is included in the Database Browser display.

From the Database Browser with Test objects selected:

1. Execute

**View->Include**

The system displays the Include dialog shown in Figure 5-6. Since we selected the Include dialog from the Database Browser with **Test** selected, the dialog will be set to filter for Test objects.



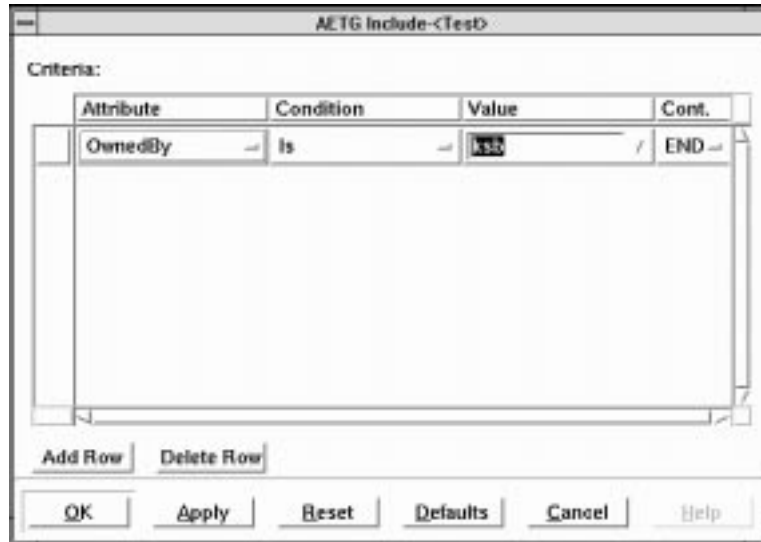
**Figure 5-6.** Test Object Include Dialog

2. To add the first row, click on the **Add Row** pushbutton.  
The system will add a row to let you set attributes and conditions.
3. To specify the **Attribute**, **Conditions** and **Values** for the first row:
  - A. Select an attribute from the first **Attribute** drop down list. (e.g., **Owned By**).
  - B. Select a condition from the **Condition** option list. (e.g., **is**)
  - C. Select a value from the **Value** option list or type a value. (e.g., the owner's name **ksb**).

**NOTE** — If the **Condition** is *Contains*, then you may type a partial value. The system will match all values that contain that partial value.



After you complete selecting items for the row, the Include dialog would look similar to the one shown in Figure 5-7.



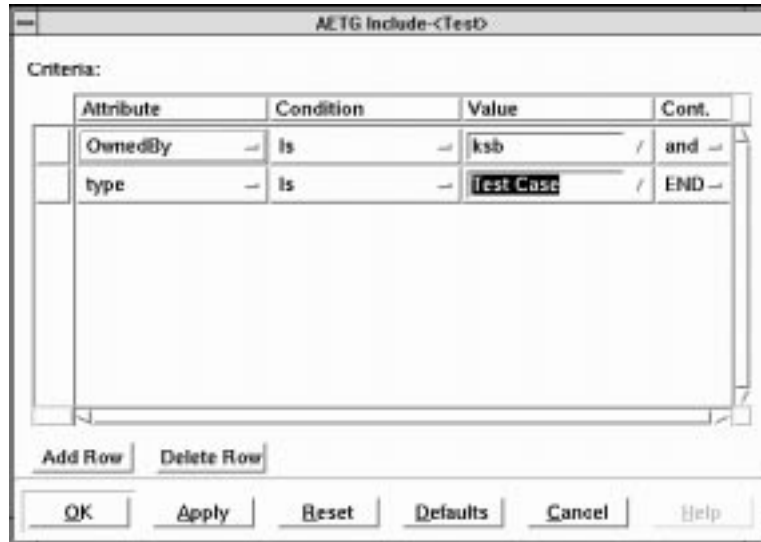
**Figure 5-7.** Include Dialog with One Condition Specified.

4. To specify the second row, click on the **Add Row** pushbutton.

The system will add another row to let you set attributes and conditions. The AETG System automatically set its relation to the first condition will be set to **and**, which is what we want.

5. To specify the **Attribute**, **Conditions** and **Values** for the second row:
  - A. Select an attribute from the first **Attribute** drop down list. (e.g., **Type**)
  - B. Select a condition from the **Condition** option list. (e.g., **is**)
  - C. Select a value from the **Value** option list. (e.g., **Test Case**)

After you complete selecting items for the row, the Include dialog would look similar to the one shown below in Figure 5-8.



**Figure 5-8.** Include Dialog with Two Conditions Specified.

6. Click the **Apply** pushbutton.

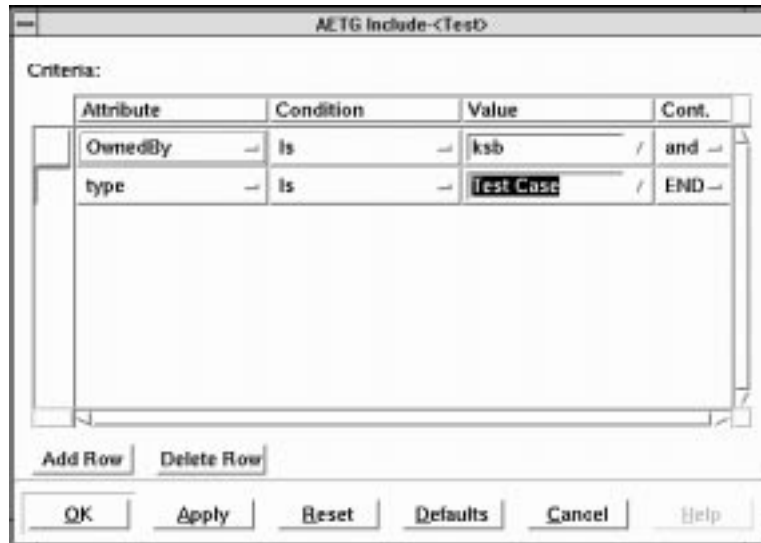
The system will retrieve all the script objects that meet the criteria specified.

The system will save this query as your default query if you do a **Save**.

### 5.2.2.5 Deleting Rows From the Include Dialog

You can delete rows from an Include query. To do this

1. Click on the square at the beginning of the row you want to delete. (See [Figure 5-9.](#))



**Figure 5-9.** Deleting a Row

2. Click on the **Delete Row** push button.

The system will delete the row you checked.

### 5.2.3 Opening Objects With the Database Browser

As we said earlier, you can open objects directly from the Database Browser window. To do this

1. Click on the object you want to open to select it.
2. Execute

#### **Open->Selected**

You can also simply double click on the object row item.

In either case, the system will open the object you selected and display its default view.

## 5.3 Using Person Objects

As we said earlier, most information in the AETG System is represented as an object. Every AETG user is represented as a Person object. That includes you!

When you first logon to the AETG System, it will prompt you for your name, login id, and telephone number. This information is stored in your Person Object.

Your system administrator maintains all person objects and in general you can't create or edit other user's Person Objects. You can however access Person objects and look at the information they contain. You can also access your own person object and make changes to the data in it.

### 5.3.1 Viewing Person Objects

To view Person Objects:

1. Execute

**Tools->Database Browser**

2. Select **Person** as the **Database Object**.

The system will display a list of Person objects.

3. Open the Person object you want to view by performing one of the following:

- Double clicking on it.
- Execute

**Selected->Open**

Figure 5-10 shows a Person object Properties view we opened. The person object we opened happens to be for a user with Administrative privileges.



**Figure 5-10.** Person Object Properties View

Note that there is no lock icon on this view because you can't change any information since you can't change information in another user's Person object. As you can see from Figure 5-10 each Person object lists:

- |                     |                                       |
|---------------------|---------------------------------------|
| <b>User ID</b>      | The UNIX user ID.                     |
| <b>Last Name</b>    | The user's last name.                 |
| <b>First Name</b>   | The user's first name.                |
| <b>MI</b>           | The user's middle initial.            |
| <b>Email</b>        | The user's email address.             |
| <b>Phone</b>        | The user's phone number.              |
| <b>Authority</b>    | System authority granted the user.    |
| <b>Default View</b> | The currently specified default view. |

### 5.3.2 Changing Data in Your Person Object.

You can change the data that appears in your Person object. The only data you can change are the data you entered when you logged on to the AETG System for the first time. You cannot change information entered by the System Administrator.

To change information in your person object:

1. Execute

#### Tools->Database Browser

2. Select **Person** as the object **Type**.
3. Select your Person object.
4. Open it by performing one of the following:
  - Double clicking on it.
  - Execute

#### Selected->Open

The system display the Person object's Properties view (Figure 5-11).



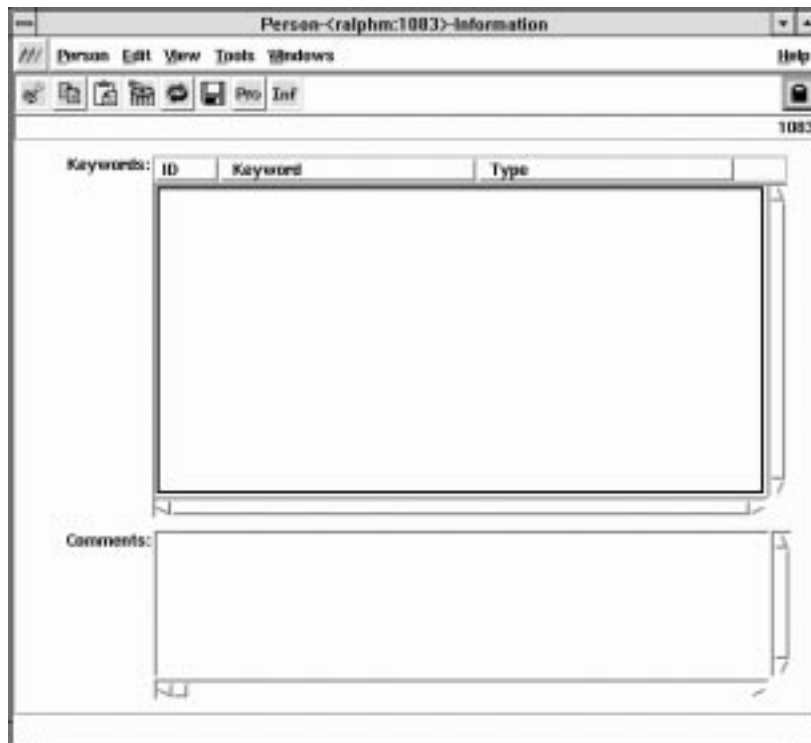
Figure 5-11. Person Object Opened for Editing

5. Click on the Lock icon to unlock the Person object.
6. Highlight the information you want to change and delete it.
7. Type in new information.
8. Repeat Steps 6 and 7 for each piece of information you want to change.
9. To save your changes, execute

**Person->Save**

### 5.3.3 Using the Person Object Information View

The Person object Information view allows you to see the Keyword objects associated with a Person object. You access the Information view by clicking on the **Inf** icon or by clicking on the **View** menu, the **Change View** menu selection and then selecting the **Information** view from the list. The system will display the Information view shown in Figure 5-12.



**Figure 5-12.** Person Object Information View

You can enter comments in the Comment area if this is your Person object. To enter comments:

- Position the pointer in the first free line of the **Comments** area
- Type in the text.

When you are finished copying in Keyword objects and entering comments, execute

**Person->Save**

to save your changes.



---

## 6. Format Object

A Format object describes a testable interface to an application or a testing situation. This interface, or situation, can represent any of the following:

- A Screen (e.g., 3270 or Asynchronous screen)
- A Message (e.g., a defined message between two applications, such as a Remote Procedure Call or a contract message)
- A Protocol (e.g., a defined protocol between any two entities, such as a communication protocol)
- A Flow (e.g., a defined flow of data or logic)
- A Configuration (e.g., this could represent a set of valid combinations of applications or it could represent all of the Tags and Values in a configuration file)
- Any other testing situation that can be represented as a set of fields.

### 6.1 Why use a Format Object

You use a Format object to document the interface that you are trying to represent and to define all of the fields on the format.

### 6.2 Working a Format Object

You create a Format object by executing one of the following from either the AETG Desktop or from within a folder.

To create a Format object ...	Then
On the AETG Desktop	On the AETG Desktop, execute <b>AETG-&gt;New -&gt;Format</b>
In an open Folder	In a Folder, execute <b>Folder-&gt;New-&gt;Format</b>
In a closed Folder	1. Select the Folder. 2. Execute <b>Selected-&gt;New-&gt;Format</b>

An untitled Format object appears on the Desktop or in a Folder. You may then open, edit, and save the Format object. Editing entails defining the Fields and Values for this Format.

You may optionally create Compounds, which group Fields so that you can treat the fields as one unit.

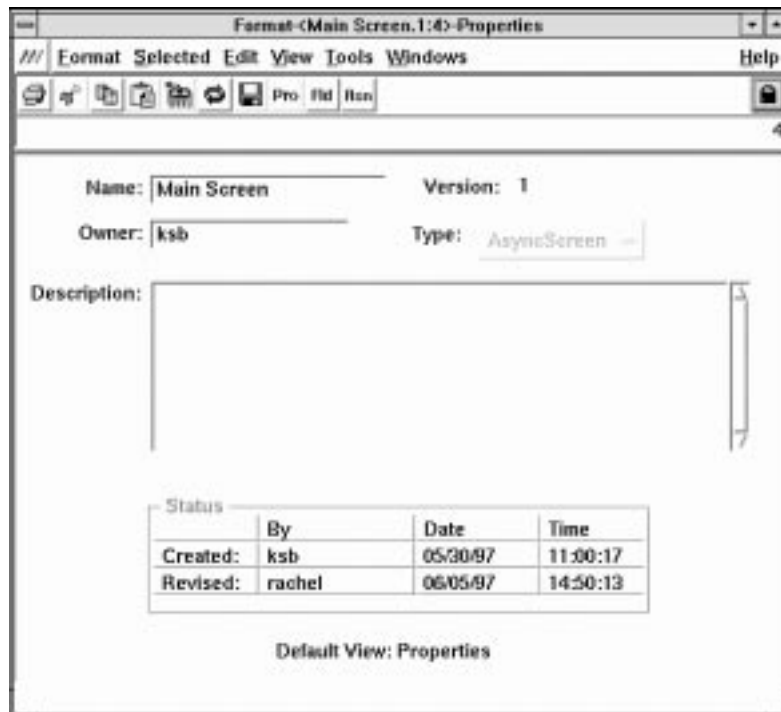
The Format object has three views:

- Properties**      Used to define the Properties for the Format object. (See [Section 6.2.1.](#))
- Fields**        Used to create Fields and, if desired, Compounds for the Format Object. (See [Section 6.2.2.](#))
- Associations**    Used to display what Test(s) you have associated with this Format object. (See [Section 6.2.3.](#))

The default view is the Properties view ([Figure 6-1](#)).

### 6.2.1 Properties View

The Format object Properties view ([Figure 6-1](#)) lets you enter or update such information as the Name, Owner, and Type of the Format object.



**Figure 6-1.** Format Object Properties View

The Properties view contains the following parameters:

<b>Name</b>	Lets you enter a name for the Format object, e.g., <i>Main Screen</i> .  This is a required parameter.
<b>Version</b>	An AETG System generated number for the version of the Format object, which helps you track changes to a Format object from release to release. The initial Version number for a new Format object is 1 and is incremented by 1 for each duplicate you make of the Format object. (See <a href="#">Section 6.2.4</a> for information on duplicating a Format object.)  This is a display only parameter.
<b>Type</b>	A drop-down menu that lets you specify the kind of interface being represented. Valid values are  <b>3270Screen</b> The Format object represents a synchronous (3270) screen.  <b>asyncScreen</b> The Format object represents an asynchronous screen.  <b>guiScreen</b> The Format object represents a screen of a GUI based application.  <b>config</b> The Format object represents a configuration screen.  <b>other</b> The Format object type is indeterminate.  Default= <b>other</b> .  This is a required parameter.  <b>NOTE</b> — The Type parameter does not affect the resulting test case matrix but provides you with an added means of documenting the test.
<b>Owner</b>	Specifies the person who is currently responsible for maintaining this Format object. By default this is the person who created the Format object  This is a required parameter.
<b>Description</b>	Lets you enter a description for the Format object, e.g., the purpose or contents of the Format object.
<b>Status</b>	Displays who created or revised the Format object and when the Format object was created or revised.  This is a display only parameter.

---

The combination of the Name and Version number uniquely creates a title for the Format object, which appears in the Title bar in the format

**name.version#**

For example, if you entered **Logon Screen** for the Name and the Version number is **1**, then

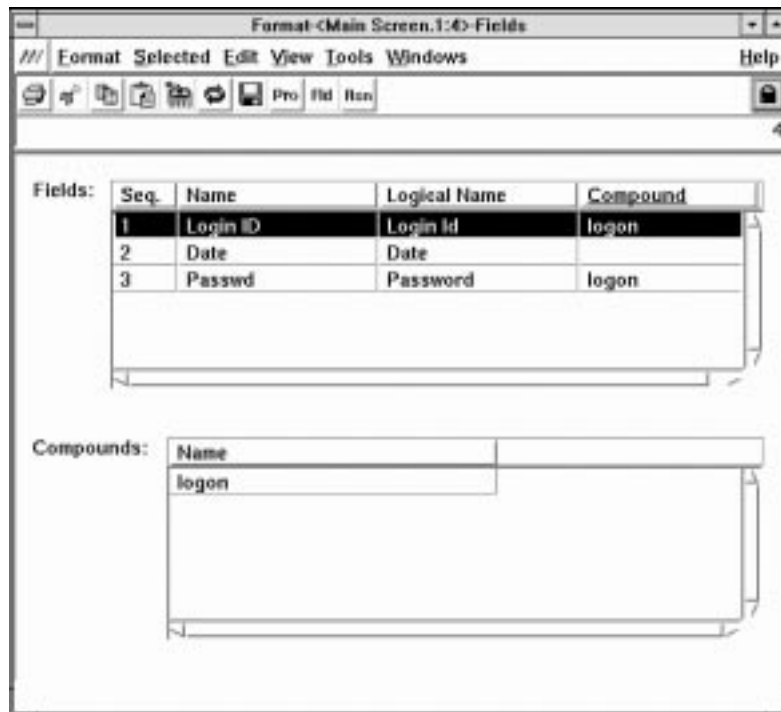
**Logon Screen.1**

appears in the Title bar.

This name (e.g., **Logon Screen.1**) also appears in the List view of the Folder in which you placed this Format object. (See [Figure 6-14](#).)

## 6.2.2 Fields View

The Fields view ([Figure 6-2](#)) lets you define the elements of the Format used to input data to an application.



**Figure 6-2.** Format Object Fields View

The Fields view contains the following parameters:

**Fields** Contains a list of Field objects associated with a Format object. Field objects define the data values that are sent as input to an application. A single Field object is called a **Simple Field**.

This is a required parameter.

See [Section 7](#) for detailed information on Fields Objects.

**Compounds** Contains a list of Compound objects associated with a Format object. Compound objects are optional and let you collect several Fields and treat them as one. A Compound is a **Complex Field**, made up of several simple Fields and specific values.

See [Section 8](#) for detailed information on Compound objects.

#### 6.2.2.1 Working with Fields

The Fields list contains the following columns:

<b>Seq.</b>	The Sequence Number within the Format for this Field. (See <a href="#">Section 6.2.2.3</a> .)
<b>Name</b>	The user-provided Field Name. This Field Name must be unique for a Format. (See <a href="#">Section 7.2.1</a> .)
<b>Logical Name</b>	The user-provided Logical Name given to the Field. (See <a href="#">Section 7.2.1</a> .)
<b>Compound Name</b>	The name of a Compound that the Field participates in. If the Field does not participate in a Compound, this column is blank. (See <a href="#">Section 8</a> .)

### 6.2.2.1.1 Creating a New Field Object

To create a new Field object

1. Make sure that the Format Object is unlocked.
2. Execute

#### Format->New Field

An untitled Field row, marked **<new field>**, appears in the Field list and a new unlocked Field object is displayed. (See Figure 6-3.) While the Field is untitled, the **Seq.** column contains dashes.

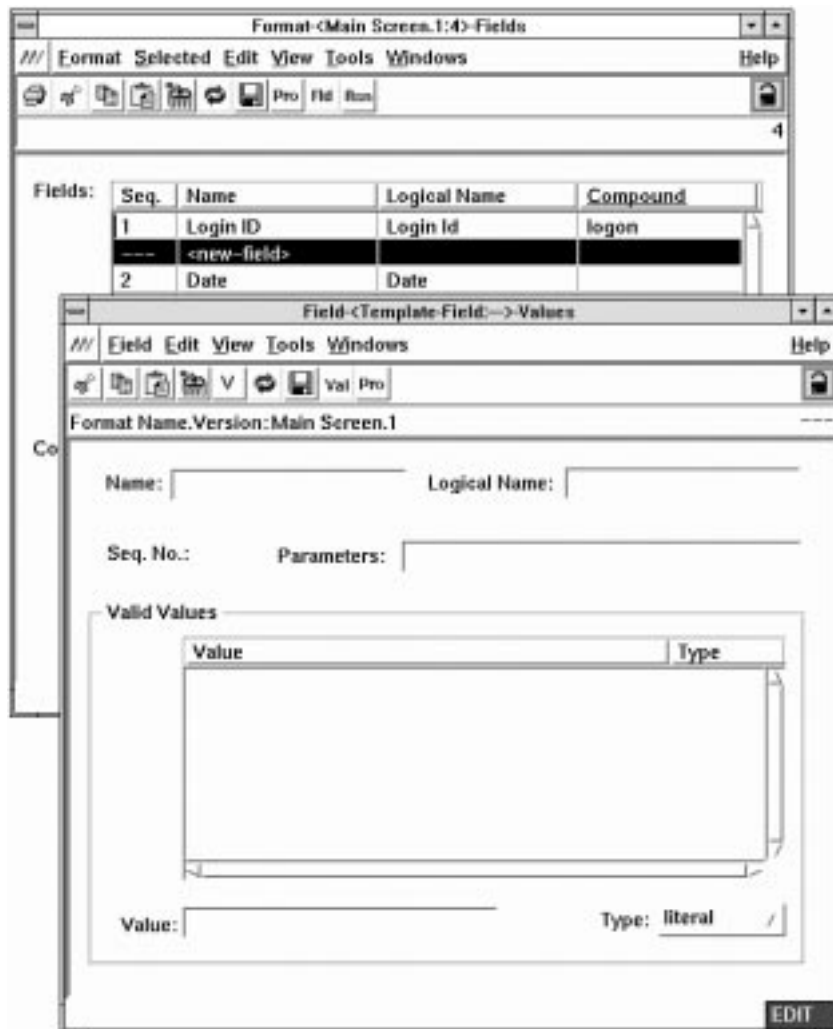


Figure 6-3. Creating a New Field Object

**NOTE** — While an Untitled row exists in the **Field** list, the **New Field** menu item is disabled.

3. Enter, at a minimum, a Field Name and at least one Value for the newly created Field object. (See [Section 7.2.1.1](#) for information on specifying values.)

**NOTE** — To specify a Value, you must first execute **Format->New Value**

**WARNING** — Two Fields in a single Format cannot have the same Field name.

4. Save the Field object.

The new Field is placed after a selected Field row or at the end of the list if no Field row is currently selected.

The Sequence Number of the new Field is populated when the Field is saved for the first time. All other Sequence numbers are updated if necessary.

#### 6.2.2.1.2 *Copying Field Objects*

Field objects can be copied from another Format object or from within the same Format object. To copy a Field object

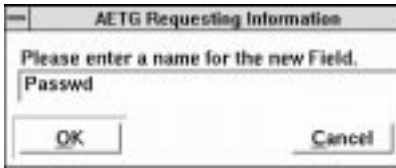
1. Select the Field object you want to copy in the **Fields** list area.
2. Execute

**Edit->Copy**

3. You can now paste the copied Field object.

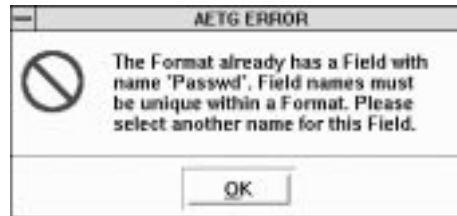
To paste the Field Object(s)...	Then
To another Format object	<ol style="list-style-type: none"><li>1. Open the Format object. If needed, change the view to the Fields view.</li><li>2. Unlock the Format object.</li><li>3. Click on the <b>Fields</b> list area, and execute <b>Edit-&gt;Paste</b></li></ol>
Within the same Format object	<ol style="list-style-type: none"><li>1. If needed, unlock the Format object.</li><li>2. Execute <b>Edit-&gt;Paste</b></li></ol>

In either case, the dialog in [Figure 6-4](#) appears, prompting you to enter a name for the pasted Field object.



**Figure 6-4.** Pasted Field Object Dialog

**NOTE** — No two Field objects within the same Format object can have the same name. If you enter a Field object name that already exists within the Format object, an error dialog such as the one in [Figure 6-5](#) appears. When you click on **OK**, the Field object name dialog in [Figure 6-4](#) reappears, and you *must* enter a unique Field object name or click on **Cancel** to cancel the paste.



**Figure 6-5.** Duplicate Field Object Name Error Dialog

When a Field object is copied, the following Field object attributes are copied:

- Logical Name
- Description
- Parameters
- Field Values.

#### 6.2.2.1.3 *Deleting Field Objects*

To delete Field Objects

1. On an unlocked Format object, select one or more Field objects in the **Fields** list.



2. Execute

**Edit->Delete**

The row corresponding to the deleted Field object(s) is removed from the Fields list.

**NOTE** — If a Field is associated with a Relation (Section 9.3.2.1), the AETG System sounds an error beep and tells you which Relation(s) are affected, otherwise the system deletes the Field and will remove the row from the display.

6.2.2.2 Working with Compound Objects

The Compounds list contains only one column, **Compound Name**, which contains the user-provided names of Compound objects.

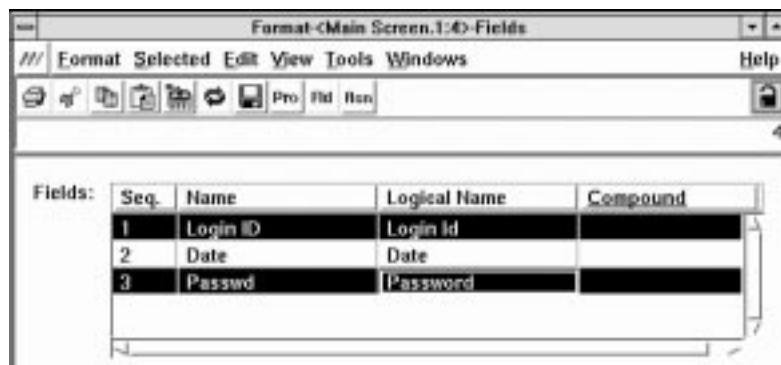
6.2.2.2.1 *Creating Compound Objects*

If you have created more than one Field objects, you can collect several Fields and treat them as one. This is done using a Compound object.

**NOTE** — Compound objects are optional, but can help enhance your test cases.

To Create a Compound object

1. Select two or more Field objects in the **Fields** list area. (See Figure 6-6.)



**Figure 6-6.** Selecting Field Objects for a Compound

See Section 4.11 for information on how to multiselect Field objects.

2. Execute

**Format->New->Compound**

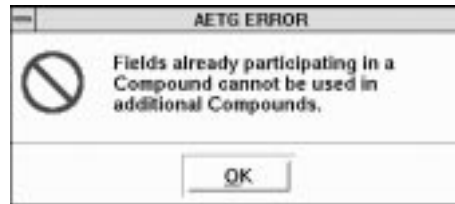
An untitled Compound row, marked <new-compound>, appears in the Compounds list and a new unlocked Compound object (Figure 6-7) appears. By default, the Compound object's Properties view appears.



**Figure 6-7.** Compound Object Properties View

**NOTE** — A Field object cannot participate in more than one Compound. If you select a Field object that participates in a Compound, the error dialog in [Figure 6-8](#) appears when you execute

**Format->New->Compound.**



**Figure 6-8.** Field used in Existing Compound Error Dialog

**NOTE** — When an Untitled Compound row exists in the Compound list, the **New Compound** menu item is disabled until you save or close and discard the Untitled Compound object.

3. You must
  - Provide a Name for the newly created Compound object
  - Generate at least one Tuple, which is a reference name given to the specific value set of a Compound. (See [Section 8.2.3.1.](#))

All other data input is optional.

See [Section 8](#) for more details about working with Compound objects.

4. Save the Compound object

The Fields view of the Format object are updated with the following:

- The Compound Name is populated in the Compound row item
- The Compound Name is populated in each participating Field row.

A Format Object cannot have two Compound objects with the same name. If you try to save a Compound object using the name of an existing Compound object, the AETG System prompts you to enter a new Compound name.

#### 6.2.2.2.2 Deleting Compound Objects

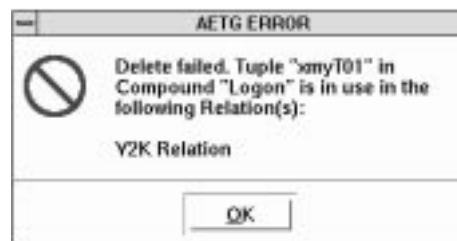
To delete Compound objects

1. Select one or more Compound rows in the **Compounds** list.
2. Execute

##### **Edit->Delete**

The row corresponding to the deleted Compound object is removed from the **Compounds** list. The indicators in the Compounds column of the Field objects that participates in the Compound are also removed.

**NOTE** — If a Compound object is associated with a Relation ([Section 9](#)) an error dialog ([Figure 6-9](#)) appears telling the you which Relation(s) use this Compound.



**Figure 6-9.** Deleting Used Compound Error Dialog

#### 6.2.2.3 Field Sequence Numbers and Rearranging Field Objects

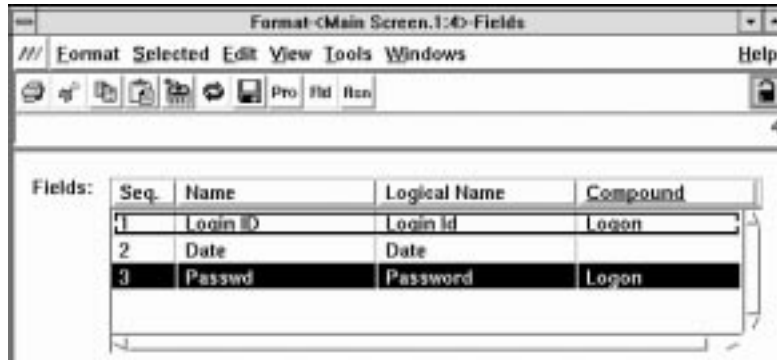
Field object sequence numbers determine the order that Field objects are displayed in a Format object. Sequence numbers also determine the initial order in a new Relation object ([Section 9](#)).

The rows in the Field list are always sorted by the Sequence number. However, you can rearrange Fields by using a simple drag and drop method. This changes the Fields sequence. You select a Field and drag it to a new position using the **Middle** mouse button.

To rearrange Fields

1. If the Format object is locked, click on the lock icon to unlock it.
2. Position the pointer on the Field you want to move.

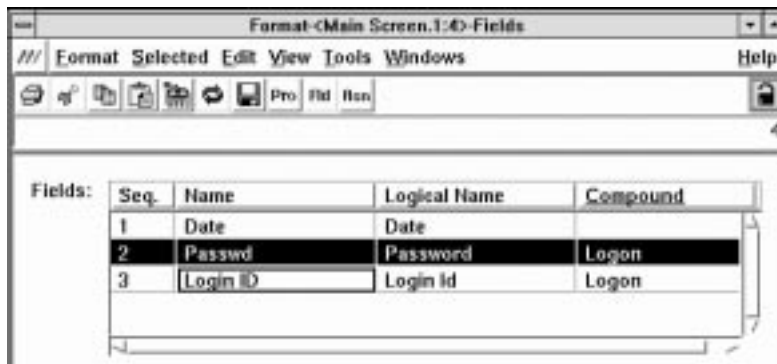
3. Press down and hold the **Middle** mouse button while dragging the Field to a new position. The Field you select is “boxed”, i.e., the lines surrounding the Field are darkened. (See [Figure 6-10](#).)



**Figure 6-10.** Selecting a Field for Rearranging

4. Release the **Middle** mouse button when the Field is in the position you want.

The system moves the Field to the new location and the system regenerates the sequence numbers of all affected Fields. (See [Figure 6-11](#).)



**Figure 6-11.** Resequenced Field Objects

### 6.2.3 Associations Views

The Format object Associations view (Figure 6-12) displays what Test(s) you have associated with this Format object, i.e., those Test objects that have access to the Fields and Values defined by this Format object. See Section 9.3.2.1 for information on how to associate a Test object with a Format object.

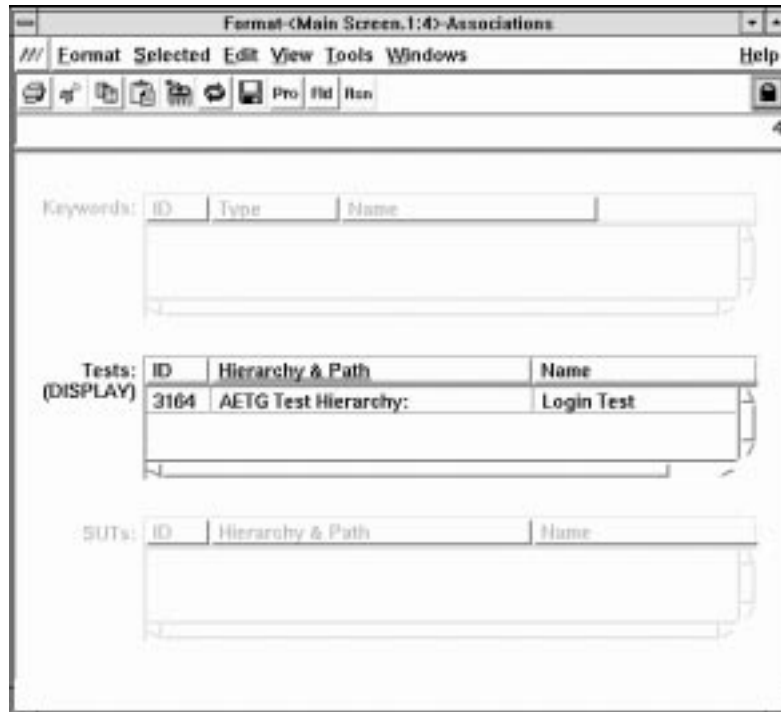


Figure 6-12. Format Object Associations View

## 6.2.4 Duplicating a Format Object

When you want to copy a Format object, you must use the **Duplicate** option on the **Selected** menu. **Duplicate** creates a new Format object with many of the attributes of the original object.

When you execute

**Selected->Duplicate**

the Duplicate Format Dialog (Figure 6-13) appears.



Figure 6-13. AETG Duplicate Format Dialog

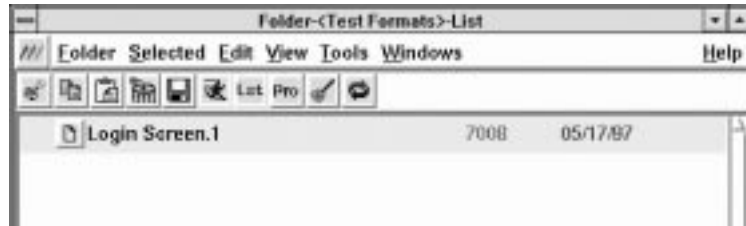
- If you select the **New Version of Existing Format Object** option, the AETG System creates an exact duplicate of the Format object with an iterated Version Number. (See [Section 6.2.1](#) for information on Format object Version Numbers.)

**NOTE** — When you use this method to duplicate a Format object, all attributes of the original object are copied over to the new Format object *except* associated Test objects, e.g., all Fields and Compounds are copied to the new Format object.

- If you select the **New Format Object** option, the AETG System creates a “clone” of the Format object.

**NOTE** — When you use this method to duplicate a Format object, all attributes of the original object are copied over to the new Format object *except* for the name of the Format object and associated Test objects.

Sections 6.2.4.1 and 6.2.4.2 assume you have the Format object in Figure 6-14.

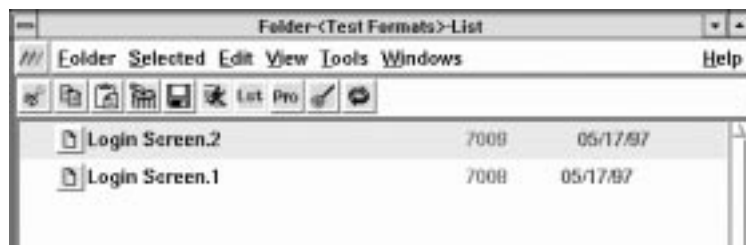


**Figure 6-14.** Original Format Object

**NOTE** — Remember, the name of a Format object as it appears in a Folder’s List view is a combination of the name you entered on the Format object Properties view and the system generated Version Number.

#### 6.2.4.1 Creating a New Version of Existing a Format Object

If you select the **New Version of Existing Format Object** option, the system creates a new Format object as a sibling of the original Format object, as shown in Figure 6-15. The Format object name is retained, but an iterated Version Number is created for the new Format object. In addition, the system assigns the duplicate Format object a new **ID** number.



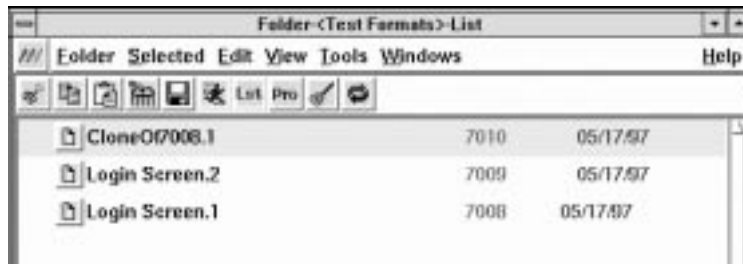
**Figure 6-15.** Creating a New Version of an Existing Format Object

#### 6.2.4.2 Creating a New Format Object

If you select the **New Format Object** option, the system creates a new Format object as a sibling of the original Format object, as shown in Figure 6-16. The system gives the new Format object the name of **Cloneof.<id>**, where **<id>** is the ID number of the original



Format object, in this case the new Format object name is **Cloneof.7008**. The system assigns the duplicate Format object a new **ID** number.



The screenshot shows a window titled "Folder <Test Formats> List" with a menu bar containing "Folder Selected Edit View Tools Windows" and "Help". Below the menu bar is a toolbar with icons for file operations. The main area contains a table with three rows of data:

Format Object Name	ID	Date
CloneOf7008.1	7010	05/17/97
Login Screen.2	7009	05/17/97
Login Screen.1	7008	05/17/97

**Figure 6-16.** Creating a Clone of a Format Object



## 7. Field Object

Field objects define the data values that are sent as input to an application. A Field object belongs to a version of a Format object.

### 7.1 Why use a Field Object

A Field object, in combination with the Format object it belongs to, defines an interface or testing situation. The Field object lets you enter such information about a Field on a Format as the name of the Field and a list of valid values for the Field.

### 7.2 Working with Field Objects

You create a Field object on a Format object ([Section 6.2.2.1.1](#)) by executing

**Format->New Field**

The Field object has two views:

- |                   |   |
|-------------------|---|
| <b>Properties</b> | Used to define the Properties for the Field object. (See <a href="#">Section 7.2.2.</a> )   |
| <b>Values</b>     | Used to enter create the values that is entered in the Fields you define for the Format object. (See <a href="#">Section 7.2.1.</a> ) |

The default view is the Values view ([Figure 7-1](#)).

## 7.2.1 Values View

The Field object Values view (Figure 7-1) lets you enter information for the Field object, such as the name and valid values for the Field on the Format.

Value	Type
ksb	literal
kjm	literal
rkr	literal
rru	literal
hyr	literal
ltr	literal

Figure 7-1. Field Object Values View

The Values view contains the following parameters:

**Name** Lets you enter a name for the Field being specified. This should be the name of the Field as it appears on the Format, e.g., *loginID*.

**NOTE** — You can not include embedded spaces in a Field object name.

This is a required field.

**Logical Name** Lets you enter an optional name for the Field that can be a name used by other users to identify this field, e.g., *login ID*.

**NOTE** — You can include embedded spaces in a name you enter in the **Logical Name** text area.

- Seq.** Displays the Sequence Number for this Field within the corresponding Format. (See [Section 6.2.2.3.](#))  
This is a display only field.
- Parameters** Lets you enter a list of parameters for this Field. (See [Section 7.2.1.2.](#))
- Valid Values** Contains a list showing all values that have been specified for this Field. This area also lets you specify valid Values and Value Types to be used as input for your test cases. (See [Section 7.2.1.1.](#))  
The fields in this parameter are required.

#### 7.2.1.1 Specifying Valid Values

The **Valid Values** area contains a list showing all defined values for the current Field. The **Valid Values** area also contains the following parameters that let you specify new values or change the specifications for existing values:

**Value** A text entry box that lets you enter the values for this field.

**Type** A drop-down menu that lets you select what type of value is being specified:  
**Literal** or **Non-Literal**.

A literal value is a string that must be entered explicitly on the Format, such as a program name or login ID.

A non-literal is a string that you do not wish to (or cannot) explicitly specify. For example, if you need to represent values that are long, you can use a non-literal value. Another example would be when you want to specify a value that cannot be typed, such as a dial-tone.

Table 7-1 provides some examples of value types.

**Table 7-1.** Field Object Value Type Examples

Entered Value	Specified Type	Interpretation
"abc"	Literal	Specifies the string "abc", including the quotes.
dog is brown	Literal	Specifies the string <b>dog is brown</b> . Note that the quotes are not necessary.
ADDRESS	NonLiteral	A non-literal value that represents an address.
DIALTONE	NonLiteral	A non-literal value that represents a dial-tone.
NULL	NonLiteral	A non-literal representation of a null value.
BLANK	NonLiteral	A non-literal representation of the absence of value.
OMIT	NonLiteral	A non-literal representation of the absence of a tag in a message.

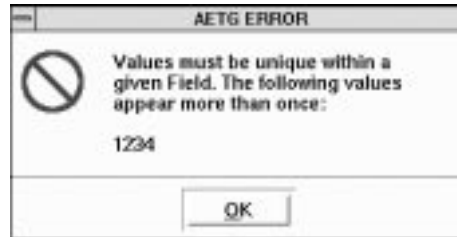
**NOTE** — Values may contain embedded and leading blanks. However, Values cannot contain trailing blanks. If a Value must contain trailing blanks, set the **Type** to **Non-Literal** and let the system set the value.

When you create a new Value (See [Section 7.2.1.1.1](#)), a new, highlighted, Value row is added to the Value list. The Value column is prepopulated with the value **<new value>**, and the Type column is prepopulated with **Literal**. The Value text box is also prepopulated with the value **<new value>**. This value is automatically highlighted, indicating that you can type in this text area to change the value. You can also select a different Type for this new Value.

The new Value row in the list is updated when you hit **Enter** or when you leave this area of the window, either by tabbing or by clicking on any other parameter in the Values view.

You must enter all values you wish to include in the valid set of values for a Field. If a Field can take a large range of values, you can enter just a few values from the range. For example, for a numeric field, you can enter the smallest value in the allowed range, the largest value, and one value from the middle of the range.

Values must be unique within a Field object. If you hit **Enter** or **Tab** from the Value text area and the value is not unique, the error dialog in [Figure 7-2](#) will appear. You must then enter a new, unique value.



**Figure 7-2.** Non-unique Value Error Dialog

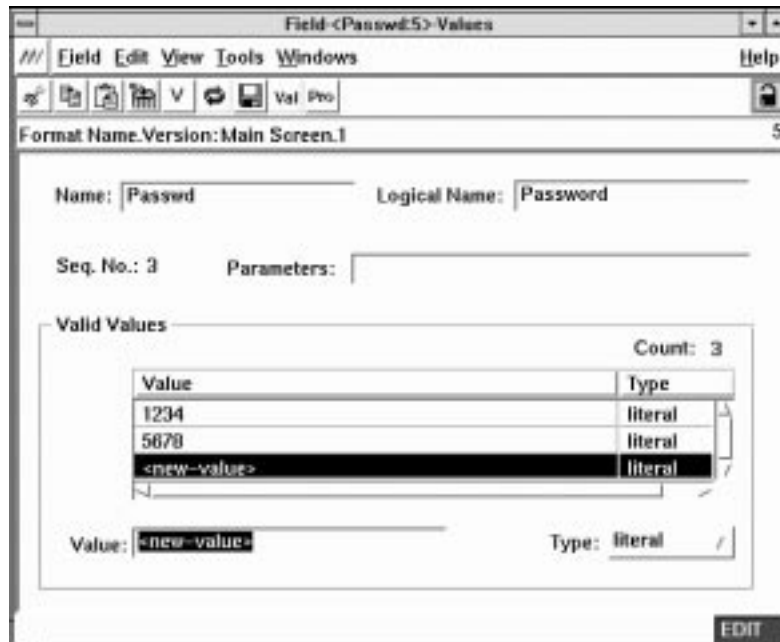
#### 7.2.1.1.1 *Creating New Values*

To create a new Value

1. On an unlocked Field object, execute

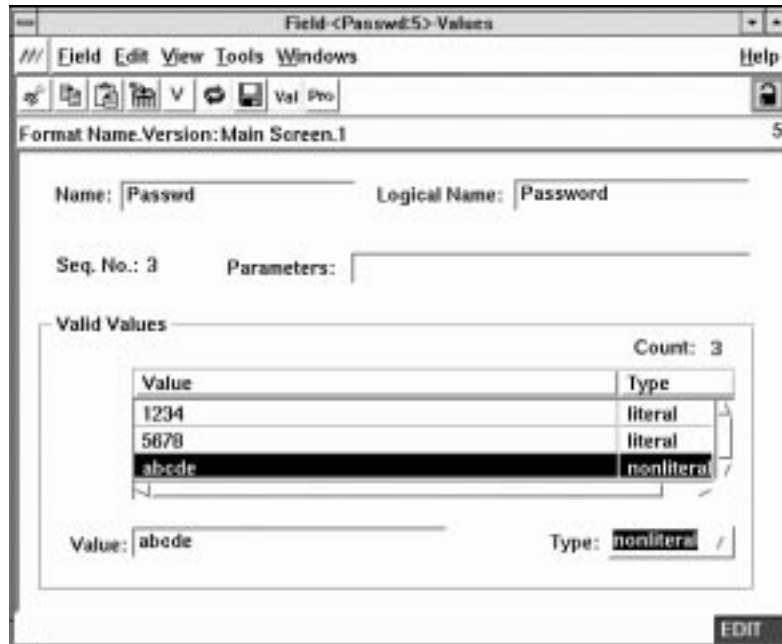
**Field->New Value**

A new Value row is added to the list, as shown in [Figure 7-3](#).



**Figure 7-3.** Creating a New Value

2. Enter a new value in the Value text box (Figure 7-4). Since the text box is already highlighted, you can start typing immediately, overwriting the prepopulated value.



