MYNAH™ System Users Guide

MYNAH™ 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.

Bellcore
Bell Communications Research

Document Feedback

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.	Issue No.	Publication Date	Revision No.	Supplement No.
BR 007-252-006	Issue 3	October 1997		

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

	Missed	Nearly Met	Met	Exceeded	Not Applicable
a. Relevance of the information to your work	☐	☐	☐	☐	○
b. Ease of finding the information that you need	☐	☐	☐	☐	○
c. Clarity of the information	☐	☐	☐	☐	○
d. Accuracy of the information	☐	☐	☐	☐	○
e. Usefulness of the information	☐	☐	☐	☐	○
f. Thoroughness of the information	☐	☐	☐	☐	○
g. Level of detail of the information	☐	☐	☐	☐	○
h. Availability of this document when you needed it	☐	☐	☐	☐	○
i. Overall quality of this document	☐	☐	☐	☐	○

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.

Trademark Acknowledgments

Adobe, Acrobat, 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

MYNAH and AETG are trademarks of Bell Communications Research

SunOs, Solaris, SPARCstation, and NFS are trademarks are Sun Microsystems, Inc.

QA Partner is a trademark of Segue Software, Inc.

QC/Replay is a trademark of CenterLine Software, Inc.

UNIX is registered trademark of Novell, Inc.

XRunner is a registered trademark of Mercury Interactive Corporation.

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

MYNAH System Users Guide

Contents

List of Figures

List of Tables

1. Introduction to the MYNAH System

The MYNAH™ System is designed to support business flow testing and task-automation. Business flow testing means that the MYNAH System supports complete end-to-end testing of all your enterprise computing resources. Task-automation means non-test script development for general purposes. The MYNAH System supports a wide range of non-test scripting activities.

The MYNAH System operates in a Motif X-Windows™ environment that runs Solaris™ 2.5.1 or HP-UX™ 10.10. Any display device that uses the X-windows environments can be use to display the MYNAH System.

You perform testing and general scripting through an easy-to-use Graphical User Interface (GUI). The MYNAH System uses a single GUI which complies with Bellcore's standard for GUIs.

The MYNAH System provides a single scripting language called the Tool Command Language (Tcl - pronounced "tickle"). Tcl supports task-automation and testing that requires scripting. The MYNAH System provides Tcl extensions, specially developed Tcl commands, to help you with your testing and task-automation. You can use Tcl through the GUI, from the system's Command Line User Interface (CLUI) or through the MYNAH Tcl shell (**xmytclsh**). We describe the GUI and the CLUI in this manual. For more information about the MYNAH Tcl language, see the *MYNAH System Scripting Guide*.

This section of the MYNAH User's Guide offers a description of:

- Test and non-test uses of the MYNAH System
- System components
- Test environments, called packages in the MYNAH System
- A road map through the remainder of this manual
- Conventions used in the manual.

1.1 MYNAH System Test Uses

The MYNAH System provides a complete test environment through which you can:

- Write test specifications
- Represent requirement.
- Keep track of issue
- Report test metrics
- Record execution of manual tests

- Create test scripts using Tcl that automate interactions with a System-Under-Test (SUT)

- Edit and execute test scripts

- Represent user-defined test cases.

1.2   MYNAH Non-Test Uses

In addition to the test operations, you can use the MYNAH System's scripting capability for some non-test operations including:

- Automate simple repetitive tasks, for example, data entry

- Access data from legacy systems automatically

- Perform data integrity checks on remote systems

- Rapid application development.

1.3   The AETG System

The Automatic Efficient Testcase Generator (AETG™) System is a testing tool that is used to generate test cases, and can be licensed with the MYNAH System or independently of the MYNAH System as a stand-alone product. In addition, the MYNAH System can run independent of the AETG System.

For information on how to use the AETG System, see BR 007-252-007, *AETG System Users Guide*.

1.4   MYNAH Components

The MYNAH System is an integrated system made up of components that control all test and non-test activities. The sub-sections that follow describe the components that make up the MYNAH System.

1.4.1   Graphical User Interface (GUI)

The MYNAH User Interface is a Motif-compliant Graphical User Interface (GUI) that uses a simple point-and-click method for most operations. You can access all the features of the MYNAH System from the user interface. The MYNAH GUI provides you with a consistent set of windows, views, dialogs and controls.

1.4.2   Command Line User Interface (CLUI)

The Command Line User Interface (CLUI) is a set of commands you can use to perform operations from the UNIX™ command line. The CLUI was designed for experienced users who prefer to use the UNIX command line rather than the GUI.

1.4.3   Background Execution Environment (BEE)

The Background Execution Environment (BEE) allows you to submit a number of scripts for simultaneous processing. It consists of Script Dispatchers and background Script Engines and provides a complete MYNAH script execution environment that can be accessed from a UNIX command line via the CLUI or through the GUI.

1.4.4   Script Engine (SE)

A Script Engine is a process that executes scripts. (The GUI contains an interactive script engine with which users can execute scripts through the GUI).

In addition, there can be a number of script engines available through the Background Execution Environment (BEE) which you can access through the CLUI or GUI. The number of Script Engines available can be set by your system administrator and is only limited by the platform the MYNAH System uses. When you run Script Engines in the BEE, they are run as background processes. This allows you to use the multi-processing capability of the UNIX system.

1.4.5   Script Dispatcher (SD)

A Script Dispatcher is a process that manages the way scripts are executed in the background. The SD receives script execution requests from the GUI or CLUI and dispatches them to a Script Engine in the BEE. It also manages concurrently running scripts in compliance with concurrency settings.

There can be a number of Script Dispatchers in the BEE. Script Dispatchers in the BEE provide you with the capability of submitting a large number of scripts to the background.

1.4.6   Extension Packages and Domains

Extension packages and Domains are extensions to the basic Tcl language that allow you to automate interactions on interfaces to SUTs. The following packages are provided with the MYNAH System.

1.4.6.1   3270 Terminal Emulation

This Synchronous terminal extension package emulates IBM ™ 3270 terminals. This includes support for the terminal's color capabilities, operator information area as well as all keys, ASCII and EBCDIC. This domain can operate with TCP/IP, Bisync, or SNA protocols.

1.4.6.2   Synchronous Printer Emulation

The Synchronous Printer extension package provides IBM 3278 printer emulation which allows you to capture printer messages sent from the SUT. This printer emulation operates with the TCP/IP, Bisync, or SNA protocol.

1.4.6.3   Asynchronous Terminal Emulation

The Asynchronous extension package emulates all the functionality of VT100 asynchronous terminals.

1.4.6.4   Application-To-Application

The Application-To-Application provides emulation for Application-to-Application interfaces. This domain uses the TOP/TCP/IP protocol. You can send messages to and receive them from a SUT using this package.

1.4.6.5   General Application-To-Application

Provides a general purpose interface collector mechanism which will allow customers to write Tcl scripts to control the testing of other types of App-to-App interfaces using MYNAH. Included in these enhancements are changes to support a Non-Broadcast (connection-based) interface to the SUT as well as enhancements to the **xmyMsgDir** capability for manipulating messages in the message directory.

1.4.6.6   DCE (Client/Server)

The Client/Server Domain provides for interaction between MYNAH and a SUT across a DCE (Distributed Computing Environment) /IDL interface. MYNAH supports all data types compatible with this interface.

1.4.6.7   Batch

The Batch Domain provides an interface to mainframe applications for batch processing. It can submit JCL jobs, determine when they are completed, and determine completion status codes for each step in a JCL job.

1.4.6.8   X-Windows Domain

The X-Windows Domain uses a third party vendor tool for automated interaction with a SUT that uses X-Windows.

1.4.6.9   PC Windows Domain

The PC Windows Domain uses a third party vendor tool for automated interaction with a SUT that uses Microsoft Windows GUI.

1.5   Example Used in This Manual

Throughout the manual we will refer to an example application under test to illustrate using MYNAH features. The example application is **crolo** an automated rolodex system which you can access from a UNIX command line.

This application was supplied to you with the MYNAH System and we entered a number of names and addresses into it. You start it by typing **crolo** at the UNIX prompt.

We created a complete test plan for this application and create the scripts to implement the test plan using the MYNAH System.

1.6   What's Next

The next four sections will help familiarize you with the MYNAH System. These sections are:

- *Section 2: A Quick Tour Through The MYNAH System* is a simple exercise in creating a folder and copying some items into it, connecting to a SUT, recording actions across the interface and including them in a script. You will then duplicate a test, identify the script you created with the test, and finally run the test.

- *Section 3: MYNAH Basics* explains the windows, views, dialogs, and controls used in the MYNAH GUI.

- *Section 4: Using MYNAH Objects* describes how MYNAH objects, which represent MYNAH features, are created, defined, edited and deleted.

- *Section 5: Using the Database Browser* describes using the Database Browser to find object.

The remaining sections of the Users Guide will provide you with detailed instructions on developing tests and scripts; executing them on a SUT; and analyzing and reporting the results of testing. You will also find information on task-automation, i.e., general scripting for non-test purposes. And finally, you will find detailed information on the CLUI.

- *Section 6: Using Requirement Objects* explains how to use MYNAH Requirement objects to document requirements you are trying to test.

- *Section 7: Defining a System-Under-Test* explains how to specify a SUT for testing.

- *Section 8: Developing Tests* provides step-by-step instructions for developing tests. This includes specifying a Test object and associating it with other objects such as Script objects.

- *Section 9: Using Issues* describes how to use the MYNAH Systems's issue object to document any issues that develop while testing.

- *Section 10: Using Script Objects* explains how to create scripts through the GUI and document them with a Script object.

- *Section 11: Using Keywords* explains how you can use Keyword objects to characterize other testing objects.

- *Section 12: Developing Script Code with the Script Builder Feature* explains how to develop script code, make connections to the SUT, record events across the SUT interface, incorporate events into scripts, and execute scripts directly from the Script Builder.

- *Section 13: Format Object* explains the AETG System Format object.

- *Section 14: Examining Results and Reporting on Test Activities* explains what results objects are and how to use them; and describes the reports available with the MYNAH System and explains how to generate them.

- *Section 15: Task Automation with the MYNAH System* explains how use the Task-Automation Builder to develop scripts that automate routine tasks.

- *Section 16: Understanding the Background Execution Environment (BEE)* explains what the Background Execution Environment is and how it can help you.

- *Section 17: Using the Command Line Interface (CLUI)* explains the CLUI and how the CLUI works with the BEE. This section also describes the commands that comprise the CLUI along with their syntax and options.

1.6.1   Related Documents

There are two other documents available with the MYNAH System that may be of interest to you these are:

- For information about using the Tcl scripting language and the MYNAH extensions to this language, refer to the *MYNAH System Scripting Guide, BR 007-252-008.*

- System administrators will find the information they need to install, maintain, and manage the MYNAH System in the *MYNAH System Administration Guide, BR 007-252-005.*

1.7   Conventions Used in This Manual

We use fonts in a specific way in this manual. Table 1-1 explains this.

Table 1-1. Fonts Used in the Manual

Use	Type	Example
Section and manual references	Italic	*Section 1: Introduction the MYNAH System*, *Section 1.6 Conventions Used in This Manual*
Menu and menu options	bold, sans-serif type.	**MYNAH** menu
Screen labels	bold	**Name** field.
Data you enter	bold, constant width font	`xmyRunMynah`

1.7.1   How Actions Are Represented

We provide step-by-step instructions for many of the tasks you will perform with the MYNAH System. In these procedures major steps are numbered. For example:

1. Click on the **MYNAH** menu and then on the **New** selection.

2. Position the cursor in the **Name** data entry field and type in a name for the test.

We will indicate sub-steps with lettered steps. For example:

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

    A. Double clicking on it.

    B. Clicking on the **Selected** menu and then on the **Open** option.

Finally, we will use a dash to indicate a system response to your action. For example:

1. Click on the **MYNAH** menu and then on the **New** selection.

   — The MYNAH System will display the Properties view.

1.7.2   How We Represent Things in the MYNAH System

Throughout this manual you will encounter the term **objects**. This concept provides us with a useful way to represent a number of things in the system. Objects are tests, scripts, or the results of tests, among other things. The MYNAH System uses objects in the GUI and stores them in the MYNAH database. We will describe objects in detail and explain how to use them in *Section 4: Using Objects*. For now, remember that objects are a key concept in the MYNAH System.

1.8   For More Help

You can get support from the MYNAH Customer Service Center Monday through Friday between 8AM and 7PM EST by calling (732) 699-2668, (option 3). If you are outside the 732 area code or are calling outside the United States, dial (800) 795-3119 (option 3).

Outside of standard support hours and on holidays we provide critical problem coverage. If you call the Hotline (800-975-3119), you will be prompted on how to get help for severity one problems. You can also contact support with e-mail at *mynah-support@cc.bellcore.com*.

1.9   On-line Versions of the MYNAH Documents

The MYNAH documents are available on-line in the Adobe® Acrobat® PDF format (Release 3.0). The PDF files are accessible from either the local file system or from an internal URL. See the MYNAH administrator for the location of the PDF files.

Viewing a PDF file requires that you have installed the Adobe Acrobat Reader®. If you need the Acrobat Reader, contact the MYNAH administrator. In addition, you can download the Acrobat Reader directly (off the Internet) from Adobe at *www.adobe.com*.

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

- Using the Acrobat Reader directly if the MYNAH System has been installed on a local system.

- Using the Acrobat Reader as plug-in to a browser if the MYNAH 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 a 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.

2. A Quick Tour Through the MYNAH System

Before we explain the system in detail we want you to do some simple tasks with the MYNAH System so that you can become familiar with it. We won't provide much detail about why we are doing something -- later we will. This is a kind of "jump start" in which we will get you working with the system so that you will see how easy it is to use.

You can follow along with the activities and perform the steps we describe. Throughout this section we use examples based on a sample application delivered with the system. The name of the application is **Crolo,** which provides an on-line Rolodex card file.

We will do three things:

1. Create a folder on the MYNAH desktop and import some demo objects into it.

2. Create a script, document it with a Script object and identify the test that this script implements.

3. Run the test.

2.1 Some Terms We Will Use

The following are basic mouse terminology when we say:

- "Click" in the procedures that follow, we mean that you should place the mouse pointer on an item we identify in a MYNAH window and press down and release the left mouse button quickly.

- "Double click" is to press down and release the left mouse button twice in rapid succession.

- "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.

- When we refer to the "pointer", we mean the arrow that indicates the mouse position.

2.2   Starting The MYNAH System

To begin using the MYNAH System GUI you need to start it up. Remember, before you start the MYNAH System, your path must be set properly to contain *$XMYDIR/bin*. Also, your XMYHOME environment variable must be set. If you need to, refer to the *MYNAH System Administration Guide*, or call your system administrator for more information about doing this.

To start the MYNAH System, type

```
xmyRunMynah
```

at the UNIX prompt. The MYNAH GUI will run as a background process.

After you start the MYNAH GUI, it will display a series of status messages as it goes through its initialization routine and then display the MYNAH Banner window.

The MYNAH banner window will appear briefly and then the system will display the MYNAH Welcome New User dialog (Figure 2-1).

2.2.1   Registering as a MYNAH User

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

Figure 2-1. New User Dialog

The system is case-insensitive to information you enter here. This means 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.**

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 a MYNAH user.