**Figure 7-4.** Entering a New Value

3. If you wish to change the Type of value, click on the **Type** drop-down menu and select the value type you desire.
4. Hit **Enter** or **Tab** to update the Value list.
5. Perform one of the following:
  - Repeat Steps 1 through 4 to enter another value.
  - Save your values to the database by executing

**Field->Save**

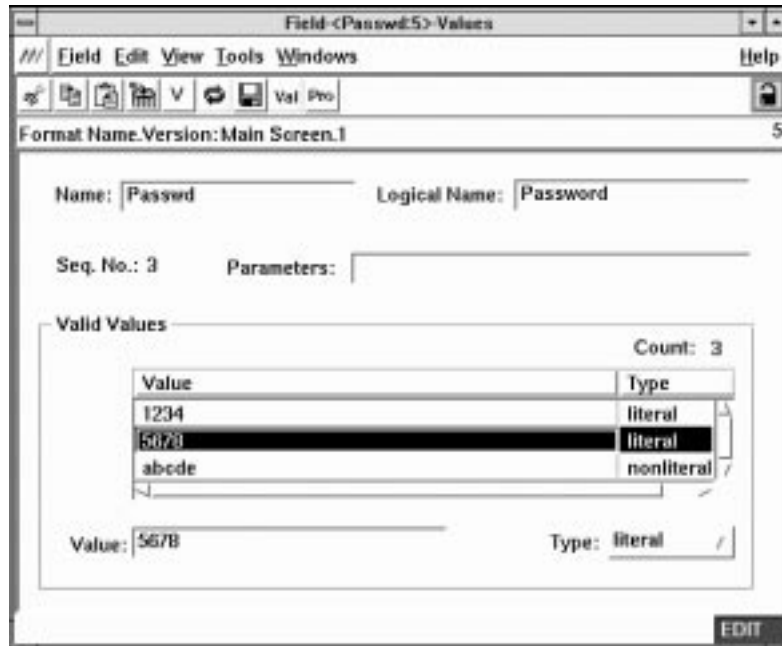
**NOTE** — If a value is not unique, the error dialog in Figure 7-2 appears when you save the Field object.



### 7.2.1.1.2 Changing Values

To update an existing Value row to change a Value

1. On an unlocked Field object, select a Value row. The **Value** text area is populated with the current **Value** and **Type**, as shown in Figure 7-5.



**Figure 7-5.** Selecting a Value to Change

2. Enter a new, unique Value (in the **Value** text box) and/or change the **Type**.
3. Hit **Enter** or **Tab**. The Value row is updated.

### 7.2.1.1.3 Copying Values

To copy an existing value

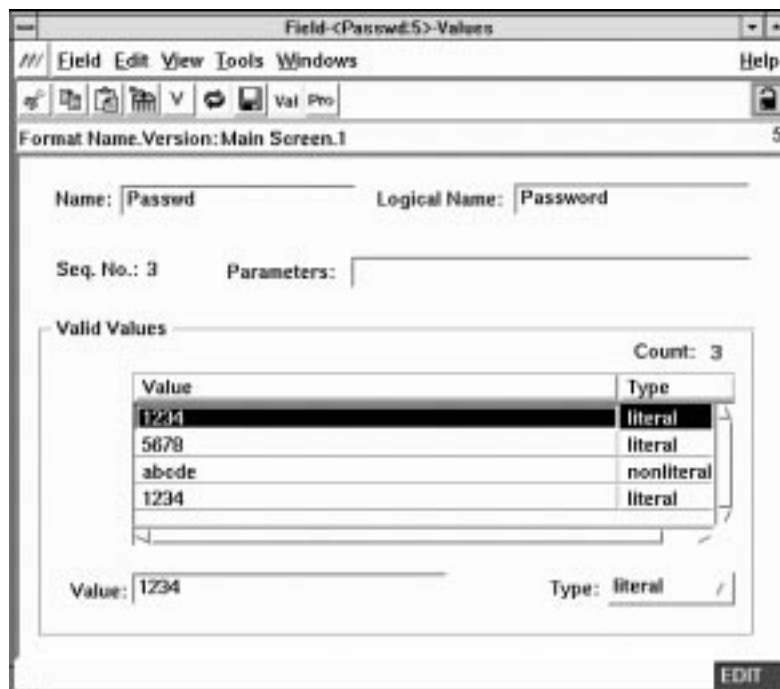
1. On an unlocked Field object, select an existing value and execute

**Edit->Copy**

2. Execute

**Edit->Paste**

The copied value appears in the **Valid Values** list. (See [Figure 7-6](#).)



**Figure 7-6.** Pasting a Copied Value

3. Change the text in the **Value** text box. (All values within a Field object must be unique.)

**NOTE** — If you save the Field object without changing the value text, the error dialog in [Figure 7-2](#) appears telling you that the value text is not unique.

#### 7.2.1.1.4 Deleting Values

To delete a Value row

1. On an unlocked Field object, select an existing row.
2. Execute

##### **Edit->Delete**

If the value is not currently being used by a Compound or Relation, the selected row is removed from the display. If the value is used by a Compound or Relation, an error dialog similar to the one in [Figure 7-7](#) appears informing you what Compound and/or Relation(s) use the value.



**Figure 7-7.** Deleting a Value Error Dialog

#### 7.2.1.2 Specifying Parameters

You can specify a list of parameters that provide more information about the Field. For example, you might enter *Row=1, Col=1* indicating that this Field is in the upper left hand corner of a screen.

The **Parameters** attribute is free form text so it may be used in any way you desire.

## 7.2.2 Properties View

The Field object Properties view (Figure 7-8) lets you enter a Description for the Field object.



Figure 7-8. Field Object Properties View

## 7.2.3 Printing a Field Object

To print a Field object, execute

```
xmyPrintFormatFields format_name.version_number
```

where *format\_name* is the name you entered for a Format object and *version\_number* is the system generated version number of the Format object.

**NOTE** — If a Format object name contains embedded spaces, replace the spaces with a backslash-escaped space character (`\` ).

For example, to print the Field object in Figure 7-8, you would execute

```
xmyPrintFormatFields Main\ Screen.1
```

For more information on **xmyPrintFormatFields**, see Section 11.3.

## 8. Compound Object

The Compound object lets you collect several Fields and treat them as one Field.

**NOTE** — Compound objects are not required. However, using Compounds can help simplify your test cases.

### 8.1 Why use a Compound Object

By collecting several Fields and treating them as one, a Compound object simplifies Formats having many blocks of interrelated fields. A Compound object can reduce the complexity of relations in the Relation Object. (See [Section 10.](#))

### 8.2 Working with Compound Objects

You can create a Compound object when you have created more than one Field object. Simply select the Fields you want added to the Compound object and execute

#### **Format->New Compound**

You can create any number of Compound objects for a Format, but once a Field is used in a Compound it is no longer eligible for use in another Compound.

Once you have created a Compound object, you must create at least one Tuple for the Compound object by executing

#### **Compound->Generate Tuple**

You can not save a Compound until you generate at least one Tuple. A **Tuple** is a reference name given to the specific value set of a Compound. This symbolic value can be expanded into the specific values of the simple Fields represented by the compound. (See [Section 8.2.3.](#))

If Tuples do exist but you have modified the Fields or Values and have not regenerated the Tuples, the AETG System ask you if want to save the changes and regenerated the Tuples.

The Compound object has three views:

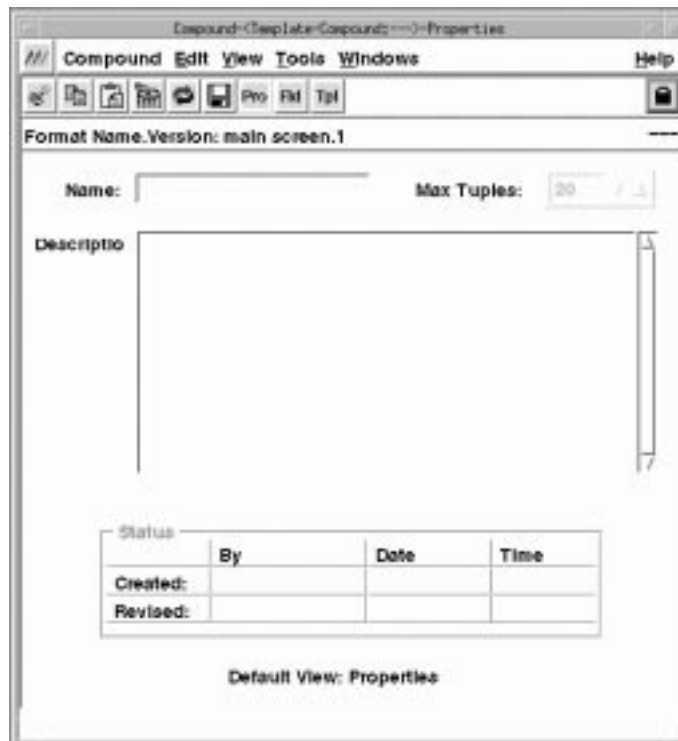
- |                   |   |
|-------------------|---|
| <b>Properties</b> | Used to define the Properties for the Compound object. (See <a href="#">Section 8.2.1.</a> )      |
| <b>Fields</b>     | Used to select the values that participate in the Compound. (See <a href="#">Section 8.2.2.</a> ) |
| <b>Tuples</b>     | Used to create Tuples for the Compound object. (See <a href="#">Section 8.2.3.</a> )              |

The default view is the Properties view ([Figure 8-1](#)).

---

## 8.2.1 Properties View

The Compound object Properties view (Figure 8-1) lets you enter the name and a description of a Compound object. You can also select a maximum Tuples (Max Tuples) value.



**Figure 8-1.** Compound Object Properties View

The Properties view contains the following parameters:

**Name** Lets you enter a name for the Compound object.  
This is a required field.

- Max Tuples** Determines the maximum number of Tuples that the AETG System generates in the Tuples view.
- See [Section 8.2.3.1](#) for information on generating Tuples.
- Valid values are in the range *1* to *30*.
- Default = *20*
- Note** — The system will not allow more than 30 Tuples because this would cause too many test cases in the ultimate Test Matrix.
- To specify a **Max Tuple** value, perform either of the following:
- Click on the Spin Button arrows to change the value
  - Type the value directly in the text box.
- Description** Lets you enter a description for the Compound object, e. g., the purpose or contents of the Compound.
- Status** Displays who create or revised the Compound object and when the Compound object was created or revised.
- This is a display only field.

## 8.2.2 Fields View

The Fields view (Figure 8-2) lets you select the fields and values that participate in the Compound.

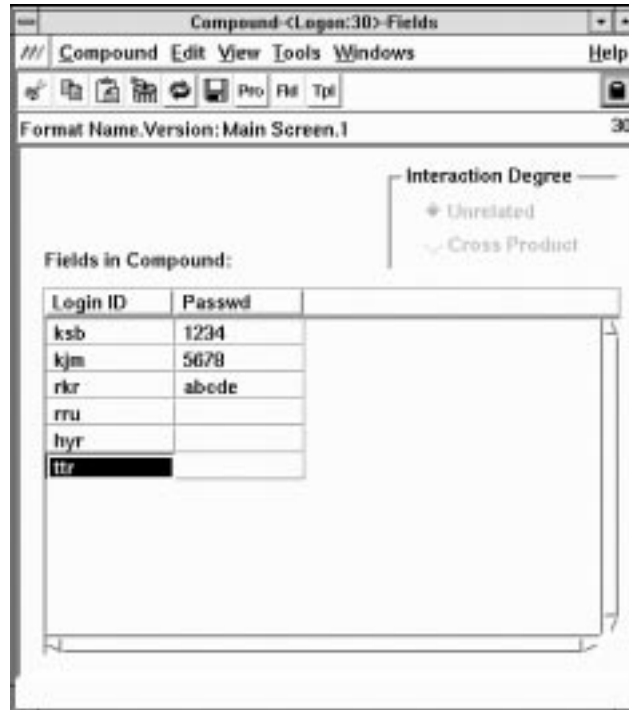


Figure 8-2. Compound Object Fields View

The Fields view contains the following areas:

**Interaction Degree** Lets you specify how the fields will interact with each other when the system generates Tuples. See [Section 8.2.2.2](#).

Click on one of the following radio buttons:

- **Unrelated**
- **Cross Product.**

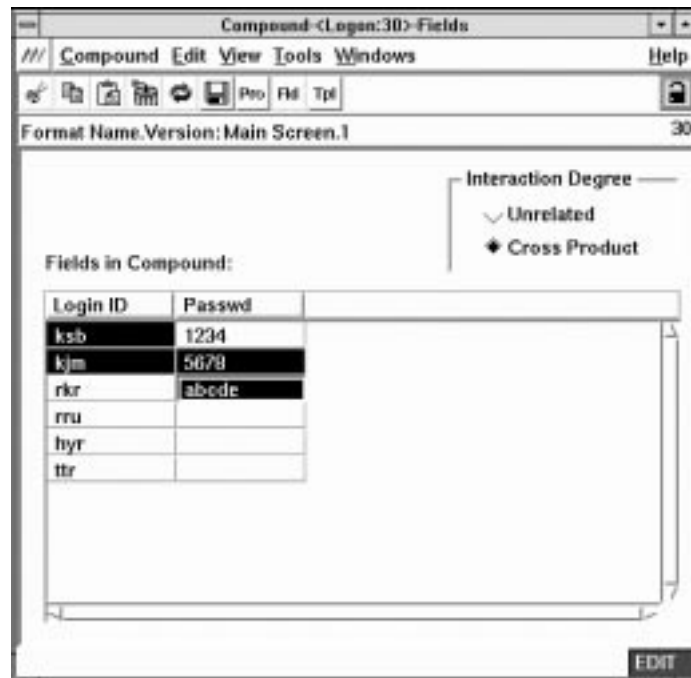
The default setting is **Unrelated**.

**Fields in Compound** Lists all of the Fields that were selected on the Format window when you created the Compound object. All values for these Fields are also populated.



### 8.2.2.1 Selecting Values to Participate in a Compound

You **Ctrl-Click** on values from each Field's value list (Figure 8-3) to multiselect the values to be assigned to the Fields for the current Compound when you generate Tuples for this Compound.



**Figure 8-3.** Multiselecting Values to be used in a Compound

**NOTE** — See [Section 4.11](#) for information on how to multiselect items.

**NOTE** — At least one value must be chosen from each Field in the Compound.

To clear all selections, execute

**Edit->Deselect All**

**NOTE** — When selecting values to participate and generating Tuples for those values, it is important to attempt to minimize the number of values you choose. A large number of values may produce a large number of Tuples, which could lead to more complexity in the ultimate Test Matrix rather than simplifying it, defeating the purpose of the Compound field.

### 8.2.2.2 Selecting an Interaction Degree

You can specify the degree of interaction between selected values that the system will use during a **Generate Tuples** operation ([Section 8.2.3.1](#)). This is done using one of the following radio buttons on the Fields views ([Figure 8-2](#)).

- |                      |  |
|----------------------|--|
| <b>Unrelated</b>     | <b>Generate Tuples</b> produces enough Tuples to include each of the values at least once. |
| <b>Cross Product</b> | <b>Generate Tuples</b> produces enough Tuples to include every combination of every value. |

### 8.2.3 Tuples View

The Tuples view (Figure 8-4) shows the Tuples that have been generated for the Compound object. See Section 8.2.3.1 for information on generating Tuples.



Figure 8-4. Compound Object Tuples View

The Tuples View contains the following parameters

- Tuples** A list containing a column listing the Name of the Tuple and a column for each value in the Tuple.
- Tuple Name** A text box displaying the name of a Tuple you selected in the **Tuples** list. The text box lets you change the name of the Tuple. (See Section 8.2.3.2.)
- Count** Displays the total number of generated Tuples.

Each Tuple is made up of a Name (which is initially generated by the AETG System) and one value for each of the Fields in the Compound. These Tuples are the values for the Compound.

For example, if you

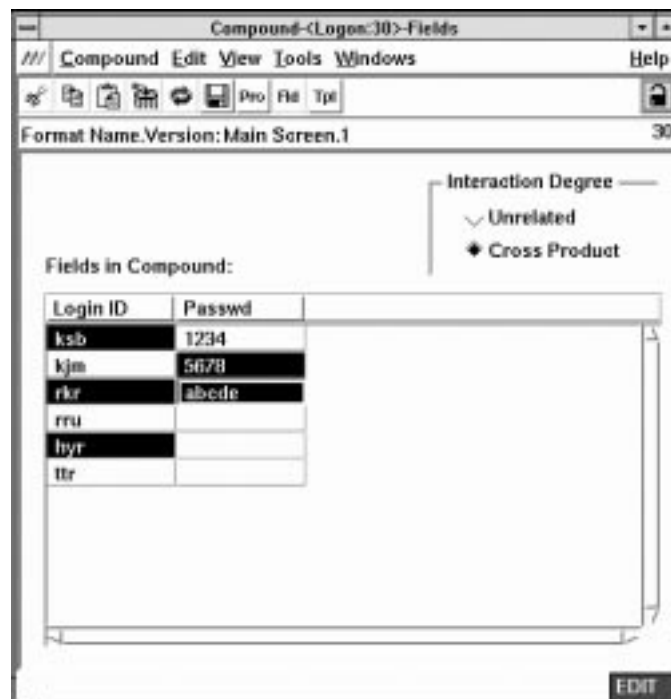
1. Have three fields (e.g., F1, F2, F3) and two values for each field
2. Select all of the values
3. Generate Tuples using **Cross Product**.

the AETG System generates eight (8) Tuples (e.g., value combinations). At this point, you can consider the Compound as one Field having eight possible values.

### 8.2.3.1 Generating Tuples

To generate a Tuple

1. On the Compound object Fields view ([Figure 8-5](#))
  - Multiselect your desired values for each Field.
  - Choose the Interaction Degree.

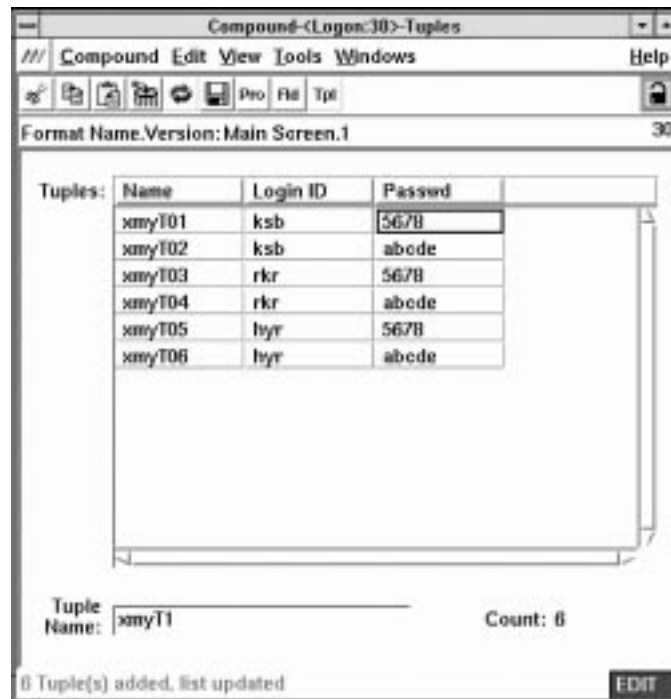


**Figure 8-5.** Selecting Fields and Interaction Degree for Generating Tuples

2. Execute

**Compound->Generate Tuples**

The AETG System generates the appropriate Tuples (Figure 8-6) based on your selections on the Fields view.



**Figure 8-6.** Generated Tuples

**NOTE** — The Tuples view does not appear automatically when you generate Tuples. To view the Tuples, you must either execute

**View->Change Views->Tuples**

or click on the **Tpl** button.

Generating Tuples can be an iterative process or it can be done all at once.

For example, if you selected all values in a Compound and chose the **Cross Product** Interaction Degree, then **Generate Tuples** produces the maximum possible number of Tuples for the value set. In this case, there would not be any point in generating more Tuples. In fact, if you tried to generate new Tuples, the new Tuples would be truncated.

However, if you want to be more selective about the Tuples to be generated, then you can generate the Tuples iteratively by

1. Selecting a few Values.
2. Generating the Tuples.
3. Selecting a different set of Values.
4. Repeating Steps 2 and 3 until you are satisfied with the resulting Tuple set.

In the iterative case, **Generate Tuples** appends any new Tuples that are generated to the list of already existing Tuples. You can also choose a different **Degree of Interaction** for each iteration. Therefore, the setting of Degree of Interaction only has meaning for a particular run of **Generate Tuples**.

For each run of **Generate Tuples**, all duplicate Field/Values are discarded. However, the AETG System will *not* discard any duplicate combinations when appending Tuples to an existing Tuple list.

For example, if you generate a set up Tuples using the selections from [Figure 8-5](#) and generate a new set of Tuples without changing the selected Values, the AETG System appends the Tuples list with a set of identical Field/Value combinations, but each new combination has a new Tuple name, as shown in [Figure 8-7](#).

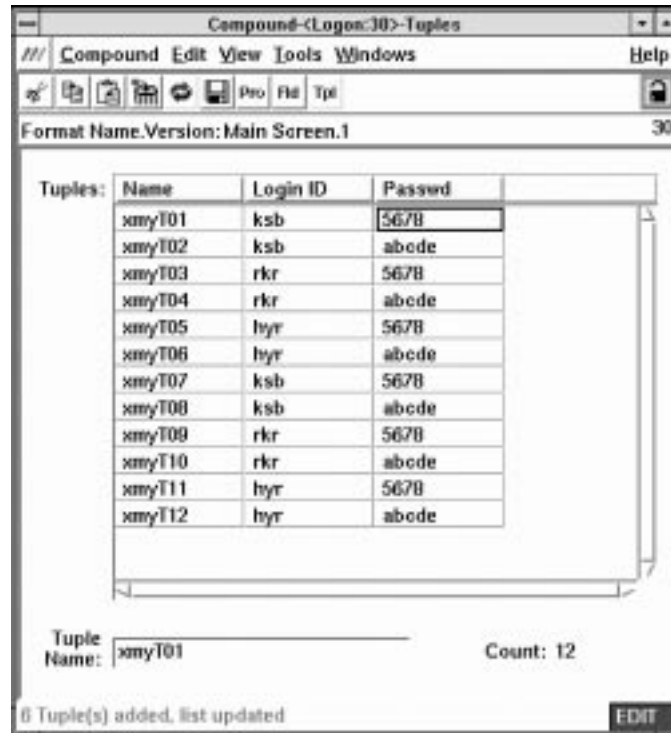


Figure 8-7. Regenerating an Identical Tuple Set

When generating Tuples, the system generates only the number of Tuples specified by **Max Tuple Count**. Once this number is reached, all other Tuples are truncated. A confirm dialog (Figure 8-8) appears if more than the maximum allowed Tuples are generated.

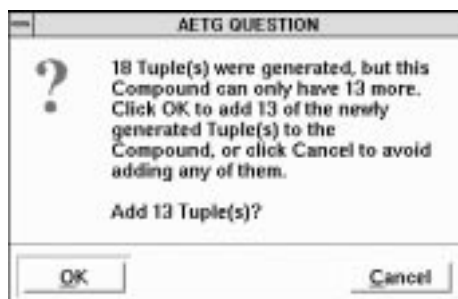


Figure 8-8. Generating Tuples Confirm Dialog

You are asked if you want to add any Tuples, and, if so, whether to add Tuples up to the **Max Tuple Count** setting.

If you have already generated the maximum number of tuples for this Compound and you attempt to add more Tuples, a warning dialog (Figure 8-9) appears.

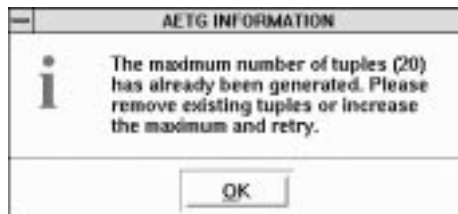


Figure 8-9. Maximum Tuples Confirm Dialog

### 8.2.3.2 Changing a Tuple's Name

The Tuple's name is initially assigned by the system and takes the form *xmyT1*, *xmyT2*, ..., *xmyTN*, where *N* is the number of Tuples generated by the system. These names do not have any particular meaning.

The names of Tuples are displayed in a Relation object and represent each value set for the Compound. You may find it useful to provide a more meaningful name for one or more of the Tuples.

The text box at the bottom of the window on the Tuple view is populated with the Name of the currently selected Tuple if one, and only one Tuple row is selected.



You can type a new Name for the Tuple in this text box. Once you leave this field (by tabbing or by clicking somewhere else on the window) the new Tuple Name is populated in the Tuple row. The system validates the name you supplied to make sure it is unique within this Compound and that it does not begin with *xmyT*.

To change the name of a Tuple

1. Click on one (and only one) Tuple in the **Tuples** list.
2. Click in the **Name** text box and delete the existing name.
3. Type a new Name.
4. Press the **Tab** key or click anywhere else on the Tuples view.

### 8.2.3.3 Regenerating System Generated Tuple Names

To regenerate all system generated names of Tuples, execute

#### **Compound->Rename Tuples**

This does not modify user edited names.

As noted before, the AETG System generates Tuples names in the form *xmyT1*, *xmyT2*, ... *xmyTN*, appending an iterated number to the suffix *xmyT*. If you used an iterative process to generate Tuples, then the system, when adding additional Tuples to the list, first finds the largest number in a system generated Name and then increments the iterated number for the new names.

If you delete one or more Tuples during the iterative process, then there will be gaps in the system generated names. For example, you may choose to keep only Tuples *xmyT3*, *xmyT9*, and *xmyT22*, and you may have edited one or more Tuple names. A sample list of names might be

- xmyT3
- xmyT9
- hsbMain
- xmyT22.

If you execute **Rename Tuples**, the resulting set of Names is

- xmyT1
  - xmyT2
  - hsbMain
  - xmyT3.
-

#### 8.2.3.4 Deleting Tuples

To delete Tuples

1. Select or multiselect rows in the Tuples view
2. Execute

##### **Edit->Delete**

If the Tuple is not used by a Relation, the system removes the rows from the display and updates the Count field. If the Tuple is used, an error dialog (Figure 8-10) appears telling the you which Relation(s) use this Tuple.



**Figure 8-10.** Deleting Used Tuple Error Dialog

### 8.3 Saving a Compound

You cannot save a Compound until at least one Tuple has been generated.

If Tuples do exist but you have subsequently modified the Fields or values and have not regenerated the Tuples, the system will ask you if you would like to save the changes and the regenerated Tuples. You can also decide to not save the changes.

An initial Save operation updates the Compound row in the Fields view of the Format (Figure 6-2).

## 9. Test Object

Test objects define the means by which a requirement is verified in a software application.

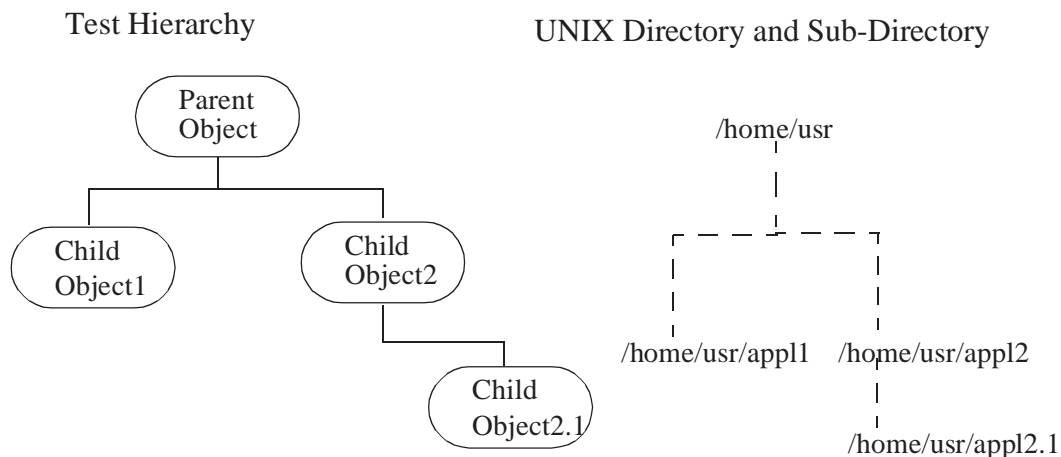
### 9.1 Why use a Test Object

You use a Test object to create

- Relation objects, which are used to define the
  - Constraints on the combinations of Fields that are tested
  - Actual values that are used in a set of generated test cases.
- Matrices, which let you generate or view the AETG generated set of test cases.

### 9.2 Using Test Hierarchies and Creating Test Objects

Test objects always appear in their own Test hierarchies. Test hierarchies are useful for organizing your work. Hierarchies are relationships between Test objects organized into levels similar to a UNIX directory structure, which has directories and sub-directories. Hierarchies are comprised of “parent” objects and “child” objects and can have several layers, i.e., child objects with other child objects below it. (See [Figure 9-1.](#))



**Figure 9-1.** *Hierarchy Concept*

Test hierarchies contain only Test objects, and Test hierarchies are the only place in the AETG System where you can place Test objects. You can, however, have several Test hierarchies and you can place the hierarchies in folders you create.

Figure 9-2 illustrates a Test object hierarchy. Hierarchies are most obvious when we look at them in an expanded list view.

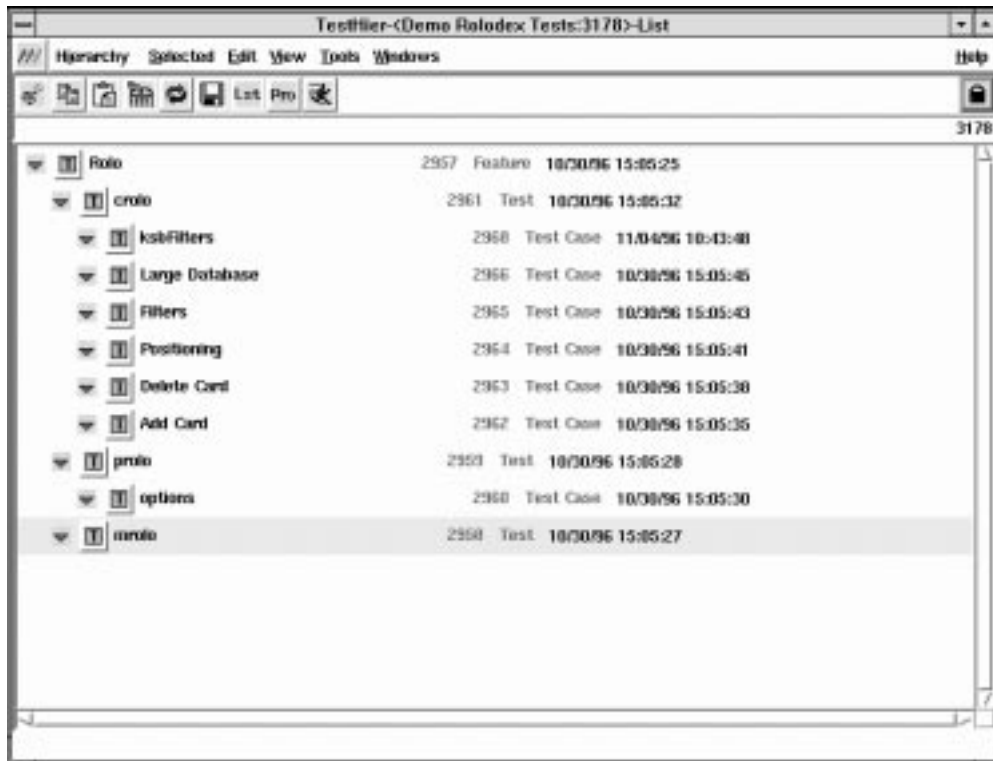


Figure 9-2. Test Object Hierarchy

### 9.2.1 Creating and Naming Test Hierarchies

You can create a Test hierarchy from the AETG Desktop or any folder you have created.

To create a new hierarchy

1. On the AETG Desktop, execute

**AETG->New->Test Hierarchy**

**NOTE** — To create a hierarchy in an open Folder, execute

**Folder->New->Test Hierarchy**

The system places the new Test hierarchy in the topmost position of the Desktop or Folder as show in [Figure 9-3](#).



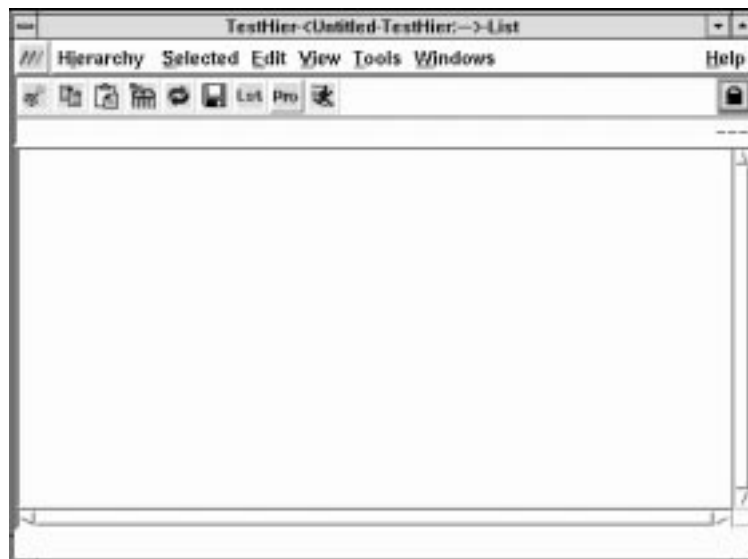
**Figure 9-3.** *New Test Hierarchy.*

**NOTE** — To create a hierarchy in a closed Folder, select the Folder and execute

**Selected->New->Test Hierarchy**

2. Double click on the new hierarchy icon to open it.

The system displays the Test hierarchy List view ([Figure 9-4](#)).

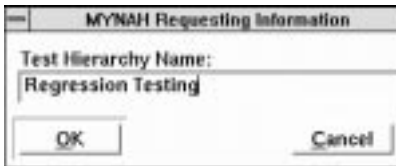


**Figure 9-4.** *Test Hierarchy List View*

3. Click on the **Lock** icon to unlock the hierarchy.

The system displays a Name dialog for the hierarchy you created.

4. Type in a name for the new Test hierarchy, e.g., *Regression Testing*, and click **OK**. (See [Figure 9-5](#).)



**Figure 9-5.** *Test Hierarchy Name Dialog.*

5. Save the hierarchy by executing  
**Hierarchy->Save**

### 9.2.2 Deleting Hierarchies

You delete hierarchies the same way you delete any other AETG object. (See [Section 4.7](#).) The only people who can delete a hierarchy is the user who created it or the AETG Administrator.

However, since hierarchies contain other objects you must be careful. When you delete a hierarchy you are deleting all the objects in the hierarchy from the AETG Database, no matter who owns them! The system always reminds you of this with a confirmation dialog.

### 9.2.3 Creating Test Objects in Hierarchies

Test objects can be created at the first level or at any level below that in a hierarchy. We continue the example we started in [Section 9.2.1](#) by creating Test objects in the Test hierarchy we created.

To create an object at the first level of a hierarchy:

1. Open the hierarchy in which you are placing the new object.
2. Make sure that no other objects are selected. If you need to, execute

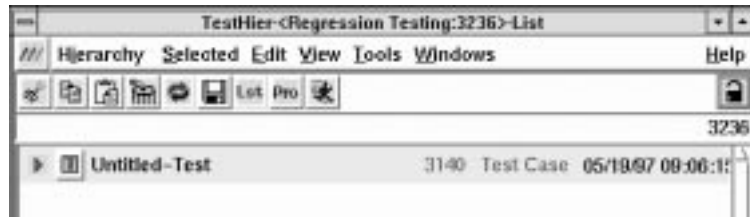
**Edit->Deselect-All**

to clear all selections.

3. Unlock the hierarchy by clicking on the **Lock** icon.
4. Execute

**Hierarchy->New Test**

The system places the new untitled Test object in the hierarchy. (See [Figure 9-6](#).)



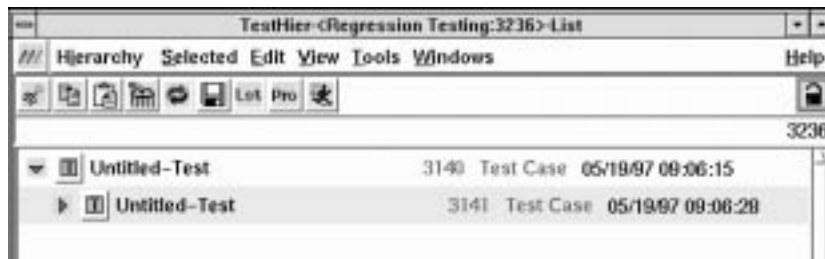
**Figure 9-6.** *New Test Object Dialog*

To place a new object at a level below the first level of a folder or hierarchy,

1. Select the object *below* which you want the new object to appear. For example, select the object you just created.
2. Execute

**Hierarchy->New Test**

The system places the new object in the Test hierarchy below the first Test object ([Figure 9-7](#)). Note that the new Test object is indented, indicating that it is a child object of the original Test object.



**Figure 9-7.** *New Object at Second Level*

If you want to place another object at the first level

1. Make sure that no other objects are selected. If you need to, execute

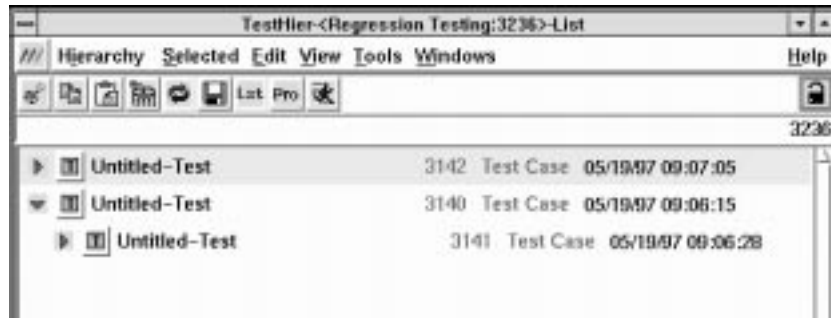
**Edit->Deselect-All**

to clear all selections.

2. Execute

**Hierarchy->New Test**

The system placed a new untitled Test object in the hierarchy. (See [Figure 9-8](#).)



**Figure 9-8.** *Another New Object at First Level*

3. Save the hierarchy by executing

**Hierarchy->Save**

**NOTE** — You can not open Test objects in an unlocked Test hierarchy.

Normally, you would open the new Tests and fill in their attributes at this point.

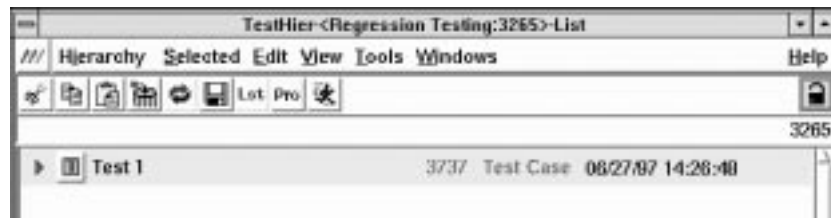
#### 9.2.4 Duplicating a Test Object

When you want to copy a Test object, you must use the **Duplicate** option on the **Selected** menu. **Duplicate** creates a new object with many of the attributes of the original.

If you duplicate a Test object, the system gives the new Test object the name of **Cloneof <id>**, where **<id>** is the ID number of the original Test object. The system assigns the duplicate Test object a new **ID** number so that you won't overwrite the original object.

As an example we will duplicate a Test object within a Test hierarchy.

1. Click on the **Lock** icon to unlock the hierarchy.
2. Select the Test Object you want to duplicate. (See [Figure 9-9](#).)



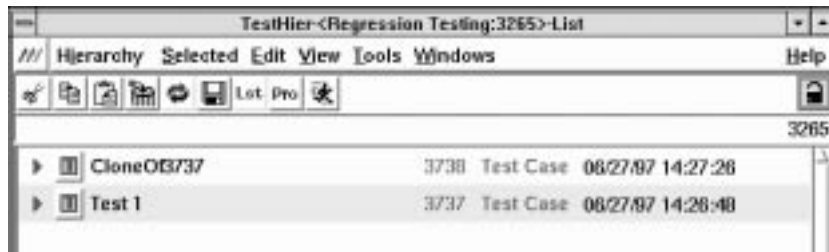
**Figure 9-9.** *Selecting a Test Object to Duplicate*



3. Execute

**Selected->Duplicate**

The system places an object labeled **Cloneof** in the hierarchy as a sibling of the object you duplicated. (See [Figure 9-10](#).)



**Figure 9-10.** *Duplicated Test Object*

[Table 9-1](#) lists the attributes that are and are not duplicated in the new Test object.

**Table 9-1.** *Duplicated/Nonduplicated Test Object Attributes*

Duplicated	Not Duplicated
<ul style="list-style-type: none"><li>• Owner</li><li>• Version number</li><li>• Test case type</li><li>• Description</li></ul>	<ul style="list-style-type: none"><li>• Name</li><li>• Matrices</li><li>• Associated Format object</li><li>• Relations</li></ul>

### 9.3 Working with Test Objects

You can create one or more Test objects to test the parts of a Format that you wish to test. For example, you can test a subset of Fields on a Format with one Test object and another subset using another Test object. You then associate a Format with each Test object that validates some aspect of the Format.

You create Test objects within Test hierarchies, as detailed in [Section 9.2](#). These hierarchies let you refine the area of a Format each Test object validates.

For each Test object, you define the set of rules that are exercised by the Test in one or more objects called Relation objects.

**NOTE** — Before any Relations can be defined, the Test must have an associated Format object.

While in the Relations view of the Test object, you can either open an existing Relation object for editing or create a new one.

Once you've specified all of the Relations for a Test, you can have the AETG System generate the efficient set of test cases for this Test.

The Test object contains the following views:

<b>Properties</b>	Used to define the Properties for the Test object. (See <a href="#">Section 9.3.1</a> .)
<b>Relations</b>	Used to define the Relations for this Test object, including associating a Format object with the Test. (See <a href="#">Section 9.3.3</a> .)
<b>Matrix</b>	Used to display the generated test cases. (See <a href="#">Section 9.3.2</a> .)

The default view is the Properties view ([Figure 9-11](#)).

### 9.3.1 Properties View

The Test object Properties view (Figure 9-11) lets you enter or update such information as the Name, Owner, and Type of a Test object.

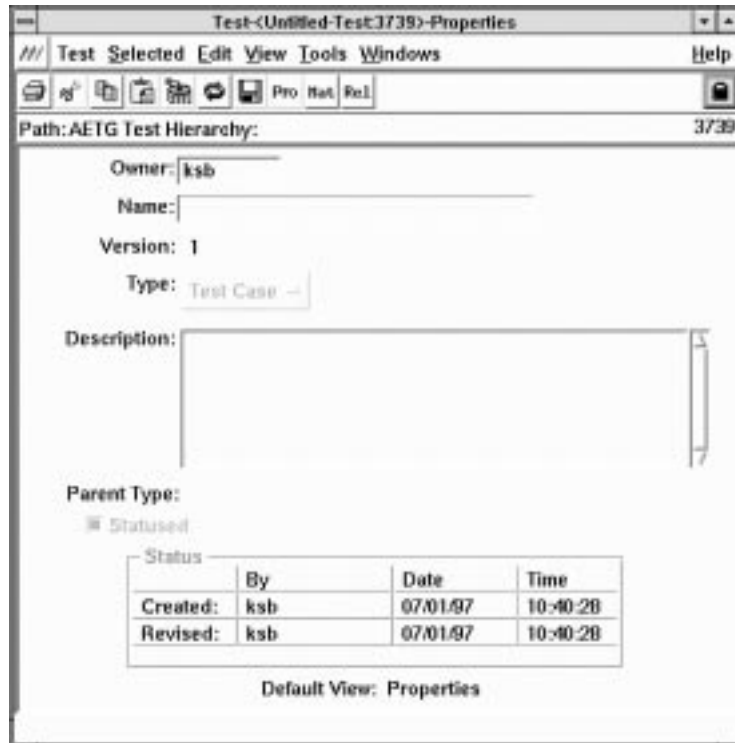


Figure 9-11. Test Object Properties View

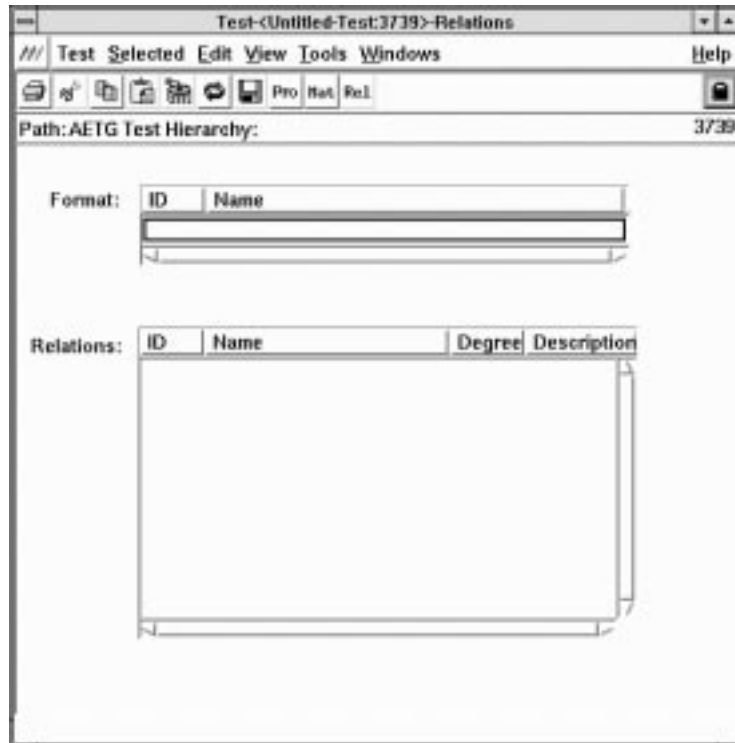
The Properties view contains the following parameters:

- Owner** Lets you specify the owner of the Test object.  
This is a required parameter, and is initially prepopulated with the login ID of the person who created the object. You can change the owner of an object in the text box.
- Name** Lets you specify the name of a Test object, e.g., *login*.  
This is a required parameter.

<b>Version</b>	<p>Displays the version number of the Test object, which is generated by the AETG System. Note that, in the AETG System, this is always initially be set to 1.</p> <p>This is a display only field</p>
<b>Type</b>	<p>Lets you specify the type of test you are creating, which helps you organize tests. You must select one of the values from the drop-down menu. The delivered values are</p> <ul style="list-style-type: none"><li>• <b>Area</b></li><li>• <b>Feature</b></li><li>• <b>Test</b></li><li>• <b>Test Case</b></li></ul> <p>The default value is <b>Test Case</b>.</p>
<b>Description</b>	<p>Lets you enter a description of the test.</p>
<b>Parent Type</b>	<p>Displays the parent Test object type, if any, for the current object.</p>
<b>Stateded</b>	<p>This parameter is not supported for this release of the AETG System.</p>
<b>Status</b>	<p>Displays who create or revised the Test object and when the Test object was created or revised.</p> <p>This is a display only field</p>

### 9.3.2 Relations View

The Test object Relations view (Figure 9-12) lets you associate a Format object to the Test and to view or create a set of Relations involving the Fields in that Format.



**Figure 9-12.** Test Object Relations View

The Relations view contains the following parameters:

**Format** Lets you associate a Format object with this Test object. This list displays the ID, Name, and Version number of the Format object.

See [Section 9.3.2.1](#) for instructions on associating a Format object with a Test object.

**Relations** Contains a list displaying all Relations defined for this Test object. Using this list you can create new Relation objects or duplicate or delete existing objects.

Each line of the list displays the ID, Name, Degree, and Description of the Relations object.