The system will now display the MYNAH Desktop (Figure 2-2). This is the first of many MYNAH System windows, views and dialog boxes. We will describe the MYNAH Desktop in *Section 3: MYNAH Basics.*

Notice that there are no objects on the MYNAH Desktop. We will create a folder and copy objects into it that we will use as examples for our *Quick Tour.*
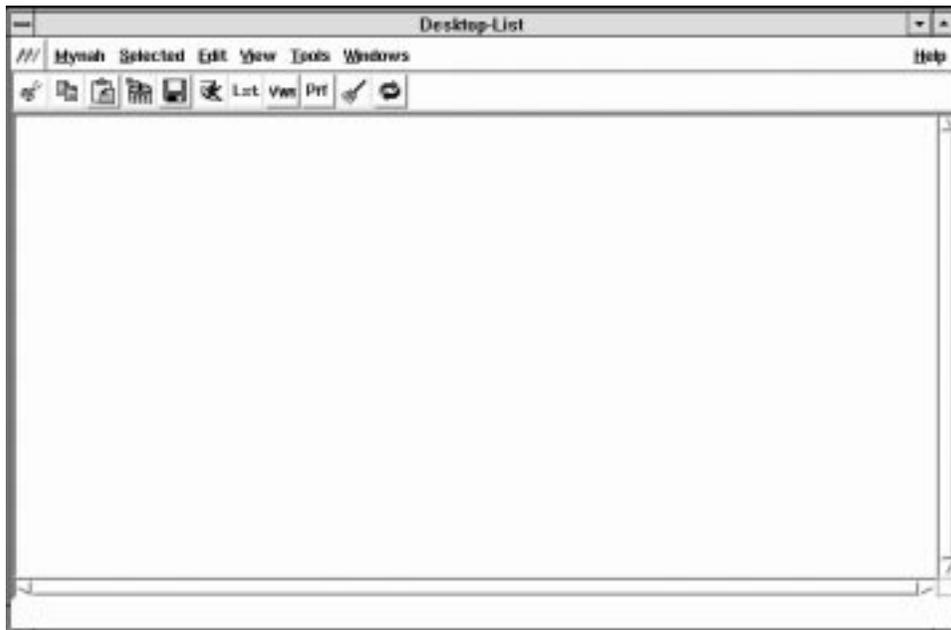
Figure 2-2. MYNAH Desktop

2.3 Creating a Folder

The MYNAH System provides you with a convenient tool with which to organize your work -- folders. Folders are objects in which you can place other objects. You can think of them as you would paper folders in which you place a number of documents.

The MYNAH System allow two menu options from the **Menu Bar** for creating a New Objects. These menu selections will be explained in more detail in *Chapter 3: MYNAH Basics.*

To create a folder:

1. Click on the **Mynah** menu.

   — The system displays a drop-down menu.

2. Click on the **New** menu selection.

3. Click on the **Folder** option.

   — A dialog box requesting a name for the folder appears. See Figure 2-3.
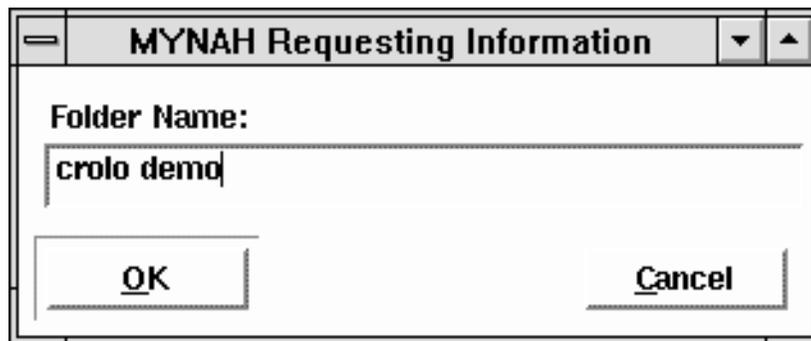
Figure 2-3. Naming a Folder

4. Position the pointer in the **Folder Name** area and type in a name for it.

   — We typed in `crolo demo`.

> **NOTE —** The MYNAH System accepts spaces in object names. The number of spaces and case of character uniquely identify objects.

5. Click on **OK**.

— The system creates a new folder and place it in the desktop display as a line item.
   See Figure 2-4.
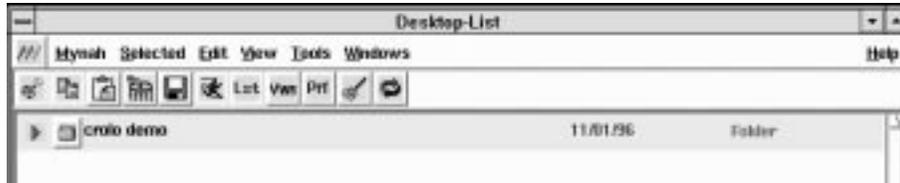
Figure 2-4. New Folder

You can open folders by double clicking on their icons. We won't do this since it is not
necessary to open the folder to do the tasks in the next sub-section.

2.4   Copying an Object Into a Folder

The MYNAH System has a database for the objects you will create and use. The system
also provides you with the **Database Browser** tool that makes finding objects in the
database easy.

We will show you how easy it is to find and copy objects using the Database Browser. We
will copy two objects into our folder: a Test Hierarchy and a SUT Hierarchy.

1. Click on the **Tools** menu.

— The system will display a drop-down list of three tools provided with the system.

2. Click on the **Database Browser** menu option.

— The system will display the **Database Browser** window shown in Figure 2-5. Notice that the selected object type is now **Keyword.**
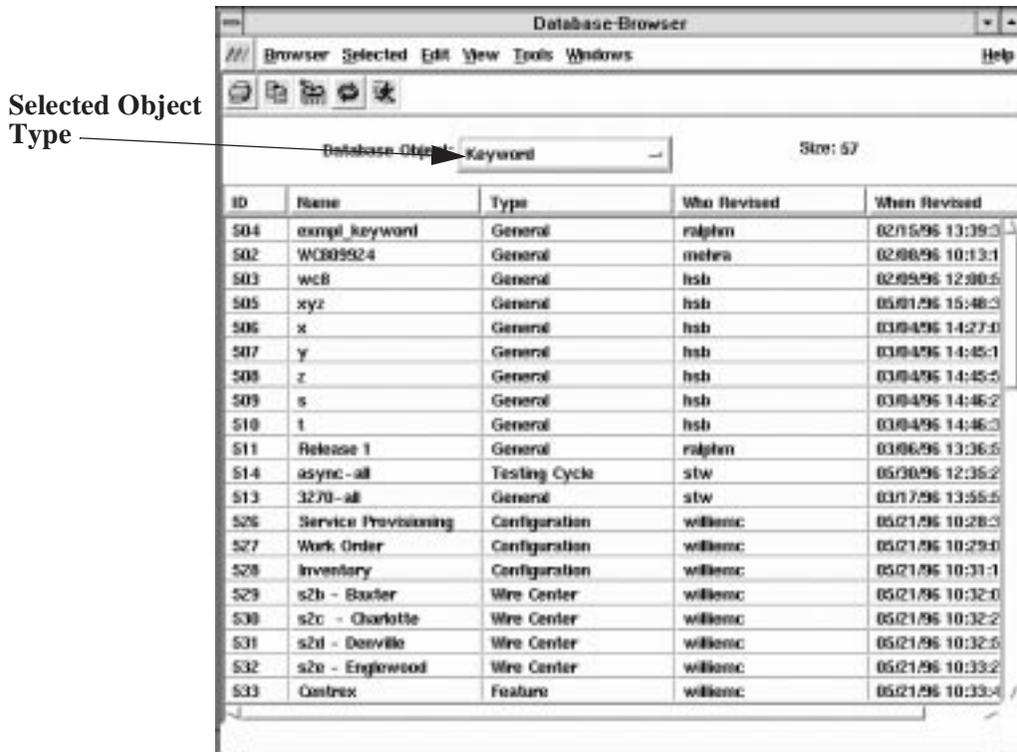
Figure 2-5. Database Browser

3. Let's change the **Database Object** to **SUTHier**, by clicking on the drop-down list and then clicking on **SUTHier**. Figure 2-6 illustrates this.
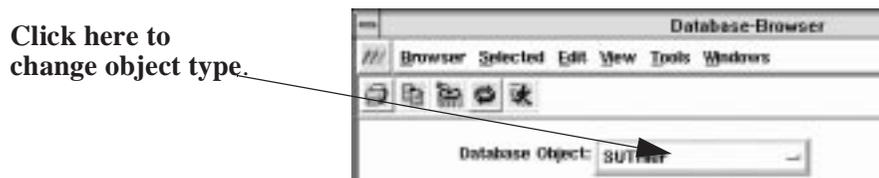
Figure 2-6. Selecting an Object Type

> **NOTE —** If the number of SUT Hierarchies exceeds the default database map number, the system will prompt you to get all or only 100.

4. Now we will select the SUTHier object we want to copy by clicking on it. See
    Figure 2-7. (We selected **Demo Rolodex SUTs**.)

We selected an
object by clicking
on it.
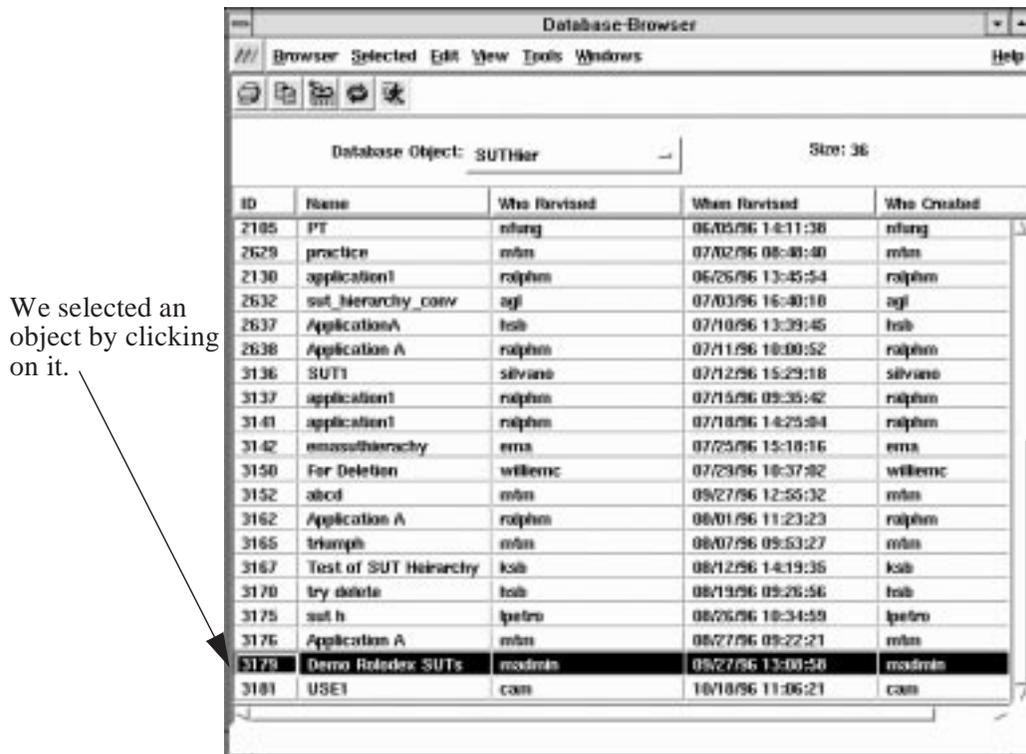
Figure 2-7. Selecting an Object From the Database Browser

5. In the **Database Browser** window, click on the **Edit** menu and then on the **Copy** menu
    selection to copy the object.

6. Make sure **crolo demo** is highlighted on the **MYNAH Desktop-List**. To do this, just
    click on it.

7. Click on the **Edit** menu and then on the **Paste** menu selection.

    — The system copies the **Demo Rolodex SUTs** object into the **crolo demo** folder.
      Figure 2-8 illustrates this. Note how the copied object appears indented below the
      crolo demo folder. This indicates that **Demo Rolodex SUTs** object is in the folder
      named **crolo demo**.

Copy from the Database Browser
to the crolo demo folder.
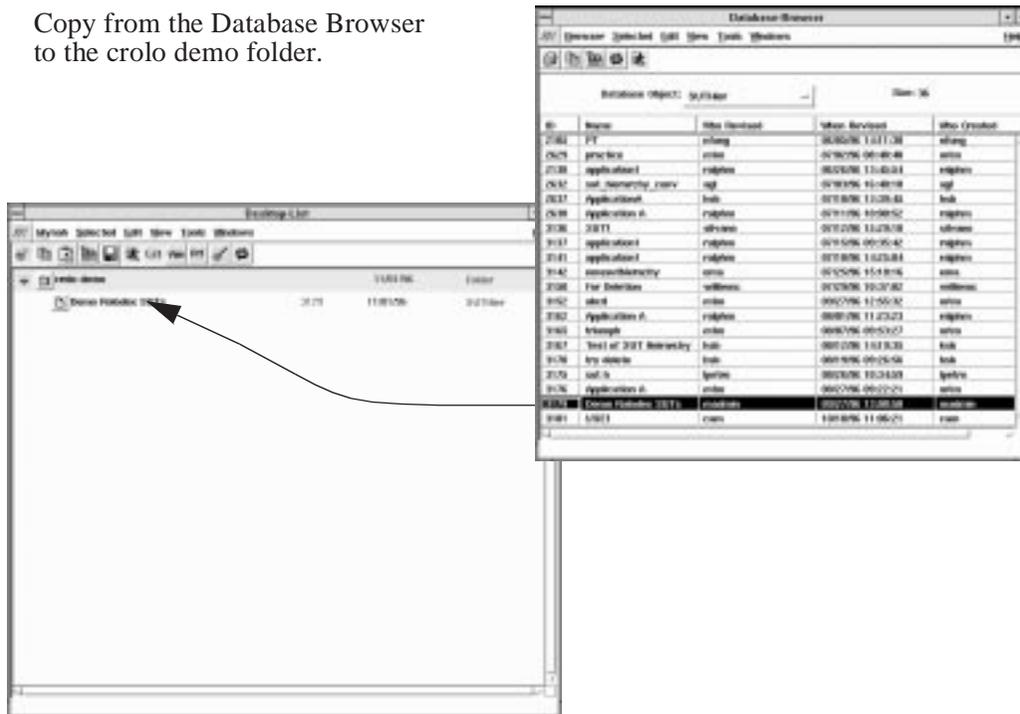
Figure 2-8. Copying a SUT Hierarchy into a Folder.

Now we will copy a Test Hierarchy into our **crolo demo** folder. To do this:

1. In the **Database Browser** window, change the object type to **TestHier** by clicking on
    the **Database Object** drop-down list and selecting **TestHier**.

    > **NOTE —** If the number of SUT Hierarchies exceeds the
    > default database map number, the system will prompt you
    > to get all or only 100.

2. Select the Test Hierarchy, by scrolling though the list and highlighting **Demo Rolodex
    Tests**.

3. In the **Database Browser** window, click on the **Edit** menu and then on the **Copy**
    selection to copy the object.

4. Make sure **crolo demo** is highlighted on the **MYNAH Desktop-List**, to do this just
    click on it.

5. Click on the **Edit** menu and then on the **Paste** selection.

— The system copies the **Demo Rolodex Tests** object into the **crolo demo** folder. Figure 2-8 illustrates this. Note how **Demo Rolodex Tests** appears above the **Demo Rolodex SUTs** object.