**Note** — See [Section 10](#) for detailed information on Relation objects.

### 9.3.2.1 Associating a Format Object

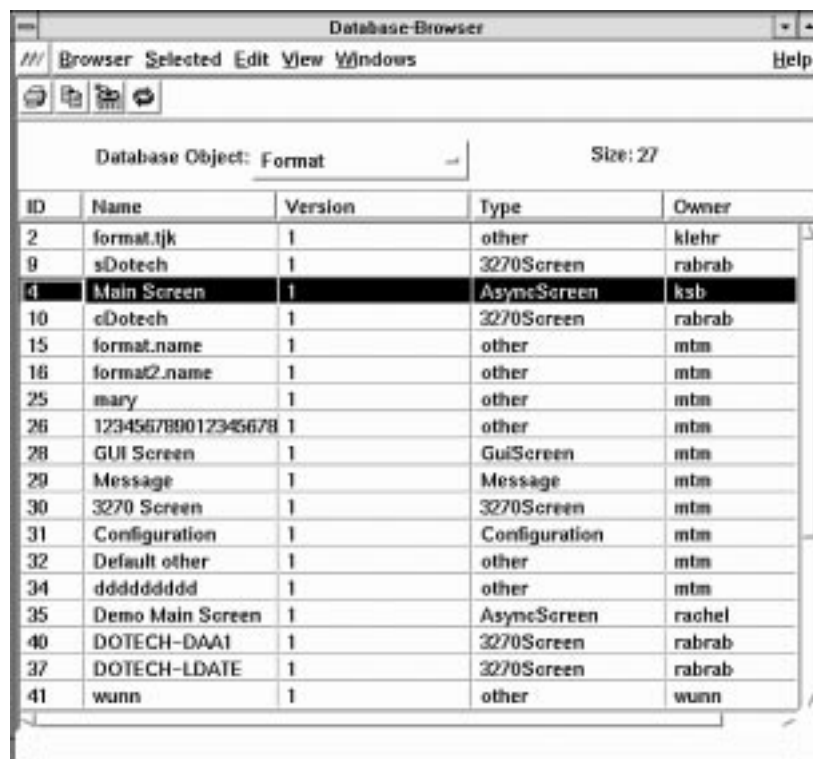
When you create a test case, you must associate a Format with a Test object, which gives the Test object access to the Fields and Values defined by the Format object.

**NOTE** — A Test object can only be associated with one Format at a time. However, a Format can be associated with more than one Test object at a time.

To associate a Format object to a Test object

1. Display the Database Browser, and display the list of Format objects.
2. Select a Format object from the **Database Browser** list (Figure 9-13) and execute

**Edit->Copy**



The screenshot shows a window titled "Database-Browser" with a menu bar (Browser, Selected, Edit, View, Windows, Help) and a toolbar. Below the toolbar, there is a dropdown menu showing "Database Object: Format" and a "Size: 27" indicator. The main area contains a table with the following data:

ID	Name	Version	Type	Owner
2	format.tjk	1	other	klehr
9	sDotech	1	3270Screen	rabrab
4	Main Screen	1	AsyncScreen	ksb
10	cDotech	1	3270Screen	rabrab
15	format_name	1	other	mtm
16	format2_name	1	other	mtm
25	mary	1	other	mtm
26	123456789012345678	1	other	mtm
28	GUI Screen	1	GuiScreen	mtm
29	Message	1	Message	mtm
30	3270 Screen	1	3270Screen	mtm
31	Configuration	1	Configuration	mtm
32	Default other	1	other	mtm
34	dddddddd	1	other	mtm
35	Demo Main Screen	1	AsyncScreen	rachel
40	DOTECH-DAA1	1	3270Screen	rabrab
37	DOTECH-LDATE	1	3270Screen	rabrab
41	wunn	1	other	wunn

**Figure 9-13.** Selecting a Format to Associate with a Test Object

**NOTE** — You can also select and copy the desired Format object from the AETG Desktop.

3. Display the Test object and click on the **Lock** icon to unlock the object (if it is not already unlocked).

4. Click in the **Format** parameter box, and execute

**Edit->Paste**

on the Test object Relations View to create the association. (See [Figure 9-14.](#))



**Figure 9-14.** *Associating a Format with a Test Object*

### 9.3.2.2 *Creating a New Relation*

To create a new Relations object in an unlocked Test object Relations view, execute

**Test->New Relation**

**NOTE** — The Test object must have an associated Format object, and there cannot be an untitled Relation row in the list.

An untitled row is added to the **Relations** list. The new Relation object is automatically opened and unlocked so that you can provide a Name. (See [Figure 9-15](#).)



**Figure 9-15.** *Creating a New Relation Object*

**NOTE** — See [Section 10](#) for detailed information on using Relation objects.

### 9.3.2.3 *Duplicating a Relation*

When you want to copy a Relation object, you must use the **Duplicate** option on the **Selected** menu. **Duplicate** creates a new object with many of the attributes of the original.

If you duplicate a Relation object, the system gives the new Relation object the name of **CloneOf <id>**, where **<id>** is the ID number of the original Relation object. The system assigns the duplicate Relation object a new **ID** number so that you won't overwrite the original object.



To duplicate a Relation

1. On an unlocked Test object Relations view, select the Relation you want to duplicate.
2. Execute

**Selected->Duplicate**

The system creates and opens a new Relation object that is a duplicate of the selected Relation. (See [Figure 9-16.](#))



**Figure 9-16.** Duplicate Relation Object

Table 9-2 lists the attributes that are and are not duplicated in the new Relation.

**Table 9-2.** *Duplicated/Nonduplicated Relation Object Attributes*

<b>Duplicated</b>	<b>Not Duplicated</b>
<ul style="list-style-type: none"><li>• The set of Fields in the original Relation</li><li>• The associated Test</li><li>• The associated Format</li><li>• Interaction degree</li></ul>	<ul style="list-style-type: none"><li>• Name</li><li>• Field Values</li><li>• Description</li><li>• Includes</li><li>• Excludes</li><li>• Invalids</li></ul>

Once you have entered a new Name, you can save the duplicated Relation. The new Relation appears in the **Relation** list on the Test object Relations view.

#### 9.3.2.4 *Deleting a Relation*

In an unlocked Test object

1. Select one or more Relation rows.
2. Execute

**Edit->Delete**

### 9.3.3 Matrix View

The Test Object Matrix view (Figure 9-17) lets you

- Generate or view the AETG generated set of test cases.
- Delete test cases from the displayed Matrix.
- Save the Matrix to a file.
- Load a previously saved matrix.
- Delete previously saved matrices.

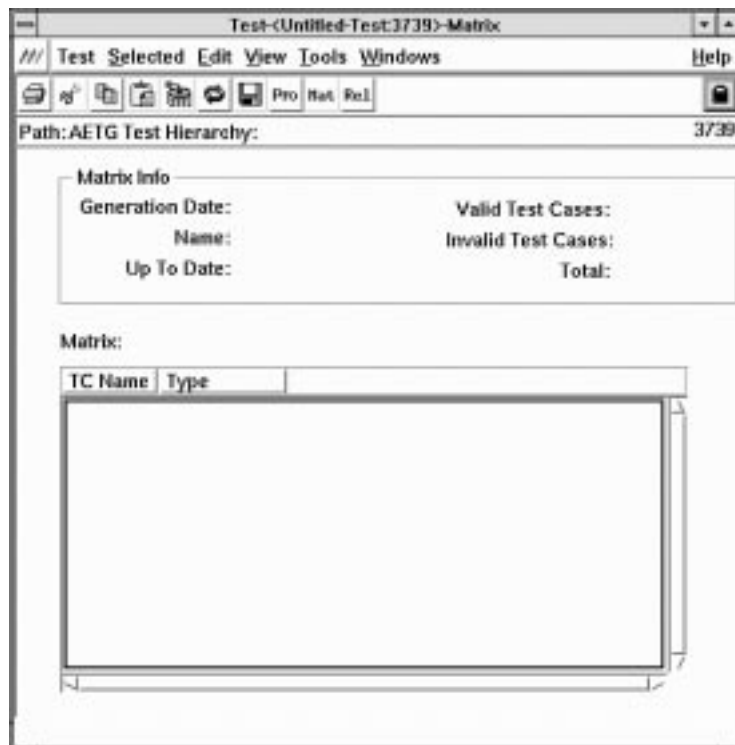


Figure 9-17. Test Object Matrix View

The Matrix view is populated when you execute

**Test->Matrix->Generate**

The displayed Test Case Matrix is the *efficient* set of test cases that is produced by the **Generate** function for the set of Relations defined for this Test.

The Matrix view contains the following parameters:

<b>Matrix Info</b>	Displays the following information:
<b>Generation Date</b>	The date and time when the Matrix was generated.
<b>Name</b>	The name of the Test Case Matrix. This is marked <i>&lt;untitled&gt;</i> if the displayed matrix has not been saved. When you save the Matrix, you are asked to enter a name for the matrix. (See <a href="#">Section 9.3.3.3.</a> )
<b>Up To Date</b>	Indicates whether or not the involved Relations have changed since the displayed Matrix was generated.
<b>Valid Test Cases</b>	The number of valid generated test cases.
<b>Invalid Test Cases</b>	The number of invalid generated test cases.
<b>Total</b>	The total number of generated test cases.
<b>Matrix</b>	Contains a list box that displays the Test Case Matrix as a set of rows. Each test case row contains the following columns:
<b>TC Name</b>	Displays the name of the test case. The system automatically generates a name using the forms <i>V-N</i> or <i>I-N</i> , where
	<i>V</i> Indicates that this is a valid test case
	<i>I</i> Indicates that this is a invalid test case
	<i>N</i> Is an automatically iteration number. For each test case type, <i>N</i> is initially <i>0</i> , e.g., <i>V-0</i> , <i>I-0</i> .
<b>Type</b>	Indicates whether this test case is valid ( <i>V</i> ) or invalid ( <i>I</i> ).
<b>&lt;Value&gt;</b>	Displays a value for each participating Field in the test case.

### 9.3.3.1 Generating a Matrix

The **Generate** menu function is enabled only when at least one Relation is defined for the Test object.

When you execute

**Test->Matrix->Generate**

the system displays the message

Test Matrix being generated...

in the Relations view status area. When the operation is completed, the set of generated test cases is displayed in the **Matrix** list (Figure 9-18) and the message

Test Matrix completed - N test cases generated

appears in the message area, where *N* is the number of generated test cases.



**Figure 9-18.** *Generated Test Case Matrix*

Any existing test cases in the list are overwritten by the new set of test cases.

### 9.3.3.2 Deleting Test Cases

To delete test cases

1. Select or multiselect test case rows (Figure 9-19).



Figure 9-19. Selecting Test Case Rows to Delete

2. Execute

**Edit->Delete**

The display-only test case counts are updated (Figure 9-20).



**Figure 9-20.** Updated Matrix View

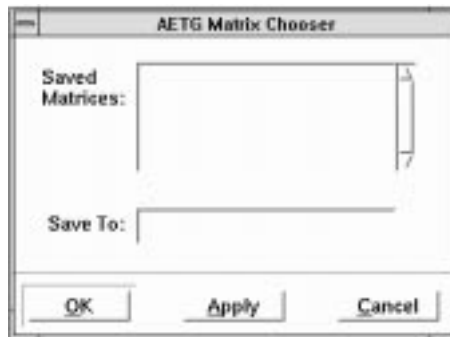
### 9.3.3.3 Saving a Matrix

You can save a Matrix when there are test cases displayed in the **Matrix** list.

1. Execute

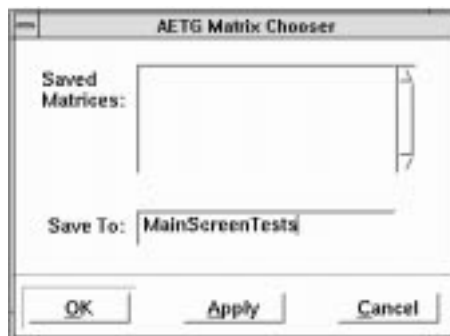
**Test->Matrix->Save**

The Save Matrix Dialog (Figure 9-21) appears.



**Figure 9-21.** Save Matrix Dialog

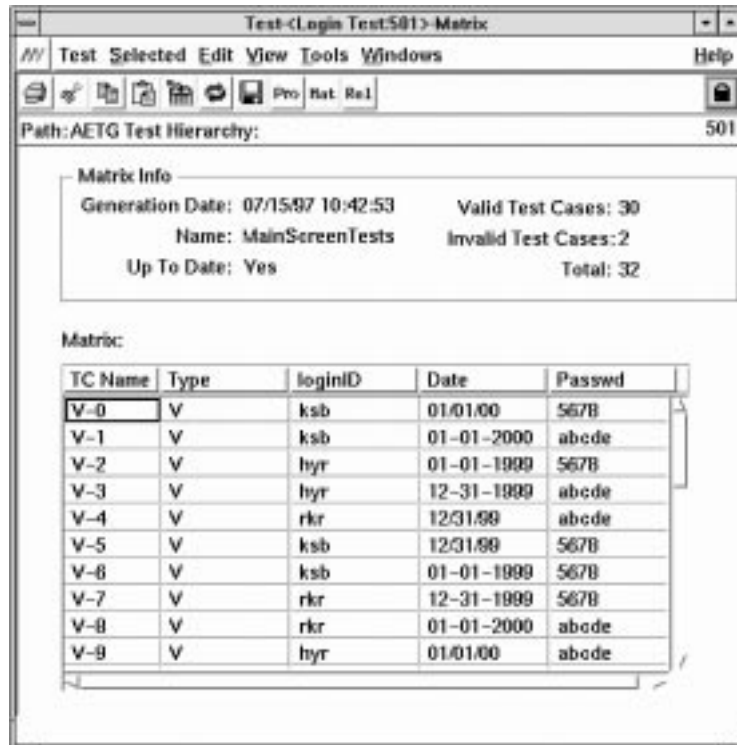
2. Enter a name for the Test Case Matrix file, e.g., *MainScreenTest* (See Figure 9-22), and click on OK.



**Figure 9-22.** Entering a Name for a Test Case Matrix File



The filename you entered in the **Name** field of the **Matrix Info** parameter (Figure 9-23).



**Figure 9-23. Saved Matrix**

The Matrix data is saved to an ASCII file in the UNIX operating system.

The directory where this file resides is determined by the system and is always  $\$XMYHOME/data/testMatrix/testID$ , where *testID* is the identification number for the Test object. For example, if you save a matrix for the Test object (ID number 3793) in Figure 9-15, your Test Case Matrix file is saved to  $\$XMYHOME/data/testMatrix/3793$ .

See Section 9.4.2 for information on the Matrix file format.

You can save multiple test case matrices, each with a different Name. You simply provide a new Name when you execute **Matrix->Save**.

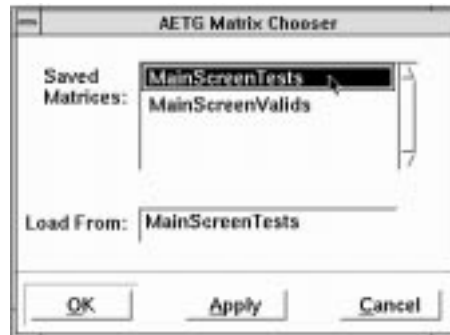
**NOTE — Matrix->Save** does not save the Test object. It only saves the Test Case Matrix data to the entered filename.

#### 9.3.3.4 Loading an Existing Matrix

To view a previously Saved matrix, execute

**Test->Matrix->Load**

A Load Matrix dialog (Figure 9-24) appears.



**Figure 9-24.** Load Matrix Dialog

Select one of the names in the list or type one in the **Load From** box and click **OK**. The data in that matrix is loaded in the **Test Case** list up to the amount specified by the AETG **Max Test Cases** parameter. (See [Section 3.7](#) and [Section 9.3.3.6](#).) The **Name**, **Generated Date**, and **Up To Date** fields are also updated.

### 9.3.3.5 Deleting a Matrix

To delete one or more previously saved matrices

1. Execute

#### Test->Matrix->Delete

A Delete Matrix dialog (Figure 9-25) appears.

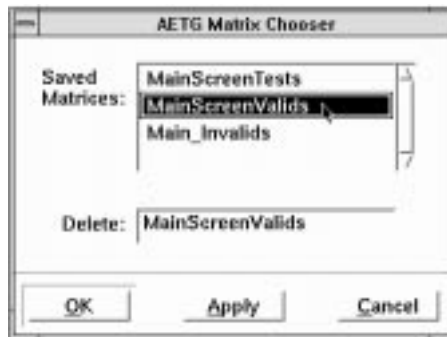


Figure 9-25. Delete Matrix Dialog

2. Select a Matrix name and click **OK**. The matrix is deleted.

### 9.3.3.6 Specifying the Maximum Number of Test Cases

The AETG Desktop's Preferences view contains the preference **Max Test Cases**. This preference indicates the number of test cases that the system automatically loads into a Test object Matrix view. If the number of test cases in the Matrix exceeds the specified **Max Test Cases** value, a dialog (Figure 9-26) appears asking you if you wish to load all of the generated test cases, load the maximum number of test cases, or cancel loading test cases.

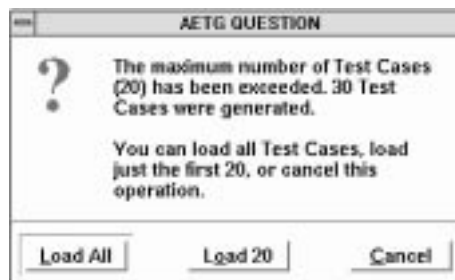


Figure 9-26. Max Testcases Exceeded Dialog

The system default **Max Test Cases** value is 500. You can change this value on the AETG Desktop Preference view. To change the **Max Test Cases** value, see [Section 3.7.1](#).

The **Max Test Cases** value effects both the **Generate Matrix** and **Load Matrix** functions.

#### 9.3.4 Test Save Implications

The Matrix Name is saved at Save time. Also, the test cases in the displayed **Test Case** list are written to the file that is currently displayed in the Name field. If you have not saved the current Test Case Matrix (i.e., *Untitled* appears in the Name field) the Save Matrix dialog appears prompting you to enter a name for the matrix.

## 9.4 Obtaining Test Case Matrix Output

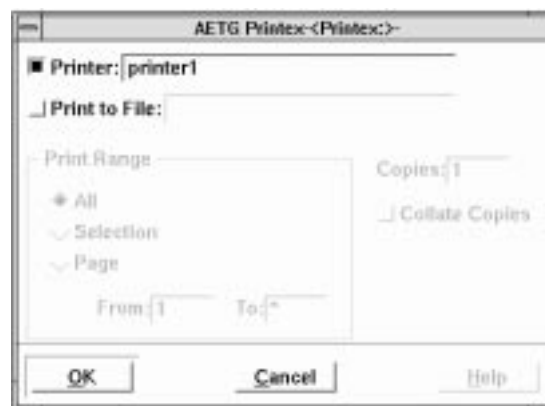
Once you have generated a Test Case Matrix, you can then use the matrix as the input to your testing tool. The AETG System provides several methods of acquiring an electronic version or hardcopy of the Test Case Matrix.

### 9.4.1 Printing a Matrix

As stated in [Section 9.3.3.3](#), saving a Test Case Matrix creates an ASCII file in the directory `$XMYHOME/data/testMatrix/testID`. However, you can print a hard copy or create a local ASCII copy of the Test case matrix by executing

**Test->Matrix->Print**

in an unlocked Test object. The Matrix Print dialog ([Figure 9-27](#)) appears. .



**Figure 9-27.** Matrix Print Dialog

**NOTE** — While the Test object is unlocked, the **Print** option on the **Test** menu is disabled.

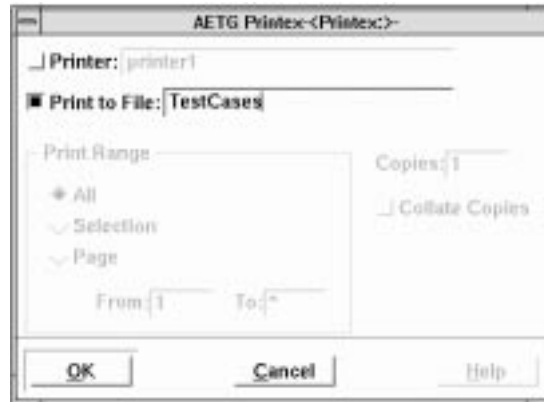
This print dialog works the same as the standard AETG print dialog ([Figure 3-21](#)), letting you send data to a printer or save the data to a file. The difference is that this dialog prints the contents of the current matrix only, while the AETG print dialog lets you print information for the entire Test object.

For more information on using the print dialog, see [Section 3.10](#).

**NOTE** — The contents of the printed file are exactly what you find in `$XMYHOME/data/testMatrix/testID`.

If you decide to print the matrix to a file, you only need to specify the directory. (This can be the full path or the name of a subdirectory to the directory in which you started the AETG System.) The system saves the file using the name of the current matrix as the filename.

For example, if you want to print a copy of the matrix in [Figure 9-23](#) to a file in the directory *TestCases*, which is a subdirectory of the directory in which you started the AETG System, then you could fill in the Matrix Print dialog as shown in [Figure 9-28](#).



**Figure 9-28.** *Printing a Matrix to a File*

Assuming your home directory is */users/kjd*, the resulting path/filename for the printed matrix would be */users/kjd/TestCases/MainScreenTests*.

## 9.4.2 Matrix File Format

The format of the Matrix file is shown in [Figure 9-29](#).

```
{id 501} {date {07/15/97 10:42:53}} {tc 32} {valid 30} {invalid 2} {staleFlag False}
{name V-0} {type V} {fields {{loginID ksb} {Date 01/01/00} {Passwd 5678}}}
{name V-1} {type V} {fields {{loginID ksb} {Date 01-01-2000} {Passwd abcde}}}
{name V-2} {type V} {fields {{loginID hyr} {Date 01-01-1999} {Passwd 5678}}}
{name V-3} {type V} {fields {{loginID hyr} {Date 12-31-1999} {Passwd abcde}}}
{name V-4} {type V} {fields {{loginID rkr} {Date 12/31/99} {Passwd abcde}}}
{name V-5} {type V} {fields {{loginID ksb} {Date 12/31/99} {Passwd 5678}}}
{name V-6} {type V} {fields {{loginID ksb} {Date 01-01-1999} {Passwd 5678}}}
{name V-7} {type V} {fields {{loginID rkr} {Date 12-31-1999} {Passwd 5678}}}
{name V-8} {type V} {fields {{loginID rkr} {Date 01-01-2000} {Passwd abcde}}}
{name V-9} {type V} {fields {{loginID hyr} {Date 01/01/00} {Passwd abcde}}}
```

**Figure 9-29.** *Matrix File Content*

All elements of the Matrix file are delimited by braces ({}).

The first line contains the information displayed in the Matrix view **Matrix Info** parameter, e.g., the **Generation Date** or **Valid Test Cases** count. Each remaining line lists the

information for a Test Case listed in the **Matrix** parameter, i.e., **TC Name**, **Type**, and **<Value>**. Each value listed in the **<Value>** parameter is also delimited by a set of braces.

### 9.4.3 Converting a Test Case Matrix File's Format

The **xmyConvMatrix** command lets you convert a Test Case Matrix file to a Comma Separated Values (CSV) format, which can be directly imported by most spreadsheet programs or you may be able to load into a testing tool.

Execute

```
xmyConvMatrix -t CSV matrixFileName
```

where *matrixFileName* is the name of the Test Case Matrix file you want to convert. This file can be a Test Case Matrix file in *\$XMYHOME/data/testMatrix/testID* or a local file you created using the Matrix Print dialog.

**NOTE** — **xmyConvMatrix** prints the converted file to the standard output, which you can route to a file.

For example if you save the Test Case Matrix file in [Figure 9-29](#) to the local file *MainScreenTests*, you could execute

```
xmyConvMatrix -t CSV MainScreenTests > MainScreenTests.txt
```

in the directory containing the *MainScreenTests*. This would produce the output in [Figure 9-30](#), saving the output to the file *MainScreenTests.txt*.

```
"501", "07/15/97 10:42:53", "32", "30", "2", "False"  
"V-0", "V", "ksb", "01/01/00", "5678"  
"V-1", "V", "ksb", "01-01-2000", "abcde"  
"V-2", "V", "hyr", "01-01-1999", "5678"  
"V-3", "V", "hyr", "12-31-1999", "abcde"  
"V-4", "V", "rkr", "12/31/99", "abcde"  
"V-5", "V", "ksb", "12/31/99", "5678"  
"V-6", "V", "ksb", "01-01-1999", "5678"  
"V-7", "V", "rkr", "12-31-1999", "5678"  
"V-8", "V", "rkr", "01-01-2000", "abcde"  
"V-9", "V", "hyr", "01/01/00", "abcde"
```

**Figure 9-30.** *xmyConvMatrix* Output

**NOTE** — Relation information is not converted and is not included in the output.

For more information on **xmyConvMatrix**, see [Section 11.1](#).





## 10. Relation Object

Relation objects let you collect requirements data to establish how fields and values on the format relate to one and another.

### 10.1 Why use a Relation Object

You use Relation objects to collect the requirements information necessary to provide input to an application under test. For example, a requirement may state, “When Field1 has a value of *blue*, Field2 may not be *pink* or *yellow*.” This input constraint is usually found in the application’s requirements document.

Relation objects also

- Let you tell the AETG System what test cases are important enough to always be included in the set of generated test cases.
- Let you define a set of invalid values that you would like tested.

### 10.2 Working with Relation Objects

As mentioned in [Section 9.3.2](#), a Relation object is accessed through a Test object’s Relations view. To create a new Relation object, execute

#### **Test->New Relation**

in an unlocked Test object.

When you create a Relation object, you decide

1. What Fields will participate in this Relation.
2. If all of the rules governing the interactions of this set of Fields are specified in this Relation or using several Relations (e.g., one rule is specified using one Relation, another using a second Relation, and so on).

For example, for a given set of Fields, you may want to do one of the following:

- Define one Relation for each rule.
- Define one Relation that contains all of the rules for the set of Fields.
- Define several Relations, each containing the rules for a subset of Fields.

When deciding what Relations to create, the AETG System enforces several rules that you must keep in mind:

- A Relation can contain only Fields in the associated Format.
- Multiple Relations defined for one Test object must have either completely disjoint Fields or must have all Fields in common.

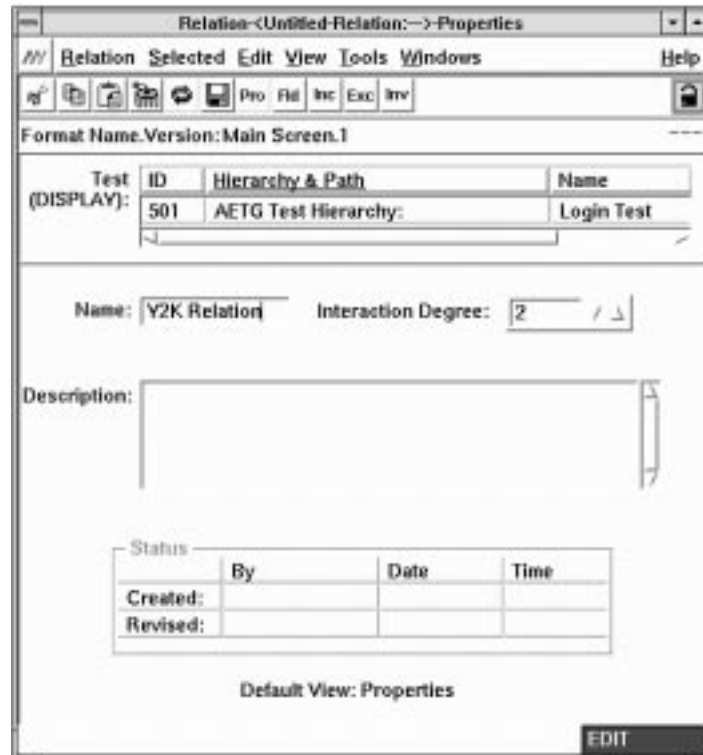
The Relation object has five views:

<b>Properties</b>	Used to define the Properties for the Relation object. (See <a href="#">Section 10.2.1.</a> )
<b>Fields</b>	Lets you choose the Fields and Values that will participate in this Relation. (See <a href="#">Section 10.2.2.</a> )
<b>Includes</b>	Lets you specify field/value sets to be included in the list of generated test cases. (See <a href="#">Section 10.2.3.</a> )
<b>Excludes</b>	Lets you specify value field/value sets to be excluded in the list of generated of test cases. (See <a href="#">Section 10.2.4.</a> )
<b>Invalids</b>	Lets you define invalid Field values to do error condition testing. (See <a href="#">Section 10.2.5.</a> )

The default view is the Properties view ([Figure 10-1](#)).

## 10.2.1 Properties View

The Relation object Properties view (Figure 10-1) lets you enter or change properties of the Relation, such as the Name and Interaction Degree.



**Figure 10-1.** Relation Object Properties View

The Properties view contains the following parameters:

- Test**                Displays the ID, hierarchy, path, and name of the associated Test. This association is automatically populated by the AETG System and is display only.
- You can display the Test object by double clicking on the ruler item.
- Name**                Lets you enter or change the name of the Relation object. The name must be unique within the set of Relations defined for the associated Test.
- This is a required parameter.

**Interaction Degree** Lets you select an Interaction Degree for the Relation, which determines how the fields in the presented generated test cases interact with each other.

For example, a degree of **2** will guarantee that for any two fields all possible value combinations have been covered in the set of generated test cases. A degree of **1** means that the fields are not interdependent. In the case where the degree is **1**, the number of test cases generated for the relation is equal to the largest number of values given to a single field.

The interaction degree is restricted to an integer from **1** to **N**, where **N** is the number of fields in the relation. Typically, choosing a pair wise (**2**) or triple (**3**) field interaction will generate test cases that yield good functional coverage.

The Interaction Degree can be different on each individual Relation associated with a Test.

Default value = **2**.

To choose an Interaction Degree, perform either of the following:

- Select the degree value using the spin controls.
- Type the degree value directly in the Interaction Degree text box.

**Description** Lets you enter a description of the Relation.

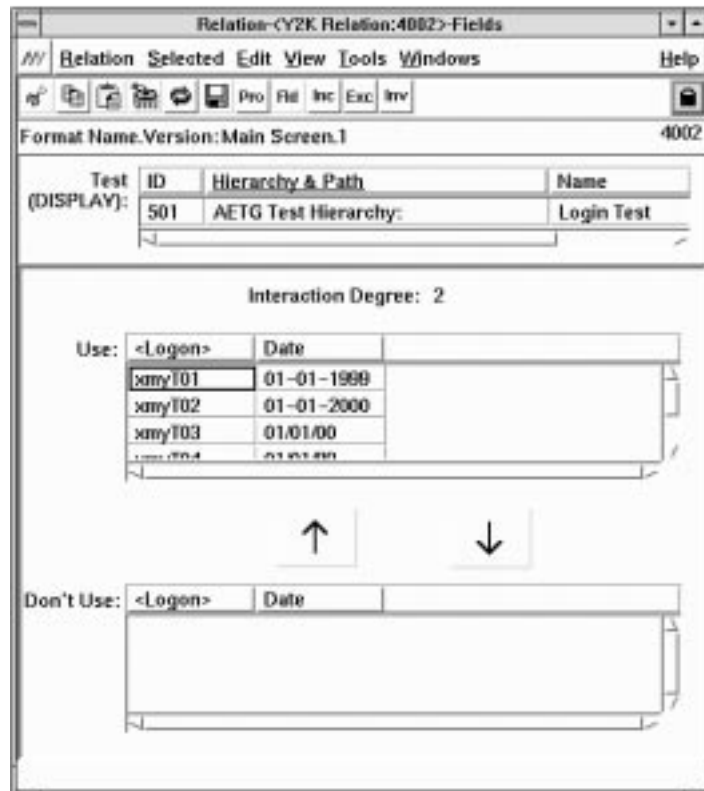
**Status** Displays who created or revised the Relation object and when the Relation object was created or revised.

This is a display only field

In addition to these areas, the associated Format name and version are displayed in the space below the Tool bar area.

## 10.2.2 Fields View

The Relation object Fields view (Figure 10-2) lets you choose the Fields and Values that participate in this Relation. After selecting the participating Fields, you may also reorder the Field columns to create a better arrangement for this Relation.



**Figure 10-2.** Relation Object Fields View

The Fields view contains the following parameters:

**Test** Displays the ID, Hierarchy, Path, and Name of the associated Test. This association is automatically populated by the AETG System and is display only.

You can display the Test object by double clicking on the ruler item.

**Interaction Degree** Displays the Interaction Degree you selected on the Relation object Properties view.

**Use** Lists the Fields and Values that participate in this Relation.  
The top of this list is a ruler divided into columns, one column for each defined Field, with each column listing the Fields in this Relation. Under this ruler is a scrollable list showing the defined values in the corresponding Field. Initially, this list is populated with all Fields and corresponding Values from the associated Format that are not already used in another Relation.

Compounds appear in the Fields ruler with the Compound Name enclosed in brackets (<>), e.g., <*compound1*>. The corresponding Values for Compounds are the Tuple names defined in the Compound object.

**Note** — The simple Fields that make up the Compound are *not* displayed.

In the simplest case, you would have all initially populated Fields and Values participate in the Relation. If that is the case then you do not have to do anything on this view. However, that is more often than not the typical case. In this case, you can

- Change what Fields participate in the Relation by executing

**Relation->Change Fields**

See [Section 10.2.2.1](#) for information on changing the participating Fields.

- Change what Values participate in the Relation using the arrow buttons below the list.

See [Section 10.2.2.2](#) for information on changing the participating Values.

**Don't Use** Lists the Fields and Values that do *not* participate in this Relation.

The Fields ruler contains the exact same set of Fields as the **Use** list. Initially, the scroll area listing the values in the Fields is empty.

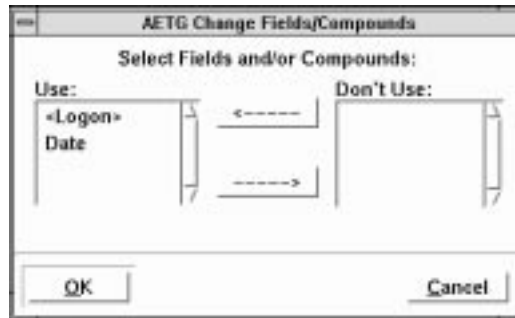
If the participating Fields or Values are changed on this view, those changes are reflected on the other views of this object even if you have not saved the changes on this view. Therefore, if you switch to another Relation object view from an unlocked Fields view, all other views are updated with any changes made on the Fields view.

### 10.2.2.1 Changing Participating Fields

To add or subtract Fields to a Relation, execute

**Relation->Change Fields**

on an unlocked Fields view. The Change Fields confirm dialog (Figure 10-3) appears.



**Figure 10-3.** Change Fields Confirm Dialog

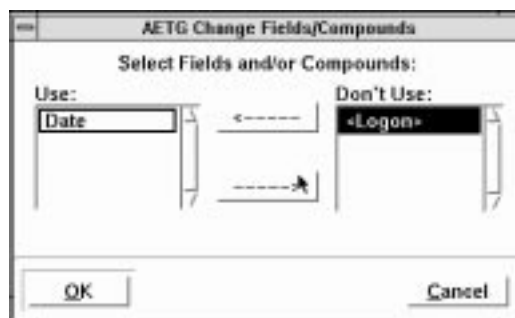
The **Use** list displays all of the Fields that will participate in the Relation, and the **Don't Use** list displays all of the remaining valid Fields that will not participate.

To move Fields from one side to the other

1. Highlight one or more Fields.

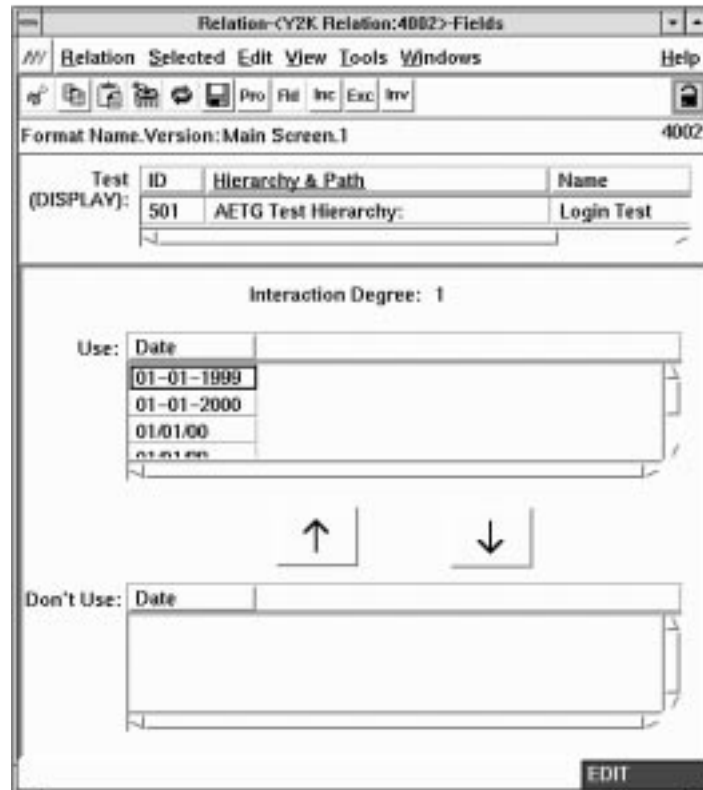
**NOTE** — You cannot remove *all* Fields from the **Use** list.

2. Click one of the arrow buttons. (See Figure 10-4.)
  - Click on the left arrow (<-) to add Fields to the **Use** list
  - Click on the right arrow (->) to add Fields to the **Don't Use** list.



**Figure 10-4.** Changing Fields

3. The Change Fields confirm dialog is modal; you must close the dialog before working on any other AETG process. You must therefore perform one of the following:
  - To cancel your changes, click on **Cancel**. The dialog closes.
  - To accept your changes, click on **OK**. The dialog closes and the Relation object's Fields view is updated to show the Fields that participates in the Relation.



**Figure 10-5.** Updated Fields View Showing Affect of Changing Fields

**NOTE** — If you change a Relation that already has data on other views, you can only add Fields to the **Use** list. Fields that are currently participating in a Relation cannot be removed and the bottom arrow is disabled.

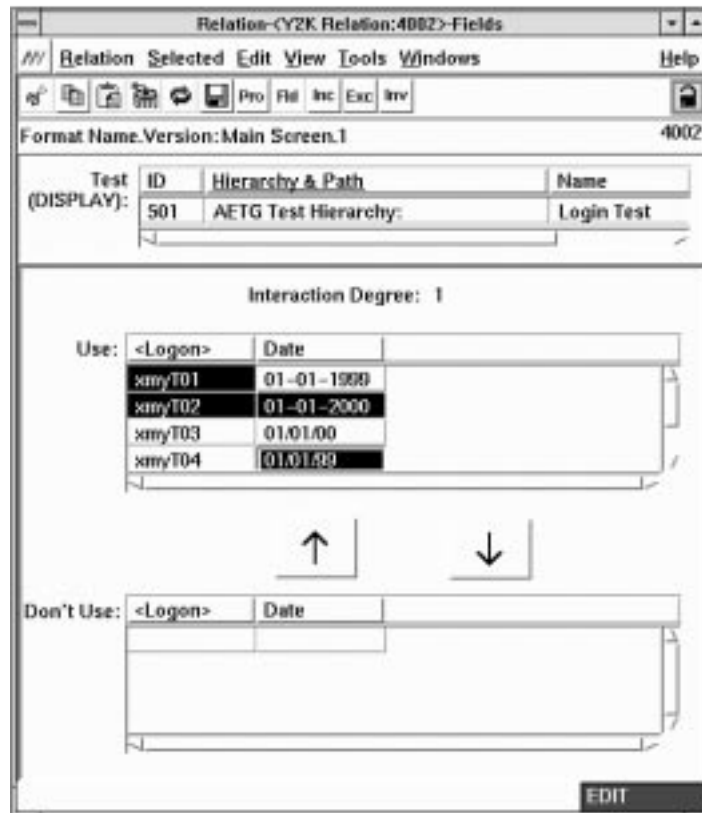


### 10.2.2.2 Changing Participating Values

You can move Values for a participating Field between the **Use** and the **Don't Use** lists.

To move values from the **Use** to the **Don't Use** list,

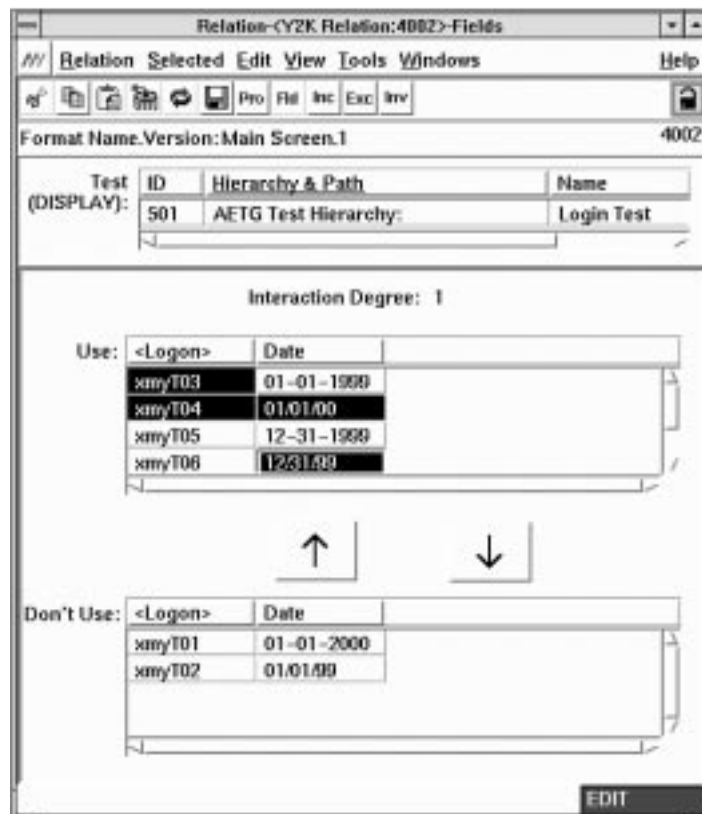
1. On an unlocked Fields view, highlight one or more Values in the **Use** scroll list. (See [Figure 10-6](#).)



**Figure 10-6.** Selecting Values to add to the Don't Use List

**NOTE** — Remember, to multiselect values, **Ctrl-Click** on the desired values.

2. Click on the down arrow. The selected Values move to the **Don't Use** list. (See [Figure 10-7.](#))



**Figure 10-7.** Values added to the Don't Use List

Conversely, you can move values from the **Don't Use** to the **Use** list by clicking on the up arrow after you highlight the desired values.

**NOTE** — You cannot remove *all* Values from the **Use** list.

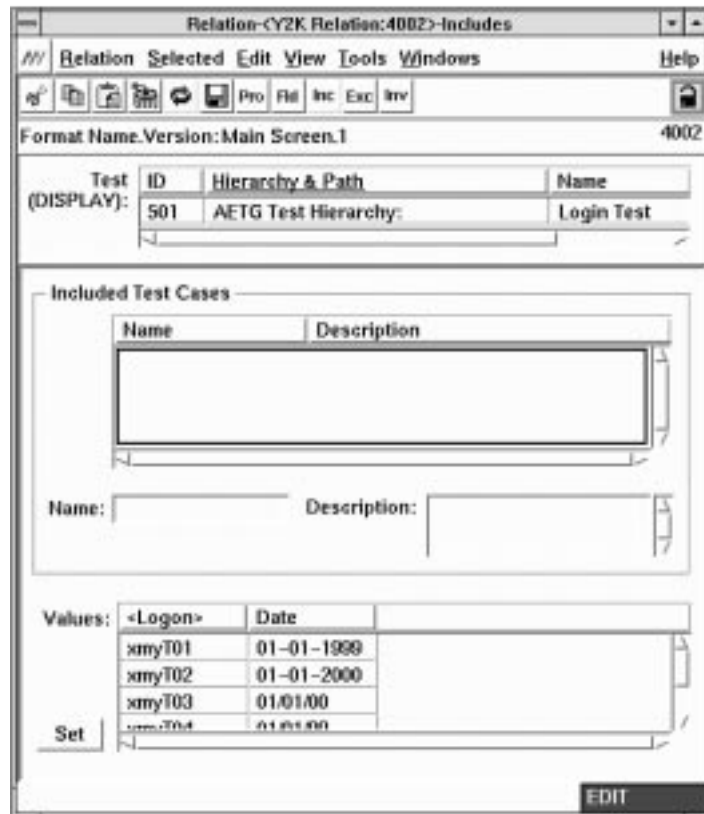
### 10.2.2.3 Rearranging Field Columns

You can rearrange the columns in an unlocked object. Rearranging the columns in one list rearranges the columns in the other list, too. However, the rearrangement is not saved with the object nor is the rearrangement propagated to the Include or Exclude views. This means that the arrangement on these other views can be different.

See [Section 3.4.5](#) for information on rearranging columns.

### 10.2.3 Includes View

The Relation object Includes view (Figure 10-8) lets you select specific field/value sets that you want to ensure are included in the list of generated test cases. You can also view, modify, or delete Include Test Cases.



**Figure 10-8.** Relation Object Includes View

The Includes view contains the following parameters:

**Test** Displays the ID, Hierarchy, Path, and Name of the associated Test. This association is automatically populated by the AETG System and is display only.

You can display the Test object by double clicking on the ruler item.

**Included Test Cases** Contains a list showing all defined Included Test Cases. You can create new Included Test cases and view, modify or delete existing Included Test Cases. When you create new Included Test Cases, you must specify a Name and, optionally, a description of the Included Test Case. (See [Section 10.2.3.1.](#))

**Note** — An Included Test Case name cannot contain embedded spaces.

**Values** The value combinations defined here are included in the generated list of tests.

This list displays the defined Fields and Values for this Relation. All Fields that participate in the Relation are represented as columns in the ruler at the top of this list, and all participating Values are populated in the appropriate columns.

Compounds appear in the Fields ruler with the Compound Name enclosed in brackets (<>), e.g., <compound1>. The corresponding Values for Compounds are the Tuple names defined in the Compound object.

Initially, this list is populated with all valid Values, i.e., all Values that you selected on the Fields view.

When selecting Values for an Included Test Case, you can select any number of values in a Field or Compound column.

**Note** — If two or more Relations have the same set of Fields and you execute

#### **Relation->Change Fields**

a pop-up error message appears stating the name of the other Relation(s) that use the same set of Fields and explaining that they can therefore not change the set of Fields.

**Set** Applies the selected values to the Included Test Case, that is, the system defines the values for this particular Included Test Case. If you do not click on the **Set** button, then the values will not be set.

For example, a Test may have the Fields F1, F2, F3, F4, and F5 and you want to ensure that there is a test case having

- F1 = blue
- F3 = red
- F5 = green.

You can then multiselect the corresponding Values for each field, and the generated test case is guaranteed to be included in the Test Matrix view of the Test Object. That is, at least one test case will have this desired combination in the list of generated test cases. This test case will not be dropped out when the AETG System generates the efficient set of test cases.

### 10.2.3.1 Adding New Includes

To add an Include row to the **Included Test Cases** list

1. On an unlocked Relation object Includes view, execute

#### **Relation->New->Include**

A new row is added to the **Included Test Cases** list and the **Name** field is populated with *<new-test-case>*. (See [Figure 10-9](#).)

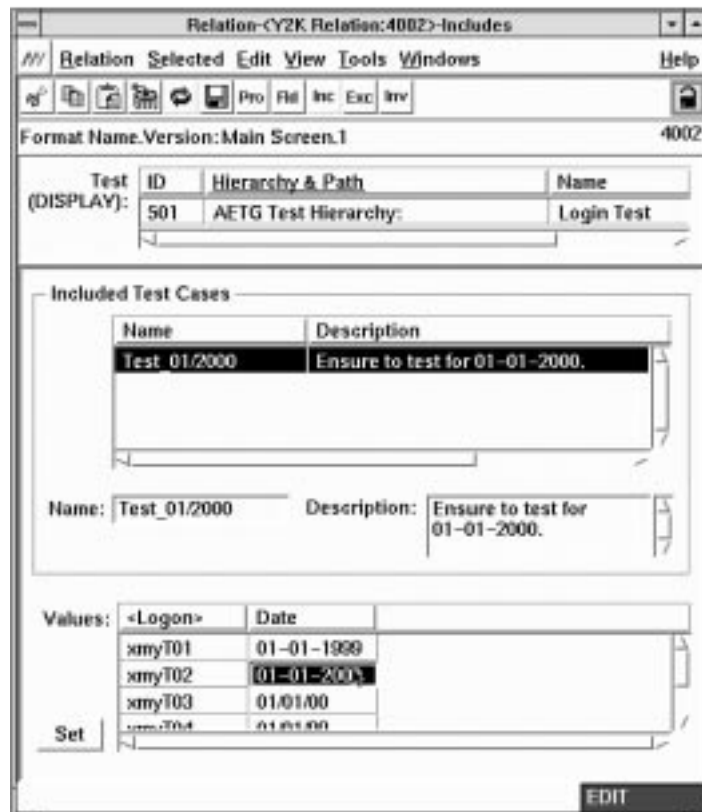


**Figure 10-9.** Creating a New Include

2. Edit the **Name** field and optionally enter text in the **Description** field.

**NOTE** — Remember, an Included Test Case name cannot contain embedded spaces.

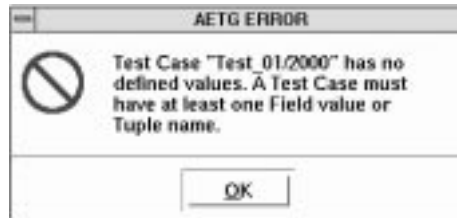
3. Select the value cells from any of the Field or Compound columns in the **Values** list to determine the values you want to participate in this Included Test Case. (See [Figure 10-10.](#))



**Figure 10-10.** Defining a new Included Test Case

4. Click on the **Set** button.

**NOTE** — If you do not click on the **Set** button and you try to save the Relation object, the error dialog in [Figure 10-11](#) appears.



**Figure 10-11.** Undefined Relation Error Dialog

**WARNING** — The AETG System does not check for duplicate value sets, e.g., if two or more Included Test Cases use the exactly same values.

**NOTE** — More than one value can be selected in any column. Selecting more than one value means that the Included Test Case will represent more than one test case.

5. When you are satisfied with your edits and selections, perform one of the following:
  - Execute
    - Relation->New->Include**
    - and repeat Steps 2 through 4 to define a new Included Test Case.
  - Save the Relation object, e.g., execute
    - Relation->Save**
    - An error dialog appears if
      - A Name is not unique among the Included Test Cases
      - There are no set values for an Included Test Case.
  - Change to one of the other Relation object views to further define your test case.

**WARNING** — If an Included Test Cases name is not unique among the Included Test Cases when you execute **New->Include** or when you save of the object, an error dialog (Figure 10-12) appears prompting you to enter a new name.



**Figure 10-12.** Non-unique Included Test Case Name Error Dialog

#### 10.2.3.2 Viewing/Modifying Existing Includes

To view an existing Include, simply select an Included Test Case row. When you do so

- The **Name** and **Description** test areas are populated with the current contents of the Included Test Case row. You may edit the **Name** or the **Description** text on an unlocked Relation object.
- The included Values are automatically highlighted in the **Values** list. To change the included Values, simply select new or different value cells on an unlocked Relation object.



### 10.2.3.3 Duplicating Include Test Cases

To create a new Included Test Case that is similar to an already existing Included Test Case, click on the existing Included Test Case and execute

#### **Selected ->Duplicate**

A new Included Test Case row (Figure 10-13) is populated with the same

- Name as the original Included Test Case, but with a 1 appended to it
- Description
- Set of Values



**Figure 10-13.** Duplicating an Included Test Case

You may want to duplicate an Included Test Case if you have many values chosen for an existing Included Test Case and you need to create another Included Test Case and change only a few of those values.

If you accidentally click in the Values area and cause all selections to become deselected, select a different Included Test Case row, then select the Included Test Case row you were working on.

#### 10.2.3.4 Deleting Existing Includes

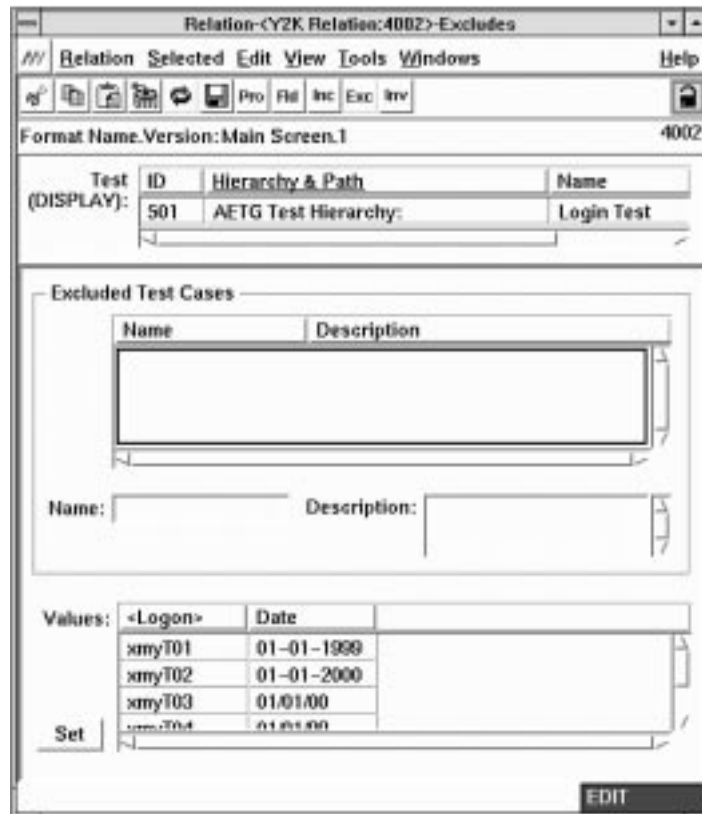
To delete an existing Included Test Case,

1. Select the appropriate row.
2. Execute

**Edit->Delete**

### 10.2.4 Excludes View

The Relation object Excludes view (Figure 10-14) lets you specify value combinations to be excluded from the generated set of test cases. You can also view, modify or delete the Exclude Test Cases.



**Figure 10-14.** Relation Object Excludes View

The Excludes view contains the following parameters:

**Test** Displays the ID, Hierarchy, Path, and Name of the associated Test. This association is automatically populated by the AETG System and is display only.

You can display the Test object by double clicking on the ruler item.

**Excluded Test Cases** Contains a list showing all defined Excluded Test Cases. You can create new Excluded Test cases and view, modify or delete existing Excluded Test Cases. When you create new Excluded Test Cases, you must specify a Name and, optionally, a description of the Excluded Test Case. (See [Section 10.2.4.2](#) for information on adding new Excluded Test Cases.)

**Note** — An Excluded Test Case name cannot contain embedded spaces.

**Values** The value combinations defined here are excluded from the generated list of tests.

This list displays the defined Fields and Values for this Relation. All Fields that participate in the Relation are represented as columns in the ruler at the top of this list, and all participating Values are populated in the appropriate columns.

Compounds appear in the Fields ruler with the Compound Name enclosed in brackets (<>), e.g., <*compound1*>. The corresponding Values for Compounds are the Tuple names defined in the Compound object.

Initially, this list is populated with all valid Values, i.e., all Values that you selected on the Fields view.

When selecting Values for an Excluded Test Case, you can select any number of values in a Field or Compound column.

**Set** Applies the selected values to the Excluded Test Case, that is, the system applies the values to this particular Excluded Test Case. If you do not click on the **Set** button, then the values will not be set.

For example, a Test may have the Fields F1, F2, F3, F4, and F5 and you want to ensure that the application under test will never encounter the following combination:

- F1 = d
- F4 = pink

You would then

1. Multiselect the Value *d* from the Field *F1* and the Value *pink* from the Field *F4*
2. Click on the **Set** push button.

This indicates that this combination is not allowed regardless of the values of other fields. The resulting matrix of test cases will not contain any test cases having the excluded field/value pairs.

If you create conflicting Include and Exclude Test Cases rules, preference is given to the exclusion rule. For example, if you define an Include Test Cases rule with the following:

- F1 = blue
- F2 = green

and an Exclude Test Cases rule of

- F2 = green.

all test cases having the Field **F2** with a Value of **green** are excluded, which also excludes your Include Test Case.

#### 10.2.4.1 Examples of Field Constraints

There are many type of constraints that can be placed on field interaction. Here are some examples:

Constraint	Effect
If Field1= x then Field2 must be > 10	The Excludes view will have <b>Field1</b> having the value <b>x</b> (only) and <b>Field2</b> having selected values less than or equal to <b>10</b> .
If Field1= blank then Field2 != blank	The Excludes view will have both <b>Field1</b> and <b>Field2</b> having the value BLANK selected only.
If Field1= x then Field2 must be either y or z	The Excludes view will have <b>Field1</b> having the value <b>x</b> (only) and <b>Field2</b> having all values except y and z selected.

When you define exclude conditions, you must be aware of the **implicit exclude conditions/problem**. This type of condition occurs when multiple excludes are combined in one relation. For example, suppose you have the following:

Exclude #	Constraint
Exclude-1	When F1=x, F2 must be x
Exclude-2	When F2=y, F3 must be x

This implies that when F1=x, then F3=x.

The AETG System will not guarantee the correct processing of implicit exclude conditions. However, when such a condition exists the system will display a warning dialog telling you that such a condition may exist.

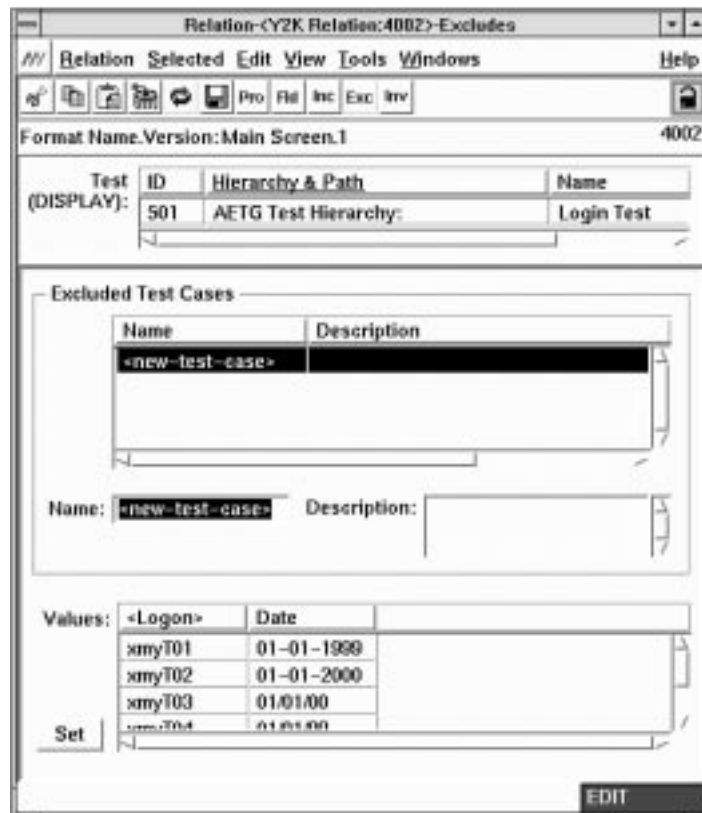
### 10.2.4.2 Adding New Excludes

To add an Exclude row to the **Excluded Test Cases** list

1. On an unlocked Relation object Excludes view, execute

**Relation->New->Exclude**

A new row is added to the **Excluded Test Cases** list and the **Name** field is populated with `<new-test-case>`. (See [Figure 10-15](#).)

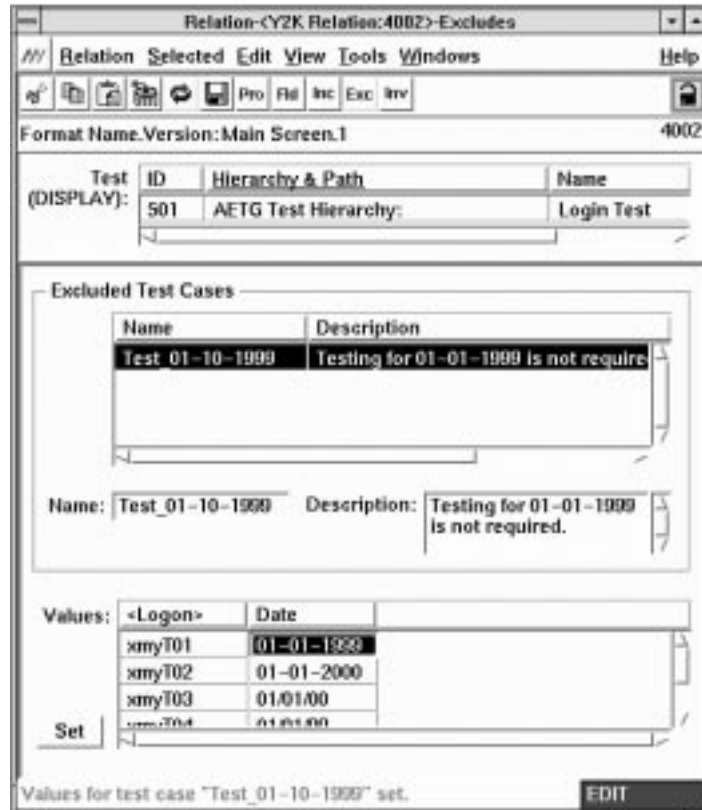


**Figure 10-15.** Creating a New Exclude

2. Edit the **Name** field and optionally enter text in the **Description** field.

**NOTE** — Remember, an Excluded Test Case name cannot contain embedded spaces.

3. Select the value cells from any of the Field or Compound columns in the **Values** list to determine the values you want to participate in this Excluded Test Case from the **Values** list. (See [Figure 10-16](#).)



**Figure 10-16.** Defining a new Excluded Test Case

4. Click on the **Set** button.

**NOTE** — If you do not click on the **Set** button and you try to save the Relation object, the error dialog similar to the one in [Figure 10-11](#) appears.

**WARNING** — The AETG System does not check for duplicate value sets, e.g., if two or more Excluded Test Cases use the exactly same values.

**NOTE** — More than one value can be selected in any column. Selecting more than one value means that the Excluded Test Case will represent more than one test case.

5. When you are satisfied with your edits and selections, perform one of the following:

- Execute

**Relation->New->Exclude**

and repeat Steps 2 through 4 to add a new Excluded Test Case.

- To save the Relation object, e.g., execute

**Relation->Save**

- Change to one of the other Relation object views to further define your test case.

**WARNING** — If an Excluded Test Cases name is not unique among the Excluded Test Cases when you execute **New->Exclude** or when you save of the object, an error dialog ([Figure 10-12](#)) appears prompting you to change enter a new name.

#### 10.2.4.3 Viewing/Modifying Existing Excludes

To view an existing Exclude, simply select an Excluded Test Case row. When you do so:

- The **Name** and **Description** test areas are populated with the current contents of from the Excluded Test Case row. You may edit the **Name** or the **Description** text on an unlocked Relation object.
- The excluded Values are automatically highlighted in the **Values** list. To change the excluded Values, simply select new or different value cells on an unlocked Relation object.



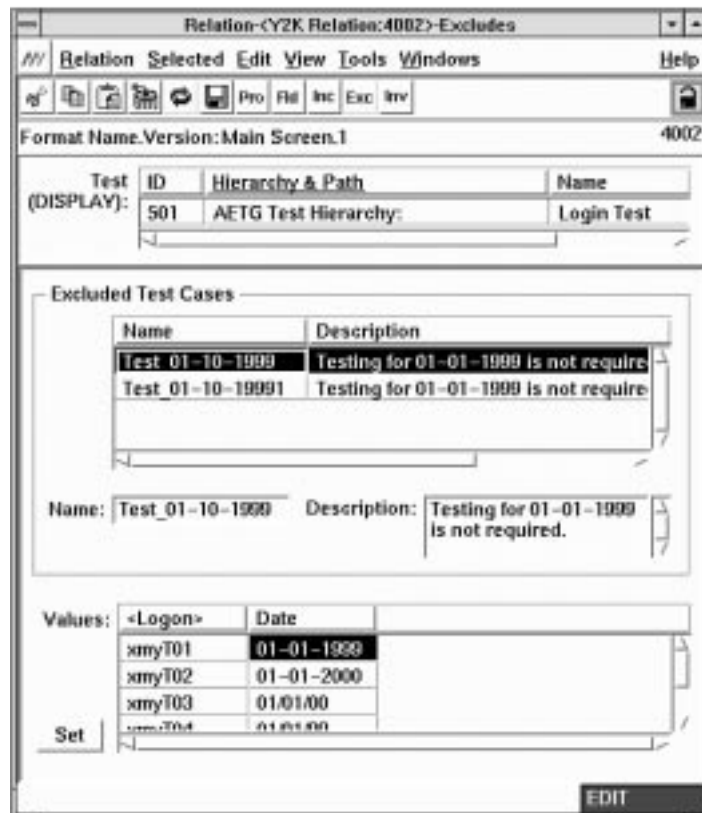
#### 10.2.4.4 Duplicating Excluded Test Cases

To create a new Excluded Test Case that is similar to an already existing Excluded Test Case, click on the existing Excluded Test Case and execute

##### **Selected ->Duplicate**

A new Excluded Test Case row (Figure 10-17) is populated with the same

- Name as the original Excluded Test Case, but with a 1 appended to it
- Description
- Set of Values.



**Figure 10-17.** Duplicating an Included Test Case

**NOTE** — If you do not change the set of values and then use the **Set** button, an error is produced when you save the Relation object.

You may want to duplicate an Excluded Test Case if you have many values chosen for an existing Excluded Test Case and you need to create another Excluded Test Case and change only a few of those values.

If you accidentally click in the Values area and cause all selections to become deselected, select a different Excluded Test Case row, then select the Excluded Test Case row you were working on.

#### 10.2.4.5 Deleting Existing Excludes

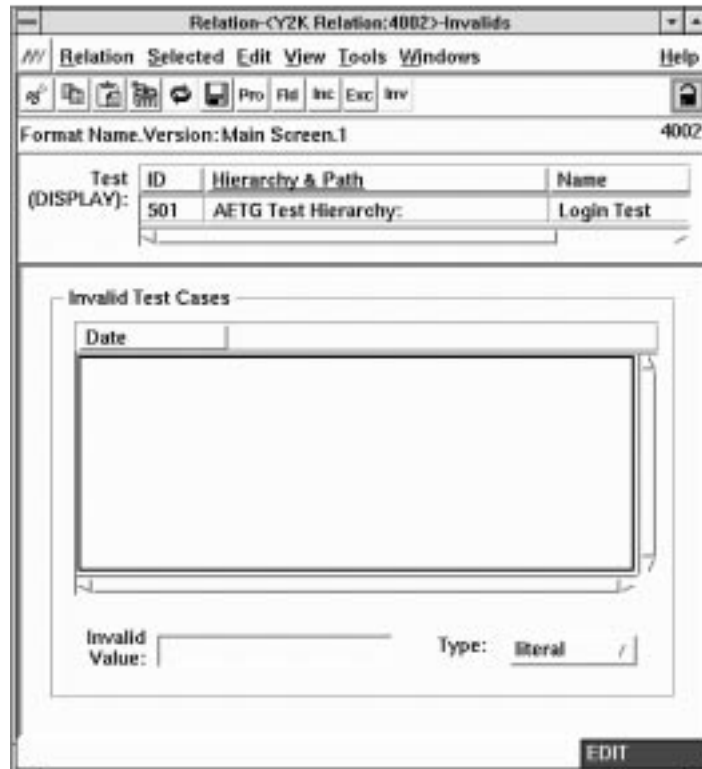
To delete an existing Excluded Test Case,

1. Select the appropriate row.
2. Execute

**Edit->Delete**

### 10.2.5 Invalids View

The Invalids view (Figure 10-18) defines invalid Field values to be tested, which are specific values you want to try on an application to do error condition testing. The values for the Fields are not supplied by the Format object. You must provide the invalid values to be used.



**Figure 10-18.** Relation Object Invalids View

The Invalids view contains the following parameters:

**Test** Displays the ID, Hierarchy, Path, and Name of the associated Test. This association is automatically populated by the AETG System and is display only.

You can display the Test object by double clicking on the ruler item.

**Invalid Test Cases** Contains a list showing all defined Invalid Test Cases. You can create new Invalid Test cases and view, modify or delete existing Invalid Test Cases. When you create new Invalid Test Cases, you must specify a Name and, optionally, a description of the Invalid Test Case. (See [Section 10.2.5.1](#) for information on adding new Invalid Test Cases.)

A column heading for each participating Field is prepopulated in the Invalid Test Cases list.

**Note** — Only one field value at a time can be made invalid. That is, you can enter only one invalid value for one field at a time; you cannot enter one invalid value for one field, another invalid value for a second field, and so on. Neither can you enter multiple invalid values for a field at one time. Each invalid value must denote a separate Invalid Test Case. Invalid combinations are not allowed in this release of the system.

The invalid test cases are included in the Test Matrix. Typically, the AETG System forms an invalid test case by duplicating a valid test case and then corrupting it, entering the values you defined for the Invalid Test Case.

**Note** — Compounds are never displayed on this view. Also, the simple Fields that participate in Compounds are never displayed on this view.

When this view is first displayed the Invalid Test Cases list is empty.

To create values for an Invalid Test Case,

#### 10.2.5.1 Adding New Invalids

To add new Invalid Test Cases

1. On an unlocked Relation object Invalids view, execute

**Relation->New->Invalid**

A new row of empty cells is added to the **Invalid Test Cases** list, one for each Field in the Relation.

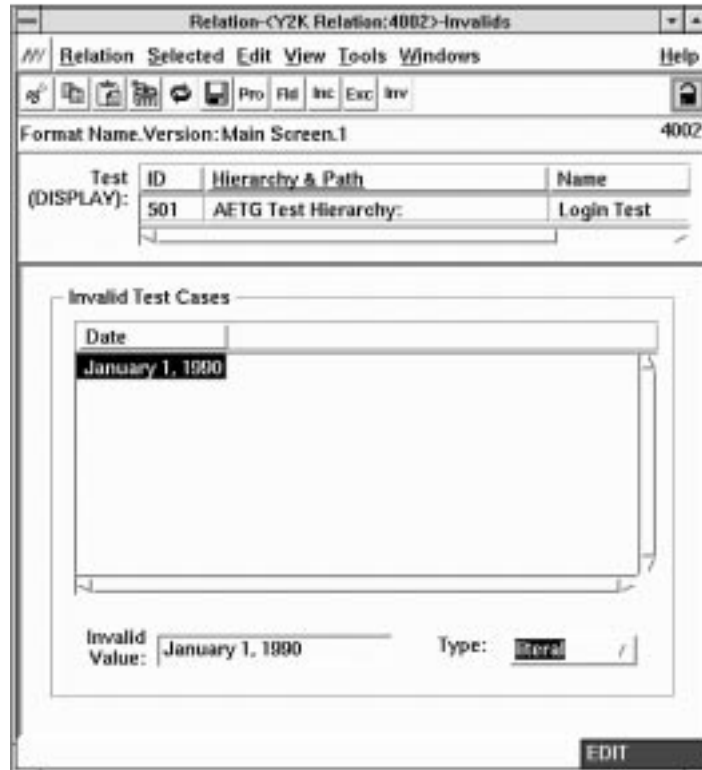
**NOTE** — Remember, Compounds are not displayed on this view.

2. Click in the empty cell for the Field for which you want to enter an invalid value (Figure 10-19).



Figure 10-19. Creating a New Invalid Test Case

3. Click in the **Invalid Value** text area and enter the invalid value for that Field.
4. Click on the **Type** menu (if needed), and select the appropriate Value type, e.g., **literal** or **non-literal** (Figure 10-20).



**Figure 10-20.** Specifying a Value for an Invalid Test Case

5. When you are finished, perform one of the following:
  - To add a new Invalid Test Case  
**Relation->New->Invalid**  
and repeat Steps 2, 3, and 4.
  - To save the Relation object, e.g., execute  
**Relation->Save**
  - Change to one of the other Relation object views to further define your test case.

#### 10.2.5.2 Viewing/Modifying Existing Invalids

To view an existing Invalid, simply select an Invalid Test Case row. When you do so the Value in the row and the Type of the Value are automatically populated in the **Invalid Value** text area and the **Type** field, respectively.

You may edit the **Invalid Value** text or may change the **Type** field on an unlocked Relation object.

#### 10.2.5.3 Deleting Existing Invalids

To delete an existing Invalid Test Case

1. Select the appropriate row.
2. Execute

**Edit->Delete**





## 11. Command Line Tools

This section describes the CLUI commands that are provided with the AETG System.

### 11.1 xmyConvMatrix

#### Syntax

```
xmyConvMatrix ?-h? -t CSV matrixFileName
```

#### Description

The **xmyConvMatrix** command reads in a Test Case Matrix file and prints the matrix information, to standard the output, in Comma Separated Values (CSV) format.

**xmyConvMatrix** takes following option:

- h** Provides a help message
- t** Determines the format of the output. Only CSV is supported in Release 5.2.
- matrixFileName*** Specifies the names of the Test Case Matrix file.

**Note** — ***matrixFileName*** can be a Test Case Matrix file saved to *\$XMYHOME/data/testMatrix/testID* or it can be a local file you created using the AETG Matrix Print Dialog ([Figure 9-27](#)).

#### Example

Assume you printed the following Test Case Matrix saved to the file */u/kjd/AETG/Payment*.

```
{id 3166} {date {06/05/97 15:07:10}} {tc 7} {valid 4} {invalid 3} {staleFlag True}
{name V-0} {type V} {fields {{C c} {P A} {DAA-02 LDB} {DAA-01 LDA}}}}
{name V-1} {type V} {fields {{C c} {P A} {DAA-02 LDB} {DAA-01 BLANK}}}}
{name V-2} {type V} {fields {{C c} {P A} {DAA-02 BLANK} {DAA-01 BLANK}}}}
{name V-3} {type V} {fields {{C c} {P A} {DAA-02 BLANK} {DAA-01 LDA}}}}
{name I-0} {type I} {fields {{C c} {P A} {DAA-02 LDAA} {DAA-01 LDA}}}}
{name I-1} {type I} {fields {{C c} {P A} {DAA-02 LDB} {DAA-01 LDAA}}}}
{name I-2} {type I} {fields {{C c} {P A} {DAA-02 BLANK} {DAA-01 BLANK}}}}}
```

Executing

**xmyConvMatrix -t CSV /u/kjd/AETG/Payment**

will create the following output:

```
"3166", "06/05/97 15:07:10", "7", "4", "3", "True"  
"V-0", "V", "c", "A", "LDB", "LDA"  
"V-1", "V", "c", "A", "LDB", "BLANK"  
"V-2", "V", "c", "A", "BLANK", "BLANK"  
"V-3", "V", "c", "A", "BLANK", "LDA"  
"I-0", "I", "c", "A", "LDAA", "LDA"  
"I-1", "I", "c", "A", "LDB", "LDAA"  
"I-2", "I", "c", "A", "BLANK", "BLANK"
```

You can save the output to a file, such as by executing

**xmyConvMatrix -t CSV /u/kjd/AETG/Payment> Payment.csv**

The CSV formatted file can be imported directly by most spreadsheet programs.

**NOTE** — Relation information is not converted and is not  
in the output.

## 11.2 xmyCreateFormats

### Syntax

```
xmyCreateFormats filename(s)
```

### Description

The **xmyCreateFormats** command lets you automatically create Format and Field objects if the information for these objects is stored in a file in a specific format.

**xmyCreateFormats** takes the following option:

*filename*            The name of a file containing one or more Format definitions.

When you execute **xmyCreateFormats**, the named file(s) are used as input for creating Format and Field objects in the AETG database.

**NOTE** — **xmyCreateFormats** processes one Format  
object per file.

## File Structure

The files containing the information about one or more Formats must follow a prescribed structure. [Figure 11-1](#) shows an example file.

```
# any comments
BEGIN MYNAH OBJECT
{FORMAT
  {NAME fmt-1}
  {TYPE 3270SCREEN}
  {DESCRIPTION "description of fmt-1"}
  {FIELDS
    {FIELD
      {NAME field-1}
      {LOGICAL-NAME "field-1"}
      {DESCRIPTION "description of field-1"}
      {PARAMETERS "location (100,100)"}
      {VALUES
        {alpha beta gamma {delta NON-LITERAL}}}
      }
    }
    {FIELD
      {NAME field-2}
      {LOGICAL-NAME "field-2"}
      {DESCRIPTION "description of field-2"}
      {PARAMETERS "location (100,200)"}
      {VALUES {one {two NON-LITERAL} three}}
    }
  }
}
END MYNAH OBJECT
# any other comments
```

**Figure 11-1.** Format File Structure

The following tags must be specified in the file:

```
BEGIN MYNAH OBJECT
FORMAT
  NAME
  TYPE
  FIELDS
    FIELD # at least one
      NAME
      VALUES # at least one
  }
END MYNAH OBJECT
```

All other tags are optional.

**NOTE** — When specifying VALUES for FIELD tags, you may specify either LITERAL or NON-LITERAL; LITERAL is the default.

If any unknown tags appear in the file, they are ignored.

If any of the following conditions are not true, an error message will be produced and the Format and Fields objects will not be produced:

1. The user id of the person running the command must be a known user in the AETG database.
2. All required tags must be present.
3. FORMAT NAME must not already exist in the database.
4. FORMAT TYPE must be a valid type, which is one of the following:
  - 3270Screen
  - AsyncScreen
  - Message
  - GuiScreen
  - Configuration
  - other.
5. FIELD NAME must be unique within the FORMAT.
6. FIELD VALUE must be unique within the FIELD.

### Formal Syntax

There can be more than one format definition embedded in the file. Each format definition must be delimited by “**BEGIN MYNAH OBJECT**” and “**END MYNAH OBJECT**”.

The formal syntax of the file is described in the following:

```
arbitrary-text
BEGIN MYNAH OBJECT
    format description
END MYNAH OBJECT
arbitrary text
BEGIN MYNAH OBJECT
    format description
END MYNAH OBJECT
...
format description:
{FORMAT format-tag*}
```

format tag:

```
{NAME text}
  {TYPE valid-format-type}
  {DESCRIPTION text}
  {FIELDS field-description*}
```

field-description:

```
{FIELD field-tag*}
```

field-tag:

```
{NAME text}
  {LOGICAL-NAME text}
  {DESCRIPTION text}
  {PARAMETERS text}
  {VALUES field-value*}
```

field-value:

```
value-name
  {value-name LITERAL}
  {value-name NON-LITERAL}
```

field-name:

text

value-name:

text

## Example

If you type

```
xmyCreateFormats formatfile
```

and *formatfile* contains the information in

```
# any comments
BEGIN MYNAH OBJECT
{FORMAT
  {NAME Order}
  {TYPE MESSAGE}
  {DESCRIPTION "this format represents a message type for appl to app2"}
  {FIELDS
    {FIELD
      {NAME Item}
      {LOGICAL-NAME "item"}
      {DESCRIPTION "the kind of item being requested"}
      {VALUES
        {box pail bucket {container NON-LITERAL}}
      }
    }
    {FIELD
      {NAME Color}
      {LOGICAL-NAME "color"}
      {DESCRIPTION "the color of the item being requested"}
      {VALUES {red blue yellow}}
    }
  }
}
END MYNAH OBJECT
# any other comments
```

**Figure 11-2.** Example Format File

then one Format object named **Order** and two Field objects named **Item** and **Color**, respectively, will be created in the AETG database.

## 11.3 xmyPrintFormatFields

### Syntax

```
xmyPrintFormatFields format_name.version_number
```

### Description

The **xmyPrintFormatFields** command takes a Format object name and Version number as input and produces a list of the Fields and Values defined in the Format object.

The output from **xmyPrintFormatFields** goes to standard output. You may redirect the output to a file or may pipe it to another program.

**NOTE** — If a Format object name contains embedded spaces, replace the spaces with a non-breaking space (`\` ).

### Example

To print Version 1 of the Format object **Decorator Screen**, execute

```
xmyPrintFormatFields Decorator\ Screen.1
```

which will print text similar to that found in [Figure 11-3](#) to the standard output.

```
format: Decorator Screen.1

field: Primary Colorseq: 1  logical: PC
      parameters:
literal value: Red
nonliteral value: Blue
literal value: Yellow

field: Other Colorseq: 2  logical: OC
      parameters:
literal value: Orange
nonliteral value: anycolor
```

**Figure 11-3.** Sample Output from xmyPrintFormatFields





## 12. Input Modeling with the AETG System

This section provides tips on how the AETG System could be used to model various types of applications. The sample modeled applications include graphical menus, 3270 Screens and command line interfaces. Although the descriptions do not include details describing how these applications would be implemented in the AETG System, you can apply these models in a similar way as the example used in the Quick Start (Section 2).

### 12.1 Modeling GUIs (Menus)

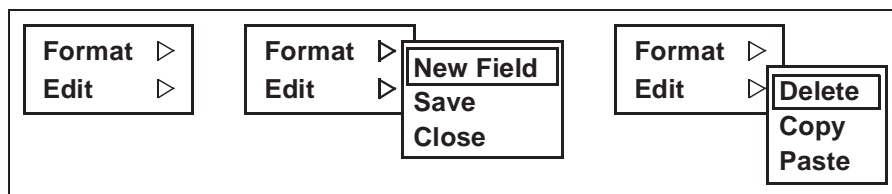
When testing menus on a GUI, it is important to verify that the appropriate functionalities are available under the various states the application may have. In most applications, menu items are disabled to prevent the use of a function that would be invalid or meaningless under certain conditions.

For example, presume you have a menu with two options: **Format** and **Edit**. Each option has a cascading submenu with the options listed in Table 12-1.

**Table 12-1.** GUI Modeling Menu Example

Menu Option	Submenu Options
Format	<ul style="list-style-type: none"><li>• New Field</li><li>• Save</li><li>• Close</li></ul>
Edit	<ul style="list-style-type: none"><li>• Delete</li><li>• Copy</li><li>• Paste</li></ul>

Figure 12-1 illustrates the various states of this menu.



**Figure 12-1.** GUI Modeling Menu Example

The objective is to verify that menu items are enabled or disabled under the “correct states”, factors that affect the menus under test. In our example, user controlled parameters that affect the menu state are the **lock**, **clipboard**, and **selection** statuses. The menu in

Table 12-1 and Figure 12-1 could be modeled using the AETG System as shown in Table 12-2.

**Table 12-2.** GUI Menu Fields and Values

Fields	NewField	Save	Close	Delete	Copy	Paste	lock	clipb	select
Values	enbld	enbld	enbld	enbld	enbld	enbld	on	full	on
	disbld	disbld	enbld	disbld	disbld	disbld	off	empty	off

Once the fields and their values have been identified, the relationships between these parameters are defined or revealed. In this example, the relations are as listed in Table 12-3. These will be used to create the Included and Excluded Test Cases.

**Table 12-3.** Menu Relations

Rel#	Description
1	<b>NewField</b> is enabled when the <b>Format</b> menu is unlocked ( <b>lock = off</b> )
2	<b>Save</b> is disabled when the <b>Format</b> menu is unlocked ( <b>lock = off</b> )
3	<b>Close</b> is always enabled ( <b>Close = enbld</b> )
4	<b>Delete</b> is enabled when <b>select = on</b> and <b>lock = off</b>
5	<b>Copy</b> is enabled when an item is selected ( <b>select = on</b> )
6	<b>Paste</b> is enabled when the object is unlocked and the clipboard is full ( <b>lock = off</b> AND <b>clipb = full</b> ).

Figure 12-2 shows an example Test Case Matrix that can be used to test this GUI menus model.

TC Name	Type	Newfield	Save	Close	Delete	Copy	Paste	lock	clipb	select
V-0	V	disbld	enbld	enbld	disbld	enbld	disbld	on	empty	on
V-1	V	enbld	disbld	enbld	disbld	disbld	enbld	off	full	off
V-2	V	enbld	disbld	enbld	enbld	enbld	disbld	off	empty	on
V-3	V	disbld	enbld	enbld	disbld	disbld	disbld	on	full	off
V-4	V	enbld	disbld	enbld	enbld	enbld	enbld	off	full	on
V-5	V	disbld	enbld	enbld	disbld	disbld	disbld	on	empty	off

Figure 12-2. Test Matrix for Testing the GUI Menus Model

## 12.2 Command Line Interface (Asynchronous Command)

To test a sample command line using the AETG System, we will use the UNIX **ls** command. The various options of the **ls** command (such as *c*, *l*, *t*, *r*, and *a*) will be used to formulate a set of parameters and values. Their respective relationships need to be observed during the definition of the Relation object.

**NOTE** — Refer to a UNIX User’s Guide for further details about the **ls** command.

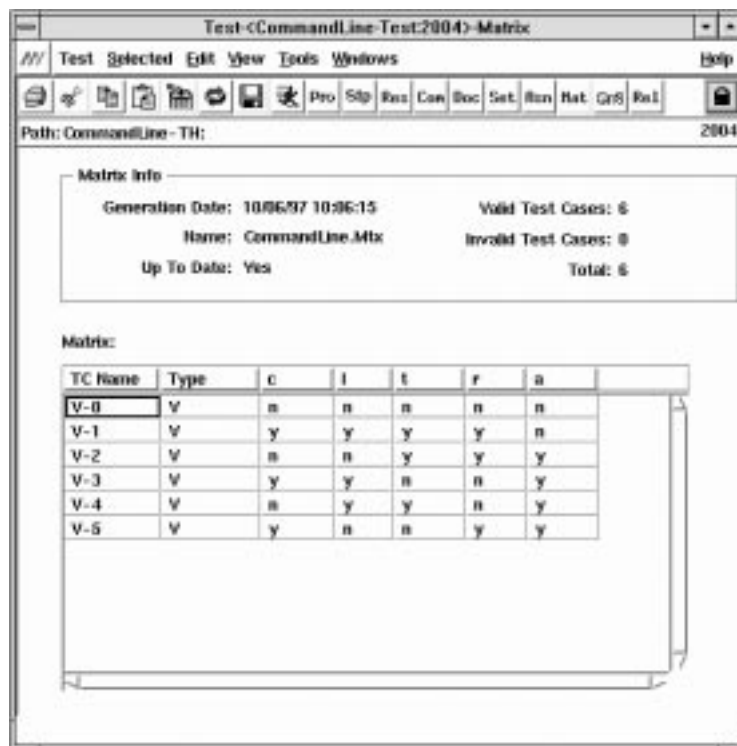
The **ls** command options will have two values: **present** or **not** (*yes/no*) as shown in Table 12-4. To model this test presume that **ls** may have no options or one or more of the

options mentioned above. Although some options are invalidated by others, we will not define any exclude conditions for this model.

**Table 12-4.** Modeling Is UNIX Command

Fields	c	l	t	r	a
Values	y	y	y	y	y
	n	n	n	n	n

Figure 12-3 shows an example Test Case Matrix that can be used to test the Is command.



**Figure 12-3.** Test Matrix for Testing the Is Command Model

### 12.3 Configurations (Config)

In most cases software products are made up of many subsystems. Often these subsystems have various versions that are designed to work together with other versions of systems. For this reason, it is important to test if the different versions of each sub-system work together as expected.

Consider a product that consist of five systems (**A**, **B**, **C**, **D**, and **E**) and three of the subsystems have three versions and the other two have four versions. [Table 12-5](#) shows the combinations of sub-system versions that can make up a product.

**Table 12-5.** Modeling Configuration Scenarios

Fields	A	B	C	D	E
Values	1	1	1	1	1
	2	2	2	2	2
	3	3	3	3	3
				4	4

[Table 12-5](#) contains the various releases of each subsystem. [Table 12-6](#) contains the constraints that exist in integrating these systems. These will be used to create the Included and Excluded Test Cases.

**Table 12-6.** System Configuration Restrictions

Rel#	Description
1	Version <i>1</i> of system <b>A</b> will not work with version <i>4</i> system <b>D</b> .
2	Version <i>2</i> of system <b>B</b> will only work with versions <i>2</i> and <i>3</i> of system <b>E</b> .
3	Version <i>4</i> of system <b>E</b> will only work with version <i>4</i> of system <b>D</b> .
4	Version <i>1</i> of system <b>C</b> will not work with versions <i>3</i> and <i>4</i> of systems <b>D</b> and <b>E</b> .

Figure 12-4 shows an example Test Case Matrix that can be used to test configurations.

Matrix Info

Generation Date: 09/26/97 15:19:24      Valid Test Cases: 17  
Name: ModelConfig.mtx      Invalid Test Cases: 0  
Up To Date: Yes      Total: 17

Matrix:

TC Name	Type	A	B	C	D	E
V-0	V	3	1	2	4	4
V-1	V	2	1	3	2	1
V-2	V	1	3	2	1	2
V-3	V	3	2	3	1	3
V-4	V	2	3	2	3	3
V-5	V	2	2	1	2	2
V-6	V	1	1	3	3	2
V-7	V	3	3	1	2	1
V-8	V	2	3	3	4	4
V-9	V	1	2	2	2	3
V-10	V	1	1	1	1	1
V-11	V	3	2	3	4	2

Figure 12-4. Test Case Matrix for Testing Configuration Model

## **Part C: Administrator Tasks**

---





## 13. Installing the AETG System

This section discusses the steps for installing the AETG System and related software packages, e.g., Telexel.

### 13.1 Introduction

In addition to the AETG System software, you must also install the Telexel software package (which the AETG System uses for interprocess communications and logging) and an Oracle database.

Table 13-1 lists, in order, the steps needed to create our example AETG installation.

**Table 13-1.** Installation Steps

Step	Section
Installing the AETG System	Section 13.6
Installing the Telexel System	Section 13.7
Installing Oracle	Section 13.8

### 13.2 Hardware and System Requirements

The AETG System runs on a SPARC™ machine running the Solaris operating system, Release 2.5.1.

**NOTE** — If you are running the Oracle software on the same machine on which you are running the AETG System, then operating system patches may also be needed. Refer to the Oracle7 Server Installation Guide for your operating system.

The end-user display devices (X-terminals, workstations running X, or PCs running an X emulator) provide graphical/windowing capabilities and a UNIX command-line interface to AETG functionality. The display device should support a minimum resolution of 1024 by 768 pixels.

Table 13-2 lists the required software products (and versions) required to use the AETG system.

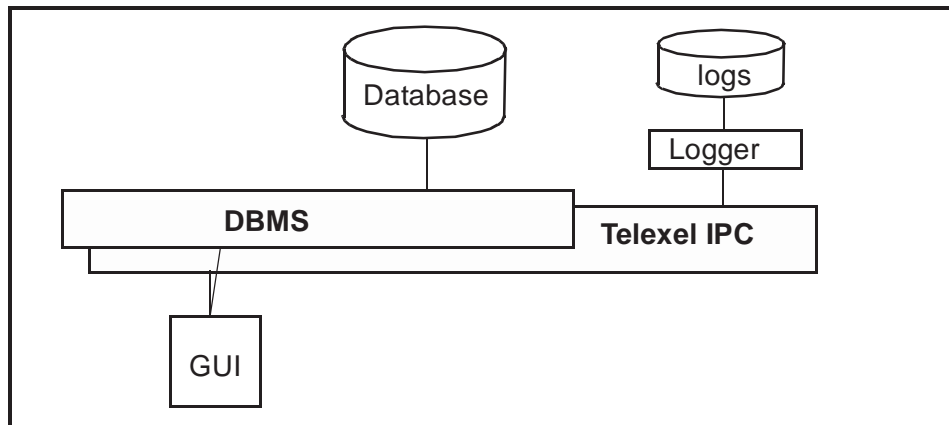
**Table 13-2.** Required Software Packages

Product	Version	Vendor
Solaris	2.5.1	Sun
Oracle DBMS	7.3.2.3	Oracle
TELEXEL IPC <sup>a</sup>	7.0	Bellcore
X Window System	X11R5	Sun

a. TELEXEL IPC is provided with the AETG System distribution. It must be installed separately.

### 13.3 Networking

AETG processes can be distributed across multiple hosts. All AETG processes use the network services of the Database Management System for database access and the network services of Telexel IPC processes for AETG interprocess communications and logging, as shown in Figure 13-1.



**Figure 13-1.** Networked Services

In addition, the AETG processes rely on a single location for executables, but these can have multiple locations for configuration information. The file system where this information is stored must be accessible (typically via NFS) by every host machine that runs a AETG process.

## 13.4 Preliminary Background Information

There is background information you need to know before you install the AETG System.

### 13.4.1 Assumptions and Recommendations

The following subsections make certain assumptions about your installation. If you do not follow these assumptions, remember to make the necessary changes while installing the software.

**WARNING** — Do not actually create any directories or links at this time.

1. All packages are installed in */opt*.
2. Under this directory you should create a directory named *SUNWxxx*, where *xxx* is the first three letters of the package, e.g., *SUNWora* for the Oracle software and *SUNWmyn* for the AETG System.
3. Under each package's *SUNWxxx* directory you create a directory for the version number, e.g., 7.3.2.3 for the Oracle software.
4. You create a symbolic link in each package's *SUNWxxx* directory pointing to the version directory, e.g., for the Oracle software you would execute

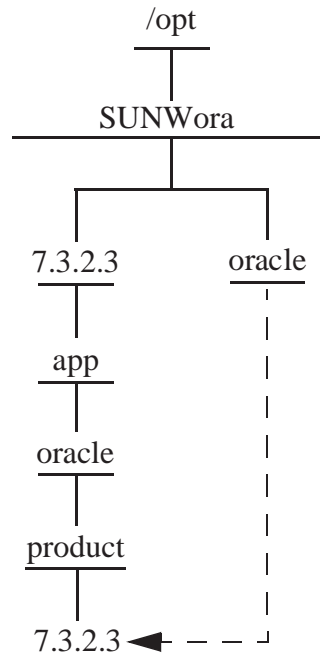
```
ln -s /opt/SUNWora/7.3.2.3/app/oracle/product/7.3.2.3 oracle  
in the Oracle directory (/opt/SUNWora).
```

You create such symbolic links so you change only the link to the new version directory when you install a new version of the software. All user references should be made to the symbolic link name. This will make software upgrades easy since no library names will have to be changed if the users are referencing the symbolic links.

**NOTE** — For the AETG and Telexel products, this will be done automatically by the BAIST installation product. You must do this yourself for Oracle.