Copy from the Database Browser to the **crolo demo** folder.
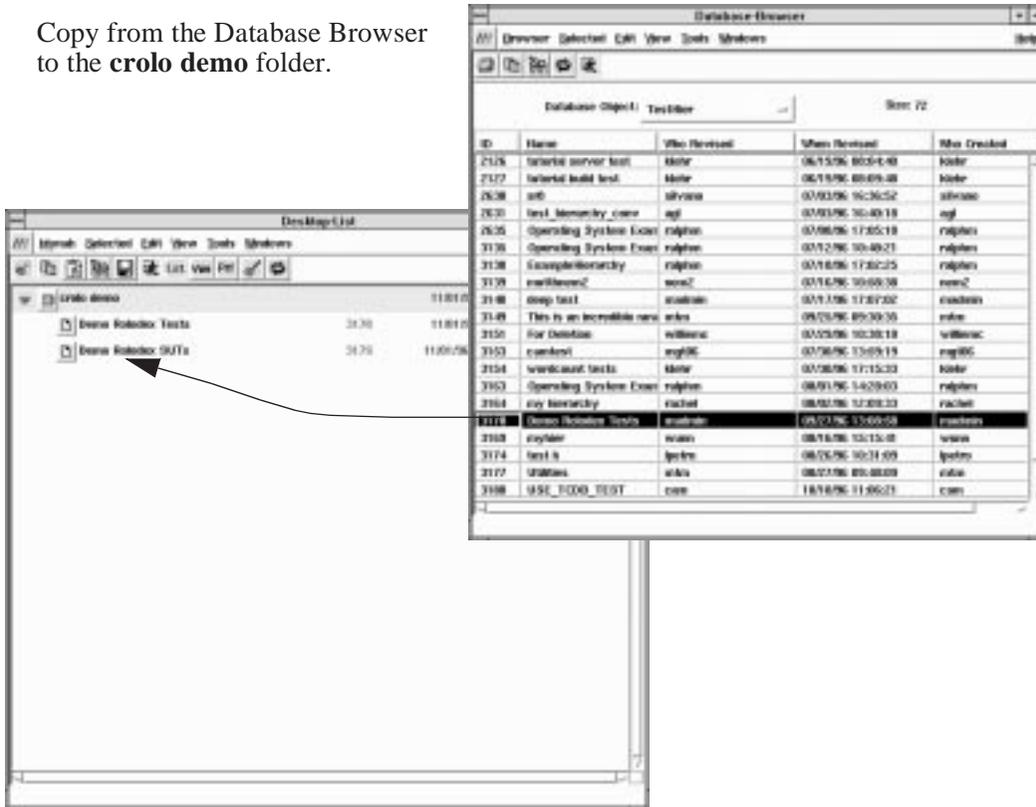
Figure 2-9. Copying a Test Hierarchy into a Folder.

We have now copied all the objects we will need for our Quick Tour.

2.5   Understanding the Default SUT

One of the most important concepts you need to understand when using the MYNAH System is the concept of the default SUT. The default SUT is the object of all your test development activities as far as the MYNAH System is concerned. What we mean by this is that when you develop tests, scripts, requirements and other objects, the system will assume that your efforts are directed at the default SUT.

There are places in the system where you have the opportunity to change the default SUT, but if you don't take the action to change the SUT, the system will assume that the current default SUT is the target.

In this sub-section we will choose a default SUT for our Quick Tour and we will use it in examples throughout the manual when we describe how to use other objects.

The current default SUT is listed on the MYNAH Desktop-Preferences view. If you click on the **Prf** icon that appears above the display area, the system will display the Desktop-Preferences view shown in Figure 2-10.
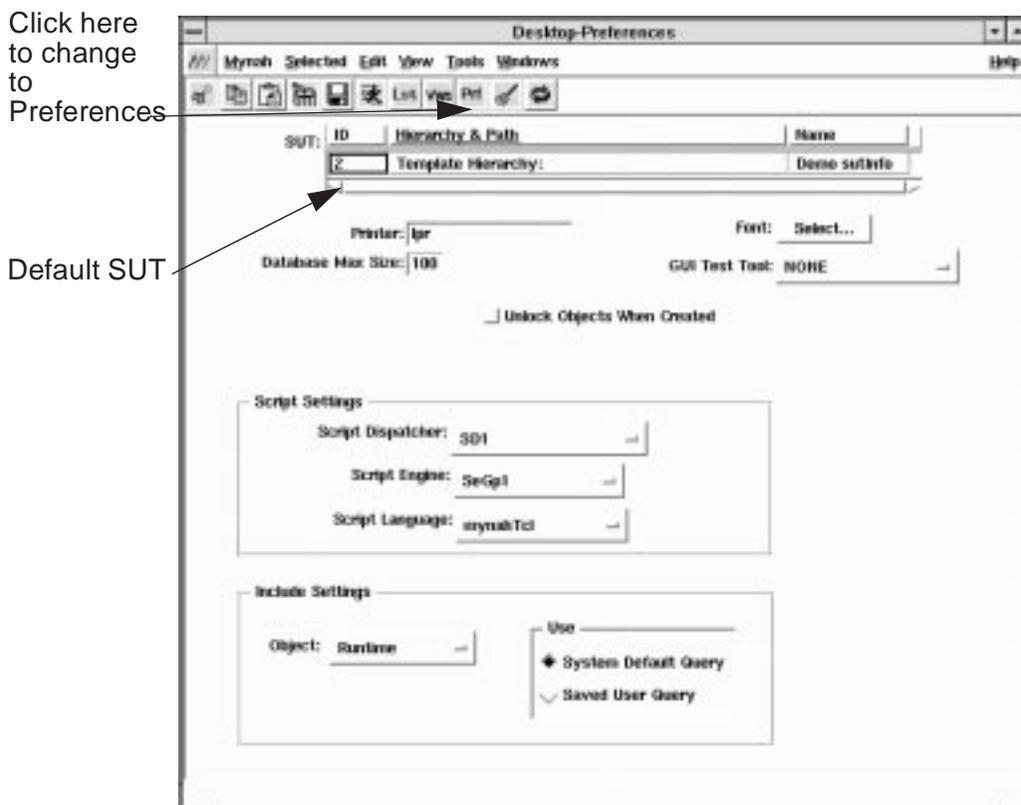
Figure 2-10. MYNAH Desktop-Preference

As you can see, the current default SUT is **Template Hierarchy.** We want to change this to the 5.4 SUT. The 5.4 SUT represents a release for our example. You can copy the SUT in two ways. You can copy it from the SUT hierarchy or from the **Database Browser**.

When you copy it from the **Database Browser**, the actions you take to change the default SUT are similar to the actions you took to copy the SUT and Test hierarchies into the **crolo demo** folder. You copy from the **Database Browser** and paste into the area where the system displays the default SUT. This area is called a "ruler." Most of the time you see this type of display area you can copy into it or from it.

To copy the SUT from the SUT hierarchy you have to open the hierarchy and then copy from it into the ruler area. Since we used the **Database Browser** earlier we will copy the new default SUT from the SUT Hierarchy.

To copy from a SUT Hierarchy:

1. In the **Desktop List** view, double click on the icon for the **Demo Rolodex SUTs** hierarchy.

    — The system opens the hierarchy and display it in a window. See Figure 2-11.

Double click
here

System will
open the SUT
Hierarchy
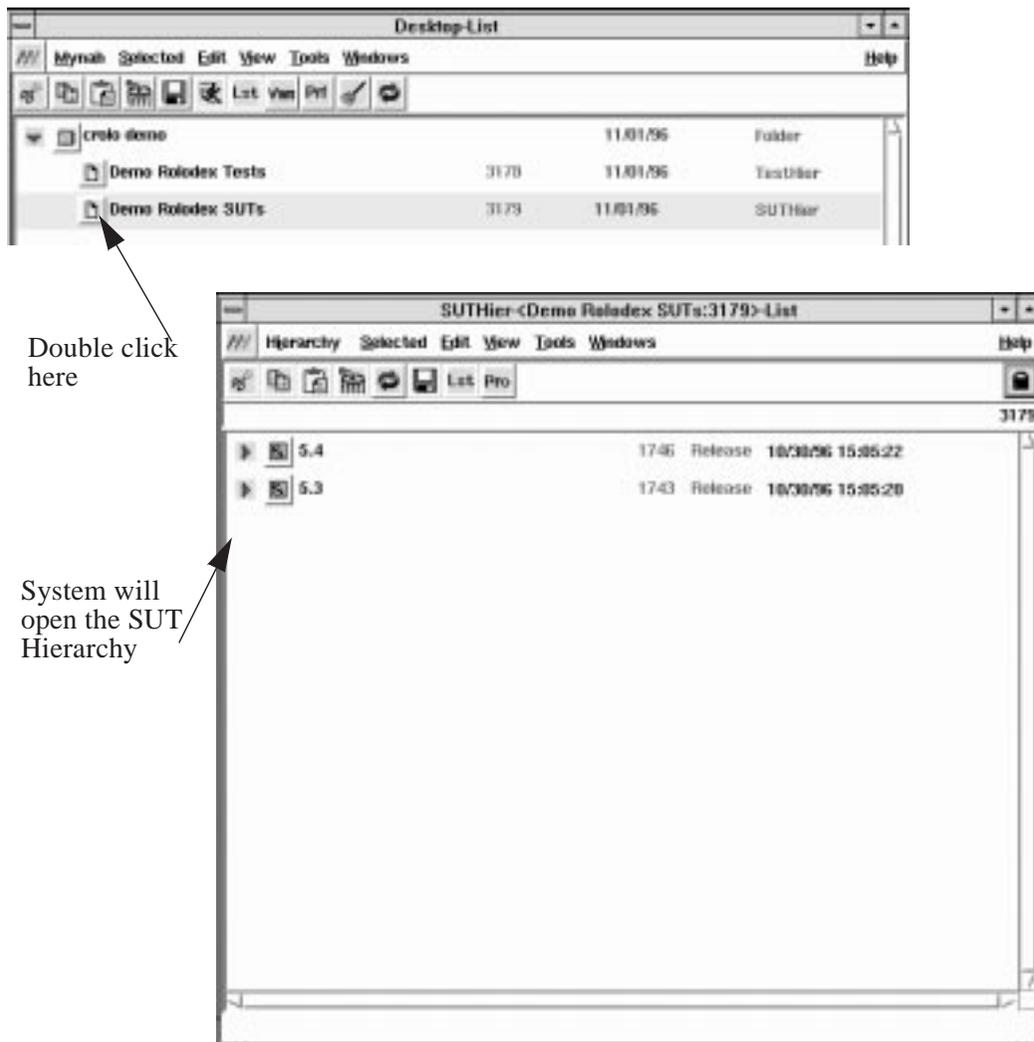
Figure 2-11. Opening a SUT Hierarchy

2. Select SUT **5.4** since this is the one we want for our default.

3. Click on the **Edit** menu and then on the **Copy** selection.

4. In the MYNAH Preferences view, click the pointer in the **SUT** ruler to make it the focus of actions.

5. Click on the **Edit** menu and then on the **Paste** selection.

— The system will paste the SUT you selected into the SUT ruler display area. Figure 2-12 illustrates this action.

Highlight
SUT and copy
here

Paste SUT
into ruler
here.
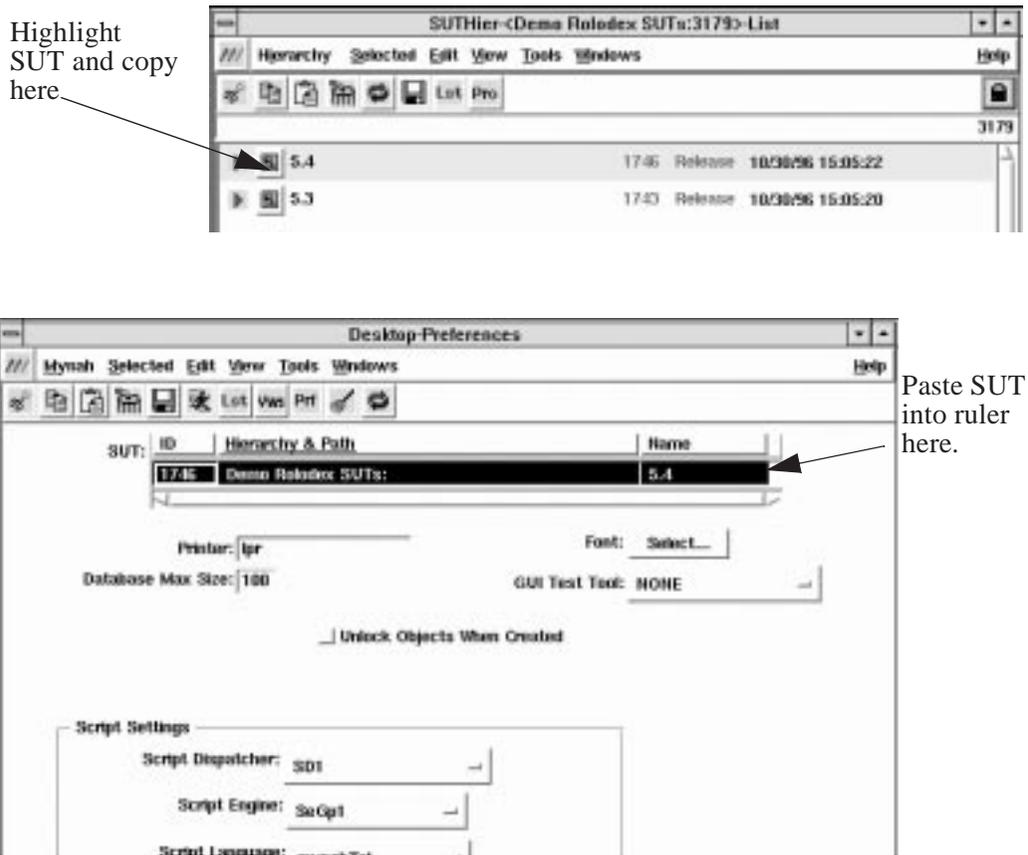
Figure 2-12. Copying a SUT from a Hierarchy

6. Change back to the List view by clicking on the **Lst** icon.

2.6   Demo Test Hierarchy

We now have our default SUT for this session. Let's look at the other hierarchy we copied into our **crolo demo** folder.

You open Test Hierarchies the same way we opened our demo SUT Hierarchy -- by double clicking on it's icon. After you do this the system will display the Test Hierarchy shown in Figure 2-13.
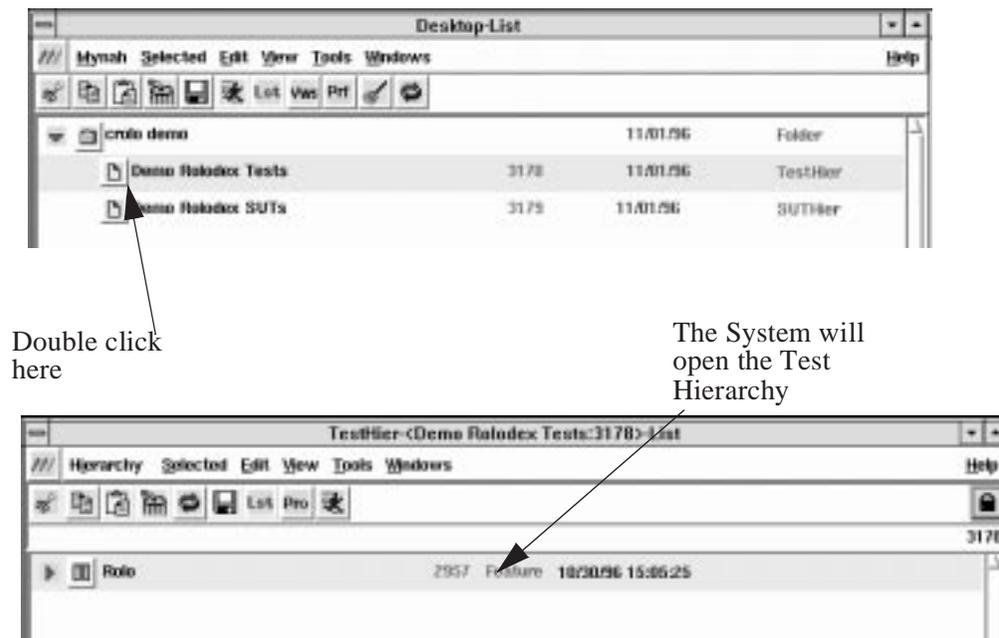
Figure 2-13. Opening a Test Hierarchy

When the system opens our **Demo Rolodex Tests** Hierarchy, it displays the first level of the hierarchy. We show the first level of the hierarchy **Rolo** selected. When you first open the hierarchy, **Rolo** won't be selected.