Figure 13-2 shows an example of the recommended Oracle directory structure. When configuring the AETG System, there are several places where you must specify the full path for the Oracle software, i.e., */opt/SUNWora/oracle*. If you later install a new version of the Oracle software (e.g., Version 8.0) and you wish to retain the old version of the Oracle software, you will have to change only the symbolic link definition for */opt/SUNWora/oracle*. However, if you used */opt/SUNWora/7.3.2.3* to specify the

path for the Oracle software, you must change each occurrence of this path declaration.



**Figure 13-2.** Recommended Oracle Software Directory Structure

In each subsection we will specify these environmental specifications (i.e., the *SUNWxxx* directory, version directory, and symbolic link names) for that package.

#### 13.4.2 Environment Settings and *xmyProfile* and *xmyLogin*

You will be prompted to enter a value for the shell variable *\$XMYHOME* during the BAIST installation process. (See [Section 13.6.2.2](#).) This variable is set in the files *xmyProfile* and *xmyLogin*, which are installed in the *\$XMYHOME/config* directory, assuming you have used our recommended directory structure.

Users who wish to access the AETG System must source either *xmyProfile* or *xmyLogin* into their *.profile* or *.login*, respectively. See the AETG postinstallation steps in [Section 13.6.2.3](#).

**NOTE** — The AETG System is installed into two *different* directory structures. The `$XMYDIR` variable designates the directory containing the AETG software. (Using the recommended directory structure, this is `/opt/SUNWmyn/aetg`.) The `$XMYHOME` variable designates the directory containing your configuration files and run logs. We require that this be a directory that all AETG users have access to since the run log directory needs to have open write permissions. The default will be `/opt/SUNWmyn/aetg_home`.

The `xmyProfile` file contains the declarations for **ksh** environment variables for the installed software based on our recommended locations. The `xmyLogin` file contains the declarations for **cs**h environment variables for the installed software based on our recommended locations. For example, the recommended location for the home of the Oracle software, `ORACLE_HOME`, is `/opt/SUNWora/oracle`.

The `xmyProfile` and `xmyLogin` files also contain “place-holders” for certain environment settings that will be unique for your installation, such as the Telexel processes port number. As you determine the values for these settings you can either enter them directly into these files or write them down and make all of your changes at one time.

Once you have completed installing the AETG System and all other required and optional software packages, you should source the `xmyProfile` or `xmyLogin` file into your `.profile` or `.login`, respectively, depending on which shell you are using.

**NOTE** — Most of the environment variables will be updated automatically during the BAIST installation process, but the `xmyProfile` and/or `xmyLogin` file need to be verified manually.

- If you are using **ksh**, source the `xmyProfile` file into your `.profile` (and all AETG user’s `.profile`’s) by including the following line:

```
. /opt/SUNWmyn/aetg_home/config/xmyProfile
```

This assumes that the user chose the default path for `XMYHOME`. (See [Section 13.6.2.2](#).)

See [Appendix C.1.3](#) for an example of the `xmyProfile` file.

- If you are using **cs**h, source the `xmyLogin` file into your `.login` (and all AETG user’s `.login`’s) by including the following line:

```
source /opt/SUNWmyn/aetg_home/config/xmyLogin
```

See [Appendix C.1.4](#) for an example of the `xmyLogin` file.

---

**NOTE** — During installation you may have to reset the environment variable `LD_LIBRARY_PATH` due to its value being lost after executing an `su` command.

**NOTE** — While the other shells (`sh` and `csch`) are supported, all example shell commands shown in the installation steps assume you are using `ksh`.

### 13.4.3 Requirements

The `$XMYDIR` and `$XMYHOME` directories must be accessible to all AETG machines and users. This can usually be accomplished by automounting these file systems on all other machines. If X-Terminals or PCs are used then the user will be logging on to the AETG server, so this will not be necessary for these users.

## 13.5 Preliminary Actions

Before you install the AETG System, there are certain actions you must perform.

### 13.5.1 Obtaining License Keys

The AETG System software uses a floating license scheme involving a license-server daemon that runs on one machine in your network and takes requests for licenses from any machine on your network.

To run the AETG System software, you must first install a licensing key.

Generating the licenses is performed by Bellcore, and the keys are then sent to the AETG customers. The contract between the customers and Bellcore defines which of the preceding products will have keys generated and for what time period.

Perform the following steps to obtain your license keys:

1. Decide which machine on your network will run the License Server daemon. (The daemon will run only on the machine for which it is licensed.) This is usually the machine on which the AETG System software is installed, although it can be any machine on your network.
2. Once you have decided which machine will run the License Server, determine its hostid by typing the either of the following:
  - `/usr/sbin/sysdef -h`
  - `/usr/sbin/hostid`

**WARNING** — The License Server software (LicenseServ) does not properly process dashes (-) in a directory path.

The hostid (in hexadecimal) is used to create the license key.

3. Once you have obtained this information, call 1-908-699-2668, Option 3 or 1-(800)-795-3119, Option 3 or send e-mail to `mynah-support@cc.bellcore.com`

You will receive a list containing your key(s). This list is sent to you via the medium of your choice—letter, e-mail (if available), phone, or fax.

**NOTE** — Save these keys until you are ready to install them. (See [Section 13.9](#).)

### 13.5.2 Creating the mynah Group and AETG Administrator logid (madmin)

Perform the following steps before you begin installing the AETG System:

1. Decide which machine will be used as the AETG server.
2. Become a superuser, i.e., type  
`su root`
3. Create a group called **mynah**.
4. Create a logid for the AETG Administrator in the group **mynah**, i.e., **madmin**.

**NOTE** — The AETG Administrator, **madmin**, should have a home directory and default shell, e.g., **ksh**.

5. Log onto the machine (as **madmin**) where you will be installing the system.

### 13.5.3 Changes to `/etc/services`

During installation, several changes must be made to the `/etc/services` file. For example, when configuring the Telexel System ([Section 13.7.3](#)), you must define and export **vxIpcPort**, which is the port used by the Telexel processes.

During installation, a file called *etc.system.changes.eg* is placed in the *\$XMYDIR/examples/admin/scripts* directory. This file contains changes that should be added to the */etc/system* file. Once the changes have been made, type

```
reboot -- -rt
```

to reboot the system. The system will be reconfigured with the changes to */etc/system* incorporated in the kernel. You must do this as **root** on each system running a AETG component and on the ORACLE server.

**NOTE** — See Appendix C.1.2 to see a copy of the *etc.system.changes.eg*.

### 13.5.4 BAIST Considerations

The AETG System is delivered using the Bellcore standard UNIX installation process, BAIST. The BAIST archive is delivered via a file archive obtained from the AETG **ftp** server or a CD-ROM. Whichever medium you use, the BAIST package contains archives for the AETG and Telexel software.

This subsection describes the BAIST preinstallation procedures.

Installation must be performed as **root** on the AETG machine, and **root** must have write permission to the file system where the installation will be performed.

If installation is performed via CD-ROM, the CD-ROM drive must be local to the machine.

Create a directory for the BAIST installation database, e.g., */usr/local/BCRDB*, if it has not already been created. This directory must be owned by **root**. You must also perform several actions to set the environment variable called BCRDB to the location of the BAIST installation database directory. How you set BCRDB depends on which UNIX shell (**sh**, **cs**, or **ksh**) you are using.

For all three shells, perform the following:

```
su root
mkdir /usr/local/BCRDB
```

If you're using **sh**, enter

```
BCRDB=/usr/local/BCRDB
LOCAL_BCRDB=yes
export BCRDB LOCAL_BCRDB
```

If you're using **cs**, enter

```
setenv BCRDB /usr/local/BCRDB
setenv LOCAL_BCRDB yes
```



If you're using **ksh**, enter

```
export BCRDB=/usr/local/BCRDB
export LOCAL_BCRDB=yes
```

Create the directory into which the installed software will go (e.g., */opt/SUNWmyn*) prior to performing the installation. This directory must be owned by a user other than **root**, e.g., the AETG System Administrator, **madmin**.

For its own scripts, BAIST uses the **ksh** in *\$BCRDB/tools*. However, some of the scripts used by the products BAIST installs, such as configuration scripts, may be using */bin/ksh*. So it will be advisable to install the newer version of **ksh**.

The commands you use to start installation from the BAIST archive depends on which medium you use and will be explained in the following subsections. Whichever method you use, after you start the BAIST archive, the BAIST **Opening Screen** ( [Figure 13-3](#)) appears.

```

Bellcore Application Installation Setup Tool
- - - - -
                BAIST 2.1

COPYRIGHT (c) 1996 Bell Communications Research Inc.,
                All Rights Reserved.

PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.

                A UNIX Packaging and Installation Tool

                For further information on BAIST
                Contact:  Raymond C. Gray
                        BAIST Project Manager
                        (908)699-7960
```

**Figure 13-3.** BAIST Opening Screen

Following the BAIST **Opening Screen**, the **BAIST Product Menu** ( [Figure 13-4](#)) appears, prompting you for which package you wish to install.

```
BAIST 2.1 FLOW CONTROL
PRODUCT MENU
  Archived Products List

      1) AETG 1.0
      2) AETG UPDATES
      3) TELEXEL 7.0

      I) Installed Products
      R) Registered Products
      E) Exit

Enter your Selection:
```

**Figure 13-4.** BAIST Product Menu

After a software package is installed, the **BAIST Product Menu** will reappear. You can then select to install one of the other packages or exit from the archive. For example, during initial installation of the AETG System, you may wish to first install the AETG software, followed by the Telexel software. You can then perform the necessary post installation or configuration steps needed by the AETG and Telexel software packages. You may instead wish to install one software package, exit from the BAIST archive, configure the installed software, and then restart the BAIST archive and install and configure another package.

Prior to installing any package, a home directory must be created for the package. You may find it convenient to create all of these directories up front rather than individually. (See [Section 13.6.1](#) and [Section 13.7.1](#).)

The following subsections detail installing one package at a time. You may also use this method if you wish to install only one of the packages, such as if you receive an updated version of one software package while retaining the currently installed versions of the other packages.

**NOTE** — If you wish to install the AETG and Telexel software packages during one BAIST session, remember to perform the necessary preinstallation steps for each package before starting the archive.

## 13.6 Installing the AETG System

### 13.6.1 AETG System Preinstallation Steps

Perform the following steps before installing the AETG System:

1. Become a superuser, i.e., type  

```
su root
```
2. Change to the */opt* directory, i.e., type  

```
cd /opt
```
3. Create the *SUNWmyn* directory, i.e., type  

```
mkdir SUNWmyn
```
4. Change the owner of */opt/SUNWmyn* to **madmin**, i.e., type  

```
chown madmin /opt/SUNWmyn
```
5. Change the group of */opt/SUNWmyn* to **mynah**, i.e., type  

```
chgrp mynah /opt/SUNWmyn
```

### 13.6.2 AETG Software Installation

The AETG System is delivered, along with the Telexel software, via a CD-ROM or a file archive obtained from the AETG **ftp** server. You are prompted for the package you wish to install.

**NOTE** — When installing the AETG System as **root** on a remote filesystem (i.e., installing the software in a filesystem that is not local to the machine on which the installation is being performed), you may not really have **root** access to that file system. The installation could fail with permissions errors.

Install the AETG software using one of the following methods:

#### 13.6.2.1 AETG Software CD-ROM Installation

For installation from CD-ROM, see the instructions in the CD-ROM case.

### 13.6.2.2 AETG Software File Archive Installation

If you are installing from a file, perform the following steps:

1. Obtain the BAIST archive file from the **ftp** server.
2. Enter the following command:

```
/bin/sh archive-file-name
```

where *archive-file-name* is the name of the BAIST archive file.

3. Select the **AETG 1.0** option from the **BAIST Product Menu** (Figure 13-4).

**NOTE** — See Appendix C.1.1 for an example of a BAIST installation session of the AETG System software.

4. Answer the following questions asked by the BAIST installation software:

Where should the AETG directory be created? **/opt/SUNWmyn**

Who should own the AETG application? **madmin**

Where should XMYHOME be? **<enter your location for XMYHOME>**

**NOTE** — *\$XMYHOME* *must* be different than the AETG software directory (*\$XMYDIR*).

5. After the AETG software is installed, perform one of the following:
  - Install the Telexel (Section 13.7) software package if it has not already been installed, and then exit from the BAIST archive.
  - Exit from the BAIST archive and perform the AETG post installation steps in Section 13.6.2.3.

### 13.6.2.3 AETG Post Installation Steps

Once you've installed the software, verify and correct, if necessary, the value assigned to XMYDIR in your working copy of the *xmyProfile* or *xmyLogin* files.

1. Become **madmin**, i.e., type

```
su madmin
```

2. Verify that a symbolic link exists pointing to the version directory (created by the BAIST installation process), i.e., type

```
cd /opt/SUNWmyn
```

```
ls -al
```

3. Verify and correct, if necessary, the value assigned to XMYDIR in your working copy of the *xmyProfile* or *xmyLogin* files.

## 13.7 Installing the Telexel System

The AETG System requires the Telexel System for inter-process communications and logging.

**NOTE** — The Telexel System must be installed on a local filesystem or on a remote filesystem that is mounted with the **suid** option.

### 13.7.1 Telexel System Preinstallation Steps

Perform the following steps before installing the Telexel System:

1. Become a superuser, i.e., type  

```
su root
```
2. Change to the */opt* directory, i.e., type  

```
cd /opt
```
3. Create the *SUNWtel* directory, i.e., type  

```
mkdir SUNWtel
```
4. Change the owner of the *SUNWtel* directory to **madmin**, i.e., type  

```
chown madmin SUNWtel
```
5. Change the group of the *SUNWtel* directory to **mynah**, i.e., type  

```
chgrp mynah SUNWtel
```

**NOTE** — The Telexel installation process automatically creates a separate directory for the version of the software being installed. In addition, the software package includes a utility program, **vxInstall**, that creates a series of symbolic links from the version directory into */opt/SUNWtel*. This is handled by the BAIST Telexel post installation step.

## 13.7.2 Telexel Installation

The Telexel System is delivered, along with the AETG software, via a CD-ROM or a file archive obtained from the AETG **ftp** server. You are prompted for the package you wish to install.

**NOTE** — When installing the Telexel System as **root** on a remote filesystem (i.e., installing the software in a filesystem that is not local to the machine on which the installation is being performed), you may not really have **root** access to that file system. The installation could fail with permissions errors.

Install the Telexel software using one of the following methods:

### 13.7.2.1 Telexel Software CD-ROM Installation

For installation from CD-ROM, see the instructions in the CD-ROM case.

### 13.7.2.2 Telexel Software File Archive Installation

If you are installing from a file, perform the following steps:

1. Obtain the BAIST archive file from the **ftp** server.
2. Enter the following command:  

```
/bin/sh archive-file-name
```

where *archive-file-name* is the name of the BAIST archive file.
3. Select the **Telexel 7.0** option from the **BAIST Product Menu** (Figure 13-4).

**NOTE** — See Appendix C.2 for an example of a BAIST installation session of the Telexel System software.

4. Answer the following questions asked by the BAIST installation software:  
Where should the TELEXEL directory be created? `/opt/SUNWtel`  
Who should own the TELEXEL application? `admin`

5. After the Telexel software is installed, perform one of the following:
  - Exit from the BAIST archive and perform the Telexel post installation, configuration, and verification steps in Sections [13.7.2.3](#), [13.7.3](#), and [13.7.4](#), respectively.
  - Install the AETG software package ([Section 13.6](#)) if it has not already been installed, and then exit from the BAIST archive.

### 13.7.2.3 Telexel Post Installation Steps

Once you've installed the software, execute the following commands to verify that a symbolic link exists to the version directory:

```
su madmin
cd /opt/SUNWtel
ls -al
```

### 13.7.3 Configuring the Telexel System

To configure the Telexel System, you must define and export the following environment variables. You must have already installed and configured the AETG software for these files to exist and before performing the following steps.

**NOTE** — Remember to edit your working copy of the *xmyProfile* or *xmyLogin file*, defining and exporting these variables.

1. Become **madmin**, i.e., type

```
su madmin
```

**NOTE** — Steps 2 through 7 can be accomplished as per [Section 13.4.2](#) or manually as described below.

2. The following constraints must be followed in the */etc/system* file:
  - A. The Message Queue parameters MSGMNB and MSGMAX must be equal, e.g.,

```
set msgsys:msginfo_msgmnb=65535
set msgsys:msginfo_msgmax=65535
```

**NOTE** — MSGMNB defines the maximum bytes on the queue, and MSGMAX sets the maximum size of the message.



B. MSGMAX must be less than or equal to the product of MSGSEG and the MSGSSZ parameters.

**NOTE** — MSGSEG sets the number of message segments, and MSGSSZ sets the segment size of message.

For example, if MSGSEG is set to 16384 and MSGSSZ is set to 64, then MSGMAX must be less than or equal to 1,048,576.

### 13.7.4 Verifying the Telexel System

To manually verify your Telexel installation, **cd** to *\$TELDIR/bin* then perform the following tasks:

1. Become **madmin** and change to the *\$TELDIR/bin* directory, i.e., type

```
su - madmin  
cd $TELDIR/bin
```

2. Type

```
./vxIpcDir
```

to start the IPC process.

3. Type

```
./vxIpcProcesses
```

to verify that the process started. You should get this:

```
IPC Registered Processes  
  
ID                PID    HOST                QUEUE USER  
==                ===    =====            =====  
=====
```

If you get this,

```
vxIpcProcesses: can't retrieve process list (error 1-IP-0024, errno 146).
```

then **vxIpcDir** did not start. If you have to restart this process, you may have to wait for the port to time-out.

4. Type

```
./vxIpcUp
```

to start the Telexel Gateway. This starts a process called **vxIpcRecvd**.

5. Type

```
./vxLogToFile $XMYHOME/syslog/adminLog
```

to start the Telexel Logger.

6. Type

```
./vxErrorServer $TELDIR/lib/errorText \  
$XMYDIR/lib/xtw_error_text $XMYDIR/lib/xmyErrorText
```

to start the Telexel Error Server.

**NOTE** — The XMYDIR environment variable must be set to the AETG 5.2 installation directory. Both the XMYHOME and XMYDIR variables must be exported for Steps 5 and 6. See Section 13.6.2.3. This was done as part of the AETG installation process.

7. Execute

```
ps -ef | grep vx
```

You should expect output similar to the following:

```
madmin 4479 1 80 Jul 01 ? 9:59 vxIpcDir  
madmin 4511 1 80 Jul 01 ? 0:31 /opt/SUNWtel/telexel/lib/vxIpcRecvd  
madmin 22281 1 80 Jul 05 ? 0:01 vxLogToFile /opt/SUNWmyn/aetg/syslog/adminLog  
madmin 22290 1 80 Jul 05 ? 0:02 vxErrorServer /opt/SUNWtel/telexel/lib/errorText
```

These four processes should always be running on the AETG server.

**NOTE** — Other vx (Telexel) processes may also appear. The preceding processes constitute the minimum set.

## 13.8 Installing Oracle

The AETG System requires the use of an Oracle database.

Although you will follow the Oracle installation procedures, there are certain factors that you must take into consideration. For example, you must install several specific Oracle packages and create specific environment variables. This subsection discusses these factors.

**NOTE** — This subsection assumes you do not have an existing Oracle database and are installing one for use with the AETG System.

### 13.8.1 Oracle Preinstallation Steps

**NOTE** — Refer to the *Oracle Installation and Configuration Guide* for the version of the software being loaded. (Hereafter we refer to this guide as the *Oracle Manual*). For this installation we will reference those items relating to Oracle 7.3.2.3.

Perform the following steps before installing the Oracle System:

**NOTE** — The following steps assume you are using the **ksh** shell.

1. Create a new user named **oracle** with a group id of **dba**. (If necessary, create the group first.)

**NOTE** — You must be `root` to do perform this step.

2. Become a superuser, i.e., type  

```
su root
```
3. Change to the `/opt` directory, i.e., type  

```
cd /opt
```
4. Create the `SUNWora` directory, i.e., type  

```
mkdir SUNWora
```
5. Change to the `/opt/SUNWora` directory, i.e., type  

```
cd /opt/SUNWora
```

6. Under `/opt/SUNWora`, create the version directory, i.e., type  

```
mkdir 7.3.2.3
```
7. Create the symbolic link pointing to the version directory in `/opt/SUNWora`, i.e., type  

```
ln -s /opt/SUNWora/7.3.2.3/app/oracle/product/7.3.2.3 oracle
```
8. The owner of the `/opt/SUNWora` directory must be set to **oracle** and the group id to **dba**. Execute the following steps:  

```
cd /opt  
chown -R oracle SUNWora  
chgrp -R dba SUNWora
```
9. When installing Oracle, ensure that the correct packages are loaded. For Oracle Release 7.3.2.3, the following packages are required:

**Solaris Version**

SUNWbtool  
SUNWspot  
SUNWtoo  
SUNWarc  
SUNWlibm  
SUNWlibms

You can use the Oracle utility **pkginfo** to ensure that a package exists. For example, to verify that the SUNWbtool package has been loaded (if you are using the Solaris version), type

```
pkginfo -i SUNWbtool
```

See Solaris version of the *Oracle Manual* for updated information.

10. Define and export the following environment variables, but update only `ORACLE_HOME` and `TWO_TASK` in your working copy of the *xmyProfile* or *xmyLogin* file:

```
export ORACLE_HOME=/opt/SUNWora/oracle  
export ORACLE_TERM=sun5  
export ORACLE_SID=mynah5  
export TWO_TASK=mynah5
```

**NOTE** — `TWO_TASK` should be unset until you perform Step 5 when configuring the AETG Oracle database ([Section 13.8.4](#)).

11. Create a directory under `$ORACLE_HOME` named `mynah5` with the owner set to **oracle** and the group to **dba**, i.e., type

```
cd $ORACLE_HOME
mkdir mynah5
chown oracle mynah5
chgrp dba mynah5
```

12. Under `$ORACLE_HOME/mynah5` create the `datafiles` and `logfiles` directories with the owner set to **oracle** and the group to **dba**:

```
cd mynah5
mkdir datafiles logfiles
chown oracle datafiles logfiles
chgrp dba datafiles logfiles
```

13. The following is the recommended Oracle database disk configuration:

- disk1 - Tables and system
- disk2 - Index and some logs and control files
- disk3 - Rollback segments.

If you have only one disk available for the Oracle database, proceed to Item A. If you have multiple disks, proceed to Item B.

- A. Under `$ORACLE_HOME/mynah5/datafiles` create the directories `d01`, `d02`, and `d03` with the owner set to **oracle** and the group to **dba**, i.e., type

```
cd datafiles
mkdir d01 d02 d03
chown oracle d01 d02 d03
chgrp dba d01 d02 d03
```

**NOTE** — Directory `d01` contains items in disk1, directory `d02` contains items in disk2, and directory `d03` contains items in disk3.

Proceed to Step 14.

- B. Under `$ORACLE_HOME/mynah5/datafiles` create the following symbolic links to the directories on other disks, e.g., type

```
cd datafiles
ln -s <disk1> d01
ln -s <disk2> d02
ln -s <disk2> d03
```

Proceed to Step 14.

14. Copy the following files from *\$XMYDIR/examples/dbadmin* to *\$ORACLE\_HOME/mynah5*:

- *configmynah5.ora* (See [Appendix C.3.2](#) for an example *configmynah5.ora*.)
- *initmynah5.ora* (See [Appendix C.3.3](#) for an example *initmynah5.ora*.)
- *crdbmynah5.sql* (See [Appendix C.3.4](#) for an example *crdbmynah5.sql*.)
- *crdb2mynah5.sql* (See [Appendix C.3.5](#) for an example *crdb2mynah5.sql*.)
- *crdb3mynah5.sql* (See [Appendix C.3.6](#) for an example *crdb3mynah5.sql*.)

After you've copied the files, you must change the permissions on the file by executing the following:

```
cd $ORACLE_HOME/mynah5
chmod 775 *
```

You may need to edit the *\*.ora* and *\*.sql* files to update the correct directories paths (such as where Oracle is installed) and possibly to change the sizing information.

**NOTE** — Do not use environment variables to define these paths.

15. Make the changes to the */etc/system* file (See [Section 13.5.3](#)) as shown in *\$XMYDIR/examples/admin/scripts/etc.system.changes.Sun.eg*.

You must reboot the system with the *ste* reconfiguration options before you install the Oracle database ([Section 13.8.2](#)).

To reboot, execute

```
reboot -- -rt
```

## 13.8.2 Oracle Installation

To install the Oracle database, follow these steps.

**NOTE** — The system must be rebooted using the **-r** option before executing the Oracle installation software.

For assistance, refer to the *Oracle Manual*.

**NOTE** — Do not use environment variables or symbolic links until Oracle is installed.

1. Mount the CD-ROM containing the Oracle software using one of the three following methods
  - Execute

```
mount -r -F hsfs /dev/dsk/cot6d0s1 /cdrom
```
  - Execute

```
volcheck cdrom
```
  - If you're using volume manager, the CD-ROM is automatically mounted when you insert the disk.
2. Become the user **oracle**, i.e., type

```
su oracle
```
3. Change to the */cdrom/oracle/orainst* directory. i.e., type

```
cd /cdrom/oracle#1/orainst
```
4. Make sure the terminal type is set to vt100, and clear the screen, i.e., type

```
export TERM=vt100
clear
```
5. Start an xterm window, i.e., type

```
xterm &
```

This will let you use the arrow keys and display used by the Oracle installation software.
6. Start the Oracle install program. i.e., type in this xterm window

```
./orainst
```

Table 13-3 lists the Oracle installation items and appropriate responses you must perform to ensure that the database will work properly with the AETG System.

**Table 13-3.** Oracle Installation Items and Responses (Sheet 1 of 2)

Item	Response
Preamble.txt	OK
Installation Activity Choice	Install, Upgrade, or De-Install
Installation Option	Install New Product
Mount Point	<i>/opt/SUNWora/7.3.2.3</i>
Home Location	7.3.2.3
DB Objects - Create?	No
Logging and Status	OK
readme.first	OK
Skip readme	OK
Install source	CD-ROM
NLS	Amer.
Relink all Exec.	No
Information	OK
On-line Help Load	All Prod (Optional)
UNIX Documentation	Yes (Optional)
Product Documentation Library	All Prod (Optional)
Oracle Documentation	<i>/opt/SUNWora/7.3.2.3/app/oracle/doc</i>
Software Asset Manager	<p>At this step of the installation process, the Oracle System displays a scrollable list showing the available Oracle packages. To select which packages to install</p> <ol style="list-style-type: none"> <li>1. Use the arrow keys to scroll through the list of packages.</li> <li>2. Press the space bar to select a package when it is highlighted.</li> <li>3. When you have selected all of your desired packages, press the TAB key to go to the Install button and press the Return key.</li> </ol> <p>Table 13-4 lists the packages that must be installed.</p>
Official Hostname	Enter your machine name (including the domain).
TCP Surf Port	8888 or any unused port number.
Password and Verify Password	any



**Table 13-3.** Oracle Installation Items and Responses (Sheet 2 of 2)

Item	Response
DBA Group	OK
OSOPER Group	OK
DBA doesn't exist	Yes (continue)
Enter Oracle Sid	mynah5
X Libraries	/usr/openwin/lib
Run root.sh	OK

**Table 13-4.** Oracle Software Asset Manager Packages

Package	Version Number
Oracle Server Manager	V2.3.2.0.0
Oracle UNIX Installer	V4.0.0.0.0
Oracle Server RDBMS	V2.3.2.3.0
PL/SQL	V2.3.2.3.0
SQL*Net (V2)	V2.3.2.1.0
SQL*Plus	V3.3.2.0.0
TCP/IP Protocol Adapter	V2.3.2.3.0

### 13.8.2.1 Verifying an Installation

To verify your actions during the installation, view the *install.log* file by executing either of the following in *\$ORACLE\_HOME/orainst*:

```
tail -f install.log  
cat install.log | grep code
```

You should not get errors from the install process (i.e., all return codes should equal 0).

### 13.8.2.2 Oracle Error Messages

If you get ORACLE error messages, type

```
oerr xxx ####
```

to find out what the error is, where **xxx** is *ORA* or *DBA* and **####** is the error number.

### 13.8.3 TNS Configuration

The Oracle environment must be configured for the TNS Listener process. These files are either stored in `/var/opt/oracle` or in `$ORACLE_HOME/network/admin`. In the latter case or if these files are in a directory other than `/var/opt/oracle`, then an environment variable, `TNS_ADMIN`, must be defined to point to this directory. See the `S96oracle.eg` file in `$XMYDIR/examples/admin/scripts` for an example.

**NOTE** — This environment variable must be updated in the `$XMYHOME/config/xmyProfile` and `$XMYHOME/config/xmyLogin` files.

Sample `tnsnames.ora.eg` and `lisenter.ora.eg` files are also included in `$XMYDIR/examples/admin/scripts` and in Appendix C. These example files must be edited for your environment, renamed without the `.eg` extension, and moved to the desired location.

### 13.8.4 Configuring the AETG System Oracle Database

**NOTE** — This subsection contains several example executions of the utilities used to configure the Oracle database. All-user supplied entries appear in bold.

Since Oracle is now installed, you are ready to configure the AETG Oracle database.

1. If you are still not a superuser, become one, i.e., type

```
su root
```

2. Create a directory in `/var/opt` named `oracle` with the owner set to **oracle** and the group to **dba**, i.e., type

```
cd /var/opt
mkdir oracle
chown oracle oracle
chgrp dba oracle
```

This directory could also be a symbolic link from `/var/opt/oracle/oratab` to `/etc/oratab`.

3. Copy the `initmynah5.ora` and the `configmynah5.ora` files from `$ORACLE_HOME/mynah5` (See preinstallation Step 14 in Section 13.8.1) to `$ORACLE_HOME/dbs`. Make sure these files have been edited and updated.
4. Change to `$ORACLE_HOME/mynah5` and become the user **oracle**, i.e., type

```
cd $ORACLE_HOME/mynah5
su oracle
```

5. Execute the following commands (being sure to complete the preinstallation Step 13 first):

```
svrmgr1
SVRMGR> connect internal
SVRMGR> startup nomount pfile=/opt/SUNWora/oracle/dbs/initmynah5.ora
SVRMGR> @crdbmynah5.sql
SVRMGR> connect internal
SVRMGR> @crdb2mynah5.sql
SVRMGR> connect internal
SVRMGR> @crdb3mynah5.sql
SVRMGR> connect internal
SVRMGR> shutdown immediate
SVRMGR> exit
```

6. Change to `$ORACLE_HOME/orainst`, i.e., type

```
cd $ORACLE_HOME/orainst
```

7. Become a superuser, i.e., type

```
su root
```

8. Run `root.sh`, which was created by the Oracle install process.

The following is a sample run.

```
./root.sh
Running ORACLE7 root.sh script...
```

The following environment variables are set as:

```
ORACLE_OWNER= oracle
ORACLE_HOME= /opt/SUNWora/oracle
ORACLE_SID= mynah5
```

Are these settings correct (Y/N)? [Y]: **Y**

Enter the full pathname of the local bin directory  
[/opt/bin]: **/usr/local/bin**

```
Checking for "oracle" user id...
ORACLE_HOME does not match the home directory for oracle.
Okay to continue? [N]: Y
```

```
Creating /var/opt/oracle/oratab file...
Updating /var/opt/oracle/oratab file...
```

Please raise the ORACLE owner's ulimit as per the IUG.

```
Leaving common section of ORACLE7 root.sh.
Setting orasrv file protections
```

9. Edit the `/var/opt/oracle/oratab` file, then change the *N* to *Y* on the actual data line (i.e., the last line).
-

---

**WARNING** — *Do not use* a symbolic link name in this file. This entry is case-sensitive, so you must enter a capital *Y*.

See the comments in the *oratab* file for more information.

10. Become the user **oracle**, i.e., type

```
su oracle
```

11. Execute the following commands:

```
svrmgrl
```

```
svrmgrl: Release 7.3.2.3.0 - Production on Thu Apr 18 11:07:43 1996
```

```
Copyright (c) Oracle Corporation 1979, 1994. All rights reserved.
```

```
Oracle7 Server Release 7.3.2.3.0 - Production Release  
With the distributed, replication and parallel query options  
PL/SQL Release 2.1.6.2.0 - Production
```

```
connect internal
```

```
Connected.
```

```
startup pfile=/opt/SUNWora/oracle/dbs/initmynah5.ora
```

```
ORACLE instance started.
```

```
Total System Global Area          5079016 bytes
```

```
      Fixed Size                    39696 bytes
```

```
      Variable Size                 4621528 bytes
```

```
Database Buffers                   409600 bytes
```

```
Redo Buffers                        8192 bytes
```

```
Database mounted.
```

```
Database opened.
```

```
exit
```

12. To verify the database creation process, execute the following:

```
svrmgrl
```

```
SVRMGR> connect internal
```

```
SVRMGR> select * from v$controlfile;
```

```
      $ORACLE_HOME/dbs/ctrl1mynah5.ctl
```

```
      $ORACLE_HOME/mynah5/datafiles/d02/ctrl2mynah5.ctl
```

```
      $ORACLE_HOME/mynah5/datafiles/d03/ctrl3mynah5.ctl
```

```
SVRMGR> select file_name,status from dba_data_files;
```

```
      $ORACLE_HOME/mynah5/datafiles/d01/systmynah5.dbf      AVAILABLE
```

```
      $ORACLE_HOME/mynah5/datafiles/d02/rbsmynah5.dbf      AVAILABLE
```

```
      $ORACLE_HOME/mynah5/datafiles/d03/toolmynah5.dbf     AVAILABLE
```

```
      $ORACLE_HOME/mynah5/datafiles/d03/my5mynah5.dbf     AVAILABLE
```

```
      $ORACLE_HOME/mynah5/datafiles/d01/usrmynah5.dbf     AVAILABLE
```

```
      $ORACLE_HOME/mynah5/datafiles/d01/tempmynah5.dbf    AVAILABLE
```

13. Become the AETG Administrator (**madmin**), i.e., type

```
su madmin
```

14. Execute *xmyProfile* or *xmyLogin* in the AETG config directory (*\$XMYHOME*). Make sure that **TWO\_TASK** is unset when the database is local and set when the database is remote to the system which you are on. Make sure **ORACLE\_SID** equals *mynah5*.
15. Create the AETG database using the **xmyCreate** commands (e.g., **xmyCreateSequences** and **xmyCreateDemoObjects**).

A. **cd** to *\$XMYDIR/dbadmin*.

B. Execute

```
xmyCreateTables
```

You should see the following:

```
Database connection opened.  
Tables do not exist, creating them.
```

C. Execute

```
xmyCreateSequences
```

**xmyCreateSequences** creates all of the sequences needed by the AETG database to operate.

See Appendix [C.3.7](#) for an example **xmyCreateSequences** execution.

D. Execute

```
xmyCreateTemplates
```

See Appendix [C.3.8](#) for an example execution.

16. Verify the TCP port in the */etc/services* file by searching for an entry such as the following:

```
tnslsnr      1521/tcp    #oracle listener
```

If this line is not present and NIS is not used, add this line to the */etc/services* file.

**WARNING** — You must do this in every */etc/services* file on all machines running the AETG System.

If NIS is used, then the following can be in the *ypservices* file to verify this */etc/services* file entry:

```
ypcat services | grep 1521
```

If the 1521 port is being used, then choose another port on all machines that is not being used.

### 13.8.5 Dropping the Oracle Database

During installation, the following commands are installed in `$ORACLE_HOME/mynah5`:

- **xmyDropSequences.**
- **xmyDropTables**

Execute these commands if something goes wrong and the database needs to be cleaned up/deleted.

If problems are encountered with the database, consult AETG support before running the previous commands.

**WARNING** — These operations are drastic in nature, and thus they should not be run if the database has been populated with important data. They may, however, be useful when first installing a database.

Once these commands have finished executing, run the **xmyCreate** commands (Section 13.8.4, Step 15) to recreate the tables in the AETG database.

### 13.8.6 Verifying Oracle

To verify if everything is up and working in Oracle, do the following:

1. Copy the `S96oracle.eg` file from `$XMYDIR/examples/admin/scripts` to `/etc/rc2.d` and then rename it `S96oracle`.
2. Set the `ORACLE_HOME` variable in `S96oracle` to the correct path, then verify all other paths.
3. Execute

```
su root
/etc/rc3.d/S96oracle start
or
/etc/rc3.d/S90oracle start
```

The following Oracle processes should be up and started:

- `tnslsnr LISTENER`
- `ora_reco_mynah5`
- `ora_smon_mynah5`
- `ora_pmon_mynah5`

- ora\_lgwr\_mynah5
- ora\_dbwr\_mynah5
- ora\_s000\_mynah5
- ora\_d000\_mynah5

4. Execute a command of the following form:

```
svrmgr1 system/manager@mynah5
```

You should get a good connection.

You can also try any of the following to verify your Oracle installation:

1. Verify the existence of the `$ORACLE_HOME/rdbms/log/alert_mynah5.log` file.
2. If problems occur, try to set up a symbolic link in `/etc`, for example,

```
ln -s /var/opt/oracle/oratab oratab
```

3. Verify

- oracle permissions 6751

in `$ORACLE_HOME/bin`.

4. If you must relink, try the following:

```
$ORACLE_HOME/rdbms/lib      /usr/ccs/bin/make -f oracle.mk install  
$ORACLE_HOME/network/lib   /usr/ccs/bin/make -f network.mk install  
$ORACLE_HOME/sqlplus/lib   /usr/ccs/bin/make -f sqlplus.mk install
```

5. To verify what is linked in Oracle, use the **adapters** and **drivers** command, as in the following:

```
adapters
```

```
Installed SQL*Net V2 Protocol Adapters are:
```

```
V2 BEQ Protocol Adapter  
V2 IPC Protocol Adapter  
V2 TCP/IP Protocol Adapter  
V2 Raw Protocol Adapter
```

## 13.9 Installing the License Keys

Before you use the AETG System, you must install the AETG System license key. (See Section 13.5.1 for information on obtaining license keys.)

Perform the following steps to install the license keys:

1. Put each key on a line by itself in a file called *xmyLicenses*.
2. Place the *xmyLicenses* file in *\$XMYHOME/config*. If *xmyLicenses* already exists and contains other keys, add the new key after the last line in the file. Expired keys should be deleted.
3. Execute the following command to check that port 5093 is not currently being used:

```
netstat | grep 5093
```

If this port is being used, call the AETG hotline. (See Section 13.5.1.) Otherwise, edit the */etc/services* file, then add the following line:

```
LicenseServ          5093/udb
```

4. Set and export the variable `LSHOST`, which is the license server machine. Remember to verify/update that `LSHOST` is set in the *\$XMYHOME/config/xmyProfile* or *\$XMYHOME/config/xmyLogin* file.
5. Selected or deselect license keys by uncommenting or commenting (#) the keys.



## 13.10 Installing the On-line Documents

This document is available on-line in the Adobe Acrobat PDF format (Release 3.0). When installing the AETG System, a tar file, *mynah\_aetg.tar* is placed in *\$XMYDIR/doc*. This archive contains the PDF file and an HTML web page, which will let users access the PDF file from non-local systems, such as if the AETG System is installed on a server and the users must **telnet** to the AETG server.

Viewing the PDF file requires that users have installed the Adobe Acrobat Reader. See [Section 13.10.2](#) for information obtaining the Acrobat Reader.

### 13.10.1 Installing the PDF Files

Unpackage the **tar** archive by executing the following commands:

```
cd $XMYDIR/doc
tar -xvf mynah_aetg.tar
```

A directory called *mynah\_aetg* is created under *\$XMYDIR/doc* containing the MYNAH and AETG documentation in PDF format.

### 13.10.2 Obtaining the Acrobat Reader

The Acrobat Reader is available directly over the Internet from Adobe at [www.adobe.com](http://www.adobe.com). In addition, the Acrobat Reader is included on the CD-ROM or as a file archive obtained from the AETG **ftp** server.

The following versions of the Acrobat Reader are available via the CD-ROM or file archive:

- Solaris
- HP-UX™
- Microsoft® Windows™ 3.1 (16-bit)
- Microsoft Windows 95 and Microsoft Windows NT® (32-bit)
- Macintosh®.

### 13.10.2.1 Obtaining the Acrobat Reader from the CD-ROM

To install the Acrobat Reader from the CD-ROM

1. Mount the CD-ROM as per the instructions in the CD-ROM case.
2. Copy the appropriate Acrobat Reader and README files from the directory */cdrom/mynah/acroread*.
3. Install the Acrobat Reader as per the instructions in the README file.

### 13.10.2.2 Obtaining the Acrobat Reader as a File Archive

To obtain the Acrobat Reader as a file archive:

1. Obtain the appropriate Acrobat Reader and README files from the FTP site.
2. Install the Acrobat Reader as per the instructions in the README file.

### 13.10.2.3 Obtaining the Acrobat Reader from Adobe

You can download the Acrobat Reader directly over the Internet from Adobe at *www.adobe.com*. Follow the instructions detailed on the web page.

## 13.10.3 Accessing the PDF Files

Once you have installed the Acrobat Reader, users can read the file

- Using the Acrobat Reader directly from *\$XMYDIR/doc/mynah\_aetg* if they are running the AETG System on the system where it was installed
- Using the Acrobat Reader as plug-in to a browser if the AETG System has not been installed on a local system.

In this case, you would move the *mynah\_aetg* directory to an internal web site. You must install the *mynah\_aetg* directory in such a way that will ensure that it shall not be accessed over a public, non-secured Internet connection or shared with any Third Party, such as through an extranet connection.

If the users access the PDF file via a browser, they may wish to download the file to their local system, which will give them direct access to the file the next time they need to read the file, rather than waiting for the browser to load it.

## 14. Configuring the AETG System

Once you've installed the software, you must configure the AETG System. This section describes the syntax and content of the AETG configuration files.

### 14.1 Introduction

The AETG configuration files are stored in the directory assigned to the variable `$XMYHOME`. This must be different from `$XMYDIR (/opt/SUNWmyn/mynah)`. For example, if multiple configurations are desired for different user communities of the same AETG installation, then multiple `$XMYHOME` locations may be created.

This section covers the two AETG configuration files, the `xmyConfig.General` and `xmyConfig.OP` files, that you can edit to customize an installation.

### 14.2 The xmyConfig.General File

During the AETG install process, an example configuration file was copied from the `$XMYDIR/examples/config` directory to the `$XMYHOME/config` directory.

An example `xmyConfig.General` entry is shown in [Figure 14-1](#).

```
General Default
DefaultSD           = SD1,           # not supported
Database            = yes,           # "no" if Oracle is not used
WelcomeNewUsers     = yes,           # "no" if Oracle is not used
NonOwnerObjectModification = yes,    # true or false
OMPort              = 5000;          #
```

**Figure 14-1.** xmyConfig.General Entry

`xmyConfig.General` file entries take the format

```
General Default
  parameter      = option,
  parameter      = option;
```

and use the following conventions:

- Each parameter listing, except for the last parameter for an entry, is delimited by a comma (,).
- Each entry is delimited by a semicolon (;).

The valid **xmyConfig.General** configuration parameters are

<b>DefaultSD</b>	This configuration parameter is not supported for this release of the AETG System.
------------------	--

<b>Database</b>	<p>Indicates whether or not this configuration of the AETG System makes use of a database. The database is required if users need to use the AETG test management abilities.</p> <p>yes    Use a database</p> <p>no     Do not use a database</p> <p>Default = yes</p>
<b>WelcomeNewUsers</b>	<p>Indicates whether or not the GUI will automatically create a person object for a new user (i.e., a user that does not exist yet in the database).</p> <p>yes    The GUI will create a Person object for the new user.</p> <p>no     The GUI will notify the user that they are not an authorized user of the AETG System, and the GUI will exit.</p> <p>Default = yes</p>
<b>NonOwnerObjectModification</b>	<p>Specifies whether a user has the ability to modify other people's objects in the AETG GUI. If this parameter is set to false, only the owner of an object or an administrator will be able to edit the object.</p> <p><b>Note</b> — All users will still be able to open the object in read-only mode.</p> <p>The ability to open Test Hierarchies for editing purposes is not affected by the setting of this configuration tag; nonowners and nonadministrators can open Test Hierarchies for editing even if <b>NonOwnerObjectModification</b> is set to <i>no</i>.</p> <p>Default = yes</p>
<b>OMPort</b>	<p>Specifies the port number used by the OA processes to communicate with the <b>xmyOM start</b>, <b>stop</b>, and <b>status</b> methods. This must be set to a valid unused port number greater than 5000 and less than 65000.</p> <p>Refer to the <i>/etc/services</i> files to determine what ports are unused.</p> <p>This is a required parameter.</p>

---

## 14.3 The xmyConfigOP File Syntax

Configuration information for the Operability Management processes is contained in the *xmyConfigOP* file. This file is located in *\$XMYHOME/config*.

**NOTE** — Operability Management (OM) gives you a single mechanism to start, stop and obtain status on all of the AETG processes from any host, including those processes not developed by AETG but that are an integral part of the AETG operation (e.g., all of the Telexel processes).

This section describes the entries of the *xmyConfigOP* file, which creates configuration information for the AETG Operability Management structure.

*xmyConfigOP* file entries use the same format as the *xmyConfig* file entries:

```
entry_name LogicalName  
  parameter      = option,  
  parameter      = option;
```

*xmyConfigOP* **entry\_names** can be one of the three following reserved names, each of which is used to create specific operability configurations:

<b>General</b>	Used to create configuration parameters that apply to the entire AETG System.
<b>Process</b>	Used to define processes to be managed.
<b>OperabilityAgent</b>	Used to define the OA for the host.

The *LogicalName* argument is used to assign a unique name to an **entry\_name**.

The following sections detail the **parameters** and their values for each **entry\_name**.

### 14.3.1 General Entry

The *xmyConfigOP* file must contain the same **General** entry as the *xmyConfig.General* file (Section 14.2). This entry is included into the *xmyConfigOP* file by entering the line

```
%INCLUDE xmyConfig.General
```

at the beginning of the *xmyConfigOP* file.

### 14.3.2 Process Entries

There are processes that must be running before you start the AETG System, and each process must have a **Process** entry in the *xmyConfigOP* file. These processes can be Telexel processes or the license server. The syntax of a **Process** entry is shown in [Figure 14-2](#).

```
Process LogicalName
  Mynah      = {yes|no},
  AutoStart  = {yes|no}, # Yes tells the OA to start
                    # process at boot time
  Start      ="start command",
  Stop       ="stop command",
  Status     ="status command";
```

**Figure 14-2.** xmyConfig Process Entry

The **Process** configuration parameters are

- Mynah** Indicates whether the process is a AETG process. For example, there are Telexel processes that are required by AETG, but they themselves are not AETG processes. In this case, this parameter is set to *no*. In addition, you can use the Operability feature to bring up your own processes.
- yes** This is a AETG process
  - no** This is not a AETG process.
- AutoStart** Indicates whether the process is automatically started when the OA is booted.
- yes** Start this process when the OA is booted.
  - no** Do not start this process when the OA is booted.
- Start** Specifies the command that starts the process.
- Stop** Specifies the command that stops the process.
- Status** Specifies the command that returns the status of the process.

The names you enter for the *LogicalName* are used as elements in the **Responsibility** list parameter for the **OperabilityAgent** entry. (See [Section 14.3.4](#)) This list tells the OA what processes it is responsible for.

The **Start**, **Stop**, and **Status** commands for the background processes are standard Telexel commands.