To see the complete hierarchy, expanded to show all levels, select **Rolo**, click on the
Selected menu and then on the **Expand Fully** menu selection. When you do this the system
will display the hierarchy as shown in Figure 2-14.
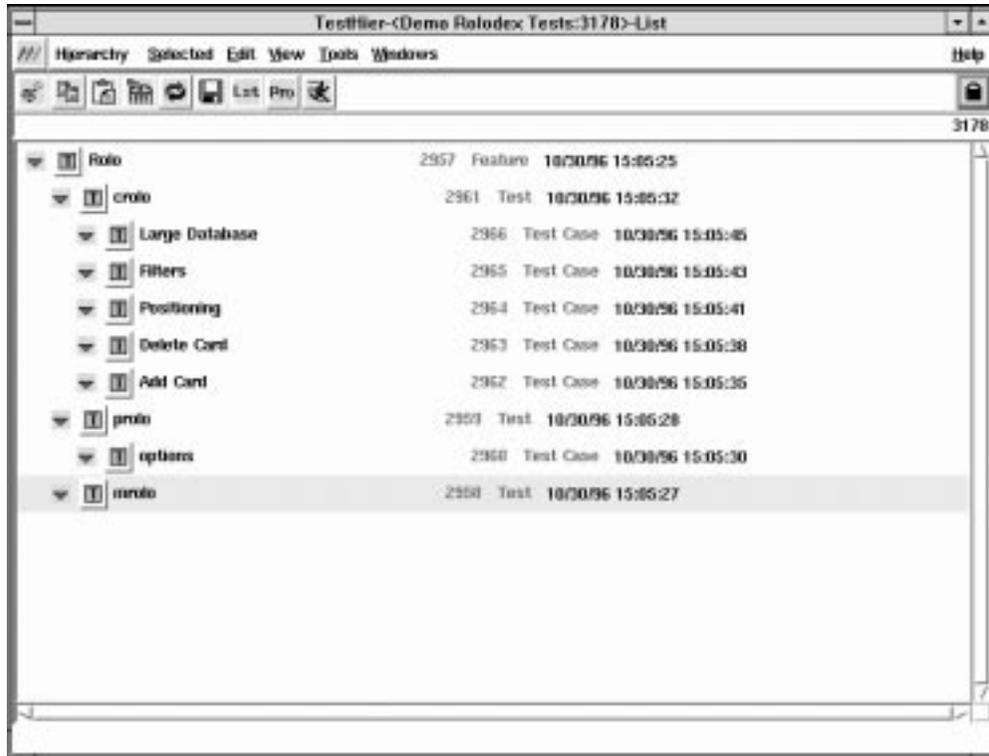
Figure 2-14. Fully Expanded Test Hierarchy

We won't go into much detail about Test Hierarchies here, but note how we have arranged
the tests in the hierarchy. The highest level test is **Rolo**. Below that are three 2nd level tests
(**crolo, prolo, mrolo**). There are a number of test cases at the third level for **crolo** and one
test case for **prolo**. All the tests for our demo system are displayed and their relationship to
other tests made clear by the hierarchical representation.

2.6.1   Duplicating the Example Test

In the remainder of this section we will develop a script that implements the **Filters** test but
we won't actually use the **Filters** test that came with the sample application. We will
duplicate the **Filters** test and give it a new name.

In the MYNAH System, when we say test, we mean documentation about a test. The test
can be performed manually or automated with a script written in Tcl or one of the other
languages compatible with the MYNAH System.

To duplicate our example test:

1. Click on the Lock icon to unlock the hierarchy.

2. Select the **Filters** test.

3. Click on the **Edit** menu and then on the **Duplicate** menu selection.

   — The system will create a new Test object and name it **CloneOf<id>**. In this
      example, it is **CloneOf2965**. Figure 2-15 illustrates this.

Click on Edit and Duplicate.
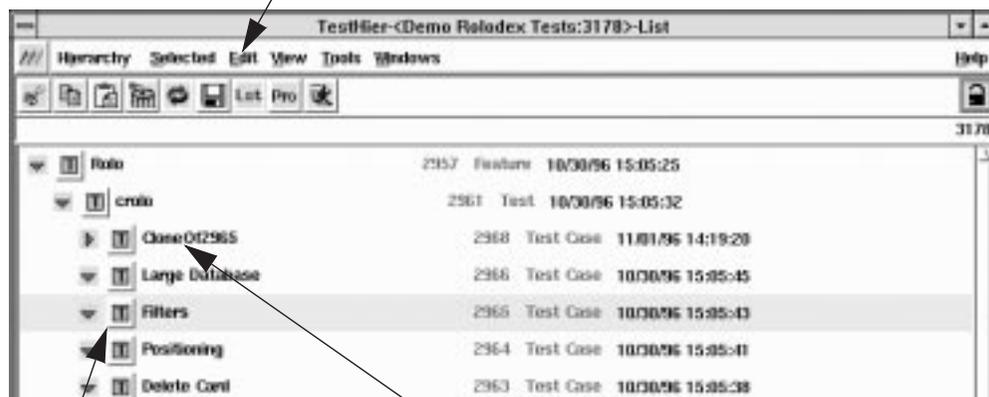
Select the Filters Test.   System creates a " CloneOf" the original.

Figure 2-15. Duplicating a Test Object

Now that we have created a new test by duplicating the **Filters** test we have to give it a new
name and make some other changes. We will name our test **<loginId>Filters** where
loginID is your login ID.

1. Click on the **Hierarchy** menu and then on the **Save** menu selection.

2. Double click on the **CloneOf2965** test icon to open it.

   — The system will display the Properties view for the **CloneOf** test shown in
      Figure 2-16.
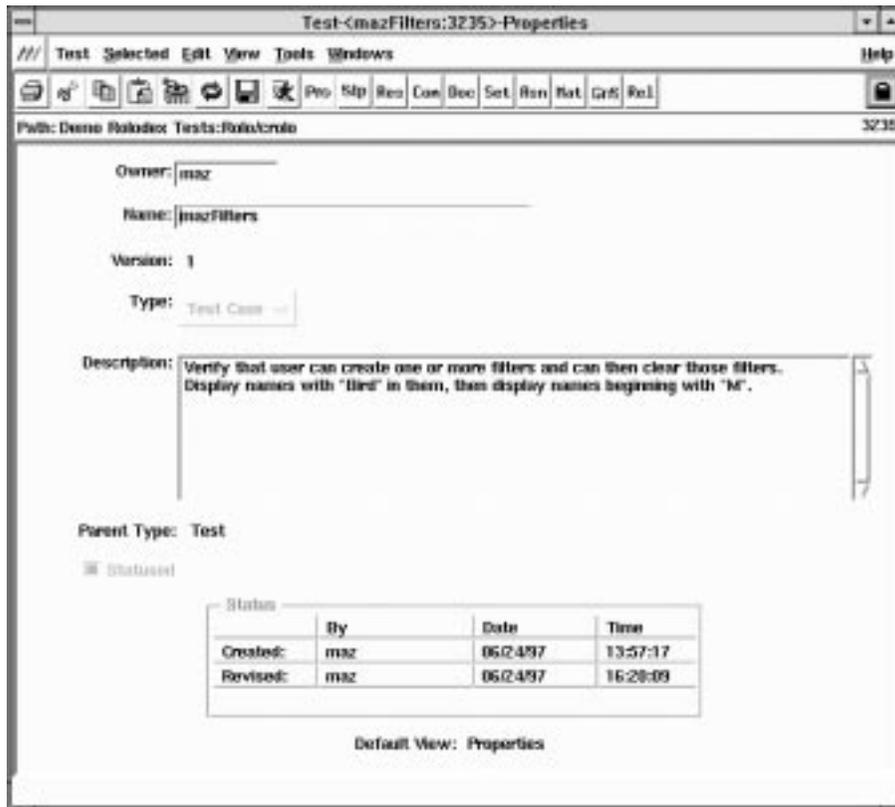
Figure 2-16. Filters Test Properties View

— The Properties view gives you basic information about the test: who owns it, its name, its type and a brief description of what the test is supposed to do. Note that the clone has the Description filled in. This was copied from the original when you duplicated it.

3. Click on the Lock icon to unlock the test.

4. Erase **CloneOf** and type in `<loginId>Filters.`

— When we finished the Properties view looked like Figure 2-17.
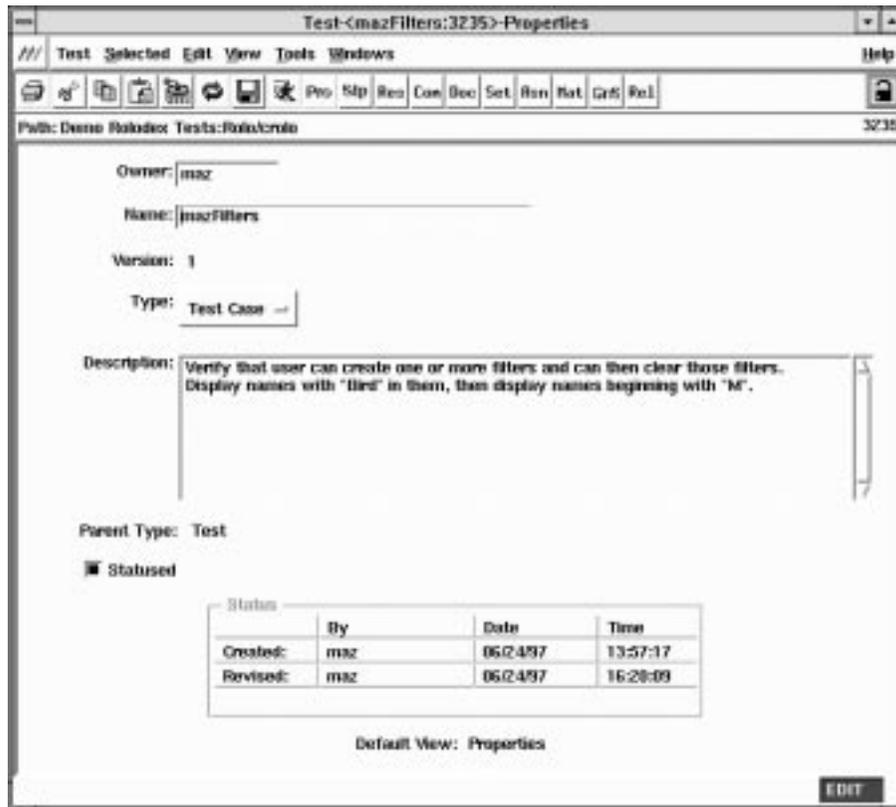
Figure 2-17. Finished Filters Test Properties View

2.6.2 Using Test Steps

The original **Filters** test object specifies all the steps needed to perform the test. These steps could be done manually, but we will create a script that will perform them for us. Now we have to copy these steps into our newly duplicated `<loginId>Filters`.

1. In the **Demo Rolodex** Test hierarchy, double click on the icon for the **Filters** test.

2. On the **Filters** test Properties view, click on the **Stp** icon.

   — The system will display the Step List view for the Filters test shown in Figure 2-18. These are the Steps we want to copy into our new test.

Figure 2-18. Filters Test Steps List View

3. On the **`<loginId>Filters`** test Properties view, click on the **Stp** icon.

   — The system will display the StepList view for the **`<loginId>Filters`** test. See
   Figure 2-19. Note that its empty.
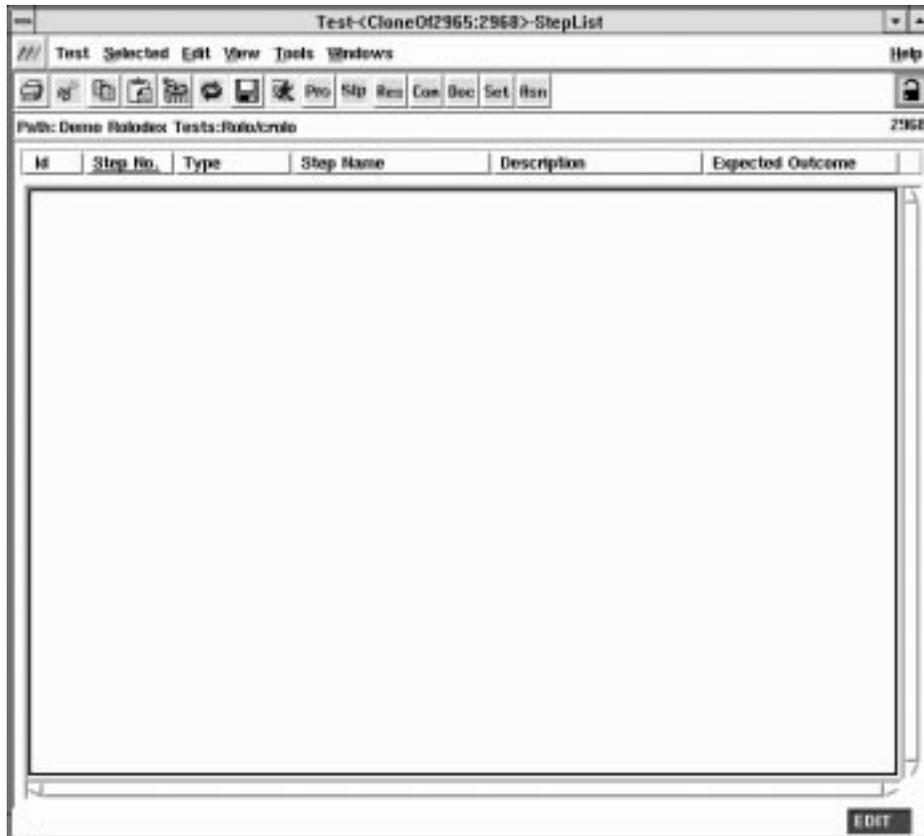
Figure 2-19. ksbFilters Test Steps List View

4. In the Filters test StepList, drag the mouse pointer across the Steps to selected them.

5. Click on the **Edit** menu and then on the **Copy** menu selections.

6. On the **`<loginId>Filters`** StepList view, click on the **Edit** menu and then on the
   Paste menu selections.

   — The system will copy the steps from the **Filters** test to the **`<loginId>Filters`**
   test. Figure 2-20 illustrates this.

7. On the **`<loginId>Filters`** StepList view, click on the **Test** menu and then on the
   Save menu selection.
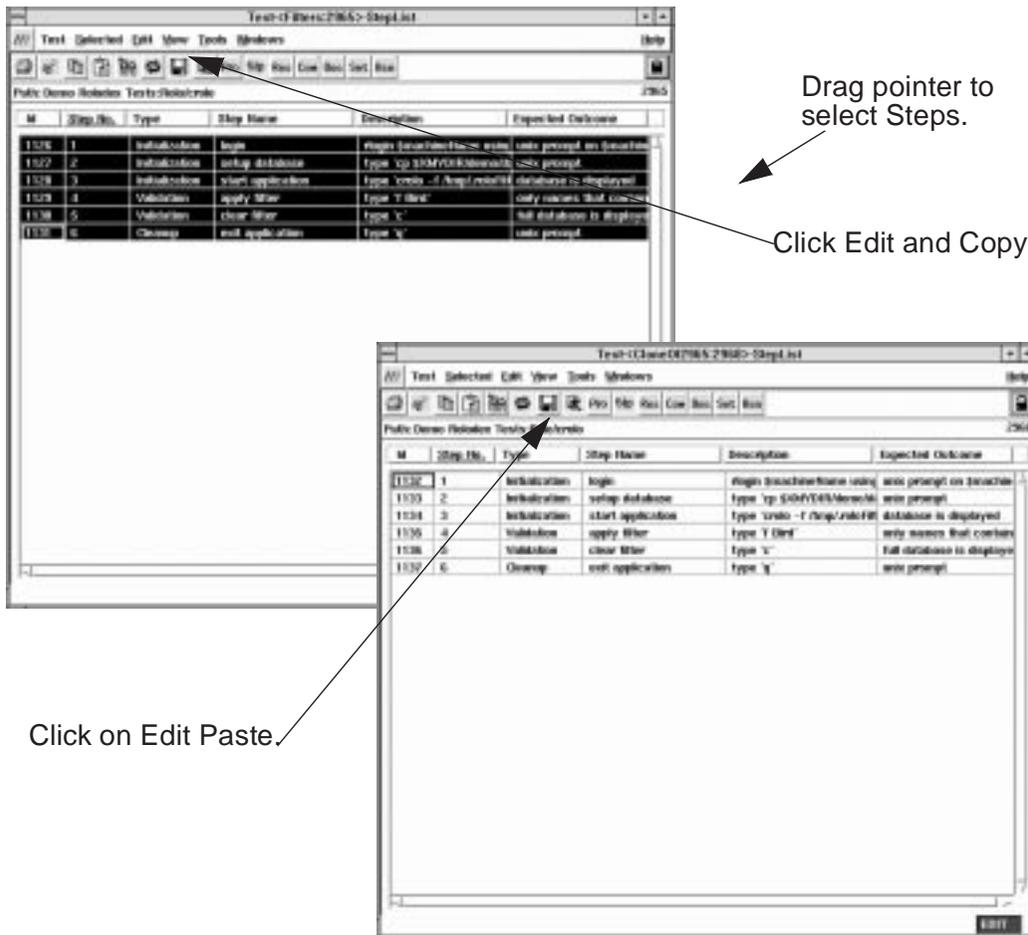
Figure 2-20. Copying Test Steps

In all there are six steps for the **Filters** test that we copied into the `<loginId>Filters`
test. We won't need to do the first step, but we will do the remaining five. These include:

- Setting up the database by typing

  `cp $XMYHOME/demo/data/rolo.db /tmp/roloFilters`

- Starting the application by typing

  `crolo -f /tmp/roloFilters`

- Filtering the **crolo** database for names with "bird" in them by typing

  `f bird`

- Clearing the filter by typing

  `c`

- Quiting the application by typing

    `q`

As you can see, the MYNAH System provides you with an easy way to essentially leave instructions for anyone who is to perform or automate a test. Next we will create the script that implements the **Filters** test.

2.7   Starting the Script Builder Feature

The Script Builder feature is your work bench for creating code. It is here that we will develop the Tcl code for our **Filters** test. This code can be saved in a script code file and documented with a Script object. To start the Script Builder:

1. Click on the **Tools** menu and then on the **Script Builder** menu selection.

    — The MYNAH System will display the Script Builder Code View shown in Figure 2-21. This is where you do most of your script development.
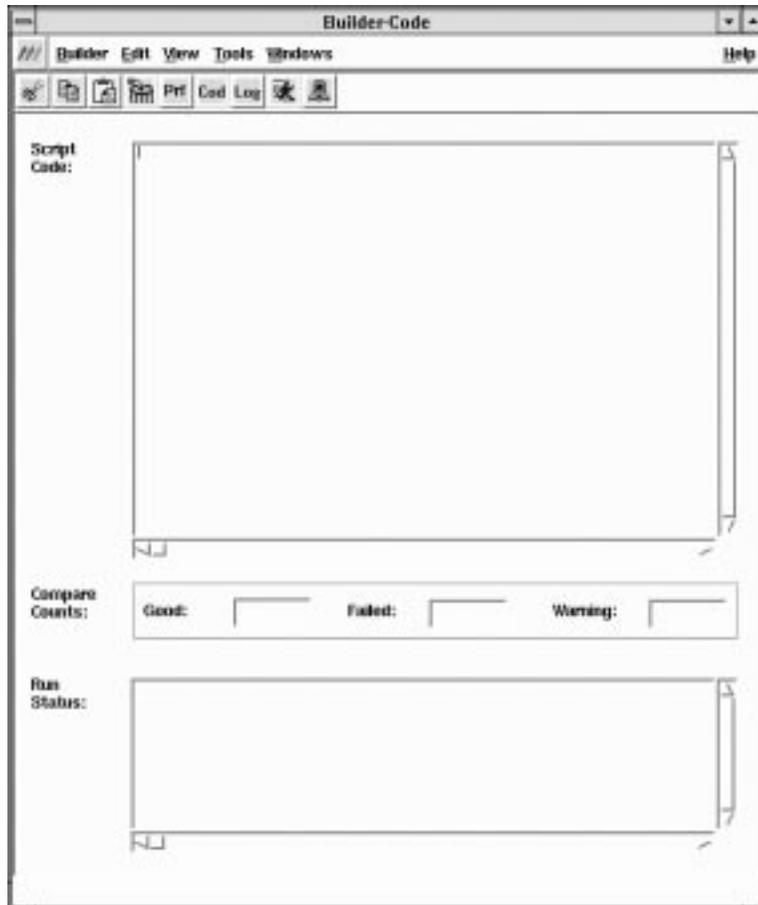
Figure 2-21. Builder-Code View Window

2.8   Opening an Asynchronous Connection

Now that we have started the Script Builder, we need to establish a connection to the UNIX system where our example application resides.

To do this:

1. Click on the **Builder** menu and then on the **Open Connection** menu selection; or click on the **Terminal** icon.

   — The system will display the dialog shown in Figure 2-22. Here we will choose part of what we want to include in our script. We will record the Tcl statements that open the connection, load the emulation package, and record events on the UNIX system. The system will automatically assign a connection variable name to the connection you just created in this example (**conn1**).
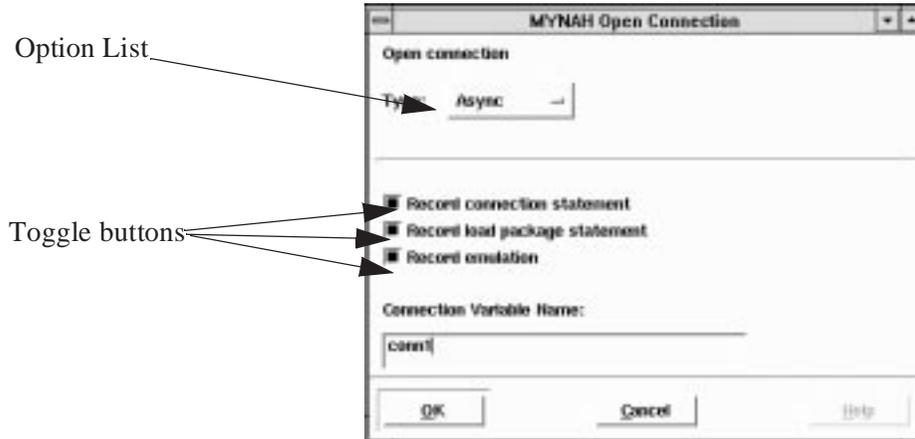
Figure 2-22. Async Terminal Setting Dialog

2. Click on the **Type** option list and select **Async**.

   — We accepted the default setting for **Record connection statement** and **Record emulation**.

3. Click on the **Record load package statement** toggle button.

4. Click on **OK**

   — The system will open an Asynchronous window to your home directory. See Figure 2-23. Note that the Tcl statements that made the connection and loaded the terminal emulation package appear in the Code View.

Recorded
Tcl Code

Set connection
statement
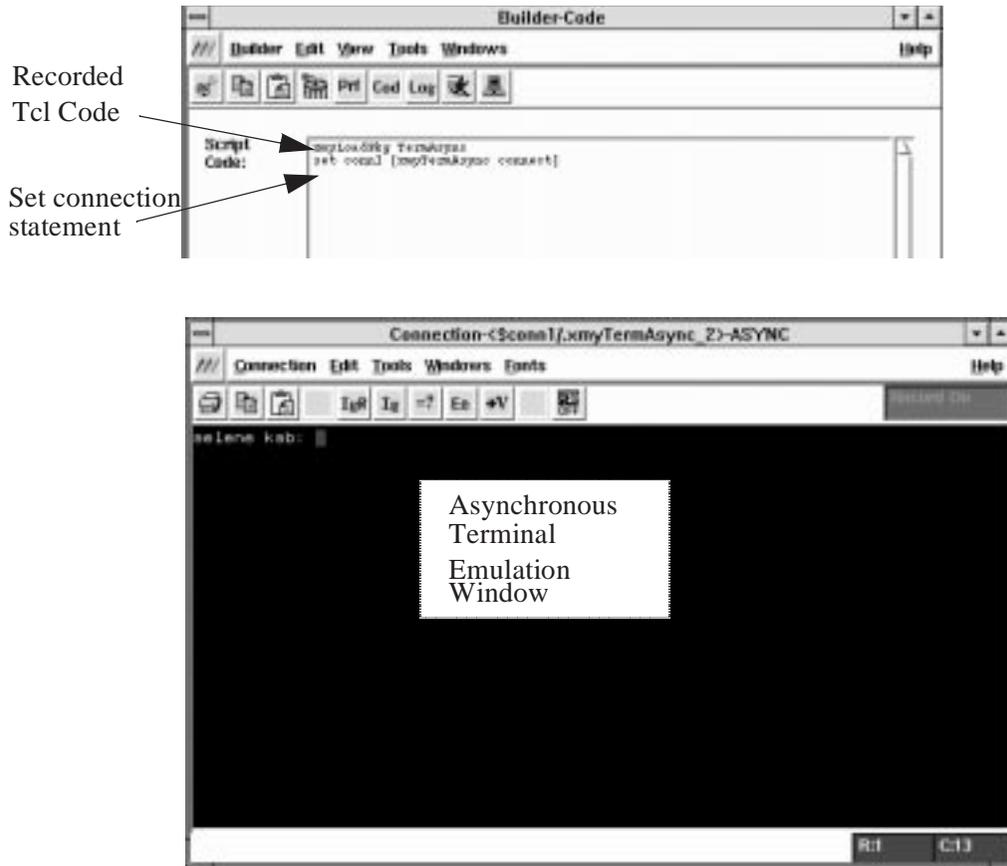
Figure 2-23. Async Terminal Emulation Window

MYNAH System Users Guide   **BR 007-252-006**
A Quick Tour Through the MYNAH System   **Issue 3, October 1997**
Release 5.2

2.9 Recording Keystrokes Made on a Remote System

From now on when you type something into the ASYNC window, the MYNAH System will record and encode it into a Tcl statements in the Code view. We will enter the keystrokes that were specified in the **Filters** Test object Step view. (See Figure 2-18.)

Remember the basic steps are:

- Copy the demo database for **crolo**

- Start **crolo**

- Filter the database for name with **bird** in them

- Clear the filter

- Exit the **crolo** application

> **NOTE —** If you use the Backspace or Delete key in the Async window, it will place control characters in the Script Code window. You can erase them in the Script Code window.

We will also define a screen region as a compare so that when our script executes it will compare what occurs on the live screen with what it expects based on what we recorded in the script. The compare will encompass all the names in the database that have the string **bird** in them.

To do this:

1. Position the pointer in the ASYNC window and type:

   `cp $XMYDIR/data/rolo.db /tmp/roloFilter`

   and press the RETURN key.

   > **NOTE —** The text you type in an emulation window **does not** appear in the Tcl script in the corresponding Code view until you type a character **after** you press the RETURN key. For example, the copy statement you just typed will not appear in the Code view until you type the letter c (of crolo) in Step 2.

2. Type

   `crolo -f /tmp/roloFilter`

   and press the RETURN key.

   — The UNIX system will start the **crolo** application while the MYNAH System records your keystrokes and encode them in the Code view.

3. Type

   f *Bird*

   and press the RETURN key.

   — The application will display all the names with bird in them and the MYNAH
     System records your keystrokes and encode them in the Code view.

4. To insert a compare:

   A. Drag the pointer to "box" the names in the ASYNC connection display.

   B. Click on the **=?** icon in the connection window.

      — Figure 2-24 illustrates this.
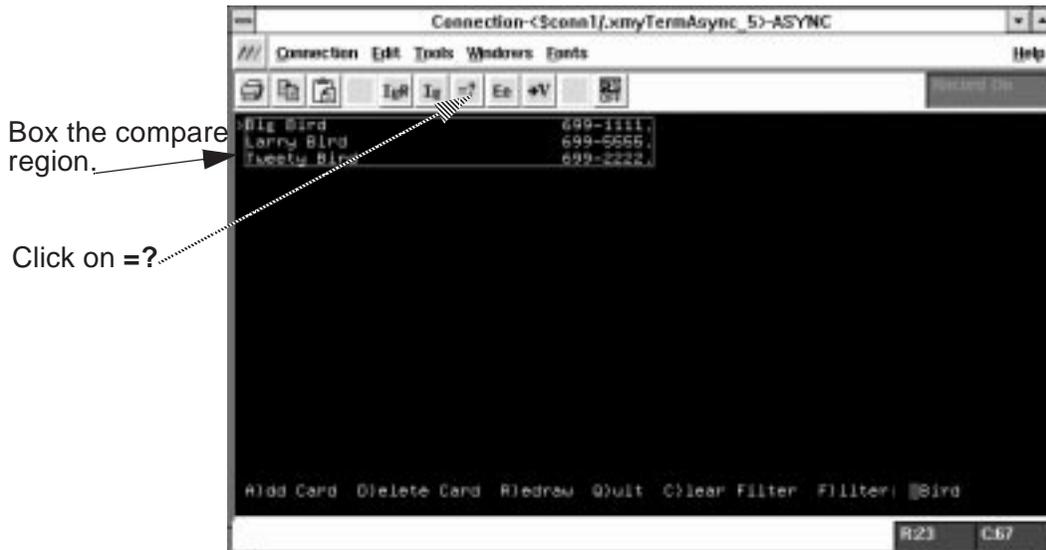
Figure 2-24. Inserting a Compare

5. Type

   c to clear the selection

6. Type

   q

   — When you complete entering the keystrokes into the ASYNC Terminal window it
     will look like Figure 2-25.
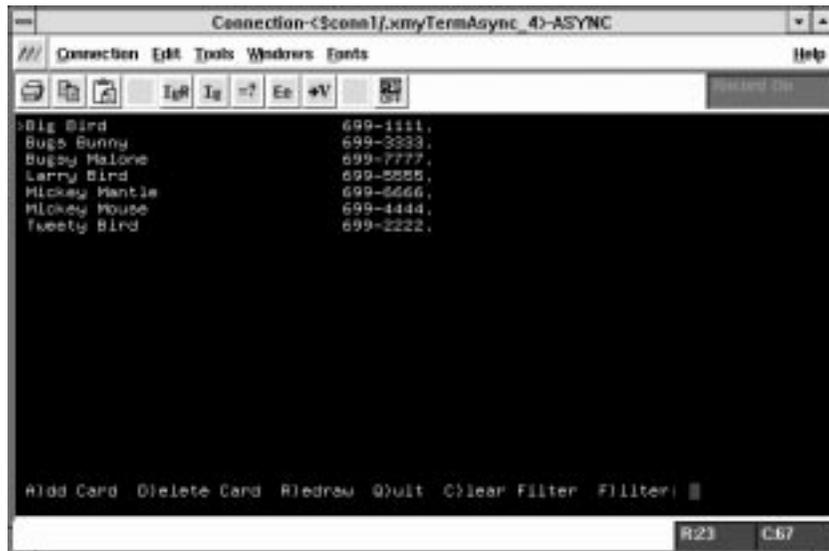
Figure 2-25. ASYNC Terminal Emulation Window with Application Screen

7. Click on the **Connection** menu and then on the **Close** selection.

8. Confirm exiting the connection when the system displays the confirmation dialog, by clicking on **OK**.

The Code view will have all the keystrokes you entered along with the Tcl connection, load
and exit statements. It will also show the screen region captured with the compare we
inserted. Figure 2-26 shows the Code view after we finished entering keystrokes and closed
the connection.