### 14.3.3 License Server Start, Stop, and Status Commands

The License Server **Start**, **Stop**, and **Status** commands are

**Start** = `xmyStartLS`

**Stop** = `xmyStopLS`

**Status** = `xmyStatusLS`

All of the following commands accept the following options:

- h** Returns a brief help message for the command.
- H** Returns a detailed help message for the command.
- R** Returns the current release number of the AETG System.

### 14.3.4 OperabilityAgent

Operability Agents (OAs) are responsible for communicating the start, stop, and status requests to individual processes.

OAs are started at host boot time or by the user using the CLUI commands.

The OA reads the `xmyConfigOP` file to determine what platform and application processes it is responsible for.

The OA relies on the **Start**, **Stop**, and **Status** parameters that are defined for each process that the OA is responsible for.

**NOTE** — Each process entry contains an **Autostart** parameter. If a process's **Autostart** parameter is set to **Yes**, the OA automatically start that process when the OA starts. By default, the delivery configuration entries for all Telexel processes are set to **Yes**.

There is one OA per host and every OA is defined in the `xmyConfigOP` file.

**NOTE** — This *must* be done on each AETG system host, and not just on the AETG server system.

The syntax of an **OperabilityAgent** entry is shown in [Figure 14-3](#).

```
OperabilityAgent OA_Hostname
    Responsibility =(list of names); # managed processes
```

**Figure 14-3.** xmyConfigOP OperabilityAgent Entry Structure

The **OperabilityAgent** configuration parameter is

**Responsibility** Specifies the names of all of the processes for which the OA has responsibility. They must appear in a comma-separated list enclosed in parentheses in the format

Responsibility = (<process1>, <process2>, ..., <processN>)

**NOTE** — The **OperabilityAgent LogicalName** must be the name of the host on which the OA runs.

Figure 14-4 contains an example of an **OperabilityAgent** entry.

```
OperabilityAgent selene
  Responsibilities = (vxDir, vxGatewaySelene, vxLogToFile,
                  vxErrorServer, xmyLS);
```

**Figure 14-4.** Example xmyConfigOP OperabilityAgent Entry

While there is only one OA per host, you must define the OAs for all hosts in the *xmyConfigOP* file. In addition, each process will be able to appear in multiple OAs' responsibility lists. This way you don't have to redefine the processes for each host.

When the OA starts, it determines its name by looking at what host it was started on. The OA then looks for its entry in the *xmyConfigOP* file to see what processes it is responsible for and immediately starts the ones with **Autostart = yes**.

Each **LogicalName** that appears in the **Responsibility** list for an OA must correspond to a defined **Process** entry.

The OA starts these processes *in the order listed* in the **Responsibility** list.

**NOTE** — This is very important because some processes *must* be up and running before other processes can start. These are

1. vxIpcDir
2. The rest of the Telexel processes



### 14.3.5 Example *xmyConfigOP* File

Figure 14-5 contains an example *xmyConfigOP* file for an AETG configuration. The *xmyConfig.General* file in Figure 14-1 has been included into the *xmyConfigOP* file via the

```
%INCLUDE xmyConfig.General
```

statement.

This example defines the following Operability processes:

- |                      |   |
|----------------------|---|
| <b>vxGatewayhost</b> | This defines the Telexel gateway daemon. One occurrence of this process must be running on each host in the AETG System configuration.                      |
| <b>vxIpcDir</b>      | This defines the Telexel directory service. One occurrence of this process must be running. It provides the directory name service for all other processes. |
| <b>vxErrorServer</b> | This defines the Telexel error server.  |
| <b>vxLogToFile</b>   | This defines the Telexel log to file process.   |
| <b>xmyLS</b>         | This defines the Operability configuration for the License Server.  |

```
%INCLUDE xmyConfig.general
#OPERABILITY ENTRIES FOR INDIVIDUAL PROCESSES ON host1

Process vxGatewayselene
    Mynah      =      no,
    Autostart  =      Yes,
    Start      =      "vxIpcUp",
    Stop       =      "vxIpcDown",
    Status     =      "vxIpcMgr";      # AETG provides an xmy shell

Process vxIpcDir
    Mynah      =      no,
    Autostart  =      Yes,
    Start      =      "vxIpcDir",
    Stop       =      "vxIpcDown -d",
    Status     =      "vxIpcMgr";

Process vxErrorServer
    Mynah      =      no,
    Autostart  =      Yes,
    Start      =      "vxErrorServer $TELDIR/lib/errorText\  
                    $TRAXWAYHOME/config/xtw_error_text\  
                    $XMYDIR/lib/xmyErrorText",
    Stop       =      "?",
    Status     =      "xmyLogStatus" ;

Process vxLogToFile
    Mynah      =      no,
    Autostart  =      Yes,
    Start      =      "vxLogToFile $XMYHOME/syslog/adminLog",
    Stop       =      "vxIpcTerm vx",
    Status     =      "vxIpcMgr";

Process xmyLS
    Mynah      =      Yes,
    Autostart  =      Yes,
    Start      =      "xmyStartLS",
    Stop       =      "xmyStopLS",
    Status     =      "xmyStatusLS";

#OPERABILITY ENTRIES FOR AGENTS

OperabilityAgent selene
    host      =      selene,
    Responsibilities=(vxDir, vxGatewayselene, vxLogToFile,  
                    vxErrorServer, xmyLS);
```

Figure 14-5. Example AETG xmyConfigOP File

## 14.4 The .xmyMYNAHrc File

The **xmyRunAetg** process creates and maintains a *.xmyMYNAHrc* file in the user's home directory. This file is used to store desktop and preference information. It is read by the process upon start-up.

## 15. Operability Management — Starting and Stopping AETG Processes

**Operability Management** gives the AETG administrator a single mechanism for starting, stopping, and getting status of the AETG processes from any host, including those processes not developed by AETG but that are an integral part of the AETG operation (e.g., all of the Telexel processes).

This mechanism is composed of an Operability Manager (OM), an Operability Agent (OA), and configuration information. The OM is invoked by typing the **xmyOM** command.

**NOTE** — See [Section 16.2](#) for a discussion of the **xmyOM** command.

### 15.1 Basic Steps

This section briefly lists the basic steps used for configuring the Operability Management files and starting AETG processes. The following sections provide detailed descriptions.

1. Set up the *xmyConfigOP* file. This file contains all of the configuration information for the **platform** processes required by the OM and the OA.
2. Start the OA on each AETG System host. Generally this is done by including the **xmyStartUp** command in a start-up file. The file *S99mynah.eg* is included in the *\$XMYDIR/examples/admin/scripts* directory. It can be updated for the correct path and machine names and then placed in the */etc/rc3.d* directory. (See [Section 15.8](#).)

The OA reads the *xmyConfigOP* file to determine which processes it is responsible for and then starts those processes on that machine that have **Autostart = Yes**, such as Telexel processes.

## 15.2 Overview

For start-up, all processes that are required by the AETG System, must be started.

For each host in the AETG configuration, these processes are composed of

- An AETG OA
- The required Telexel processes, e.g., the Telexel IPC and logger processes

## 15.3 Operability Design

As mentioned earlier, the design is composed of an OM, an OA, and configuration information.

There is only one OA per AETG host machine, per AETG configuration. The OA “manages” all AETG required processes on a host. The OM, via the **xmyOM** subcommands, provides a user interface to the OAs. The basic design is depicted in [Figure 15-1](#).

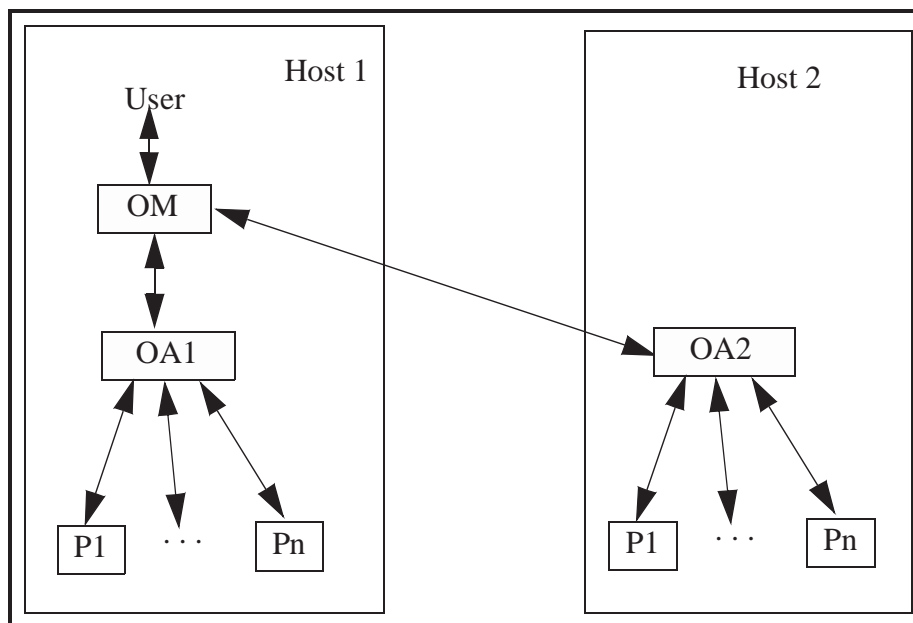


Figure 15-1. Operability Architecture

### 15.3.1 Operability Manager (OM)

The OM provides you with a set of commands to manage all AETG processes. The management is very high level. You can start or stop a process or determine if a process is running properly.

### 15.3.2 Operability Agent (OA)

The OAs are responsible for communicating the start, stop, and status requests to individual processes, and then communicating the reply back to the OM.

OAs are started at host boot time or by the user using the CLUI commands.

**NOTE** — This *must* be done on each AETG system host and not just on the AETG server system.

The OA reads the *xmyConfigOP* file to determine what platform and application processes it is responsible for. The *xmyConfigOP* tells the OA:

- The port number to use to listen on for messages from the OM. (This port number must be greater than 5000 and less than 65000.)
- The name of the individual processes it is to manage.
- All information it needs to know about each individual process that it is to manage.

The OA relies on the **Start**, **Stop**, and **Status** parameters that are defined for each process that the OA is responsible for.

**NOTE** — Each process entry contains an **Autostart** parameter. If a process's **Autostart** parameter is set to **Yes**, the OA automatically start that process when the OA starts. By default, the delivery configuration entries for all Telexel processes.

### 15.3.3 Operability xmyOM Subcommands

The CLUI's **xmyOM** commands takes a series of subcommands that let you manage the AETG processes. [Table 15-1](#) lists the **xmyOM** subcommands and their functions. See [Section 16.2](#) for complete descriptions of each subcommand.

**Table 15-1.** xmyOM Sub-commands

Command	Function
<b>autostart</b>	Causes the OA on the specified host to start up all <b>Autostart</b> processes defined for that OA.
<b>autostop</b>	Causes the OA on the specified host to shut down all the <b>Autostart</b> processes defined for that OA.
<b>query</b>	Provides information on all managed processes.
<b>readconfig</b>	Lets you request that all OA processes re-read the <i>xmyConfigOP</i> file.
<b>recycle</b>	Shuts down and restarts all <b>Autostart</b> processes on the specified host.
<b>shutdown</b>	Terminates an OA.
<b>stop</b>	Stops a process.
<b>start</b>	Starts a process.
<b>status</b>	Sends a status request to a process.

When a subcommand is executed, the OM takes the appropriate action.

- If the request is a query, the OM simply produces the output information.
- If the request is to **start**, **stop**, or obtain the current **status** of a process, the OM forwards the request to the appropriate OA(s) (the OA for the host machine that the targeted process runs on).
- If the request is to reread the *xmyConfigOP* file (i.e., **readconfig** subcommand), the OM sends a request to all OAs defined in the *xmyConfigOP* file, telling each OA to reread the *xmyConfigOP* file.

This request provides a convenient method for you to get *xmyConfigOP* changes to take effect. Without this request in the OM, you would have to bring down all OAs and then restart all OAs to effect a configuration change.

- If the request is to **shutdown**, **autostart**, **autostop**, or **recycle**, the OM forwards the request to the OA indicated on the command-line.

## 15.4 Licensing

This section describes the commands and utilities used to control AETG licensing.

### 15.4.1 AETG Licensing Commands

The AETG System provides the **xmyStartLS**, **xmyStopLS**, and **xmyStatusLS** commands to let you stop and start the License server and obtain information about the status of the License server.

<b>xmyStartLS</b>	Starts the License server. <b>xmyStartLS</b> will look for the file <i>\$XMYHOME/config/xmyLicenses</i> for the licensing keys.
<b>xmyStopLS</b>	Stops the License server.
<b>xmyStatusLS</b>	Displays the key information in <i>xmyLicenses</i> in a readable format.

### 15.4.2 Starting the License Server

After the licensing code is installed, start the license server by typing

```
xmyStartLS
```

while logged in to the machine that will run the server. **xmyStartLS** assumes that the license server resides in *\$XMYDIR/bin* and that the licensing key resides in the directory *\$XMYHOME/config* in a file called *xmyLicenses*.

The license server runs in the background and does not require regular monitoring. If a problem does develop, use **xmyStatusLS** (see [Section 15.4.4.1](#)) to confirm that the server is responding properly. If not, try stopping and restarting the server, after bringing down all licensed AETG System products.

**NOTE** — The License Server can also be started or stopped using the *xmyConfigOP* file.

### 15.4.3 Stopping the License Server

To stop the license server, type

```
xmyStopLS
```

If problems persist after restarting the license server, contact the AETG System support staff for assistance.

---

## 15.4.4 License Utilities

### 15.4.4.1 Monitoring

The **xmyStatusLS**<sup>1</sup> program provides information about the installed licenses. It has the following syntax:

```
xmyStatusLS ?machine_name?
```

where *machine\_name* is the name of the machine on which the server is installed. *machine\_name* is optional if:

- The environment variable LSHOST is set.
- A file called *LicenseServ* in the current directory contains the name of the machine running the server.
- The server runs on the local machine.

**xmyStatusLS** writes the information to the standard output.

**xmyStatusLS** provides information about all software and users that are currently licensed through *xmyLicenses* and all licenses in use, as in the following:

```
xmyStatusLS
lsmonitor for LicenseServ 3.0 Copyright (c) Viman Software

Feature Name: SCRIPTEXEC(v5.0) (floating license) No expiration date.
Concurrent licenses: 65535
Available unreserved   : 65535           In Use: 0
Available reserved    : 0               In Use: 0
Number of subnets    : 0
Site License info     : *.*.*.*
Hostid based locking

Feature Name: XMYTERMASync(v5.0) (floating license) No expiration date.
Concurrent licenses: 65535
Available unreserved   : 65535           In Use: 0
Available reserved    : 0               In Use: 0
Number of subnets    : 0
Site License info     : *.*.*.*
Hostid based locking

Users:
wunn                  (DefaultGrp)
Host name             : ariel
X display name       : ariel:0
Shared ID name       : viman default shared id
Number of keys       : 1
Q wait time          : 10
Hold time            : 0 minute(s)
```

---

1. The **xmyStatusLS** program uses the **lsmonitor** utility from Viman Software.



#### 15.4.4.2 Decoding

A vendor utility called **lsdecode** is shipped with the AETG System in the directory *\$XMYDIR/bin*. It decrypts part of the information in the file containing licensing codes (usually *xmyLicenses*). If you have trouble using AETG System products after installing the licensing code, use **lsdecode** to confirm that the installed license code is correct.

The **lsdecode** utility has the following syntax:

```
lsdecode ?-s license_key_file?
```

where **-s license\_key\_file** is the name of the file containing the license, as in the following example:

```
lsdecode -s $XMYHOME/config/xmyLicenses

          License Decoding Utility
    Copyright (c) Viman Software 1992, 1993

Reading license codes from file: "/opt/SUNWmyn/mynah/config/xmyLicenses"

License code: "YEXSUIHFTTLFKWTTBACXT9K6TIM8PX49BKUATTMS"
Feature Name: "SCRIPTEXEC", Feature Version Number: "5.0"
Exclusive license (will override additive licenses).
Floating license with
  Server host ID: "8E" (2 least significant hex characters)
Maximum concurrent users : 65535
Vendor Info               : ""
Lifetime of keys          : 2 minute(s)
Held licenses disabled.
Shared licenses disabled.
License has no expiration.
```

## 15.5 Starting Processes

As stated earlier, the OAs are started at host boot time. The OA has its `setuid` and `setgid` bits set. (This was done at installation time.) **xmyOA** should be owned by an AETG administrator. This means that when an OA starts, it changes its `uid` and `groupid` to be the same as the `uid` and `groupid` of the `logid` that owns **xmyOA**. All processes that an OA starts will run with the same `uid` and `groupid` as the OA.

**NOTE** — The OA will *not* run as **root**, so if the owner of **xmyOA** is **root**, the OA will not come up. Normally, all processes will be owned by the AETG Administrator, **madmin**.

**NOTE** — Normally, it is recommended that all processes start and stop via the `xmyConfigOP` file and either the `/etc/rc3.d/S99mynah` file (see [Section 15.8](#)).

### 15.5.1 Solaris Start-up Mechanism

At start-up, Solaris looks in the `/etc/init.d` directory for the scripts used to start the system. Solaris then executes, in order, the start and kill process files in the `rc?.d` (where `?` can be one of `S` or `0-6`, e.g., `rcS.d`, `rc0.d`, or `rc4.d`) directories. Each `/etc/rc[S0 - 6].d` directory is designed for a specific function or state. For example, user-defined processes are stored in `/etc/rc3.d`.

The file names for start processes begin with `S`, such as `/etc/rc3.d/S99mynah`. All start files are executed in numeric and alphabetic order, i.e., `S01`, then `S02`, etc.

The file names for termination processes begin with `K`, such as `/etc/rc0.d/K01mynah`. All start files are executed in numeric and alphabetic order, i.e., `K01`, then `K02`, etc.

### 15.5.2 Starting at Boot Time

An AETG start-up file, `S99mynah` ([Section 15.8.1](#) and [Appendix C.1.5](#)) is delivered with the system. You are not required to use this file, but it is strongly recommended since it contains environmental settings (such as library paths, hostports, and the hostname) that you will have to manually set if you do not use this file.

**NOTE** — Remember to edit this file so that settings reflect site-specific values.

If you do not use the `S99mynah` file but still want to start AETG processes at boot time, you must create your own start-up process file in the `/etc/rc3.d` directory on each OA host (each

host that will run platform or application processes), and each file must contain two AETG commands.

#### 15.5.2.1 .xmyRemovePips

The first command is **.xmyRemovePips**.

The **.xmyRemovePips** command removes “local” pip files (pip files of processes that run on the local machine) from the run directories in the configuration. This includes the *\$XMYHOME/run/oa* directory

The **.xmyRemovePips** command takes the following options:

- h** Provides a usage statement
- v** Provides verbose output (shows what's about to be removed)
- oa** Remove pip file in the OA's run directory for the OA on this host
- all** Remove pip files for all processes that run on this host

The **.xmyRemovePips** command is run from the start-up script and should be performed as the AETG Administrator, **madmin**. Therefore, you should add a line such as the following to your start-up file:

```
/bin/su - madmin -c '/opt/SUNWmyn/mynah/bin/.xmyRemovePips all'
```

For example, if the machine is called **selene**, then **.xmyRemovePips** removes the following file (assuming the OA runs on **selene**) *\$XMYHOME/run/oa/pip.oa.selene*.

#### 15.5.2.2 xmyStartUp

To start the OAs at boot time, the start-up file must contain the following command after the **.xmyRemovePips** command:

```
xmyStartUp
```

The OA starts those processes it is responsible for that have **Autostart = yes**. The OA simply executes the indicated **Start** command to start a process.

The OA then sits idle, listening on the OMPort for any incoming **start**, **stop**, **status**, **readconfig**, **shutdown**, **autostop**, **autostart**, or **recycle** requests.

As with **.xmyRemovePips**, **xmyStartUp** is run from the start-up script and should be performed as the AETG Administrator, **madmin**. Therefore, you should add a line such as the following to your start-up file:

```
/bin/su - madmin -c '${XMYDIR}/bin/xmyStartUp'
```

If all other processes are started via the *xmyConfigOP* file, then nothing else needs to be done. Otherwise see Section 15.5.3.

### 15.5.3 Starting a Specific Process

The **start** subcommand (Section 16.2.7) lets you start specific processes. The basic syntax for **xmyOM start** is

```
xmyOM start ?-o oa_name? logical_process_name
```

where

- **-o oa\_name** specifies the name of an OA in the *xmyConfigOP* file.
- **logical\_process\_name** is the name of the process as it appears in the **Responsibility** list of the OA in the *xmyConfigOP* file

If you do not supply an OA name, **xmyOM start** verifies that the **logical\_process\_name** appears in exactly one OA's responsibility list.

For example, the **logical\_process\_name** **xmyLS** in the example *xmyConfigOP* file in Figure 14-4 specifies the start command for the License Server. Therefore, to start the License Server, execute the following:

```
xmyOM start xmyLS
```

The OM process reads the *xmyConfigOP* file, sending a message to an OA, which also knows the contents of the *xmyConfigOP* file. The OA executes the command and sends output back to the OM. The OM prints the output to the user and quits.

### 15.5.4 Starting Autostart Processes

The **autostart** subcommand (Section 16.2.1) lets you have the OA on a specific host start all processes for that OA that have the **Autostart = yes**. The basic syntax for **xmyOM autostart** is

```
xmyOM autostart oa_name
```

where **oa\_name** is the name of the OA whose **Autostart** processes you want to start.

For example, to start all Autostart processes for the OA **selene** (as defined in the example *xmyConfigOP* file in Figure 14-4), you would execute

```
xmyOM autostart selene
```

## 15.6 Stopping Processes

### 15.6.1 Stopping a Specific Process

The **stop** subcommand (Section 16.2.7) lets you stop specific processes. The basic syntax for **xmyOM stop** is

```
xmyOM stop ?-o oa_name? logical_process_name
```

where

- **-o *oa\_name*** specifies the name of an OA in the *xmyConfigOP* file.
- ***logical\_process\_name*** is the name of the process as it appears in the **Responsibility** list of the OA in the *xmyConfigOP* file.

If you do not supply an OA name, **xmyOM start** verifies that the ***logical\_process\_name*** appears in exactly one OA's responsibility list.

For example, the ***logical\_process\_name*** **xmyLS** in the example *xmyConfigOP* file in Figure 14-4 specifies the stop command for the License Server. Therefore, to stop the License Server, you would execute the following:

```
xmyOM stop xmyLS
```

### 15.6.2 Stopping Autostart Processes

The **autostop** subcommand (Section 16.2.2) lets you have the OA on a specific host stop all processes for that OA that have the **Autostart = yes**. The basic syntax for **xmyOM autostop** is

```
xmyOM autostop host_name
```

where ***host\_name*** is the name of the host whose **Autostart** processes you want to stop.

For example, to stop all **Autostart** processes for the OA **selene** (as defined in the example *xmyConfigOP* file in Figure 14-4), execute

```
xmyOM autostop selene
```

**xmyOM autostop** shuts down the processes in the reverse order that they were started up. The OA continues to run after stopping the processes.

### 15.6.3 Stopping an OA and all Autostart Processes on the Local Host

The **xmyShutDown** command lets you gracefully terminate (stop and shut down) an OA on the local host. The basic syntax for **xmyShutDown** is

```
xmyShutDown
```

**xmyShutDown** first stops all **Autostart** processes the local OA is responsible for and then shuts down the local OA itself. This is in contrast to the **autostop** subcommand, which leaves the OA up and running.

### 15.6.4 Stopping an OA and all Autostart Processes on a Specific Host

The **shutdown** subcommand ([Section 16.2.6](#)) lets you gracefully terminate (stop and shut down) an OA on a specific host. The basic syntax for **xmyOM shutdown** is

```
xmyOM shutdown oa_name
```

where *oa\_name* is the name of an OA in the *xmyConfigOP* file.

As with **xmyShutDown**, **xmyOM shutdown** stops an OA and all related processes, but does so for a specified host. For example, to stop the OA **selene**, you would execute

```
xmyOM shutdown selene
```

**xmyOM shutdown** first stops all **Autostart** processes the OA **selene** is responsible for and then shuts down the OA **selene** itself. This is in contrast to the **autostop** subcommand, which leaves the OA up and running.

### 15.6.5 Stopping an OA

In addition to **xmyShutDown** and **xmyOM shutdown**, both of which stops an OA and all of the **Autostart** processes the OA is responsible for, there is also the **xmyStopOA** command, which only stops the OA. All **Autostart** processes continue running.

The basic syntax for **xmyStopOA** is

```
xmyStopOA
```

**xmyStopOA** only takes the *?-hHR?* options listed in [Section 14.3.3](#). In order to stop the OA on a particular machine, you must be on that machine.

### 15.6.6 Stopping and Restarting a Host

The **recycle** subcommand (Section 16.2.5) lets you stop and restart the **Autostart** processes on a specific host. The basic syntax for **xmyOM recycle** is. `xmyOM recycle`  
`host_name`

where *host\_name* is the name of the host whose **Autostart** processes you want to stop and restart.

This subcommand is useful to reinitialize log files that have grown very large.

For example, to stop and restart the **Autostart** processes for the OA **selene**, execute

```
xmyOM recycle selene
```

## 15.7 Obtaining Information about Processes

The AETG CLUI provides several subcommands to the **xmyOM** command for obtaining information on the AETG processes and configuration.

### 15.7.1 Obtaining the Status of Processes

The **status** subcommand (Section 16.2.7) lets you get the status of specific processes. The basic syntax for **xmyOM status** is

```
xmyOM status ?-o oa_name? logical_process_name
```

where

- **-o oa\_name** specifies the name of an OA in the *xmyConfigOP* file.
- **logical\_process\_name** specifies the name of the process as it appears in the **Responsibility** list of the OA in the *xmyConfigOP* file.

If you do not supply an OA name, **xmyOM start** verifies that the *logical\_process\_name* appears in exactly one OA's responsibility list.

For example, the *logical\_process\_name* **vxLogToFile** in the example *xmyConfigOP* file in Figure 14-4 specifies the status command for **vxLogToFile**. Therefore, to obtain the status information **vxLogToFile**, execute

```
xmyOM Status vxLogToFile
```

The result takes the form

IPC Registered Processes

ID	PID	HOST	QUEUE	USER
==	===	====	=====	=====
vxLogDestFile	28517	selene	3001	madmin

Another example would be to obtain the status information for the process **vxGateway** (which appears in Figure 14-4 in the responsibility list for the OA **luna**) by executing

```
xmyOM status -o luna vxGateway
```

The results take the form:

```
vxIpcMgr: vxErSrv00 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000000 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000001 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000002 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000003 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000004 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000005 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000006 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000007 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000008 on luna .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000009 on luna .. selected. no action.
```

## 15.7.2 Displaying Operability Configuration Settings

The **query** subcommand (Section 16.2.3) displays the configuration settings from the *xmyConfigOP* file for OAs and the managed processes. The basic syntax for **xmyOM query** is

```
xmyOM query ?-o oa_name | -p logical_process_name | -s?
```

where

- **-o oa\_name** specifies the name of an OA in the *xmyConfigOP* file.
- **-p logical\_process\_name** specifies the name of a process as it appears in the **Responsibility** list of the OA in the *xmyConfigOP* file.
- **-s** displays a list of all of the OAs defined in the *xmyConfigOP* file.

You can use only one of these options at a time, e.g., you can not use the **-s** option with the **-o oa\_name** option.

For example, to see what OAs are defined in your installation, execute

```
xmyOM query -s
```

which generates output of the form:

```
selene  
luna
```



Then, if you wanted to see what processes the OA **selene** is responsible for, execute

```
xmyOM query -o selene
```

which generates output of the form:

```
The OA on selene is responsible for the following processes:  
vxIpCDir  
vxGateway  
vxIpCclean  
vxLogToFile  
vxErrorServer
```

If you do not specify any options, **xmyOM query** lists the defined OAs in your configuration and the processes that appear in the **Responsibility** lists for those OAs in the *xmyConfigOP* file. This is equivalent to executing **xmyOM query -s** followed by **xmyOM query -o oa\_name**, except that instead of seeing the defined processes for one OA only, you would see the defined processes for all OAs in the system.

For example, if you type

```
xmyOM query
```

and you have defined the OAs **selene** and **luna**, you would see output of the form:

```
-----  
The OA on selene is responsible for the following processes:  
vxGateway  
-----  
The OA on luna is responsible for the following processes:  
vxIpCDir  
vxGateway  
vxIpCclean  
vxLogToFile  
xmyCollector  
vxErrorServer  
-----
```

To see settings for the process **vxIpCDir**, execute

```
xmyOM query -p vxIpCDir
```

which generates output of the form:

```
Process      : vxIpCDir  
StartCommand : /opt/SUNWmyn/mynah/bin/.xmyStartup  
StopCommand  : vxIpCDown -d  
StatusCommand: vxIpCProcesses  
Autostart    : Yes
```

## 15.8 Starting and Stopping AETG Software Packages

After installation, the `$XMYDIR/examples/admin/scripts` directory contains a start-up script for the various AETG software packages, e.g., `S99mynah.eg`.

To start these packages when the system is booted, copy this file from the `$XMYDIR/examples/admin/scripts` directory to the `/etc/rc3.d` directory.

If you want the software packages to terminate correctly when the system is rebooted, you can create logical links from the files in the `/etc/rc3.d` directory to the `/etc/rc0.d` directory, preappending the link with the kill prefix, `K`.

**NOTE** — The owner of the start files (and the terminating links) must be **root**, and you must set the permissions on these files (and links) to 755.

### 15.8.1 Automatically Starting the AETG System

To automatically start-up the AETG and Telexel processes at boot time, perform the following tasks:

1. Copy the example start-up file `S99mynah.eg` from the `$XMYDIR/examples/admin/scripts` directory to the `/etc/rc3.d` directory and rename the copy `S99mynah`.

**NOTE** — See Appendix C.1.5 for a copy of `S99mynah.eg`.

2. Edit `S99mynah` for library paths, hostport, hostname, etc.
3. Create a logical link to `S99mynah` in `/etc/rc0.d` to terminate the AETG processes correctly at reboot time, e.g.,

```
cd /etc/rc0.d
ln -s /etc/rc3.d/S99mynah K01mynah
```

### 15.8.2 Automatically Starting Oracle

To automatically start-up the Oracle processes on a client machine at boot time, perform the following tasks:

1. Copy the example start-up file `S96oracle.eg` from the `$XMYDIR/examples/admin/scripts` directory to the `/etc/rc3.d` directory and rename the copy `S96oracle`.

**NOTE** — See Appendix C.3.1 for a copy of *S96oracle.eg*.

2. Edit the *S96oracle* file to correct directories and paths, etc.
3. Create a logical link to *S96oracle* in */etc/rc0.d* to terminate the Oracle processes correctly at reboot time, e.g.,

```
cd /etc/rc0.d  
ln -s /etc/rc3.d/S96oracle K02oracle
```

4. It is recommended that you change the **shutdown** command in */opt/SUNWora/oracle/bin/dbshut* from **shutdown** to **shutdown immediate**.

## 15.9 User Defined Processes

If users want to add their own processes to the *xmyConfigOP* file, their start, stop, and status commands must follow these rules:

- They must exit with a 0 return code upon success.
- They must exit with a non-zero on failure. Any output to standard out or standard error will be included in the reply to the OM from the OA that ran this command.
- These commands must exit after performing their task. That is, when starting up a process, they must exit when that process is “up” and running by whatever definition makes sense for that process. They can *not* stay around or the OA will hang, waiting for a response from the command in terms of its exit.

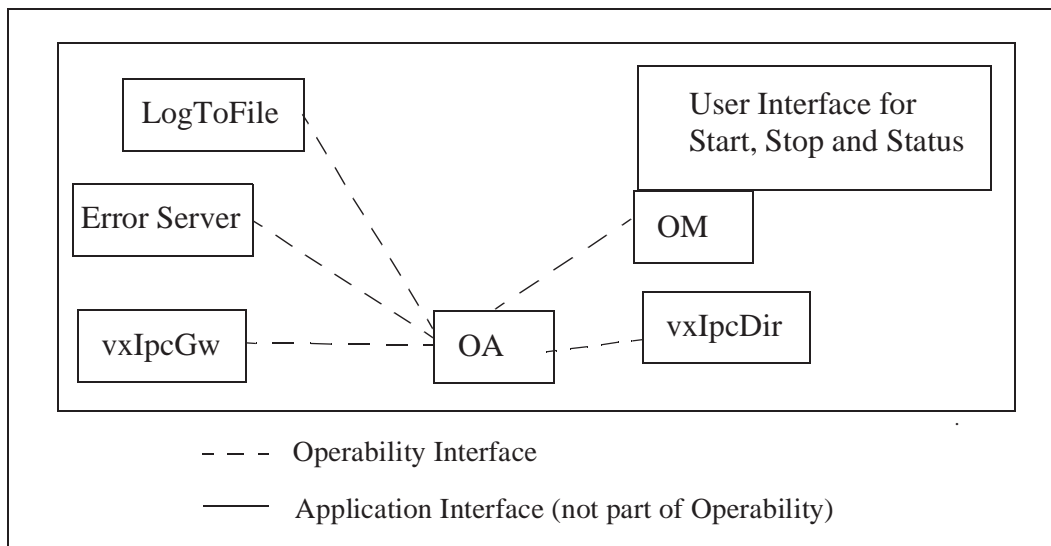
## 15.10 Operability Summary

An administrative user needs to be able to start, stop, or determine the status of any AETG application process, and any platform process that a AETG application process relies on. The operability design allows a user, from a single user interface and a single host machine, to perform this function.

The flexibility of the configuration portion of the operability design allows the AETG Administrator to add new types of managed processes as the need arises.

Only one process for each host needs to be explicitly started at boot time, the OA process. The OA process will start everything else that should be started at boot time.

The Operability feature is summarized in Figure 15-2.



**Figure 15-2.** Operability Feature Summary

## 16. Administrative CLUI Commands

This section describes the Administrative Command Line User Interface (CLUI) commands. Administrative CLUI commands are only available to the AETG user who has administrative privileges, e.g., *madmin*. All other CLUI commands are available to all AETG users.

### 16.1 CLUI Command Help Messages

All CLUI commands and subcommands have an **-h** option that displays a brief help message listing the usage and syntax of the subcommand. For example

```
xmyOM query -h
query: display info about the AETG configuration
Usage: xmyOM query [-h] [-p<process_name> | -o<oa_name> | -s ]
      -h                : display this message
      -p<process_name> : process name as in config file
      -o<oa_name>      : Operability Agent name as in config file
      -s                : List all systems (OAs) defined in xmyConfigOP
```

In addition, each main command has an **-h** option to tell you what subcommands exist for the command, for example

```
xmyOM -h
Usage:
      xmyOM command_name command_args
The command_args depend on which command_name you use.
To obtain usage information on an individual command_name,
simply type:
      xmyOM command_name -h
Here is a list of all the valid xmyOM command_names and a brief
description of what they do:
start      - start a platform/application process
stop       - stop a platform/application process
status     - display status of platform/application process
query      - display AETG configuration
readconfig - request OA to re-read the configuration file
autostop   - request OA to bring down all autostart processes
autostart  - request OA to start up all autostart processes
recycle    - request OA to re-cycle (bring down then up)autostart proc's
shutdown   - stop OA (brings down all autostart processes too)
```

## 16.2 xmyOM

The **xmyOM** command subcommands can be used to control and administer the platform and application processes that constitute the AETG System. These subcommands are implemented as part of the Operability Manager (OM), which is discussed in [Section 15](#).

A user, at any point in time, can invoke the **xmyOM** command from any machine in the network.

### 16.2.1 autostart

#### Syntax

```
xmyOM autostart host_name
```

#### Description

The **autostart** subcommand causes the OA on the specified host (*host\_name*) to start up all autostart processes defined for that OA, i.e., all processes in that OA's responsibility list whose **Autostart** entry is set to *yes*.

#### Example

```
xmyOM autostart selene
```

## 16.2.2 autostop

### Syntax

```
xmyOM autostop host_name
```

### Description

The **autostop** subcommand causes the OA on the specified host (*host\_name*) to shut down all the autostart processes defined for that OA. It shuts them down in the reverse order from which they were started up, however, the OA continues to run after shutting them down. This distinguishes the **autostop** subcommand from the **shutdown** subcommand in that with the **shutdown** subcommand, not only does the OA shut down all its autostart processes, but it also shuts itself down.

### Example

```
xmyOM autostop selene
```

### 16.2.3 query

#### Syntax

```
xmyOM query ?-o oa_name | -p logical_process_name? | -s?
```

#### Description

The **query** subcommand displays configuration file entries (from the *xmyConfigOP* file) for OAs and the managed processes.

**query** takes the following options:

- |                                |  |
|--------------------------------|--|
| <b>-o oa_name</b>              | Displays the processes the entered Operability Agent (OA) is responsible for. (See Section 15.3.2 for information on OAs.) |
| <b>-p logical_process_name</b> | Displays information for the entered process as defined in the <i>xmyConfigOP</i> file.                                    |
| <b>-s</b>                      | Displays a list of all OAs (i.e., all hosts) defined in the <i>xmyConfigOP</i> file.                                       |

If no option is used, then all OA entries in the configuration file are displayed.

#### Example

```
xmyOM query -o mimicr
```

The OA on mimicr is responsible for the following processes:

vxGateway

vxIpcClean



## 16.2.4 readconfig

### Syntax

```
xmyOM readconfig
```

### Description

The **readconfig** subcommand instructs the OAs on all hosts to read the configuration file. **readconfig** should be used only if some changes have been made to the *xmyConfigOP* file. The OM broadcasts **readconfig** to all the OAs in the AETG configuration.

### Example

```
xmyOM readconfig  
OA(selene): config file read  
OA(mimir): config file read
```

## 16.2.5 recycle

### Syntax

```
xmyOM recycle host_name
```

### Description

The **recycle** subcommand shuts down and restarts all Autostart processes on the specified host (*host\_name*). This is identical to an **xmyOM autostop *host\_name*** followed by an **xmyOM autostart *host\_name***.

### Example

```
xmyOM recycle selene
```

## 16.2.6 shutdown

### Syntax

```
xmyOM shutdown oa_name
```

### Description

The **shutdown** subcommand gracefully terminates an OA. The OA first stops all autostart processes it is responsible for, terminating them in the reverse order from which they were brought up. Lastly, the OA shuts itself down.

If you want to bring only the OA and not the Autostart processes down, you must be logged into the OA's machine and then use the **xmyStopOA** command. This will only work if you are **root** or the person who started the OA, usually **madmin**.

*oa\_name* is the name of the OA as it appears in the *xmyConfigOP* file.

## 16.2.7 start/stop/status

### Syntax

```
xmyOM start ?-o oa_name? logical_process_name  
xmyOM stop ?-o oa_name? logical_process_name  
xmyOM status ?-o oa_name? logical_process_name
```

### Description

These subcommands can be used to start, stop, or get the status of the application or platform processes. The application or platform processes on which these subcommands operate must be present in the responsibility list of at least one OA in the *xmyConfigOP* file.

*logical\_process\_name* is the name of the process as it appears in the responsibility list of the OA in the *xmyConfigOP* file.

If **-o oa\_name** is not supplied, **xmyOM** verifies that the *logical\_process\_name* appears in exactly one OA's responsibility list and forwards the request to the OA. If **-o oa\_name** is supplied, **xmyOM** makes sure that the *logical\_process\_name* appears in that OA's responsibility list and forwards the request to that OA.

### Example

```
xmyOM status -o cardinals vxGateway  
vxIpcMgr: vxErSrv00          on selene    .. selected. no action.  
vxIpcMgr: vxErrMsgClient000000 on selene    .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000001 on selene    .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000002 on luna      .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000003 on cardinals .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000004 on cardinals .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000005 on cardinals .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000006 on luna      .. selected. no action.  
vxIpcMgr: vxErrMsgClient0000007 on yankees   .. selected. no action.  
vxIpcMgr: vxErrorServer       on selene    .. selected. no action.  
vxIpcMgr: vxLogDestFile        on selene    .. selected. no action.  
vxIpcMgr: xmyBDcardinals       on cardinals .. selected. no action.  
vxIpcMgr: xmyBDluna            on luna      .. selected. no action.  
vxIpcMgr: xmyBDselene          on selene    .. selected. no action.  
vxIpcMgr: xmySDSD1             on selene    .. selected. no action.  
vxIpcMgr: xmySDa               on cardinals .. selected. no action.  
vxIpcMgr: xmySE0000SD1         on luna      .. selected. no action.  
vxIpcMgr: xmySE0000a           on cardinals .. selected. no action.
```

## 17. Person Object

The Person object is a AETG GUI object that lets an AETG Administrator control the AETG database settings for the AETG users. These functions include creating Person objects and assigning **Inactive** status to a user.

A Person object must exist in the database for each AETG user. A Person object is created in one of two ways:

- If you set the **Welcome New Users** parameter in the *xmyConfig.General* file to **yes** (see [Section 14.2](#)), the GUI automatically creates a Person object when a new user starts the GUI. In fact, if the **Welcome New Users** parameter is set to **no**, the GUI exits.

Therefore, setting the **Welcome New Users** parameter to **no** lets you prevent nonauthorized users from using the AETG GUI.

- You can create a Person object in the GUI before the new user starts the GUI. This way, if you set the **Welcome New Users** parameter to **no** (e.g., to limit the number of new users) you can create a Person object for a specific user before she/he starts the GUI.

**NOTE** — There can be only one Person object in the database per UNIX ID.

AETG users can edit their general properties attributes (i.e., their name, phone number, and e-mail address) and the keywords associated with their Person object only. As a AETG Administrator, however, you can change these attributes for other users. In addition, you can edit the authority attribute for other users.

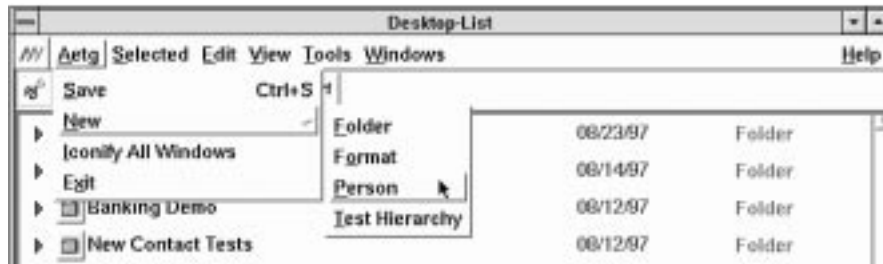
**NOTE** — While users can create untitled Person objects on the AETG Desktop, they cannot edit or save these Person objects. AETG users can, however, access the Person objects for all AETG users and edit their own Person object from the Database Browser.

## 17.1 Creating a Person Object

In the AETG Desktop, execute

**AETG->New ->Person**

as shown in [Figure 17-1](#).



**Figure 17-1.** Creating a New Person Object

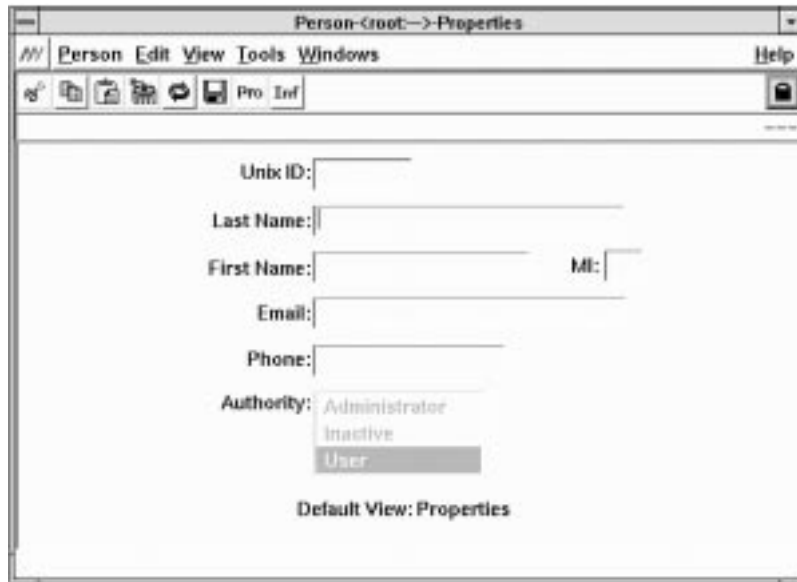
### 17.1.1 Editing Properties Attributes

Once you've created a Person object, you can enter user information.

1. Open the new Person object by performing either of the following:
  - Double clicking on the Person object icon
  - Selecting the icon and then executing

**Selected->Open**

An empty Person object opens. By default, the Person object opens in the Properties View ([Figure 17-2](#)).



**Figure 17-2.** New Person Object Properties View

2. Unlock the Person object.

**NOTE** — An AETG user can only unlock his or her own Person object. An AETG Administrator can unlock (and edit) any AETG user's Person object.

3. Enter the user's UNIX ID, name, e-mail address, and phone number. As an AETG Administrator, you can also set the user's authority level by selecting one of the three following options in the **Authority** list:

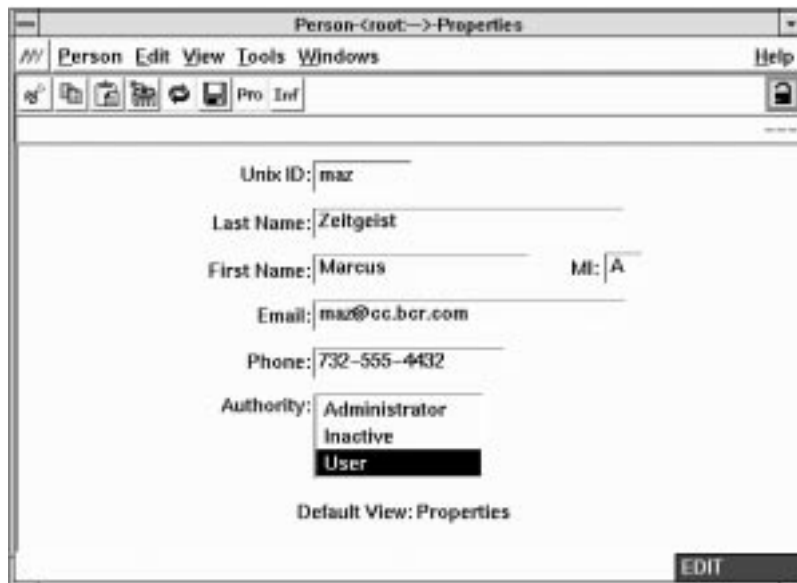
**Administrator** This gives administrative privileges to a user. An administrative user can grant other users Administrator Authority. In addition, an Administrator can edit the attributes of other users.

**Inactive** This gives a user inactive status, meaning that the user can no longer access the AETG System. The Person object remains in the database. If you want to reactivate a user, use the **Database Browser** to grant the user either Administrator or User Authority.

**NOTE** — You cannot cut a Person object from the database. Giving a Person object inactive status is the only way of removing AETG privileges for a user, e.g., if a user leaves.

**User** This gives the user the standard User Authority, such as when you are removing administrative privileges. Someone with User Authority can edit her/his attributes, but cannot change their Authority. In addition, someone with User Authority can view but cannot edit the Person objects for other users.

Figure 17-3 shows an example of an edited Person object Properties View.



**Figure 17-3.** Edited Person Object Properties View

The Person object is not added to the database until you save it. Execute

**Person->Save**

The window exits Edit Mode. The **Authority** field is grayed out.