Figure 2-26. Code View with Recorded Keystrokes

MYNAH System Users Guide   BR 007-252-006
A Quick Tour Through the MYNAH System   Issue 3, October 1997
Release 5.2

2.10   Inserting Tcl Commands Using a Template

We have recorded all the keystrokes we need. Now we want to insert a Tcl sleep command
that will pause the script while it executes.

1. In the Code view, position the pointer at the beginning of the

    ```
    $conn1 sendWait "fbird\r" -expect "6C"
    ```

    line and click.

    — Figure 2-27 illustrates this.

Position pointer
here and click.
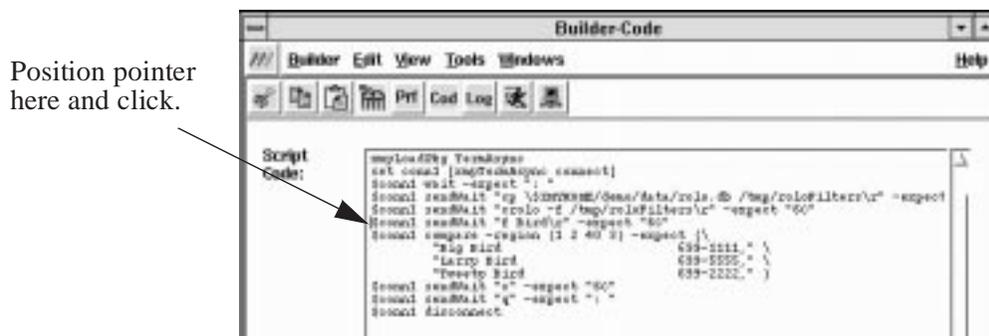
Figure 2-27. Positioning the Pointer to Insert a Tcl Statement

2. Click on the **Edit** menu and then on the **Insert Template** menu option.

    — The system will display the Insert Template dialog shown in Figure 2-28.

Figure 2-28. Insert Template Dialog

3. Scroll through the list of Tcl commands and click on `xmySleep`.

4. Drag the pointer across the **Example** to highlight it.

— See Figure 2-29.

Scroll through
list and select.

Drag pointer to
highlight the
example.
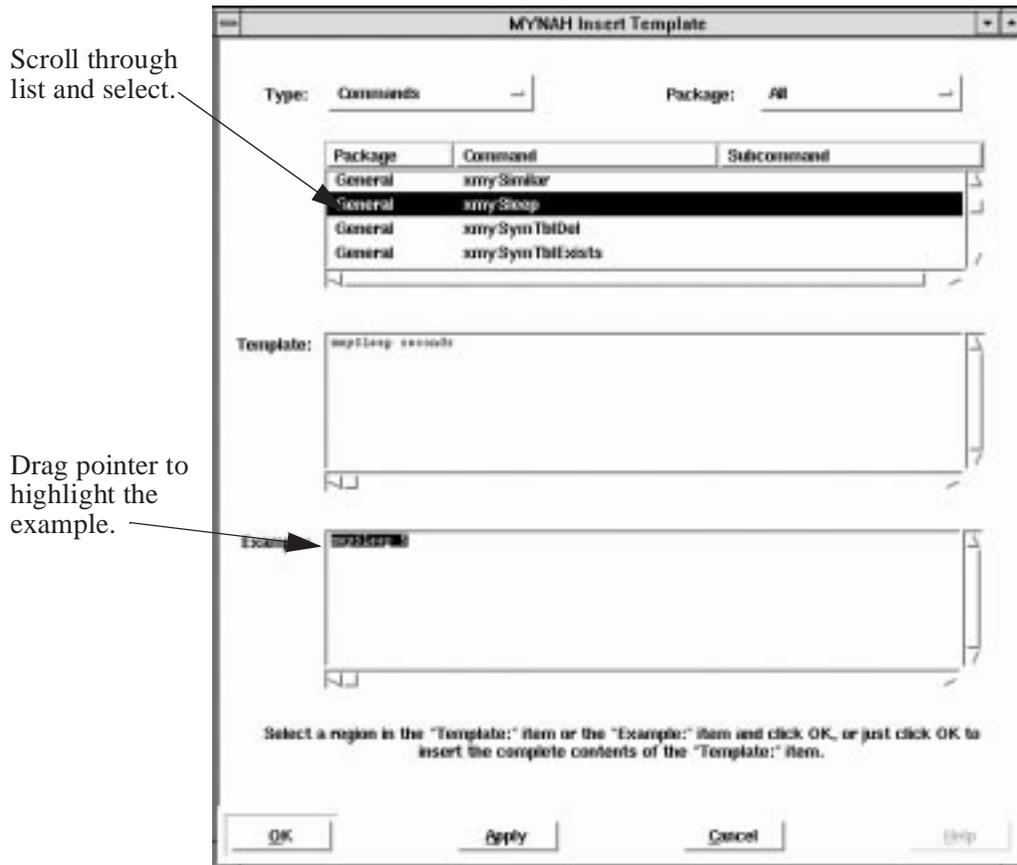
Figure 2-29. Selected Template and Example

5. Click **OK**.

— The system will place the example **xmySleep 5** command in the Code view above
where you positioned the pointer. See Figure 2-30.

> **NOTE —** There is a default value for this command, 5
> (seconds). We simply accepted that, but you can change
> this if you care to.
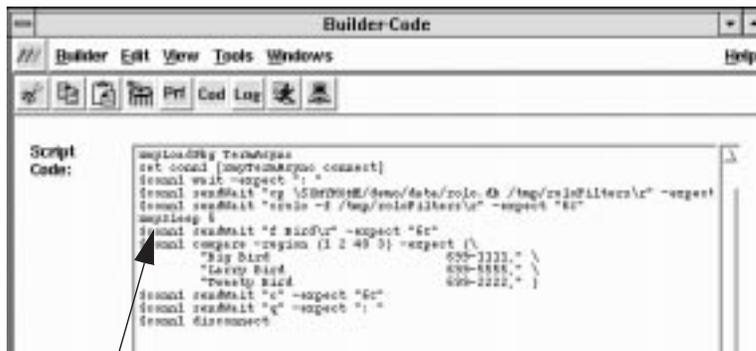
Template example in Code

Figure 2-30. Template Example Inserted in the Code View

2.11   Running The Example Script

We have created a script. Now we will execute the script directly from the Code view. This is a good practice when scripting. It allows you to make sure that the script functions properly.

1. Click on the **Builder** menu and then on the **Run Code** menu selection or click on the **Runner** icon.

   — The system will display the Run dialog shown in Figure 2-31.
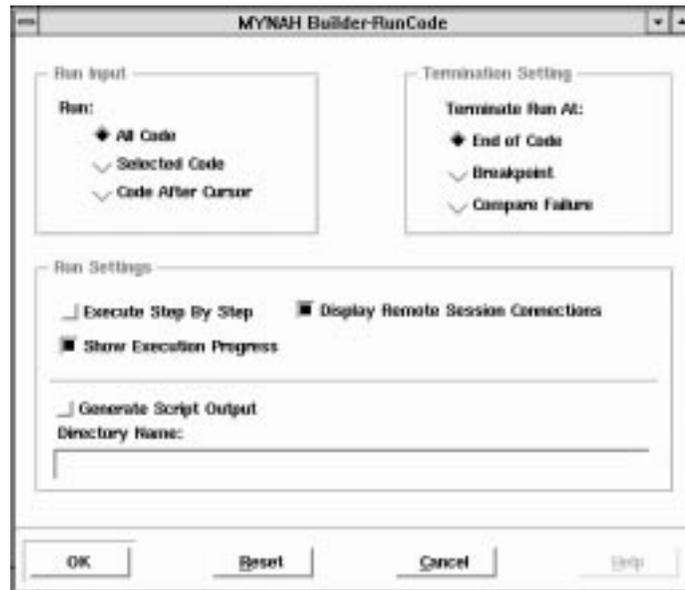
Figure 2-31. Run Code Dialog

2. We will accept the default settings, so click **OK**.

   — As the script executes, the system will display the Run Progress and dismiss it
   when the script completes.

   — You should be able to see the script open a connection and perform all the actions
   we encoded in the script. When the script finishes the Code viewwill look similar
   to Figure 2-32.

Figure 2-32. Code View When Run Is Finished

NOTE — Note the bottom portion of the Code view. In the **Run Status** area there is a message telling us that the run succeeded. This message is repeated in the Status Bar at the very bottom of the window. There is **1** in the *Compare Counts Good field*. This shows that the compare we entered succeeded for this test.

2.12   Saving Scripts

Now that we created a script and ran it to make sure that it operates the way we expect it to operate, we want to save the script to a file in the UNIX system.

1. Click on the **Builder** menu, then the **Save Code** menu selection, and finally on the **To file....** menu selection.

   — The system will display the **Save To File** dialog shown in Figure 2-33.
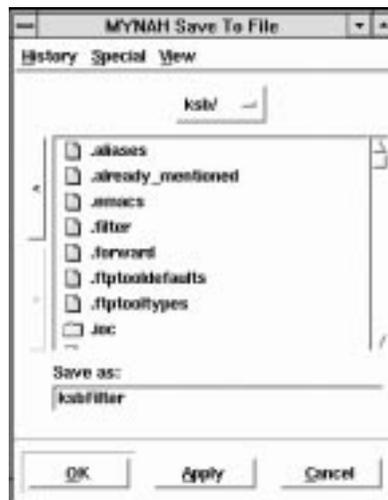
Figure 2-33. Save To File Dialog

2. Type in a name for the file in which you will save the code we just created. (We typed in **ksbFilters**.) and click OK.

   — The system will save the code in the file you specified.

2.13 Creating a Script Object to Document Your Script

We created, executed and stored a script though the script builder, but as far as the MYNAH System goes, that script doesn't exist. We have to let the MYNAH System know that there is a script and indicate where it is located. We will do this by creating a Script object. To do this:

1. On the MYNAH Desktop, select the **crolo demo** folder. (If it isn't selected already.)

2. Click on the **Selected** menu; then on the **New** menu selections; and finally, on the sub-menu that pops up, click on **Script**.

    — The system will place a new Script object labelled "Untitled-Script" in the first position of the **crolo demo** list as shown in Figure 2-34.

Double Click on
the icon for the new"Untitled- Script" object.
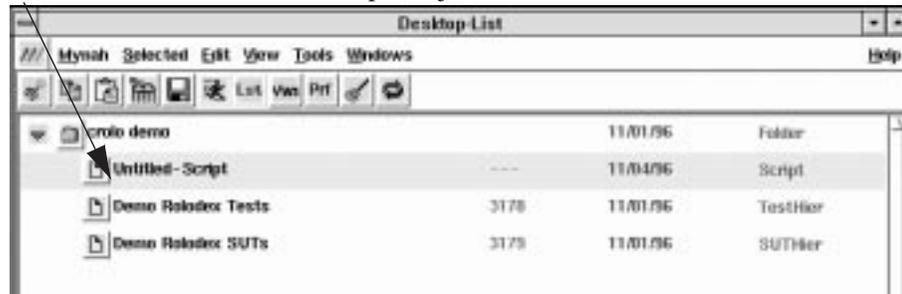
Figure 2-34. New Script Object

3. Double click on the icon for the new script object.

    — The system will display the Script Properties view shown in Figure 2-35. Note that the **Owner** field has your login ID.

Figure 2-35. Script Properties View

4. Click on the Lock icon that appears below the **Help** menu.

5. Position the pointer in the **Logical Name** data field and type in a name for the Script object. We typed in `ksbFilters`.

6. To specify a script code **File Name**, perform one of the following:

   • Position the pointer in the **File Name** data field and type in the file name for the script we created. Include the complete path to the file, e.g., */bdfs/users/ksb/SCRIPTS/ksbFilters*.

   • Click on the **?** icon at the end of the **File Name** data field and select a file name from the list. When you are done Click **OK**. (See Figure 2-36.) We didn't use this because we didn't have an existing file.

Click here to go up one
directory level.

Click here to go down one
directory level.
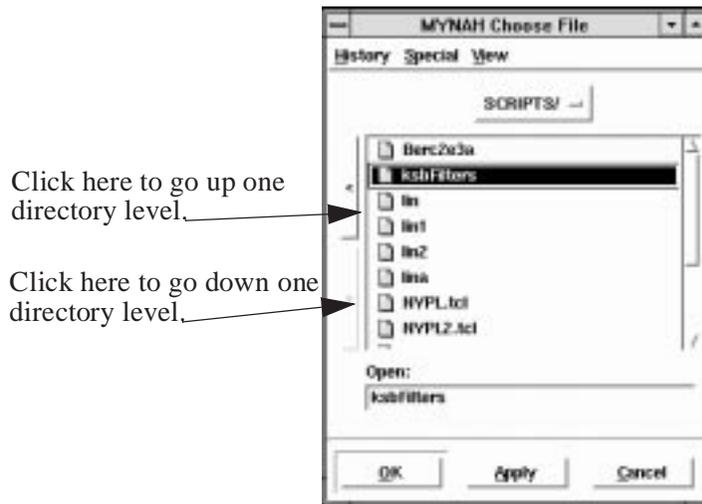
Figure 2-36. Select a FIle

7. Position the pointer in the **Description** data field of the **Properties View** and type in a description for you script. We typed in `Script created for Quick Tour`.

— We finished entering this information in our Properties view so that is looks like Figure 2-37. Note that we accepted the default Script Language **mynahTcl** because this is the scripting language we used.

Figure 2-37. Script Properties View Filled Out

8. Click on the **Script** menu and then on the **Save** selection. (You can also click on **Close** and click on **Yes** when prompted to save changes.)

— The new Script object we created will look like Figure 2-38 in the **crolo demo** folder.
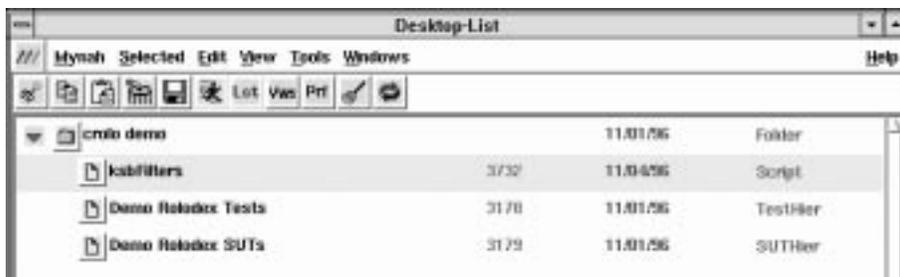
Figure 2-38. Completed Script Object

2.14   Identifying a Script to a Test

Now that we have created the script that will implement the **ksbFilters test,** we need to identify the script to the test. Right now, the MYNAH System doesn't know that we created a script to implement the **ksbFilters test.**

To do this:

1. In the **crolo demo** folder, double click on the **Demo Test** Hierarchy.

   — The system will open the **Demo Rolodex Test** Hierarchy

2. Select the **Rolo** test and the click on the **Selected** menu and the **Expand Fully** menu selection.

   — The system will expand the **Demo Rolodex Test** Hierarchy to show all levels.

3. Double click on the icon for the **ksbFilters test.**

   — The system will open the **ksbFilters test** script to its Properties view.

4. Click on the **Set** icon and when it appears, click on the Lock icon to unlock it.

   — The system will change to the test Settings view shown in Figure 2-39.