At a minimum, you must enter the User ID for a new Person object. You can then

- Fill in the rest of the fields on the Properties view.
- Save the object.
- Close the window by executing

**Person->Close**

(The system asks you if you want to save your changes if you have not already done so.)

- Change to the Information View to edit Keyword associations for the Person object.

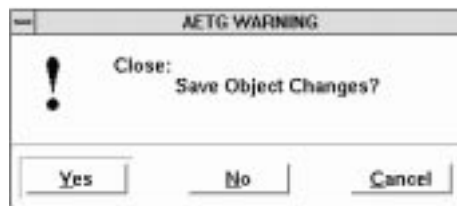


There can be only one Person object in the database per UNIX ID. If you try adding a Person object with a UNIX ID that already has a Person object, the error dialog box in Figure 17-4 appears.



**Figure 17-4.** Duplicate UNIX ID Error Box

If you've edited a Person object and then try closing it without first saving it, the dialog box in Figure 17-5 appears.



**Figure 17-5.** Close Dialog Box

- If you click on **Yes**, your changes are saved and the Person object window closes.
- If you click on **No**, your changes are discarded and the Person object window closes.
- If you click on **Cancel**, the Person object window remains open.

### 17.1.2 Editing Information Attributes

The Person object Information view (Figure 17-6) lets you enter comments regarding the Person object.

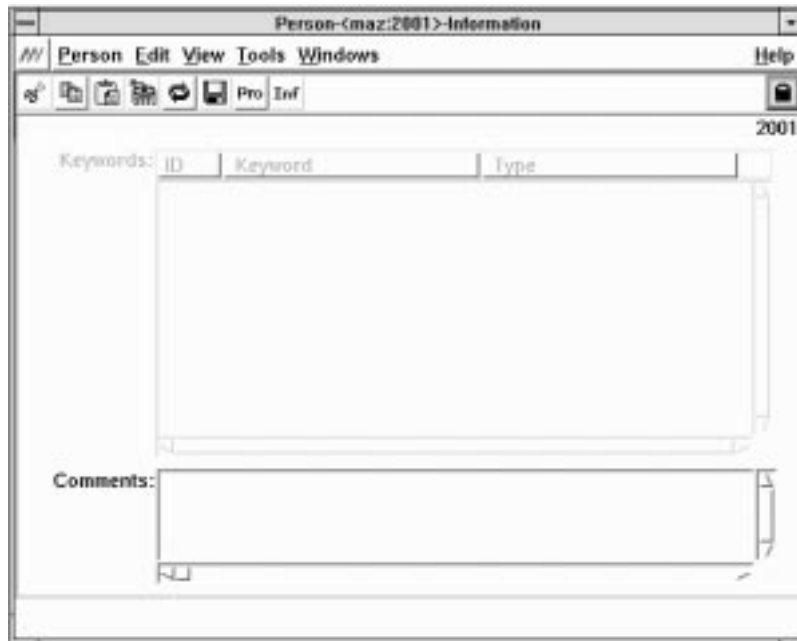


Figure 17-6. Person Object Information View

**NOTE** — The Keywords parameter is not supported by the AETG System.

## 17.2 Editing an Existing Person Object

As a AETG Administrator, you can edit Person objects for other users. Your main reason for editing a user's Person object is to change their Authority, e.g., to grant or remove administrative privileges.

**WARNING** — You can remove administrative privileges for *all* users, including yourself.

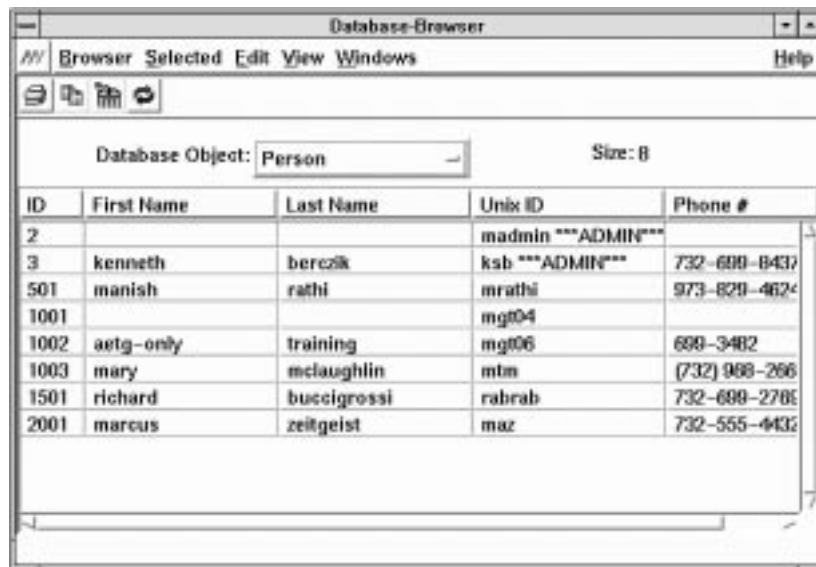
While you can create Person objects in your AETG window, all Person objects for a AETG installation are listed in the Database Browser. In fact, if the **Welcome New Users** parameter in the *xmyConfig.General* file is set to **yes**, the resulting Person objects can be found *only* in the Database Browser. Therefore, you will most often want to edit a user's Person object from the Database Browser.

In our example, if you wish to apply inactive status to a user who was added automatically, follow these steps.

1. Execute

**Tools->Database Browser**

If the list of Person objects is not visible when the Browser appears, select **Person** from the Database object menu to access the list of Person objects (Figure 17-7).



The screenshot shows a window titled "Database-Browser" with a menu bar (Browser, Selected, Edit, View, Windows, Help) and a toolbar. Below the toolbar, there is a "Database Object:" dropdown menu set to "Person" and a "Size: 8" label. The main area contains a table with the following data:

ID	First Name	Last Name	Unix ID	Phone #
2			admin ***ADMIN***	
3	kenneth	berczik	ksb ***ADMIN***	732-688-8437
501	manish	rathi	mrathi	973-829-4624
1001			mg04	
1002	aetg-only	training	mg06	688-3482
1003	mary	mclaughlin	mtm	(732) 988-266
1501	richard	buccigrossi	rabrab	732-688-2768
2001	marcus	zeitgeist	maz	732-555-4436

**Figure 17-7.** Database Browser - List of Person Objects

2. Open the Person object you want to edit. You can do this by either
  - Clicking on the row listing the object and executing **Selected->Open**
  - Double clicking on the row listing the object.

- When the Person objects opens, unlock the window, and select the **Inactive** option from the **Authority** list (Figure 17-8).

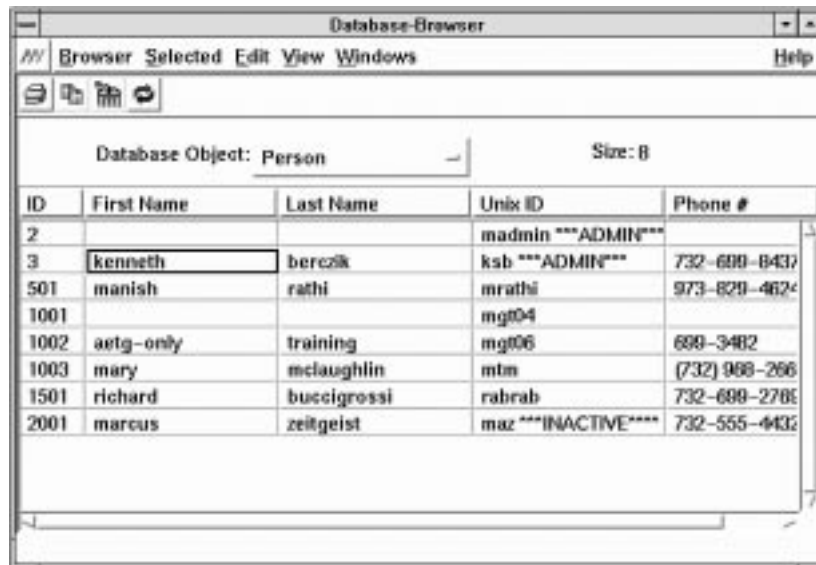


**Figure 17-8.** Edited Person Object — Granting Administrative Privileges

- Save and close the object.
- If you look at the Database Browser, you will not see any change in the Authority for the Person object. To view the change, execute

**View->Refresh**

The Database Browser is redisplayed, and you will see that the user is now listed as Inactive (See Figure 17-9).



The screenshot shows a window titled "Database-Browser" with a menu bar containing "Browser", "Selected", "Edit", "View", "Windows", and "Help". Below the menu bar are several icons. The main area of the window displays "Database Object: Person" and "Size: 8". A table with the following data is shown:

ID	First Name	Last Name	Unix ID	Phone #
2			admin ***ADMIN***	
3	kenneth	berczik	ksb ***ADMIN***	732-688-8437
501	manish	rathi	mrathi	973-828-4624
1001			mg04	
1002	aetg-only	training	mg06	688-3482
1003	mary	mclaughlin	mtm	(732) 968-266
1501	richard	buccigrossi	rabrab	732-688-2785
2001	marcus	zeitgeist	maz ***INACTIVE***	732-555-4433

Figure 17-9. Refreshed Database Browser



## Appendix A: Using the AETG System with the MYNAH System

The AETG System can be licensed with the MYNAH System, letting you generate script code that executes your test cases.

In this configuration, the AETG System is full integrated into the MYNAH System, and you see several changes to the GUI. This Appendix describes the changes you encounter.

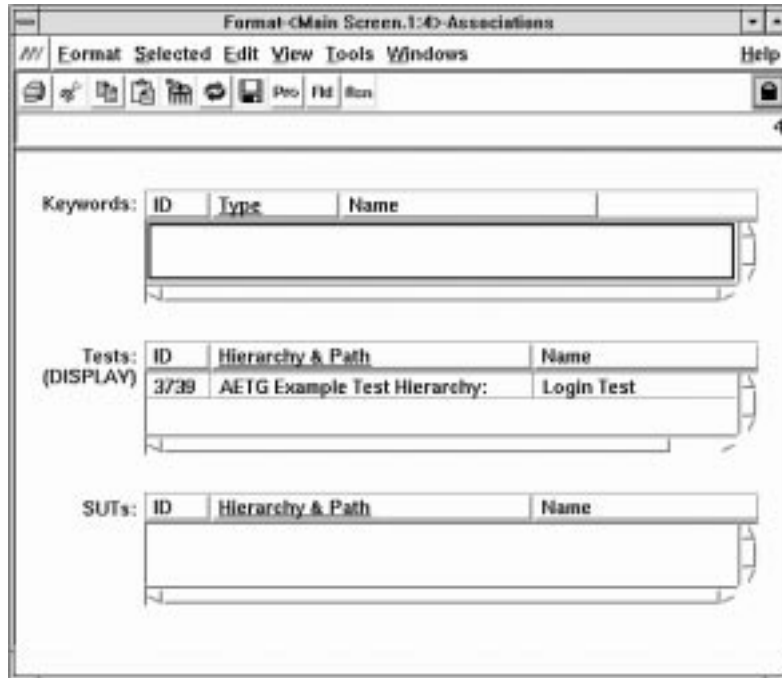
### A.1 Related Documentation

The following documents are available, detailing the use of the MYNAH System:

- *BR 007-252-005, MYNAH System Administration Guide*
- *BR 007-252-006, MYNAH System Users Guide*
- *BR 007-252-008, MYNAH System Scripting Guide*

## A.2 Changes to Format Objects

The Keywords and SUTs parameters on the Format object Associations view (Figure A-1) are enabled, letting you create associations to these object.



**Figure A-1.** Format Object Associations View

**NOTE** — See the *MYNAH System Users Guide* for information Keywords and SUTs and on how to associate an object with another object.

Format objects can be associated with SUT objects to indicate that the defined Format is valid for that particular version of the System Under Test.

More than one Format can be associated with one SUT. You can associate all of the Formats that are valid with a particular SUT.

One Format can be associated with more than one SUT. You may want to do this when a Format does not change from one SUT to the next. In that case, you may associate the old Format with the new SUT.

If a Format changes however (e.g., a new Field is added), you would create a new version of the Format by executing

**Selected ->Duplicate**

and associating the new Format with the appropriate SUT.



### A.3 Changes to SUT Objects

The following changes occur when you work with SUT objects.

The Formats parameter on the SUT object Associations view (Figure A-2) is enable, displaying what Formats are associated with this SUT.

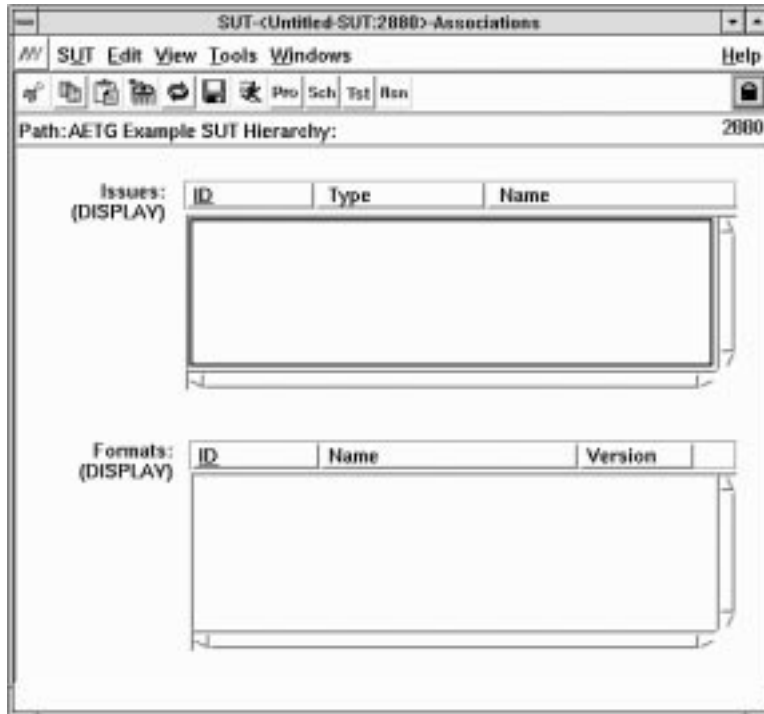


Figure A-2. SUT Object Associations View

## A.4 Changes to Keyword Objects

The Formats parameter on the Keyword object Associations view is enable, displaying what Formats are associated with this Keyword (Figure A-3).

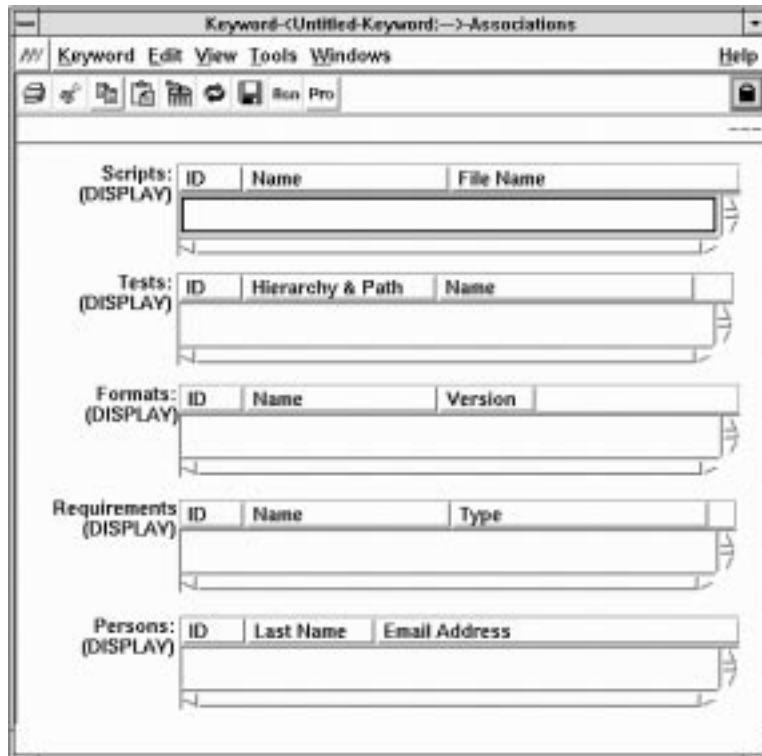


Figure A-3. Keyword Object Association View

## Appendix B: Generate Tcl Scripts for AETG Testcases

If you license the AETG System with the MYNAH System, you can automate test cases by directly reading the Test Case Matrix file from a Tcl script using standard TclX keyed list commands.

**NOTE** — See the *MYNAH System Scripting Guide* for information on Tcl and TclX keyed lists.

However, you may want to have the MYNAH System generate the automation code. This section describes the MYNAH script generation process.

### B.1 Test Object Script Generation View

The Test object Script Generation view ([Figure B-1](#)) lets you specify the parameters the MYNAH System uses to generate a script that automates the test cases in the Test Matrix view ([Section 9.3.3](#)).



Figure B-1. Test Object Script Generation View

The Script Generation view contains the following parameters:

**Tcl Procedure Names** Lets you enter the names of Tcl procedures that will use the test case matrix to create your testing script.

**Note** — You must provide a Body procedure. The Setup and Cleanup procedures are optional.

**Setup** The name of the user provided procedure that will bring the SUT to the appropriate state to accept input. In addition, this procedure should perform any other necessary setup operations.

One setup operation the **Setup** procedure must perform is setting the actual values for nonliteral values. The **Setup** procedure will always be passed the name of an array that must contain the actual values for the non-literal values. For example a **Setup** procedure might contain the line

```
set arr(daynum) 555-1212
```

where *daynum* is a nonliteral value and *555-1212* is the actual value

**Body** The name of the user provided procedure that constitutes the body of the Tcl script. This procedure performs the following functions when given an input list:

- Builds the input
- Sends the input to the SUT
- Captures the value in the Validation Field
- Reinitializes to the input state
- Returns the captured validation value.

**Cleanup** Name of the user written procedure that performs all cleanup operations, for example, logging off of the SUT.

**Validation** Lets you enter strings and Field names that will be used to determine if a test case is valid or invalid.

**Valid Field** The SUT field name that contains the information that will indicate the success or failure of a valid test case.

<b>Invalids Field</b>	<p>The SUT field name that contains the information that will indicate the success or failure of an invalid test case.</p> <p><b>Note</b> — This may be the same as the Valids Field entry but it also may be different.</p>
<b>Valids String</b>	<p>The value that the validation field should contain if the valid test case is successful. This string is passed to the Body procedure. This string is also used in the automatically generated <b>compare</b> statement.</p>
<b>Invalids String</b>	<p>The value that the validation field should contain if the invalid test case is successful. This string is passed to the Body procedure. This string is also used in the automatically generated <b>compare</b> statement.</p>
<b>Pass Parameters</b>	<p>Lets you specify if the value(s) of the Field object's <b>Parameters</b> attribute (<a href="#">Section 7.2.1.2</a>) should be included in the Field data in the generated script.</p>
<b>Code Compares</b>	<p>Lets you specify if the generated code should contain an <b>xmyCompare</b> statement for each test case.</p> <p>If the <b>xmyCompare</b> statements are generated, it is assumed that the Body procedure sets a variable called <b>result</b>. This variable is compared to the Validation String. The result of the <b>compare</b> will be recorded in the Result object for the Test and will be “labeled” with the test case name.</p>

When you execute

**Test->Script Generate**

the MYNAH System creates Tcl code that

1. Calls the Setup procedure once
2. Calls the Body procedures for each test case
3. Calls the Cleanup procedure.

The code also contains **xmyBegin** and **xmyEnd** statements for the Test object so that labeled compare statements can be used for the validation steps. If you enable **Code Compares**, the MYNAH System generates a labeled compare for each test case in the Test Matrix, using the TC Name as the label.

All valid test cases are generated first, then the invalid test cases are generated.

## B.2 Example

Assume you are generating a test case with the following:

- You have a Test Matrix with the following two test cases:

<u>Testcase</u>	<u>Type</u>	<u>eveAC</u>	<u>eveNum</u>	<u>dayAC</u>	<u>dayNum</u>
V-0	V	908	evenum	917	daynum
I-0	I	201	evenum	609	daynum

- The following procedures are in a procedure library:

```
proc phone_setup {upArr} {  
    upvar $upArr arr  
    set arr(daynum) 555-1212  
    set arr(evenum) 555-1212  
    logon  
}
```

```
proc phone_body {upList} {  
    upvar $upList localList  
    # build, enter, and get result string  
    keylset localList result "[join [$conn1 screen \  
        -label "Info" -dimension {3 1}]]"  
}
```

```
proc cleanup { } {  
    logoff  
}
```

- Disable the **Pass Parameters** parameter.
- Enable the **Code Compares** parameter.

The Script Generation view will appear as in [Figure B-2](#).



**Figure B-2.** Edited Script Generation view

When you execute

**Test->Script Generate**

the code in [Figure B-3](#) will be created.

<pre># # Call user-defined Setup procedure # setup data # # Test begins # xmyBegin -testid 2389 # # Testcase: V-0 # keylset list tcName V-0 tcType V expectField info keylset list expectString "transaction processed" keylset list fields.dayAC.value 908 fields.dayNum.value \$data(daynum) keylset list fields.eveAC.value 917 fields.eveNum.value \$data(evenum) body list set res "" keylget list result res xmyCompare -expr {\$res == [keylget list expectString]} -label V-0 # # Testcase: I-0 # keylset list tcName I-0 tcType I expectField info keylset list expectString "transaction error" keylset list fields.dayAC.value 201 fields.dayNum.value \$data(daynum) keylset list fields.eveAC.value 609 fields.eveNum.value \$data(evenum) body list set res "" keylget list result res xmyCompare -expr {\$res == [keylget list expectString]} -label I-0 # # Test complete # xmyEnd -testid 2389 # # Call user-defined Cleanup procedure # cleanup</pre>	<ul style="list-style-type: none"><li>- The Setup Procedure is passed an array called <i>data</i> as an argument.</li><li>- The overall Test begins</li><li>- The testcase name is printed in the comment</li><li>- A list is constructed that contains the info for the first testcase</li><li>- The Body procedure is called with the list as an argument</li><li>- The result is analyzed</li><li>- Everything is repeated for the second testcase</li><li>- The overall test ends</li><li>- The Cleanup procedure is called</li></ul>
---	---

**Figure B-3.** Sample Generated Script Code

After running this script, there will be one Result object for the Test with testid of 1. The Result object will contain the results of the two labeled compares, one for **V-0** and one for **I-0**.



## B.2.1 Effects of Code Compares

If you change the **Code Compares** parameter to *No*, the resulting script code would not contain the lines

```
set res ""
keylget list result res
xmyCompare -expr {$res == [keylget list expectString]}
           -label V-0
```

for any of the test cases. In this case, it is assumed that the **Body** procedure will perform the validation.

## B.2.2 Effects of Pass Parameters

The **Parameters** parameter on the Field object Values view ([Figure 7-1](#)) lets you can specify a list of parameters that provide more information about the Field ont. For example, you might enter *Row=1, Col=1* indicating that this Field is in the upper left hand corner of a screen.

The **Parameters** parameter is free form text so it may be used in any way you desire.

You can specify on the Test object GenScript view whether **Parameters** attribute information is passed to an automation script along with the Field name and Value.

If you set the **Pass Parameters** parameter to *Yes*, **Parameters** parameter information is passed to the automation script along with the Field name and Value. The generated code includes the Field Parameter information for each involved field. For example,

```
keylset list fields.dayAC.value 908 fields.dayAC.parms "0,0"
keylset list fields.eveAC.value 917 fields.eveAC.parms "0,1"
```

where the *fields.<fieldname>.parms* entry (e.g., *fields.dayAC.parms*) contains the value of the Parameters attribute of the specific Field object. In the example code, the numbers might represent offset values for where the Body procedure is to type into a field on a screen.

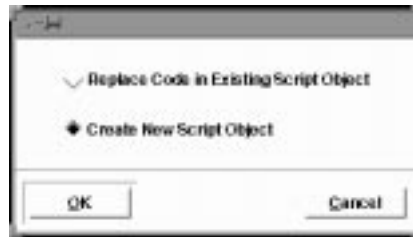
The Body procedure would be written to make use of the parameter information.

## B.3 Generate Script Processing

When you execute

### Test->Script Generate

the Script Generate Confirm Dialog ([Figure B-4](#)) appears so that the you can indicate where you would like to place the code.



**Figure B-4.** Script Generate Confirm Dialog

If a Script object is already associated with the Test, you can choose to overwrite the Script object's code, or create a new Script object.

When you click on **Ok**, the chosen Script object is opened and the generated code is placed in the Code view. You can save the Script object or close the Script object without saving changes.

**NOTE** — If this is a new Script object, you must provide a Name and Filename prior to saving the Script object.

## B.4 Using Script Generation

Script generation can be performed for the entire Test Matrix. However, it can also be performed on any subset of the test cases in the Test Matrix. This is done by first highlighting the desired test cases and executing

### Test->Generate Script

There are several potential circumstances under which it would be desirable to generate a script for a subset of the test cases.

#### B.4.1 Automation Considerations

There may be a large number of test cases in the Test Matrix and it would be inefficient to generate a single script that automates all of them. In this case, a subset could be selected and new script code could be generated. The new Script object could be associated with a different Test object that indicates the general category, if any, of the selected sub-set.

## B.4.2 Debugging Procedure Code

You may want to test the **Setup**, **Body**, and **Cleanup** procedures and the generated script code by generating one or two test cases. Again, you can highlighting one or two test cases and execute

### Test->Generate Script

The resulting script code could be loaded into the Script Builder and run.

Once satisfied that the procedures and generated script code work well together, you would generate code for all of the test cases in the matrix.

## B.4.3 Analyzing Failures

Generated script code may contain code for a large number of test cases. If, after an execution of the script, you can see that one or more of the test cases have failed, you may wish to rerun just these test cases.

You can do this by highlighting the test cases of interest and executing

### Test->Generate Script

You would choose **Create New Script Object** on the Script Generate Confirm Dialog ([Figure B-4](#)) because there is only a temporary need for this code. You could then load the new Script code into the Script Builder and run the code until it finds a failed comparison.



## Appendix C: Example Installation Files

This appendix contains examples of start-up scripts and example profile and system files mentioned in Section 13.

### C.1 Example AETG System Files

#### C.1.1 Example AETG Installation Session

This is an example BAIST installation session of the AETG System software.

```
Bellcore Application Installation Setup Tool
- - - - -
                BAIST 2.1

COPYRIGHT (c) 1996 Bell Communications Research Inc.,
                All Rights Reserved.

PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.

                A UNIX Packaging and Installation Tool

                For further information on BAIST
                Contact: Raymond C. Gray
                        BAIST Project Manager
                        (908)699-7960
```

```
                Please Wait ...
WARNING: You are using LOCAL BCRDB
***** BAIST Installation Tool Release 2.1 *****
Initializing. Please wait.
```

BAIST 2.1 FLOW CONTROL  
PRODUCT MENU  
Archived Products List

- 1) AETG 1.0
- 2) AETG UPDATES
- 3) TELEXEL 7.0

- I) Installed Products
- R) Registered Products
- E) Exit

Enter your Selection: **1**

Load AETG? [Y]: **y**

Extracting Application Profile

Getting the name of PRODUCT\_HOME dir

Where should the AETG directory be created ?

Enter a full pathname starting with /, or q to Quit processing

: **/opt/SUNWmyn**

Getting the name of RELEASE\_HOME dir

Making the PRODUCT\_HOME dir

Making the RELEASE\_HOME dir

Who should own the AETG application ?

Enter a valid login name : **admin**

Running PRELOAD script

Extracting the AETG application

Please wait. This may take a while ...

100226 of 100226 blocks loaded 100% COMPLETED

Running POSTLOAD script

Where should XMYHOME be installed? [/opt/SUNWmyn/mynah/releases/aetg\_home]

**/u/sol/XmyHome**

is /u/sol/XmyHome correct yes/no:

**yes**

Remember to:

- 1) edit /u/sol/XmyHome/config/xmyProfile
  - edit port number
  - edit LSHOST machine name
  - edit ORACLE server machine name (if using ORACLE)
  - edit vxIpcDirectory machine name

add . /u/sol/XmyHome/xmyProfile to user profiles

- 2) export your DISPLAY variable
  - export DISPLAY=machine:0.0

Product AETG AETG\_1.0 loaded successfully

BAIST 2.1 FLOW CONTROL  
PRODUCT MENU  
Archived Products List

- 1) AETG 1.0
- 2) AETG UPDATES
- 3) TELEXEL 7.0

- I) Installed Products
- R) Registered Products
- E) Exit

Enter your Selection:**E**

## C.1.2 Example System Changes File

This file contains changes that should be added to the */etc/system* file.

Once the changes have been made, type

```
reboot -- -rt
```

to reboot the system. The system will be reconfigured with the changes to */etc/system* incorporated in the kernel. You must do this as **root** on each system running a AETG component or on the ORACLE server.

```
set maxusers=128
set pt_cnt=60
# Settings for Message Queue parameters
#   MSGMNI : # of message queue identifiers
#   MSGTQL : # of system message headers
#   MSGMAP : # of entries in message map
#   MSGSSZ : segment size of message
#   MSGMNB : maximum bytes on queue
#   MSGMAX : maximum message size
#   MSGSEG : # of message segments
set msgsys:msginfo_msgmni=800
set msgsys:msginfo_msgtql=2000
set msgsys:msginfo_msgmap=1600
set msgsys:msginfo_msgssz=64
set msgsys:msginfo_msgmnb=65535
set msgsys:msginfo_msgmax=65535
set msgsys:msginfo_msgseg=16384
# Settings for Shared Memory parameters
#   SHMSEG : segments per process
#   SHMMAX : maximum shared memory segment size
#   SHMMNI : # shared memory identifiers
set shmsys:shminfo_shmseg=20
set shmsys:shminfo_shmmax=234881024
set shmsys:shminfo_shmmni=300
# Settings for Semaphore parameters
#   SEMMNS : # of semaphores in system
#   SEMMNI : # of semaphores identifiers
#   SEMMNU : # of "undo" in system
#   SEMUME :
#   SEMMAP :
set semsys:seminfo_semmns=7500
set semsys:seminfo_semmni=300
set semsys:seminfo_semmnu=300
set semsys:seminfo_semume=20
set semsys:seminfo_semmap=300
#
set s_xxx:max_ccbs=16
set s_xxx:x29_default=1
```



### C.1.3 Example AETG xmyProfile File

This is an example of the **ksh** *xmyProfile* files as it appears immediately following installation of the AETG software.

You must make the following changes to this file to reflect your environment:

- Update the directories may need to be updated
- Replace `<hostname>` with the actual machine names

**NOTE** — For the *oracleservername* entry, which is the name of your system running Oracle, this may or may not be the same as the system machine on which you installed the AETG System software.

- Replace `<port>` with the actual Telexel port (ex. 22100)

After you have made the necessary changes to this file it can be sourced into the *.profile* files for all AETG users.

```
# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)xmyProfile.eg52.3
# created on 97/10/07 at 15:37:52
#
# @(#)sample profile for madmin 96/06/11 SOL
#
# This is an example of a MYNAH System profile that
# can be sourced into MYNAH user's profiles to give
# them access to the system.
#
#
# XMYDIR is the path to a directory which is symbolically linked to
# the current release of MYNAH, e.g.,
# ln -s /opt/XXXXmyn/MYNAH/releases/MYNAH_5.0 /opt/XXXXmyn/mynah
#
XMYDIR=/opt/XXXXmyn/mynah
#
# XMYHOME is the path to a directory which is the MYNAH run environment
#
XMYHOME=/opt/XXXXmyn/mynah
```

---

```
#
#   create or append to LD_LIBRARY_PATH/SHLIB_PATH
#
if test -n "${LD_LIBRARY_PATH}"
then
    LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${XMYDIR}/lib:/usr/openwin/lib
else
    LD_LIBRARY_PATH=${XMYDIR}/lib:/usr/openwin/lib
fi

if test -n "${SHLIB_PATH}"
then
    SHLIB_PATH=${SHLIB_PATH}:${XMYDIR}/lib
else
    SHLIB_PATH=${XMYDIR}/lib
fi

#
#   create or append to MANPATH
#
if test -n "${MANPATH}"
then
    MANPATH=${MANPATH}:${XMYDIR}/man
else
    MANPATH=${XMYDIR}/man
fi

PATH=${PATH}:${XMYDIR}/bin

TCL_LIBRARY=${XMYDIR}/lib/tcl
TCLX_LIBRARY=${XMYDIR}/lib/tcl

export XMYDIR XMYHOME LD_LIBRARY_PATH SHLIB_PATH PATH TCL_LIBRARY
TCLX_LIBRARY

#
#   The host name that the MYNAH license server runs on
#
LSHOST=<hostname>
export LSHOST

#
#   Required Telexel variables.
#
#   1) Verify the host name for the environment variable
#       vxIpcDirectory.
#   2) Verify that vxIpcPort is a valid, unused port number.
#       a) Valid numbers are in the range 1024 and 65000.
#       b) If port number is already in use you will get an error
#          message similiar to "address already in use". The command
#          netstat will show port numbers currently in use. The file
#          /etc/services shows well known used ports.
#   3) If /opt/XXXXtel/telexel is not where you installed Telexel, update
#       the environment variable TELDIR to the correct location.
#

vxIpcDirectory=<hostname>
```

---

```
vxIpcPort=<port>
TELDIR=/opt/XXXXtel/telexel
PATH=${PATH}:${TELDIR}/bin

#
#   append to LD_LIBRARY_PATH/SHLIB_PATH
#
LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${TELDIR}/lib
SHLIB_PATH=${SHLIB_PATH}:${TELDIR}/lib

#
#   append to MANPATH
#
MANPATH=${MANPATH}:${TELDIR}/man

export vxIpcDirectory vxIpcPort TELDIR LD_LIBRARY_PATH SHLIB_PATH PATH
MANPATH

#
#   If Oracle is being used with MYNAH uncomment the following 3 statements,
#   however, leave the last 2 statements of the following 3 statements
#   commented if Oracle and MYNAH are installed on the same machine.
#
#oracleservername=<hostname>
#TWO_TASK=mynah5
#export TWO_TASK

#
#   If ORACLE is being used with MYNAH uncomment the following statements.
#   Verify and correct if necessary that ORACLE_HOME is being set to the
#   correct path. TNS_ADMIN must only be set if the TNS Listener files
#   installed during the Oracle installation are not installed in one of
#   locations that are searched automatically by Oracle.
#

#ORACLE_HOME=/opt/XXXXora/7.3.2.3
#ORACLE_TERM=sun5
#PATH=${PATH}:${ORACLE_HOME}/bin
#export PATH ORACLE_HOME ORACLE_TERM
#export EPC_DISABLE=TRUE

#export TNS_ADMIN=${ORACLE_HOME}/network/admin
#
#   If TOPCOM is being used uncomment the following and update
#   /opt/XXXXtop/topcom is not where TOPCOM is installed
#
#   TOPCOM=/opt/XXXXtop/topcom
#   TBIN=${TOPCOM}/bin
#   ETCPATH=/usr/public/isode/etc
#   PATH=${PATH}:${TBIN}
#   export TOPCOM TBIN PATH ETCPATH

#
#   If I/O Concepts is being used uncomment the following statement and
#   update /opt/XXXXioc/ioconcepts if this is not where I/O Concepts is
#   installed. Also set IOCLM_HOST to be the name of the host name on
#   which is I/O Concepts license server runs.
```

---

```
#
# LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/dt/lib
# IOCLM_HOST=<hostname>
# export LD_LIBRARY_PATH IOCLM_HOST

#
#
# optional command line editor enable
#
EDITOR=vi
set -o vi
```

### C.1.4 Example AETG xmyLogin File

This is an example of the **cs**h *xmyLogin* file as it appears immediately following installation of the AETG software.

You must make the following changes to this file to reflect your environment:

- Update the directories may need to be updated
- Replace `<hostname>` with the actual machine names
- Replace `<oracleservername>` with the name of the machine on which you installed the Oracle software.
- Replace `<port>` with the actual Telexel port (ex. 22100).

After you have made the necessary changes to this file it can be sourced into the *.login* files for all AETG users.

```
# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)xmyLogin.eg52.3
# created on 97/10/07 at 15:37:23
#
# @(#)sample login for madmin 96/06/11 SOL (csh version)
#
# This is an example of a MYNAH System profile that
# can be sourced into MYNAH user's .login to give
# them access to the system.
#
#
# XMYDIR is the path to a directory which is symbolically linked to
# the current release of MYNAH, e.g.,
# ln -s /opt/XXXXmyn/MYNAH/releases/MYNAH_5.0 /opt/XXXXmyn/mynah
#
setenv XMYDIR /opt/XXXXmyn/mynah
#
# XMYHOME is the path to a directory which is the MYNAH run environment
#
setenv XMYHOME /opt/XXXXmyn/mynah
#
# create or append to LD_LIBRARY_PATH/SHLIB_PATH
#
```

---

---

```
if ( $?LD_LIBRARY_PATH ) then
    setenv LD_LIBRARY_PATH
"${LD_LIBRARY_PATH}:${XMYDIR}/lib:/usr/openwin/lib"
else
    setenv LD_LIBRARY_PATH "${XMYDIR}/lib:/usr/openwin/lib"
endif

if ( $?SHLIB_PATH ) then
    setenv SHLIB_PATH "${SHLIB_PATH}:${XMYDIR}/lib:/usr/openwin/lib"
else
    setenv SHLIB_PATH "${XMYDIR}/lib:/usr/openwin/lib"
endif

#
#   create or append to MANPATH
#

if ( $?MANPATH ) then
    setenv MANPATH "${MANPATH}:${XMYDIR}/man"
else
    setenv MANPATH "${XMYDIR}/man"
endif

setenv TCL_LIBRARY "${XMYDIR}/lib/tcl"
setenv TCLX_LIBRARY "${XMYDIR}/lib/tcl"
set path = ($path $XMYDIR/bin)

#
#   The host name that the MYNAH license server runs on
#
setenv LSHOST <hostname>

#
#   Required Telexel variables.
#
#   1) Verify the host name for the environment variable
#       vxIpcDirectory.
#   2) Verify that vxIpcPort is a valid, unused port number.
#       a) Valid numbers are in the range 1024 and 65000.
#       b) If port number is already in use you will get an error
#          message similiar to "address already in use". The command
#          netstat will show port numbers currently in use. The file
#          /etc/services shows well known used ports.
#   3) If /opt/XXXXtel/telexel is not where you installed Telexel, update
#       the environment variable TELDIR to the correct location.
#

setenv vxIpcDirectory <hostname>
setenv vxIpcPort <port>
setenv TELDIR /opt/XXXXtel/telexel
set path = ($path $TELDIR/bin)
setenv LD_LIBRARY_PATH "${LD_LIBRARY_PATH}:${TELDIR}/lib"
setenv SHLIB_PATH "${SHLIB_PATH}:${TELDIR}/lib"
setenv MANPATH "${MANPATH}:${TELDIR}/man"

#
#   If Oracle is being used with MYNAH uncomment the following statement.
```

---

```
#
setenv TWO_TASK mynah5

#
#   If ORACLE is being used with MYNAH uncomment the following statements.
#   Verify and correct if necessary that ORACLE_HOME is being set to the
#   correct path.  TNS_ADMIN must only be set if the TNS Listener files
#   installed during the Oracle installation are not installed in one of
#   locations that are searched automatically by Oracle.
#

#setenv ORACLE_HOME /opt/XXXXora/7.3.2.3
#setenv ORACLE_TERM sun5
#set path = ( $path $ORACLE_HOME/bin)
#setenv EPC_DISABLE TRUE
#
#setenv TNS_ADMIN ${ORACLE_HOME}/network/admin

#
#   If TOPCOM is being used uncomment the following and update
#   /opt/XXXXtop/topcom is not where TOPCOM is installed
#
#   setenv TOPCOM /opt/XXXXtop/topcom
#   setenv TBIN ${TOPCOM}/bin
#   setenv ETCPATH /usr/public/isode/etc
#   set path = ( $path $TBIN)

#
#   If I/O Concepts is being used uncomment the following statement and
#   update /opt/XXXXioc/ioconcepts if this is not where I/O Concepts is
#   installed. Also set IOCLM_HOST to be the name of the host name on
#   which is I/O Concepts license server runs.
#
#   setenv LD_LIBRARY_PATH "${LD_LIBRARY_PATH}:/usr/dt/lib"
#   setenv IOCLM_HOST <hostname>

#
#
# optional command line editor enable
#
setenv EDITOR vi
```

---

### C.1.5 Example Solaris AETG Start-up File (S99mynah.eg)

This is a example Solaris AETG processes start-up file. Rename this file, e.g., *S99mynah*, and place it in */etc/rc3.d* with **root** as the owner. This is for the server only. You may need to update the required paths and directories.

**NOTE** — A logical link to this file should be set up in */etc/rc0.d* as follows:

```
ln -s /etc/rc3.d/S99mynah K01mynah

#!/bin/ksh

# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)S99mynah.eg 50.7
# created on 96/09/18 at 15:16:02

#
# This is a sample startup file for AETG. It should be copied
# to the /etc/rc3.d directory with root as the owner. This is
# for the server only. Directories by need to be updated.
#
# Also a logical link to this file should be set
# up in /etc/rc0.d as follows:
# ln -s /etc/rc3.d/S99mynah K01mynah

vxIpcPort=<IPC Port>
vxIpcDirectory=<hostname>
TELDIR=/opt/SUNWtel/telexel
XMYDIR=/opt/SUNWmyn/mynah
XMYHOME=/opt/SUNWmyn/mynah
export vxIpcPort vxIpcDirectory TELDIR XMYDIR XMYHOME
LD_LIBRARY_PATH=${XMYDIR}/lib:${TELDIR}/lib:/usr/openwin/lib
PATH=/usr/bin:/usr/sbin:/usr/ccs/bin:/usr/openwin/bin:/usr/ucb:/etc:${XMYDIR}/bin
export LD_LIBRARY_PATH PATH

case "$1" in
'start')
    # Start up AETG and Telexel

# removing AETG pip files that might be left over
if [ -f ${XMYHOME}/run/oa/pip.oa.<hostname> ]; then
    (echo 'Deleting stale AETG pip files.') >/dev/console
    (echo 'Deleting stale AETG pip files.') >>/var/adm/messages
    ( /bin/su - madadmin -c "${XMYDIR}/bin/.xmyRemovePips all"; )
```



```
fi

if [ -f ${XMYDIR}/bin/xmyStartOA ]; then
    (echo 'starting AETG & Telexel processes.') >/dev/console
    (echo 'starting AETG & Telexel processes.') >>/var/adm/messages
    ( /bin/su - madmin -c "${XMYDIR}/bin/xmyStartOA"; )
#    (echo 'starting AETG Collector Process.') >/dev/console
#    (echo 'starting AETG Collector Process.') >>/var/adm/messages
#    ( /bin/su - madmin -c "${XMYDIR}/bin/xmyStartCL -n <collector name>"; )
fi

    ;;

'stop')

if [ -f ${XMYDIR}/bin/xmyStopOA ]; then
    (echo 'stopping AETG processes.') >/dev/console
    (echo 'stopping AETG processes.') >>/var/adm/messages
    ( /bin/su - madmin -c "${XMYDIR}/bin/xmyOM shutdown <hostname>"; )
#    ( /bin/su - madmin -c "${XMYDIR}/bin/xmyStopCL -n <collector name>"; )
fi

    ;;

*)

    ;;

esac
```

## C.2 Example Telexel Installation Session

This is an example BAIST installation session of the Telexel System software.

```
Bellcore Application Installation Setup Tool
- - - - -
                BAIST 2.1

COPYRIGHT (c) 1996 Bell Communications Research Inc.,
          All Rights Reserved.

PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.

          A UNIX Packaging and Installation Tool

          For further information on BAIST
          Contact:  Raymond C. Gray
                   BAIST Project Manager
                   (908)699-7960
```

Please Wait ...

```
WARNING: You are using LOCAL BCRDB
***** BAIST Installation Tool Release 2.1 *****
Initializing. Please wait.
```

```
BAIST 2.1 FLOW CONTROL
PRODUCT MENU
  Archived Products List
```

- 1) AETG 1.0
- 2) AETG UPDATES
- 3) TELEXEL 7.0

- I) Installed Products
- R) Registered Products
- E) Exit

```
          Enter your Selection: 3
Load TELEXEL? [Y]: Y
Extracting Application Profile

Getting the name of PRODUCT_HOME dir

Where should the TELEXEL directory be created ?
Enter a full pathname starting with /, or q to Quit processing
: /opt/SUNWtel
Getting the name of RELEASE_HOME dir
```

Making the PRODUCT\_HOME dir  
Making the RELEASE\_HOME dir

Who should own the TELEXEL application ?  
Enter a valid login name :**madmin**

Running PRELOAD script

Extracting the TELEXEL application  
Please wait. This may take a while ...  
9096 of 9096 blocks loaded 100% COMPLETED  
Product TELEXEL TELEXEL\_6.0 loaded successfully

BAIST 2.1 FLOW CONTROL  
PRODUCT MENU  
Archived Products List

- 1) AETG 1.0
- 2) AETG UPDATES
- 3) TELEXEL 7.0

- I) Installed Products
- R) Registered Products
- E) Exit

Enter your Selection:**E**

## C.3 Delivered Example Oracle Files

This section contains all of the example Oracle start-up and configuration files that are delivered with the AETG System.