Set icon

Click
here to
unlock
it.
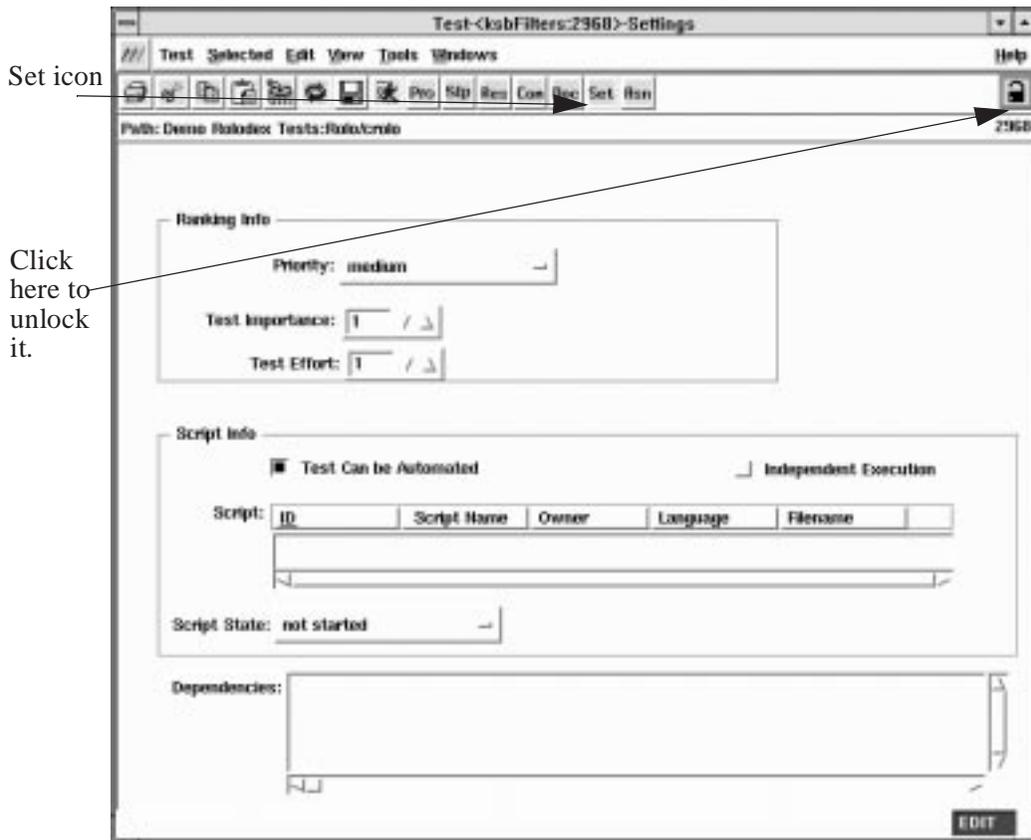
Figure 2-39. ksbFilters Test Setting View

5. In the **crolo demo** folder, select the **ksbFilters** script.

6. Click on the **Edit** menu and then on the **Copy** selection.

7. In the **ksbFilters** Test Settings view, click in the **Script** data entry/display area.

8. Click on the **Edit** menu and then on the **Paste** selection. (Figure 2-40 illustrates this process.)

Select ksbFilters Script;
Click Edit and Copy.

Copy from folder
to Setting view.

Click here for focus:
click Edit and Paste.
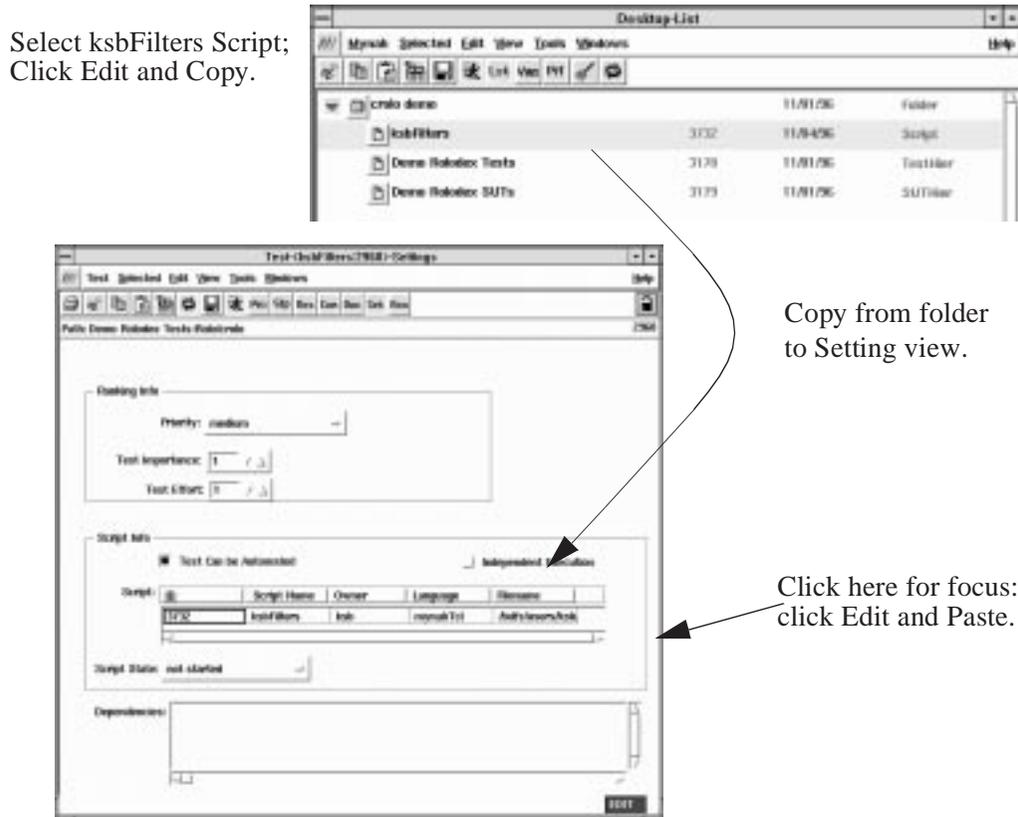
Figure 2-40. Identifying a Script to a Test

9. Click on the **Independent Execution** check box.

10. Click on the **Script State** option list and select **complete** since we have completed a script for this test.

11. Click on the **Test** menu and then on the **Close** selection.

— The system will display a confirm dialog asking you if you want to save the changes. See Figure 2-41.

Figure 2-41. Confirm Dialog

12.　Click **Yes**.

Now that we have an automated test, every time you run the test or script we just created, the MYNAH System will create *result object* for the test.

If you want to run the test, select the test in the Test Hierarchy, click on **Selected** menu and then on the **Run** menu selection. The system will send the script for this test to the Background Execution Environment and execute it. The system will also pop-up a Job Status window to let you know when the script was submitted for execution and when it completed.

2.15　What's Next?

This quick exercise should have given you a feel for the MYNAH GUI. You can go on to the next *Section 3: MYNAH Basics.* If you really feel comfortable about how the GUI works and what the controls are, you can even go on to *Section 4: Using MYNAH Objects.*

3.　MYNAH 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 MYNAH System.

This section covers:

- What folders are

- The layout and operation of the system's major functions, View windows and dialog boxes.

- Basic actions needed to navigate around the MYNAH System

- Controls used in the MYNAH GUI

- How to perform some routine tasks

We won't tell you much about objects. We cover them in detail in the next section, *Section 4: Using MYNAH Objects.*

3.1　MYNAH 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 we had 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 MYNAH System, the filing cabinet itself would be the MYNAH Desktop. 0Within the MYNAH Desktop there could be folders that correspond to drawers in the filing cabinet. Each folder could have requirements objects, issue objects, and script objects related to different releases of an application.

- 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.
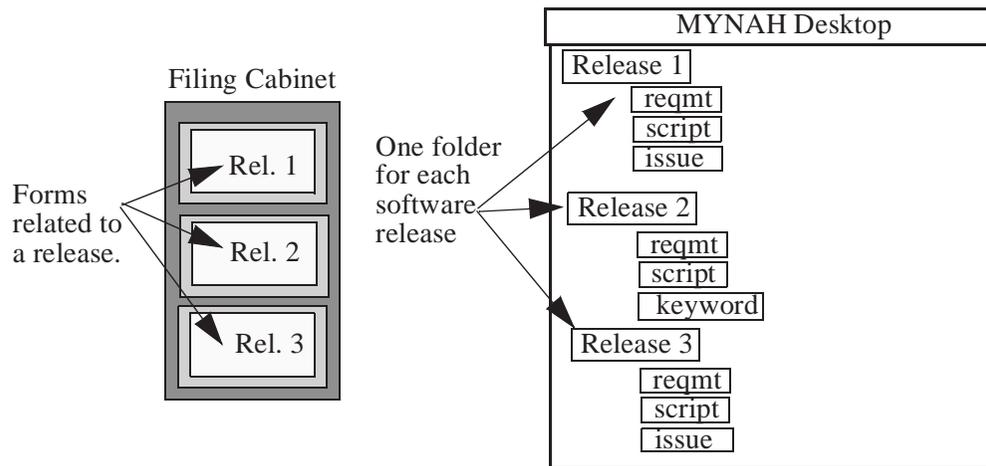
Figure 3-1. What Are Folders?

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

The MYNAH Desktop is the first window you encounter when you start the MYNAH System and serves as the main entry point to the system. (See Figure 3-2.)

3.1.1   What is the MYNAH Desktop?

You can think of the MYNAH Desktop as you would the top of your desk. The top of your desk contains folders and tools, such as pens, a telephone, and a computer, which help you do your job. Also, the MYNAH Desktop is a work space where you can collect all the objects and folders you will need for testing or task-automation.

The MYNAH Desktop has folders and tools which help you do your testing and scripting jobs. -

In this chapter we will discuss some of these tools such as dialog boxes and view functionwindowwithin the MYNAH system. In *Section 4: Using MYNAH Object*s we will discuss more of them.
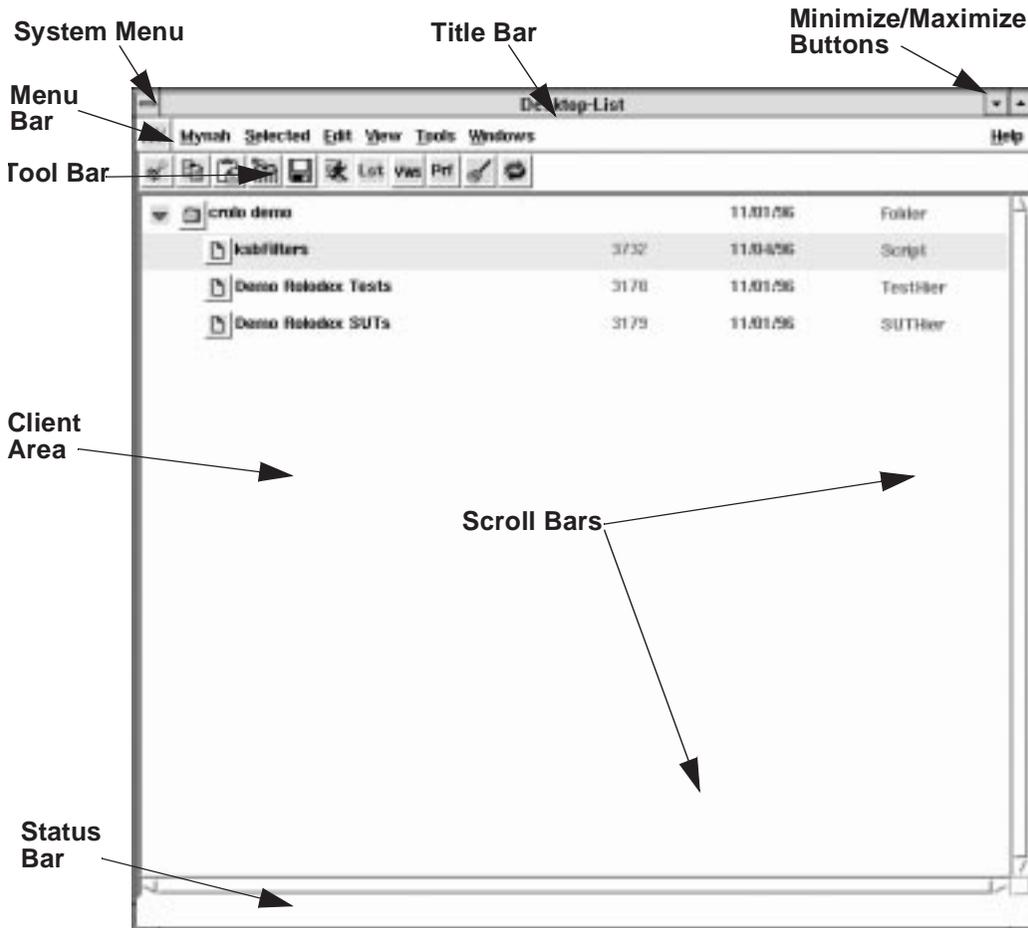
Figure 3-2. MYNAH Desktop (List View)

The MYNAH Desktop we have depicted above shows the List View. We will explain more about Views in Section 3.5.

The MYNAH Desktop main window consists of Title Bar, Menu Bar, ToolBar, Scroll Bar Status Bar, and Client Area. The basic window layout is the same for all windows in the MYNAH System. Your Desktop View may vary but the layout is simular, this will depend on the function or task in the MYNAH System.

3.1.2   Title Bar

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

On the left-hand side of **Title Bar** is 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:

Restore	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.
Move	Allows you to move the window using the arrow keys or by dragging it.
Size	Allows you to re-size the window using the arrow keys or mouse.
Minimize	Reduces the window to an icon
Maximize	Increases the window size to the maximum allowed by the window manager.
Close	Closes the window.
Switch to...	Allows you to switch to any open program

On the right-hand side of **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.

On the MYNAH Desktop Menu bar the menu's are as follows: **MYNAH, Selected, Edit, View, Tools, Windows,** and **Help.** The first name on a Menu bar will vary, this will depend on the window function. For example, the MYNAH Desktop (List View) has **MYNAH** as the first menu. The **Selected, Edit, View, Window and Help** menus appear on most windows.

> **NOTE —** In most cases when creating **New** objects. the MYNAH System lets you choose from two menu options selections. You can choose a **New object** from the first Menu selection (e.g., MYNAH, Folder & Hierarcy), or you can also choose a **New** object from the **Selected** menu option. The **Selected** menu option is primarily for creating child objects.

3.1.3.1   MYNAH Icon

The MYNAH System icon appears to the far left of the menu bar. See Figure 3-3.

Figure 3-3. MYNAH Icon

If you click on the MYNAH icon, the system will display the MYNAH Banner window.

We will describe the MYNAH Desktop menus and their selections in some detail here. In the following sections of this Users Guide we will discuss variations in menu selections where appropriate. You can also find a comprehensive listing of all the menus and their selections in *Appendix A: MYNAH Menus.*

3.1.3.2   MYNAH Menu

The menu selections for the **MYNAH** Menu are:

Save	Saves the contents of the MYNAH Desktop, and any objects preferences you set with the MYNAH window views.
New	Displays a cascade menu that allow you to create new MYNAH objects on the MYNAH Desktop.
Reports	Causes the system to display the Reports dialog with which you can create reports.
Iconify All Windows	Reduces all open windows to icons.
Exit	Closes the MYNAH Desktop and folders, then exits to the UNIX prompt.

The MYNAH System allow two entries on the **Menu Bar** for creating a New Objects. On the Desktop you will alway have the View menu name (e.g. MYNAH, Folder or Test Hierarchy) and the **Selected** menu name. Two general rules to follow when creating a new object:

- When creating a new object in the Desktop View menu name. The **New** menu selection will appear on the first menu selection of the menu bar. This new will ignore any selected items on the Desktop view and will place the new object at the top of the Desktop display.

- When creating a new object in the **Selected** menu bar, then the **New** menu selection, the New menu selection will be disabled when nothing is selected. If an object is selected on the Desktop, the new object will be created as a child of the selected object.

3.1.3.2.1   *What Happens When You Save MYNAH Desktop*

You save a MYNAH Desktop by clicking on the **Save** option in the MYNAH menu that appears on the far left-hand side of the Menu bar. The system saves the contents of the desktop window.