### C.3.1 Example Solaris Oracle Start-up File (S96oracle)

This is an example Solaris Oracle start-up file. It should be moved to */etc/rc3.d* with **root** as the owner. You may need to update the required directories.

```
#!/bin/ksh

# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)S96oracle.eg50.5
# created on 96/09/26 at 17:24:32

# This is a sample startup file for oracle. It should
# be moved to the /etc/rc3.d directory with root as the
# owner. Directories may need to be updated.

#
# Required update
#
# If /opt/SUNWora/oracle is not where you installed Oracle, update
# the environment variable ORACLE_HOME to the correct location.
#
ORACLE_HOME=/opt/SUNWora/oracle
export ORACLE_HOME

case "$1" in
'start')
    # Start up Oracle

# Delete nasty Oracle tmp directory
if [ -d /var/tmp/o ]; then
    rm -rf /var/tmp/o
fi

if [ -f ${ORACLE_HOME}/bin/dbstart ]; then
    (echo 'starting oracle7.') >/dev/console
    (echo 'starting oracle7.') >/var/adm/messages
    ( /bin/su - oracle -c "${ORACLE_HOME}/bin/dbstart"; )
    (echo 'starting oracle sqlnet server.') >/dev/console
```

```
        (echo `starting oracle sqlnet server.`) >/var/adm/messages
        ( /bin/su - oracle -c "export PATH=${ORACLE_HOME}/bin:$PATH ;
${ORACLE_HOME}/bin/tcpctl start"; )
        # Start sqlnet V2 listner
#        (echo `starting oracle sqlnet V2 server.`) >/dev/console
#        (echo `starting oracle sqlnet V2 server.`) >/var/adm/messages
        ( /bin/su - oracle -c " export PATH=${ORACLE_HOME}/bin:/usr/ccs/bin:$PATH
; ${ORACLE_HOME}/bin/lsnrctl start"; )

fi

        ;;
`stop`)

if [ -f ${ORACLE_HOME}/bin/tcpctl ]; then
        (echo `stopping oracle7.`) >/dev/console
        (echo `stopping oracle7.`) >/var/adm/messages
        ( /bin/su - oracle -c "${ORACLE_HOME}/bin/dbshut"; )
#        ( /bin/su - oracle -c " export PATH=${ORACLE_HOME}/bin:$PATH ; \
#          ${ORACLE_HOME}/bin/lsnrctl stop"; )
        ( /bin/su - oracle -c "${ORACLE_HOME}/bin/tcputl Q"; )
fi
        ;;
*)
        ;;
esac
```

### C.3.2 Example Oracle Configuration File

This is an example Oracle configuration file, *configmynah5.ora*. For this example the Oracle software is installed into */opt/SUNWora/7.2.3.2*. If you use a different directory, the directory statements in this example will need to be updated.

```
# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)configmynah5.ora 50.2
# created on 96/07/16 at 17:26:27

#
# $Header: cnfg.orc 7001200.2 93/04/26 14:58:22 eruben Osd<unix> $ Copyr (c) 1992
Oracle
#

#####
# NOTE:
# This is an example configmynah5.ora file. Directories will need
# to be updated if different. For this example the oracle software
# is installed into /opt/SUNWora/oracle.
#####

# cnfg.ora - instance configuration parameters

control_files          = (/opt/SUNWora/oracle/dbs/ctrl1mynah5.ctl,
                        /opt/SUNWora/oracle/mynah5/datafiles/d02/ctrl2mynah5.ctl,
                        /opt/SUNWora/oracle/mynah5/datafiles/d03/ctrl3mynah5.ctl)
# Below for possible future use...
#init_sql_files        = (?/dbs/sql.bsq,
#                          ?/rdbms/admin/catalog.sql,
#                          ?/rdbms/admin/expvew.sql)
background_dump_dest   = /opt/SUNWora/oracle/rdbms/log
core_dump_dest         = /tmp
user_dump_dest         = /opt/SUNWora/oracle/rdbms/log
#log_archive_dest= /opt/SUNWora/oracle/dbs/arch/arch.log
db_block_size          = 2048

db_name                = mynah5
```

### C.3.3 Example Oracle Initialization Script (initmynah5.ora)

This is an example Oracle initialization file, *initmynah5.ora*. Directories will need to be updated if different. For this example the oracle software is installed into */opt/SUNWora/7.2.3.2*.

```
# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)initmynah5.ora 50.2
# created on 96/07/16 at 17:26:40

# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)initmynah5.ora 50.1
# created on 96/07/03 at 15:58:57

#
# $Header: initx.orc 7001300.3 93/06/16 12:28:26 mkrishna Osd<unix> $ Copyr (c)
1992 Oracle
#

#####
# NOTE:
# This is an example initmynah5.ora file. Directories will need
# to be updated if different. For this example the oracle software
# is installed into /opt/SUNWora/oracle.
#####

# include database configuration parameters
ifile = /opt/SUNWora/oracle/dbs/configmynah5.ora

#rollback_segments = (r01)
rollback_segments= (r01,r02,r03,r04)

#####
# Example INIT.ORA file
#
# This file is provided by Oracle Corporation to help you customize
# your RDBMS installation for your site. Important system parameters
# are discussed, and example settings given.
```

---

---

```
#
# Some parameter settings are generic to any size installation.
# For parameters that require different values in different size
# installations, three scenarios have been provided: SMALL, MEDIUM
# and LARGE. Any parameter that needs to be tuned according to
# installation size will have three settings, each one commented
# according to installation size.
#
# Use the following table to approximate the SGA size needed for the
# three scenarios provided in this file:
#
# -----Installation/Database Size-----
#           SMALL           MEDIUM           LARGE
# Block      2K      4500K           6800K           17000K
# Size       4K      5500K           8800K           21000K
#
# To set up a database that multiple instances will be using, place
# all instance-specific parameters in one file, and then have all
# of these files point to a master file using the IFILE command.
# This way, when you change a public
# parameter, it will automatically change on all instances. This is
# necessary, since all instances must run with the same value for many
# parameters. For example, if you choose to use private rollback segments,
# these must be specified in different files, but since all gc_*
# parameters must be the same on all instances, they should be in one file.
#
# INSTRUCTIONS: Edit this file and the other INIT files it calls for
# your site, either by using the values provided here or by providing
# your own. Then place an IFILE= line into each instance-specific
# INIT file that points at this file.
#####

# tuning parameters

db_files =1020

db_file_multiblock_read_count = 8           # SMALL
# db_file_multiblock_read_count = 16       # MEDIUM
# db_file_multiblock_read_count = 32       # LARGE

db_block_buffers = 200                     # SMALL
# db_block_buffers = 550                   # MEDIUM
# db_block_buffers = 3200                  # LARGE

shared_pool_size = 3500000                 # SMALL
# shared_pool_size = 6000000              # MEDIUM
# shared_pool_size = 9000000              # LARGE

log_checkpoint_interval = 10000

processes = 50                             # SMALL
# processes = 100                         # MEDIUM
# processes = 200                         # LARGE

dml_locks = 100                            # SMALL
# dml_locks = 200                         # MEDIUM
# dml_locks = 500                         # LARGE
```

---



```
log_buffer = 8192                # SMALL
# log_buffer = 32768             # MEDIUM
# log_buffer = 163840           # LARGE

sequence_cache_entries = 10     # SMALL
# sequence_cache_entries = 30   # MEDIUM
# sequence_cache_entries = 100  # LARGE

sequence_cache_hash_buckets = 10 # SMALL
# sequence_cache_hash_buckets = 23 # MEDIUM
# sequence_cache_hash_buckets = 89 # LARGE

# audit_trail = true            # if you want auditing
# timed_statistics = true       # if you want timed statistics
max_dump_file_size = 10240      # limit trace file size to 5 Meg each

# log_archive_start = true      # if you want automatic archiving

mts_dispatchers="ipc,1"
mts_max_dispatchers=10
mts_servers=1
mts_max_servers=10
mts_service=mynah5
mts_listener_address="(ADDRESS=(PROTOCOL=ipc)(KEY=mynah5))"
nls_date_format = "DD-MON-YYYY"
remote_os_authent = true
```

### C.3.4 Example Oracle *crdbmynah5.sql* File

This is an example Oracle *crdbmynah5.sql* file. For this example the Oracle software is installed into */opt/SUNWora/7.2.3.2*. If you use a different directory, the directory statements in this example will need to be updated.

```
# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)crdbmynah5.sql50.2
# created on 96/07/16 at 17:26:15

REM #####
REM # NOTE:
REM # This is an example crdbmynah5.sql file. Directories will need
REM # to be updated if different. For this example the oracle software
REM # is installed into /opt/SUNWora/oracle.
REM #####

REM * Set terminal output and command echoing on; log output of this script.
REM *
#set termout on
#set echo on
spool /opt/SUNWora/oracle/dbs/crdbmynah5.lst

REM * Start the <sid> instance (ORACLE_SID here must be set to <sid>).
REM *

REM * Create the <dbname> database.
REM * SYSTEM tablespace configuration guidelines:
REM *   General-Purpose ORACLE RDBMS    5Mb
REM *   Additional dictionary for applications10-50Mb
REM * Redo Log File configuration guidelines:
REM *   Use 3+ redo log files to relieve ``cannot allocate new log...'' waits.
REM *   Use ~100Kb per redo log file per connection to reduce checkpoints.
REM *
create database "mynah5"
maxinstances 1
maxlogfiles 16
character set "US7ASCII"
datafile
  '/opt/SUNWora/oracle/mynah5/datafiles/d01/systmynah5.dbf' size 25M
logfile
  '/opt/SUNWora/oracle/mynah5/logfiles/log1mynah5.dbf' size 500k,
  '/opt/SUNWora/oracle/mynah5/logfiles/log2mynah5.dbf' size 500k,
  '/opt/SUNWora/oracle/mynah5/logfiles/log3mynah5.dbf' size 500k;

disconnect
spool off
```

### C.3.5 Example Oracle *crdb2mynah5.sql* File

This is an example Oracle *crdb2mynah5.sql* file. For this example the Oracle software is installed into */opt/SUNWora/7.2.3.2*. If you use a different directory, the directory statements in this example will need to be updated.

This script takes care off all commands necessary to create an OFA compliant database after the **create database** command has succeeded.

```
# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)crdb2mynah5.sql50.2
# created on 96/07/16 at 17:26:03

# COPYRIGHT (c) 1996 Bell Communications Research Inc.,
# All Rights Reserved.
#
# PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
#
# This document contains proprietary information that shall
# be distributed or routed only within Bell Communications
# Research (Bellcore), and its authorized clients, except
# with written permission of Bellcore.
#
# @(#)crdb2mynah5.sql50.1
# created on 96/07/03 at 15:58:45

REM #####
REM # NOTE:
REM # This is an example crdb2mynah5.sql file. Directories will need
REM # to be updated if different. For this example the oracle software
REM # is installed into /cowboyl/opt/SUNWora/oracle.
REM #####

REM * This script takes care off all commands necessary to create
REM * an OFA compliant database after the CREATE DATABASE command has
REM * succeeded.

REM * Set terminal output and command echoing on; log output of this script.
REM *
#set termout on
#set echo on
#spool 2-rdbms.lst
spool /cowboyl/opt/SUNWora/oracle/dbs/crdb2mynah5s1.lst

connect internal
```

---

```
REM # install data dictionary views:
@/cowboy1/opt/SUNWora/oracle/rdbms/admin/catalog.sql

REM * Create additional rollback segment in SYSTEM before creating tablespace.
REM *
connect internal
create rollback segment r0 tablespace system
  storage (initial 16k next 16k minextents 2 maxextents 20);

REM * Use ALTER ROLLBACK SEGMENT ONLINE to put r0 online without shutting
REM * down and restarting the database.
REM *
alter rollback segment r0 online;

REM * Create a tablespace for rollback segments.
REM * Rollback segment configuration guidelines:
REM *   1 rollback segments for every 4 concurrent xactions.
REM *   No more than 50 rollback segments.
REM *   All rollback segments the same size.
REM *   Between 2 and 4 homogeneously-sized extents per rollback segment.
REM *   Attempt to keep rollback segments to 4 extents.
REM *
create tablespace rbs datafile
  '/cowboy1/opt/SUNWora/oracle/mynah5/datafiles/d02/rbsmynah5sl.dbf' size 4M
default storage (
  initial      128k
  next         128k
  pctincrease  0
  minextents   2
);

REM * Create a tablespace for temporary segments.
REM * Temporary tablespace configuration guidelines:
REM *   Initial and next extent sizes = k * SORT_AREA_SIZE, k in {1,2,3,...}.
REM *
create tablespace temp datafile
  '/cowboy1/opt/SUNWora/oracle/mynah5/datafiles/d01/tempmynah5sl.dbf' size 550k
default storage (
  initial      256k
  next         256k
  pctincrease  0
  optimal      1M
);

REM * Create a tablespace for database tools.
REM *
create tablespace tools datafile
  '/cowboy1/opt/SUNWora/oracle/mynah5/datafiles/d03/toolmynah5sl.dbf' size 15M;

REM * Create a tablespace for mynah5 databases.
REM *
create tablespace my5 datafile
  '/cowboy1/opt/SUNWora/oracle/mynah5/datafiles/d03/my5mynah5sl.dbf' size 15M;

REM * Create a tablespace for miscellaneous database user activity.
REM *
create tablespace users datafile
```

---

```
    '/cowboy1/opt/SUNWora/oracle/mynah5/datafiles/d01/usrmynah5s1.dbf' size 1M;

REM * Create rollback segments.
REM *
create rollback segment r01 tablespace rbs;
create rollback segment r02 tablespace rbs;
create rollback segment r03 tablespace rbs;
create rollback segment r04 tablespace rbs;

REM * Use ALTER ROLLBACK SEGMENT ONLINE to put rollback segments online
REM * without shutting down and restarting the database. Only put one
REM * of the rollback segments online at this time so that it will always
REM * be the one used. When the user shuts down the database and starts
REM * it up with initSID.ora, all four will be brought online.
REM *
alter rollback segment r01 online;
REM * alter rollback segment r02 online;
REM * alter rollback segment r03 online;
REM * alter rollback segment r04 online;

REM * Since we've created and brought online 2 more rollback segments,
REM * we no longer need the second rollback segment in the SYSTEM tablespace.
alter rollback segment r0 offline;
drop rollback segment r0;

REM * Alter SYS and SYSTEM users.
REM *
alter user sys temporary tablespace temp;
#revoke resource from system;
#revoke resource on system from system;
#grant resource on tools to system;
alter user system default tablespace tools temporary tablespace temp;

REM * For each DBA user, run DBA synonyms SQL script. Don't forget that EACH
REM * DBA USER created in the future needs dba_syn.sql run from its account.
REM *
connect system/manager
@/cowboy1/opt/SUNWora/oracle/rdbms/admin/catdbsyn.sql

spool off
```

### C.3.6 Example Oracle *crdb3mynah5.sql* File

This is an example Oracle *crdb3mynah5.sql* file. For this example the Oracle software is installed into */opt/SUNWora/7.2.3.2*. If you use a different directory, the directory statements in this example will need to be updated.

```
/*
 * COPYRIGHT (c) 1997 Bell Communications Research Inc.,
 * All Rights Reserved.
 *
 * PROPRIETARY - BELLCORE AND AUTHORIZED CLIENTS ONLY.
 *
 * This document contains proprietary information that shall
 * be distributed or routed only within Bell Communications
 * Research (Bellcore), and its authorized clients, except
 * with written permission of Bellcore.
 */

/*
 * @(#)crdb3mynah5.sql52.1
 * created on 97/05/27 at 09:58:52
 */

static char crdb3mynah5_sql[] = " @(#)crdb3mynah5.sql52.1";

/* Prevent warning message from CC about sccs string defined but not used */
#ifdef __cplusplus
inline void static dummy_crdb3mynah5_sql() { if (crdb3mynah5_sql); }
#endif

Rem
Rem cd $ORACLE_HOME/rdbms/admin
Rem
Rem $ORACLE_HOME/bin/sqlplus sys/change_on_install << "EOF
connect sys/change_on_install
@$ORACLE_HOME/rdbms/admin/standard
@$ORACLE_HOME/rdbms/admin/catproc
@$ORACLE_HOME/rdbms/admin/catalog
@$ORACLE_HOME/rdbms/admin/dbmspipe

create user mynah identified by mynah
default tablespace my5
temporary tablespace temp;

grant execute any procedure to mynah;

grant connect to mynah;

grant resource to mynah;

commit;
Rem exit;
Rem EOF

Rem cd $ORACLE_HOME/sqlplus/admin
Rem $ORACLE_HOME/bin/sqlplus system/manager << "EOF
```

---

```
connect system/manager
@$ORACLE_HOME/sqlplus/admin/pupbld
exit;
Rem EOF
```

### C.3.7 Example xmyCreateSequences Execution

```
Continue [y/n]: y
Database connection opened.
Sending sql: CREATE SEQUENCE xmyRunTime_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyScript_Sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyPerson_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyTestVersion_Sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyHierarchyNode_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyHierarchy_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmySutInfo_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyTest_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyUserEnumValue_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyCompareResult_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyResult_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyResultDelta_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyKeyword_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyIssue_Sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyProcedure_Sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyProcedureLibrary_Sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyRequirement_Sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyCompound_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyField_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyFieldGroup_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyFieldValue_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyRelation_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmySutFormat_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyData_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyDocument_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyResource_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyResourceUsage_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyStep_sequence MINVALUE 1 CACHE 500
Sending sql: CREATE SEQUENCE xmyStepList_sequence MINVALUE 1 CACHE 500
```



### C.3.8 Example xmyCreateTemplates Execution

```
Creating template objects.  
Mynah administrator successfully created  
madmin successfully created  
UserEnumValue template successfully created  
UserEnumValue template successfully locked  
defaults userEnumValues installed.  
ScriptOid=1  
Script template successfully created  
Script template successfully locked  
Hierarchy template successfully created  
Hierarchy template successfully locked  
HierarchyNode template successfully created  
HierarchyNode template successfully locked  
HierarchyNode for demo successfully created  
SutInfo template successfully created  
SutInfo template successfully locked  
sutInfo demo successfully created  
RunTime template successfully created  
RunTime template successfully locked
```

### C.3.9 Example root.sh Run

```
# ./root.sh
Running ORACLE7 root.sh script...
The following environment variables are set as:
    ORACLE_OWNER= oracle
    ORACLE_HOME=  /opt/SUNWora/V7.2.3.2
    ORACLE_SID=   mynah5
Are these settings correct (Y/N)? [Y]: Y

Enter the full pathname of the local bin directory [/opt/bin]:
/usr/local/bin

Checking for "oracle" user id...
ORACLE_HOME does not match the home directory for oracle.
Okay to continue? [N]: Y

Creating /var/opt/oracle/oratab file...
Updating /var/opt/oracle/oratab file...

Please raise the ORACLE owner's ulimit as per the IUG.

Leaving common section of ORACLE7 root.sh.
Setting orasrv file protections
```

## Glossary

### A

**AETG** — Automatic Efficient Testcase Generator. A Bellcore developed system to generate automatically and efficiently an optimum set of test cases from a set of requirements.

### C

**CLUI** — Command Line User Interface

**Compound** — A database object that represents a complex field. It is made up of multiple simple Fields and specific values

**Config file** — The AETG Configuration File (named *xmyConfig.General*) that resides in the directory *\$XMYHOME/config*.

### F

**Field** — A database object that is used to hold information about the data input to be applied to an application (sometimes also referred to as a SIMPLE FIELD).

**Format** — A database object that represents a collection of fields for input purposes; the interface into which data is input to an application.

### G

**GUI** — Graphical User Interface.

### I

**Icon** — An X-Window that has been closed using a window manager function.

**Iconified** — The state of an X-Window

after it has become an icon.

### L

**List** — An ordered collection of elements.

**Log File** — A file containing a record of activity for a software product.

### R

**Relation** — A database object that is used to express the rules governing the data input (formal requirements) to the application under test

### S

**Script** — A file that contains one or more instructions to be performed by a domain.

**SUT** — System Under Test.

**System Under Test** — A database object that is used to represent information about a System Under Test. The System Under Test being represented can be an application, a specific release of an application, a platform that an application runs on, a testing cycle for an application, a group of applications, or any combination of these things.

### T

**Test** — A database object that is used to track and verify compliance with specific system requirements

**Tuple** — Reference name given to the specific value set of a Compound. This symbolic value can be expanded into the specific values of the simple Fields represented by the compound.



---

## Index

### A

Accelerator keys  
  description, 3-13  
  list and functions, 3-13

Add Row pushbutton in Include dialog, 5-8

Administrative Privileges  
  Granting, 17-3  
  Removing, 17-4

AETG Desktop, 2-4, 3-1  
  Preferences view, 3-22  
    edited, 3-24

AETG Processes, Starting and Stopping  
  See Operability Management

AETG System  
  AETG Desktop Preference view, 3-22  
  Creating the Database, 13-29  
  Database  
    Creating, 13-29  
    Dropping, 13-30  
  Desktop description, 3-2  
  Dropping the Database, 13-30  
  Exiting, 2-40, 3-5, 3-34  
  Folder restrictions, 3-2  
  Icon description and use, 3-4  
  Installation, 13-11 to 13-13  
    CD-ROM, 13-11  
    Configuring, 14-1  
    File Archive, 13-12  
    Installing the Software, 13-11  
    Post-Installation Steps, 13-13  
    Pre-Installation Steps, 13-11  
  Menu description and menu options, 3-5  
  Networking, 13-2  
  Overview, 1-1  
  Printing, 3-30  
  Processes  
    Automatically Starting, 14-4  
    Responsibility List, 14-6  
    Starting, 14-4  
      Commands, 14-5  
    Status of, Obtaining, 14-4

  Commands, 14-5  
  Stopping, 14-4  
    Commands, 14-5  
  Starting, 1-1, 2-3  
  System menu description, 3-4

Attributes copied when duplicating objects,  
  4-7

Authority Level, 17-3  
  Administrator, 17-3  
  Changing, 17-6  
  Inactive, 17-3  
  *See Also* Administrative Privileges  
  *See Also* Person Objects  
  User, 17-4

Autostart Processes, 14-5, 15-1, 15-3  
  Starting, 15-10  
  Stopping, 15-11

### B

BAIST, 13-8  
  Creating Version Directories, 13-3  
  Pre-installation Considerations, 13-8  
  Setting Installation Directory, 13-8  
  UNIX Shell, 13-8

Blanks in values, 7-4

### C

CD-ROM, Installation  
  AETG System, 13-11  
  Telexel System, 13-15

Changing a Folder's name, 3-28

Clipboard, 4-8

Cloning  
  Format object, 6-16  
  Relation object, 9-14  
  Test object, 9-6

CLUI, 1-7  
  Administrative Commands, 16-1 to 16-8  
  Help Messages, 16-1  
  xmyOM, 16-2 to 16-8

- 
- Displaying Configuration
    - Processes, 16-4
  - OA Reading the Configuration File, 16-5
  - Shutting Down an OA, 16-7
  - Start, Stop, Status Subcommands, 16-7
  - Starting Autostart Processes, 16-2
  - Stopping and Restarting Autostart Processes, 16-6
  - Stopping Autostart Processes, 16-3
- Comma separated values, 9-29
- Command Line Interface, Modeling, 12-3
- Complex Field, see Compound object
- Compound object
  - Creating on a Format object, 8-1
  - Entering a description, 8-3
  - Fields in Compound, listing, 8-4
  - Fields view, 8-4
  - Interaction Degree, 8-4
    - Selecting, 8-6
    - Unrelated, 8-6
  - Interaction degree
    - Tuples
      - Cross Product, 8-6
      - Selecting, 8-4
  - List of in a Format object, 6-5
  - Maximum number of Tuples
    - Adding up to, 8-12
    - Setting, 8-3
  - Name, specifying, 8-2
  - Properties view, 8-2
  - Specifying a name, 8-2
  - Status, displaying, 8-3
  - Tuples, 8-1
    - Deleting, 8-14
    - Generate Tuples menu option, 8-9, 8-10
    - Generating, 8-8
      - At one time, 8-9
      - Iteratively, 8-10
    - Generating before saving a Compound object, 8-14
    - List, 8-7
    - Names, 8-7
      - Changing a Tuples name, 8-12
      - Regenerating, 8-13
    - System generated, 8-12
    - See also Tuples
    - Tuples view, 8-7
    - Values, clearing, 8-5
    - Values, selecting, 8-5
    - Why use, 8-1- Compound objects
  - Creating, 6-9
  - Deleting, 6-12
- Compounds, Naming, 8-2
- Configuration
  - AETG System, 14-1
  - Configuring AETG Database, 13-26
  - Configuring the Telexel System, 13-16
  - Database, Specifying Whether to Use, 14-2
  - General Entries, 14-1
  - OA, 14-6
  - OM Port Number, 14-2
  - Port Numbers, 14-2
  - Telexel, 13-16
  - xmyConfig.General File
    - General Entries, 14-1
  - xmyConfigOP File
    - General Entries, 14-3
    - See Also xmyConfigOP File
    - Syntax, 14-3 to 14-7
- Configuration Parameters
  - xmyConfig.General File
    - Database, 14-2
    - DefaultSD, 14-1
    - WelcomeNewUsers, 14-2
  - xmyConfigOP File
    - AutoStart, 14-4
    - Mynah, 14-4
    - Responsibility, 14-6
    - Start, 14-4
    - Status, 14-4
    - Stop, 14-4
- Configurations, Modeling, 12-5
- Control key, 4-10
- Copying closed objects, 4-6
- Copying opened objects, 4-6
- Creating a folder, 2-5
- Creating objects on the AETG Desktop, 4-3
- Creating Test objects in a hierarchy, 9-4

---

Creating the AETG System Database, 13-29  
csh  
    Setting BAIST Installation Directory,  
        13-8  
    Use of xmyLogin File, 13-5  
Customer Support, 1-8

## D

Database  
    AETG System  
        Configuring, 13-26  
        Running Without a Database, 14-2  
    Deleting hierarchies, 9-4  
    Deleting objects from, 3-6, 4-8  
    Dropping Oracle, 13-30  
    Duplicating objects, 4-7  
    Installing Oracle, 13-19 to 13-31  
    Maximum Size, 3-23  
    Searching for objects, 3-7  
    *See Also* Oracle  
    Specifying Whether to Use, 14-2  
    Storing objects in the database, 4-2  
    Verifying, 13-30  
Database Browser  
    Accessing from Tools Menu, 5-2  
    Description, 3-7  
    general description, 5-1  
    Include dialog  
        Add Row pushbutton, 5-8  
        Delete Row pushbutton, 5-11  
    Object type, selecting, 5-3  
    Opening objects, 5-11  
    queries, default, 5-6  
    *See Also* Include dialog  
    Sorting objects, 5-3  
    uses, 5-1  
    Using the Include Dialog, 5-5  
    Using to associate a Format object with a  
        Test object, 9-12  
Decoding licensing codes, 15-7  
Default settings  
    Database, Maximum size, 3-23  
    Fonts, 3-22  
    printer, 3-22  
    Saving, 3-23

Delete Row pushbutton in Include dialog, 5-11  
Deleting Folders, 3-28  
Deleting hierarchies, 9-4  
    Consequences, 9-4  
Deleting objects, 3-6, 4-8  
    consequences, 4-8  
Dialog boxes  
    description, 3-11  
    message type, 3-11  
    setting type, 3-11  
    transaction type, 3-11  
Differences between copying and duplicating  
    objects, 4-7  
Displaying Operability configuration settings,  
    15-14  
Duplicating  
    Excluded Test Cases, 10-25  
    Format object, 6-15  
        Creating a new object, 6-16  
        Creating a new version, 6-16  
    Included Test Cases, 10-17  
    Relation object  
        Duplicated/nonduplicated attributes,  
            9-16  
        Excludes, 10-25  
        Includes, 10-17  
    Test object, 9-6  
        Duplicated/nonduplicated  
        Attributes, 9-7

## E

Edit menu selections description, 3-6  
Editing Person Object Attributes, 17-2  
Environment setting for GUI, 2-3  
/etc/services, Changes, 13-7  
Excluded Test Cases, 10-20  
    Adding, 10-22  
    Deleting, 10-26  
    Duplicating, 10-25  
    Example constraints, 10-21  
    Modifying values, 10-24  
    Saving, 10-24  
    Setting values, 10-20  
    Specifying values, 10-20

Exiting the AETG System, 2-40, 3-5, 3-34

## F

### Field object

- Copying, 6-7
  - Copied attributes, 6-8
- Creating, 6-6
- Creating on a Format object, 7-1
- Deleting, 6-8
- Logical name, 7-2
- Name, specifying, 7-2
- Parameters
  - Entering a list of, 7-3
  - Specifying, 7-9
- Properties view, 7-10
- Rearranging in a Format object, 6-12
- Sequence numbers, 7-3
- Specifying a name, 7-2
- Values
  - Blanks in values, 7-4
  - Changing, 7-7
  - Copying, 7-8
  - Creating, 7-5
  - Deleting, 7-9
  - Entering, 7-3
  - Specifying valid values, 7-3
- Type
  - Examples, 7-4
  - Literal, 7-3
  - Non-Literal, 7-3
- Values view, 7-2
- Version number, 6-3
- Why use, 7-1

### Fields

- Listing all in a compound, 8-4
- Logical name, 7-2
- Name, 7-2
- Parameters, 7-3
- Sequence numbers, 7-3
- Valid values, 7-3

File Archive Installation, 13-12, 13-15

### Folders

- Analogy to filing cabinet, 3-1
- Changing name of, 3-28
- Client area description, 3-9
- Creating new folders, 3-27

- Deleting, 3-28
- general description, 3-1
- List View layout and screen objects, 3-3
- moving object in and out, 3-2
- naming, 3-27
- Saving, 3-5, 3-28
- Status area, 3-9
- title bar description, 3-3

### Fonts, 3-22

- Selecting default, 3-25

### Format object

- Associating a Test object, 9-11, 9-12
- Associations view, 6-14
- Cloning, 6-16
- Copying a Field object, 6-7
  - Copied attributes, 6-8
- Copying Field objects, 6-7
- Creating, 6-1
- Creating a Field object, 6-6
- Creating Compound objects, 6-9
- Defining an interface, 6-1
  - Specifying an interface type, 6-3
- Deleting a Field object, 6-8
- Deleting Compound objects, 6-12
- Deleting Field objects, 6-8
- Duplicating, 6-15
  - Creating a new Format object, 6-16
  - Creating a new version of a Format object, 6-16
- Entering a description, 6-3

### Fields

- Compound name, 6-5
  - Field name, 6-5
  - Logical name, 6-5
  - Sequence numbers, 6-5, 6-12
- Fields view, 6-4
- Interfaces, types, 6-1
- Name, specifying, 6-3
- Object version number, 6-3
- Owner, specifying, 6-3
- Properties view, 6-2
- Rearranging Field objects, 6-12
- Specifying a name, 6-3
- Status display, 6-3
- Title format, 6-4
- Why use, 6-1
-



## G

- General definition, 9-1
- Granting administrative privileges, 17-6
- GUI
  - Controls
    - Pushbuttons, 3-15
    - Radio Buttons, 3-15
    - Spin buttons, 3-14
    - Toggle Buttons, 3-14
  - Creating a Person Object, 17-2
  - Default Setting, unlock new objects, 3-23
  - Environment setting, 2-3
  - Exiting, 2-40, 3-5, 3-34
  - Fonts, 3-22
    - Selecting default, 3-25
  - Person Object, 17-1 to 17-8
  - Removing objects, 4-8
  - Rulers, 3-16
  - Starting, 2-3
- GUIs, Modeling, 12-1

## H

- Hardware Requirements, 13-1
- Help menu selection, description, 3-7
- Hierarchies, 9-1
  - Child object definition, 9-1
  - Creating and naming, 9-2
  - Creating Test objects, 9-4
  - Deleting, 9-4
    - Consequences, 9-4
  - Naming, 9-4
  - Parent object description, 9-1
  - Test Hierarchy description, 9-1
  - Uses, 9-1

## I

- Icons
  - Minimizing and maximizing, 3-12
  - Tool Bar icons, 3-8
  - Using, 3-12
- Inactive Status, 17-3
- Include dialog

- Adding rows, 5-8
- Attribute/values conditions, 5-7
- Conditions between attributes and values, 5-7
- Deleting rows, 5-11
- Relations between rows, 5-7
- See also* Database Browser, 5-5
- specifying attributes, 5-8
- Using, 5-5
- Included Test Cases, 10-12
  - Adding, 10-13
  - Deleting, 10-18
  - Duplicating, 10-17
  - Modifying, 10-16
  - Saving values, 10-15
  - Setting values, 10-12
  - Specifying values, 10-12
- Installation
  - AETG System, 13-11 to 13-13
  - Assumptions and Recommendations, 13-3
  - BAIST considerations, 13-8
  - Changes to /etc/services, 13-7
  - Configuring the AETG System, 14-1
  - Environment Settings, 13-4
  - Installing License Keys, 13-32
  - Installing the AETG Software, 13-11
  - madmin logid, 13-7
  - mynah Group ID, 13-7
  - Obtaining License Keys, 13-6
  - Oracle, 13-19 to 13-31
  - Recommended Directory Structure, 13-3
  - Steps, 13-1
  - Telexel, 13-14 to 13-18
- Interaction degree
  - Relations, 2-30, 10-4
  - Tuples, 8-4
    - Cross Product, 8-6
    - Selecting, 8-4, 8-6
    - Unrelated, 8-6
- Invalid Test Cases, 10-28
  - Adding, 10-28
  - Deleting, 10-31
  - Modifying, 10-31
  - Saving values, 10-30
  - Specifying values, 10-30

## K

- ksh
  - Setting BAIST Installation Directory, 13-8
  - Use of xmyProfile File, 13-5

## L

- License Keys
  - Installing, 13-32
  - Obtaining, 13-6
- License Server, 13-6
- Licenses
  - Installing Keys, 13-32
  - License Server, 13-6
  - Obtaining License Keys, 13-6
- Licensing, 15-5
  - Decoding licensing codes, 15-7
  - Monitoring installed licenses, 15-6
  - Starting the license server, 15-5
  - Stopping the license server, 15-5
- List view, expanding object listing, 3-19

## M

- Menu Bar description, 3-4
- Menus
  - Accelerator keys, 3-13
    - list and functions, 3-13
  - AETG menu, 3-5
  - Bar description, 3-4
  - Edit menu, 3-6
  - Help menu, 3-7
  - making selections, 3-12
  - Mnemonic keys, 3-12
  - Selected menu, 3-6
  - System, 3-4
  - Tools Menu, 3-7
  - View menu, 3-6
  - Window menu, 3-7
- Mnemonic keys
  - description and use, 3-12
- Modeling
  - Command Line Interface, 12-3

- Configurations, 12-5
- GUIs, 12-1

- Monitoring installed licenses, 15-6
- Mouse button actions, 3-11
- Multiselecting Objects, 4-10

## N

- Naming
  - Folders, 2-5
  - Hierarchies, 9-4

## O

- OA, 15-1
  - Configuration Parameter, 14-6
  - Definition, 15-3
  - Stopping an OA, 15-12
    - On a specified host, 15-12
    - On the local host, 15-12
  - Stopping and restarting a host, 15-13
  - xmyConfigOP Entry, 14-5
- Object status and default view, 4-9
- Objects
  - changing information in your person object, 5-14
  - Clearing, 4-8
  - Clearing from desktop, 3-6
  - Copying, 3-6
  - Creating at first level of hierarchy, 9-4
  - Creating at lower levels of a hierarchy, 9-5
  - Creating new, 3-6
  - Cutting, 3-6, 4-8
  - Deleting, 3-6, 4-8
    - consequences, 4-8
  - Description, 4-1
  - Deselecting all on desktop, 3-6
  - Expanding, 3-6
  - General description, 4-2
  - Individual definitions, 4-1
  - managing objects include dialog, 5-5
  - Modifying another person's objects, 4-9
  - Multiselecting, 4-10
  - New objects, unlocking, 3-23
  - Opening, 3-6

- Opening in Database Browser, 5-11
  - Pasting, 3-6
  - Person object
    - Viewing, 5-12
  - Person object description, 5-12
  - Relationship to views, 4-2
  - Removing from GUI, 4-8
  - See Also* Include Dialog
  - Selecting all on desktop, 3-6
  - Sorting in Database Browser, 5-3
  - Viewing
    - Viewing Attribute/values conditions, 5-7
    - Viewing, *See Also* Include dialog
  - Objects duplicating objects (clones), 4-7
  - Obtaining the status of processes, 15-13
  - OM, 15-1
    - Definition, 15-3
    - OM Port number, 15-3
    - Port Number, Specifying, 14-2
    - xmyOM Subcommands, 15-4
  - Operability Agent
    - See* OA
  - Operability Management, 15-1 to 15-18
    - Autostart Processes, 14-5, 15-1, 15-3
      - Starting, 15-10
      - Stopping, 15-11
    - Basic Steps, 15-1
    - Design, 15-2
    - Displaying Operability configuration settings, 15-14
    - Licensing, 15-5
      - Decoding licensing codes, 15-7
      - Monitoring installed licenses, 15-6
      - Starting the license server, 15-5
      - Stopping the license server, 15-5
  - OA, 15-1
  - Obtaining the status of processes, 15-13
  - OM, 15-1
  - Overview, 15-2
  - See Also* OA
  - See Also* OM
  - Specifying if Process is a AETG Process, 14-4
  - Starting processes, 15-10
  - Stopping processes
    - Stopping a specific host, 15-11
    - Stopping an Autostart process, 15-11
    - Stopping an OA, 15-12
    - Stopping an OA and Autostart Processes
      - On a specified host, 15-12
      - On the local host, 15-12
    - Stopping and restarting a host, 15-13
  - xmyOM Subcommands, 15-4
    - autostart, 15-10
    - autostop, 15-11
    - query, 15-14
    - recycle, 15-13
    - shutdown, 15-12
    - start, 15-10
    - status, 15-13
    - stop, 15-11
  - Operability Manager
    - See* OM
  - Operating System Requirements, 13-1
  - Operating System, Solaris, 13-1
  - Options, 14-1
  - Oracle
    - Configuring AETG Database, 13-26
    - Database Disk Configuration, 13-21
    - Environment Variables, 13-20
    - Installation
      - Error Messages, 13-25
      - Installing the Software, 13-23
      - Verifying, 13-25
    - Pre-Installation Steps, 13-19
    - Required Packages, 13-20
    - Required Processes, 13-30
    - Verifying, 13-30
- P**
- Parameters, 14-1
  - Person Objects, 17-1 to 17-9
    - Attributes, 5-13
    - Creating, 17-1
      - From the GUI, 17-2
    - Description, 5-12
    - Duplicate IDs, 17-5
    - Editing
      - Attributes, 17-2, 17-6
      - Authority Level, 17-6
    - Granting Administrative Privileges, 17-3

- Granting and removing administrative privileges, 17-6
  - Information View, 17-6
  - Information view, 5-15
  - Limits per UNIX ID, 17-1
  - Properties View, 17-3
  - Properties view, 5-13
  - Removing Administrative Privileges, 17-4
  - Restrictions on Users, 17-1
  - Saving, 17-4
  - Viewing, 5-12
  - Port Numbers, 14-2
  - Post-Installation Steps
    - AETG System, 13-13
    - Telexel, 13-16
  - Pre-Installation Steps
    - AETG System, 13-11
    - Oracle, 13-19
    - Telexel, 13-14
  - Printer, default, 3-22
  - Printing
    - General description for AETG System, 3-30
    - Test Case Matrix, 9-27
  - Privileges, Specifying, 17-3
  - Processes
    - Automatically Starting, 14-4
    - Responsibility List, 14-6
    - Specifying if a AETG Process, 14-4
    - Starting, 14-4
      - Commands, 14-5
    - Status of, Obtaining, 14-4
      - Commands, 14-5
    - Stopping, 14-4
      - Autostart, 15-11
      - Commands, 14-5
      - On local host, 15-12
      - On specific host, 15-12
      - Specific processes, 15-11
  - Pushbuttons
    - Description and use, 3-15
    - Functions, 3-15
- R**
- Radio button, description, 3-15
  - Rearranging Field objects, 6-12
  - Registering as an AETG user, 2-3
  - Relation object
    - Changing participating fields, 10-7
    - Changing participating values, 10-9
    - Cloning, 9-14
    - Creating, 10-1
    - Description, entering, 10-4
    - Duplicating, 9-14
      - Duplicated/nonduplicated attributes, 9-16
  - Excluded Test Cases, 10-20
    - Adding, 10-22
    - Deleting, 10-26
    - Duplicating, 10-25
    - Example constraints, 10-21
    - Modifying values, 10-24
    - Saving, 10-24
    - Setting values, 10-20
    - Specifying values, 10-20
  - Excludes view, 10-19
  - Fields
    - Changing participating fields, 10-7
    - Rearranging, 10-10
  - Fields view, 10-5
  - Fields, listing nonparticipating, 10-6
  - Fields, listing participating, 10-6
  - Included Test Cases, 10-12
    - Adding, 10-13
    - Deleting, 10-18
    - Duplicating, 10-17
    - Modifying, 10-16
    - Saving values, 10-15
    - Setting values, 10-12
    - Specifying values, 10-12
  - Includes view, 10-11
  - Interaction degree, 10-4
  - Invalid Test Cases, 10-28
    - Adding, 10-28
    - Deleting, 10-31
    - Modifying, 10-31
    - Saving values, 10-30
    - Specifying values, 10-30
  - Invalids view, 10-27
  - Listing nonparticipating fields, 10-6
  - Listing participating fields, 10-6
  - Name, specifying, 10-3

Properties view, 10-3  
Rearranging Fields, 10-10  
*See also* Test object  
Specifying a name, 10-3  
Test Cases  
    Excluding, 10-19  
    Including, 10-11  
    Invalids, 10-27  
Values  
    Changing participating values, 10-9  
    Modifying Excludes, 10-24  
    Setting  
        Excluded Test Cases, 10-20  
        Included Test Cases, 10-12  
    Specifying for Excluded Test Cases,  
        10-20  
    Specifying for Included Test Cases,  
        10-12  
    Specifying for Invalid Test Cases,  
        10-30  
    Why use, 10-1  
Removing administrative privileges, 17-6  
Removing local pip files, 15-9  
Required Software Packages, 13-2  
Requirements  
    Hardware, 13-1  
    Operating System, 13-1  
    Required Software Packages, 13-2  
Ruler column headings  
    in displays, 3-16  
    using, 3-16

## S

Saving Default setting on Preference view,  
    3-24  
Saving Folders, 3-5, 3-28  
Selecting default fonts, 3-25  
Set button, 10-14, 10-23  
Setting Administrative Privileges  
    Using the GUI, 17-1 to 17-8  
sh, Setting BAIST Installation Directory, 13-8  
Simple Field, 6-5  
Software Packages  
    Required, 13-2

Solaris, 1-1, 13-1  
    Delivered Oracle Start-up Files, 15-16  
    Delivered Start-up Files, 15-16  
    Oracle Packages, 13-20  
    Start-up Mechanism, 15-8  
Spin button, description and use, 3-14  
Starting and Stopping AETG Processes  
    See Operability Management  
Starting processes, 15-10  
Starting the AETG System, 1-1, 2-3  
Starting the license server, 15-5  
Stopping an OA, 15-12  
    On a specified host, 15-12  
    On the local host, 15-12  
Stopping and restarting a host, 15-13  
Stopping processes  
    Stopping a specific host, 15-11  
    Stopping an Autostart process, 15-11  
    Stopping an OA, 15-12  
    Stopping an OA and Autostart Processes  
        On a specified host, 15-12  
        On the local host, 15-12  
    Stopping and restarting a host, 15-13  
Stopping the license server, 15-5  
System Overview, 1-1  
System Requirements, 13-1

## T

Telexel  
    Configuring, 13-16  
    Installation, 13-14 to 13-18  
        CD-ROM, 13-15  
        File Archive, 13-15  
        Post-Installation Steps, 13-16  
        Pre-Installation Steps, 13-14  
    Verifying, 13-17  
Test Case Matrix  
    Deleting, 9-20, 9-25  
    Displayed values, 9-18  
File  
    Deleting, 9-25  
    Format, 9-23, 9-28  
        Converting, 9-29  
    Loading, 9-24

- Location, 9-23
- Printing, 9-27
- Saving, 9-23
- Generating, 9-17, 9-19
- Loading, 9-24
- Matrix information, 9-18
- Name, format, 9-18
- Printing, 9-27
- Saving, 9-22
- Specifying maximum number, 9-25
- Type, 9-18
- Test object
  - Area test types, 9-10
  - Associating a Format object, 9-11, 9-12
  - Cloning, 9-6
  - Creating a new Relation object, 9-13
  - Creating at first level of hierarchy, 9-4
  - Creating at lower levels of a hierarchy, 9-5
  - Creating in a hierarchy, 9-4
  - Description, entering, 9-10
  - Displaying Relation objects, 9-11
  - Duplicating, 9-6
    - Duplicated/nonduplicated Attributes, 9-7
  - Duplicating a Relation object, 9-14
    - Duplicated/nonduplicated attributes, 9-16
  - Featuretest types, 9-10
  - Matrix view, 9-17
  - Name, specifying, 9-9
  - Naming, 9-4
  - Owner, specifying, 9-9
  - Parent type, 9-10
  - Properties view, 9-9
  - Relation
    - Deleting, 9-16
  - Relation object, 9-11
    - Creating a new Relation object, 9-13
    - Duplicating, 9-14
      - Duplicated/nonduplicated attributes, 9-16
  - Relations view, 9-11
  - See also* Hierarchies
  - See also* Relation object
  - See also* Test Case Matrix
  - Specifying a name, 9-9
  - Specifying an owner, 9-9
- Status, displaying, 9-10
- Test Case Matrix
  - Deleting, 9-20, 9-25
  - Displayed values, 9-18
  - File
    - Deleting, 9-25
    - Format, 9-23, 9-28
      - Converting, 9-29
    - Loading, 9-24
    - Location, 9-23
    - Printing, 9-27
    - Saving, 9-23
  - Generating, 9-17, 9-19
  - Loading, 9-24
  - Matrix information, 9-18
  - Name, format, 9-18
  - Printing, 9-27
  - Saving, 9-22
  - Specifying maximum number, 9-25
  - Type, 9-18
- Test Case test types, 9-10
- Test test types, 9-10
- Types
  - Displaying Parent object type, 9-10
  - Specifying, 9-10
- Version number, 9-10
- Why use, 9-1
- Tests
  - Defining test properties, 9-9
- Toggle buttons, description and use, 3-14
- Tool Bar
  - description, 3-7
  - icons, description of function, 3-8
- Tools Menu, 3-7
  - Database Browser, 3-7, 5-2
- Tuples, 6-11
  - Adding up to Maximum number, 8-12
  - Count, 8-7
  - Creating on a Format object, 8-1
  - Definition, 8-1
  - Deleting, 8-14
  - Generate Tuples menu option, 8-9, 8-10
  - Generating, 8-8
    - At one time, 8-9
    - Iteratively, 8-10
  - Generating before saving a Compound object, 8-14

Interaction Degree  
    Selecting, 8-6  
    Unrelated, 8-6  
Interaction degree, 8-4  
    Tuples  
        Cross Product, 8-6  
        Selecting, 8-4  
Listing, 8-7  
Names, 8-7  
    Changing a Tuples name, 8-12  
    Regenerating, 8-13  
    System generated, 8-12  
Setting Maximum number, 8-3

## U

Using Include Dialog Add Row pushbutton,  
    5-8  
Using Include Dialog Delete Row pushbutton,  
    5-11  
Using the Include Dialog, 5-5

## V

Values  
    Blanks in values, 7-4  
    Changing, 7-7  
    Clearing in a Compound, 8-5  
    Copying, 7-8  
    Creating, 7-5  
    Deleting, 7-9  
    Entering, 7-3  
    Selecting for a Compound, 8-5  
    Specifying valid values, 7-3  
    Type  
        Examples, 7-4  
        Literal, 7-3  
        Non-Literal, 7-3  
Verifying  
    Oracle, 13-25  
    Telexel, 13-17  
Views  
    AETG Desktop Preference view, 3-22  
    Changing, 3-6  
    changing, 3-26  
    definition, 3-18

description, 3-18  
List  
    description, 3-19  
    expanding object listing, 3-19  
Person object  
    Information, 5-15  
    Properties, 5-13  
Preferences view  
    AETG Desktop, 3-22  
    Edited, 3-24  
Refreshing  
    Database Browser, 3-7  
    desktop, 3-6  
    Folders and objects, 3-6  
Test Properties, 9-9

## W

Window menu selections, description, 3-7  
Window scroll bars, 3-9

## X

X-windows, 1-1  
xmyConfig.General File  
    Configuration Parameters  
        General Entries  
            Database, 14-2  
            DefaultSD, 14-1  
            WelcomeNewUsers, 14-2  
    Options, 14-1  
    Parameters, 14-1  
xmyConfigOP File  
    Configuration Parameters  
        AutoStart, 14-4  
        Mynah, 14-4  
        Responsibility, 14-6  
        Start, 14-4  
        Status, 14-4  
        Stop, 14-4  
    Example, 14-7  
    Syntax, 14-3  
xmyConMatrix, 9-29  
\$XMYDIR, 13-5, 13-12  
\$XMYHOME, 13-5, 13-12  
xmyLogin, 13-4

xmyMYNAHrc file, 3-5  
xmyProfile, 13-4  
xmyRunAetg, 1-1, 2-3  
xmyShutDown, 15-12  
xmyStartUp, 15-9