The system saves the organization of object, folders and any object 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. Everytime 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 MYNAH System, it saves the contents of folders, the desktop, and the object preferences you selected every 60 seconds to 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 MYNAH objects you have selected and to create new objects. Selections include:

New   Displays a cascade menu that allows you to create new MYNAH objects on the MYNAH Desktop or in a folder. (We will explain more about objects in the next section, *Section 4: Using MYNAH Objects*.)

Open   Allows you to open a selected object.

Duplicate   Create a new Test or SUT object that is similar to an existing one.

Run   Runs the selected script in the BEE. The system enables this menu selection only when a script or test is the 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 are:

Undo

Cut   Removes the selected object from the display and temporarily places it on a clipboard.

Copy	Places a copy of a selected object on a clipboard.
Paste	Retrieves an object 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 objects in a selected state.
Deselect All	Deselects all visible selected objects.

3.1.3.5   View Menu

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

Change View	Allows you to change the view for the currently opened object.
Refresh	This refreshes the current view by removing objects from the display that are no longer in the database; include folders and objects, when you refresh you get the latest copy of object from the database; and the **Database Browser**, re-issues any defined or default queries.

3.1.3.6   Tools Menu

The **Tools** Menu contains a number of useful tools or functions, these have separate window view. These include:

Database Browser	Allows you to search for objects in the MYNAH database. You can open or run objects directly from the Database Browser or copy them to a folder.
Job Status	Lists all the scripts running in the BEE.
Script Builder	Helps you to develop scripts, capture events on a remote system, and run scripts interactively.
Start GUI Test Tool	Starts the GUI test tool you have specified in the Preferences view.

3.1.3.7   Windows Menu

The **Window** Menu helps you to manage the windows associated with the MYNAH System. The drop down menu will *show all the currently opened windows* even if they are

iconized. 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 MYNAH folder window is the **Help** Menu. The selections are:

Contents	Displays a list of help topics.
Procedures	Displays a list of hypertext links to help on tasks and activities. (Available in future releases.)
Keyboard	Lists key accelerators and their uses.
On Help	Displays help messages on the help system.
On Icons	Displays the icons that appear on the MYNAH Desktops windows and Toolbars, and gives a brief description of each.
On Version	Displays product information.

3.1.4   The Toolbar

As you can see in Figure 3-2, icons appear directly below the Menu Bar in what we call the Toolbar. These icons represent often used functions that correspond to menu selections. We placed them here for your convenience. The features available on Toolbar change depending on the window. Here we will describe the tools available with the MYNAH Desktop in some detail (See Table 3-1.) and 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 object and temporarily places it on a clipboard. This does not remove objects from the database.

Table 3-1. Tool Bar Icons and Functions

Icon	Function	What It Does
	Copy	Places a copy of a selected object on the clipboard.
	Paste	Retrieves an object from the clipboard and places it in the client area of folder.
	Delete	Deletes a selected object from the database and clears it from the display.
	Save	Saves the contents of the MYNAH Desktop windows, any object preferences you set.
	Run	Causes the system to display the Run Dialog with which you can execute tests and scripts.
Lst	Change View	Changes the current view to the view represented by the icon, e.g. Lst changes view to the List view. Icons vary with the views available with an object.
Vws	Default Views	Allow you to set the default views for objects, folders, and Tools. Default view means that the view you select here will appear when you open an object of the selected type.
Prf	Preference	Allows you to set items that affect all objects and processing done during the session.
	Clear	Removes the selected object from the desktop, but does not place it on the clipboard.

Table 3-1. Tool Bar Icons and Functions

Icon	Function	What It Does
	Refresh	Refreshes the Mynah Desktop view.

You use the Tool Bar by clicking on the icon that corresponds to the function you want. Depending on the icon you selected, the MYNAH 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 Mynah Desktop window between the Toolbar and Status line 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 MYNAH System displays the contents of GUI objects in this area (e.g folders and scripts).

3.1.6   Status Bar

The bottom line of the MYNAH Desktop is the Status Bar which displays text messages on the left-hand side. Two areas appear on the right-hand side of the Status Bar: one on objects which displays the cursor location and the other which displays the current mode MYNAH 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 MYNAH 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 MYNAH display can have a virtual display area, i.e., text is present which appear beyond the visible display. Scroll bars allow you to see this text.

Figure 3-4 shows how to use scroll bars.
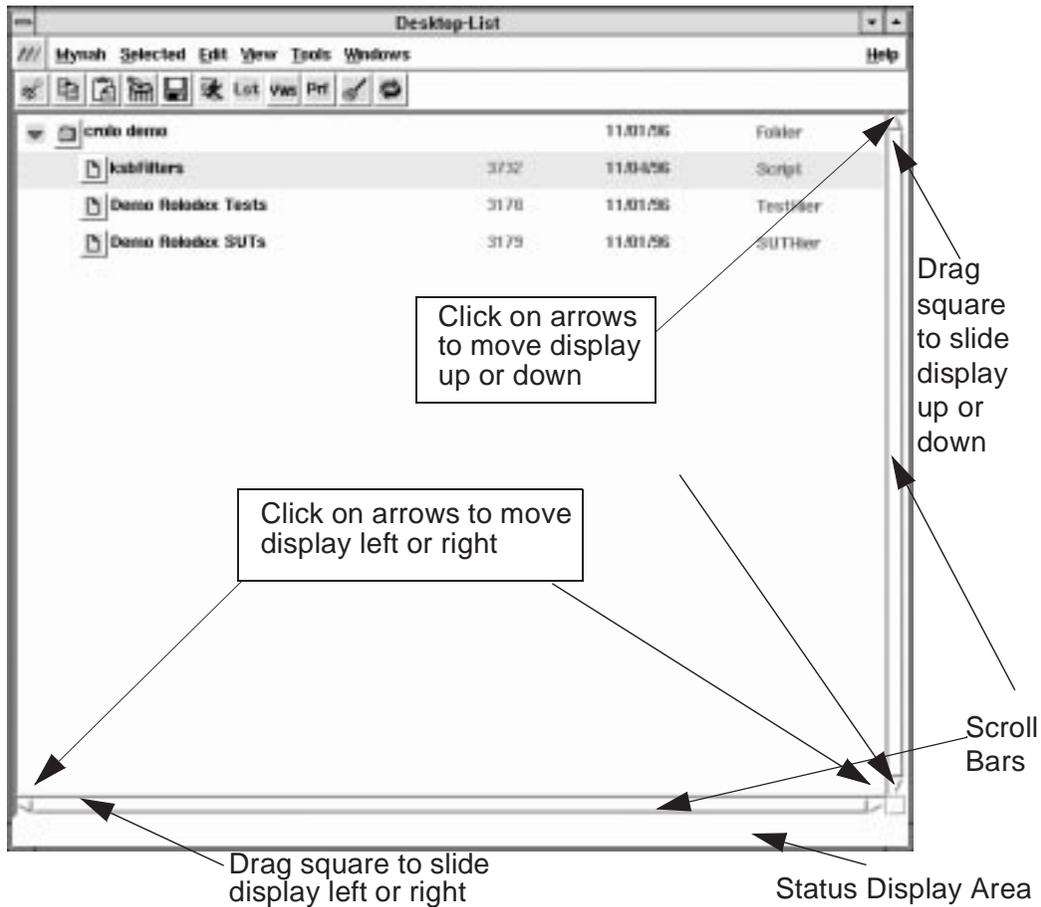
Figure 3-4. Using Scroll Bars

3.2   MYNAH Dialogs

The MYNAH System uses a number of Dialogs (often called dialog boxes) to help you perform test and task-automation activities. In general, dialogs are MYNAH fuction windows accessed from a Desktop which present you with a number of controls with which to perform actions supporting the functions of the Desktop window from which they were called.

MYNAH uses three basic type of dialogs:

- Dialogs that allow you to complete or control operations you initiated from a Desktop window or object view.

- Dialogs that allow you to 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 MYNAH System

The MYNAH System or 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 Desktop window to another is by making a menu selection or activating an icon. You used some of these controls in *Section 2: A Quick Tour Through MYNAH*, but here we will go through them in a more detail. This will help you when we give you instructions on how to use MYNAH features.

3.3.1 Using the Mouse

We will use certain terms when describing mouse actions throughout this manual. The table below describes what these terms mean. All these terms refer to action you take with the **left mouse button**. Do not use the **right mouse button** for any MYNAH 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 which appear in the client area. They also represent tools in the Toolbar. We have already discussed the icons in the Tool bar 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.

An arrow appears next to an icon if the icon represents a folder which contains other objects. For example, if we had a folder which contained a number of other objects, an arrow would appear next to it as we have illustrated below. 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 below. (See *Section 3.6 List View*)

Figure 3-5. MYNAH Icon

To transform an open object into an icon:

*   Click on the first menu and then click on the **Close** menu selection

To activate a window that has been iconized:

*   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. Often 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 the Tab Key

All text entry fields in the MYNAH GUI, with the exception of Script object Code view and Script Builder Code view, do *not* support the **Tab** key. When you hit the **Tab** key, the screen entry cursor advances to the next field in the **Tab** sequence.

3.3.6 Using Accelerator Keys

The MYNAH GUI provides accelerator keys. Accelerator keys are key combinations which you can use instead of mouse actions to make menu selections and perform other common actions. The common accelerator keys for the MYNAH folder appear in Table 3-3.

Table 3-3. Key Accelerators

Keys	Function
Ctrl+P	Print
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Ctrl+/	Select all objects
Ctrl+\	Deselect all objects
Ctrl+F4	Close
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   MYNAH GUI Controls

If you are familiar with common GUI controls, you can skip most of this section and go on to Section 3.4.6.

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   Option Menu

The MYNAH System presents you with option menus when there is a relatively short selection of items. Figure 3-6 shows a series of Option menu, used with the Mynah Default view.

Figure 3-6. Option Menu

You select items from this list by performing one of the following:

- Clicking on drop down list box and then on the menu selection you want.

- Dragging the mouse pointer from the drop-down icon to the selection you want and releasing the mouse button when the selection you want is highlighted.

3.4.2   Using Spin Buttons

Spin buttons are another GUI control that helps you to make settings. You use spin buttons to set values when the values are an ordered exclusive set. Figure 3-7 shows some typical spin buttons.

Decrease Value — Increase Value
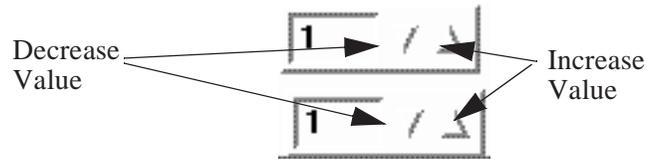
Figure 3-7. Spin Buttons

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

- Click on the up or down arrows which 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.

3.4.3   Using Toggle Buttons

Toggle button are simple on/off buttons. Toggle buttons can appear in a group or individually. Figure 3-8 shows a toggle button.

Toggle Button - OFF

Display Remote Session Connections

Toggle Button - ON

Display Remote Session Connections

Figure 3-8. 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.

3.4.4   Using Radio Buttons

Radio buttons allow you to select one option from a group of options. They are exclusive settings -- you can only select one from a group. Figure 3-9 shows a group of radio buttons with **All Code** selected.

Run:
- ● All Code
- Selected Code
- Code After Cursor
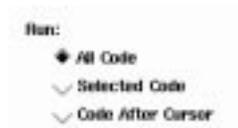
Figure 3-9. Radio Buttons

3.4.5   Using Pushbuttons

Pushbuttons are GUI controls used to execute actions that affect the entire window. They generally appear at the bottom of a MYNAH window or dialog. You activate a pushbutton by clicking on it. Figure 3-10 shows typical pushbuttons arranged at the bottom of a window.

Figure 3-10. Pushbuttons

Table 3-4 lists the pushbuttons used with the MYNAH System.

Table 3-4. Pushbuttons and Their Functions

PushButton	Function
OK	Acknowledges messages from the system and approves changes to settings. Usually closes the window.
Apply	Commits the current setting on a window without closing the window
Reset	Cancels changes made to the window content and returns the window to the last approved settings.
Stop	Stops any ongoing processes.
Continue	Continues a process that has been interrupted.
Retry	Allows users to retry a process interrupted by the operating environment
Pause	Suspends an ongoing process without terminating it.
Resume	Continues a process that was paused (See **Pause** above.)
Close	Closes a window without affecting a process.
Cancel	Closes a window without applying any changes to a window's content that have not been applied (See **Apply** above.)
Yes	Indicates a positive response to information contained in a message.
No	Indicates a negative response to information contained in a message.
Defaults	Use default values.
Help	Displays contextual help information for a window.

3.4.6   Using Ruler Column Headings

In many MYNAH window views and dialogs we use what are called Ruler Column Headings. Figure 3-11 shows a window from the **Database Browser** 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.

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 3-11 shows Script objects sorted in alphabetical order based on the "Owners" name.
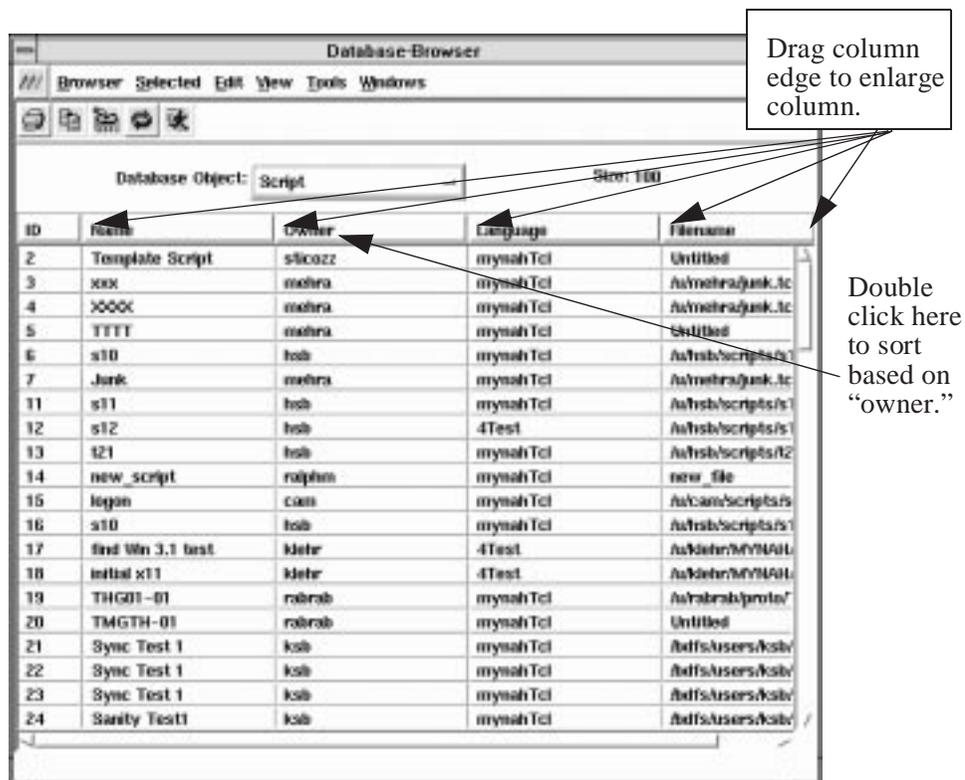
Figure 3-11. Database Browser Display Sorted by Script Name

3.5   What Are Views?

The MYNAH System uses what we call views. Viewss appear as a windows on your terminal screen when you open a MYNAH object. Views organize information about objects. Each views 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 allow you to enter information about objects and associate objects with other objects.

The MYNAH Desktop we showed in Figure 3-2 uses a List views to represent objects as line items with icons and text descriptions. This is one of three MYNAH Desktop views. The other two are Preferences views and Default view.

3.6   List View

List viewss are very important in the sense that List viewss are where you begin with the MYNAH System and they are views to which you will return often. List views show you all the objects available in a folder. Figure 3-12 shows a List view for the MYNAH desktop.
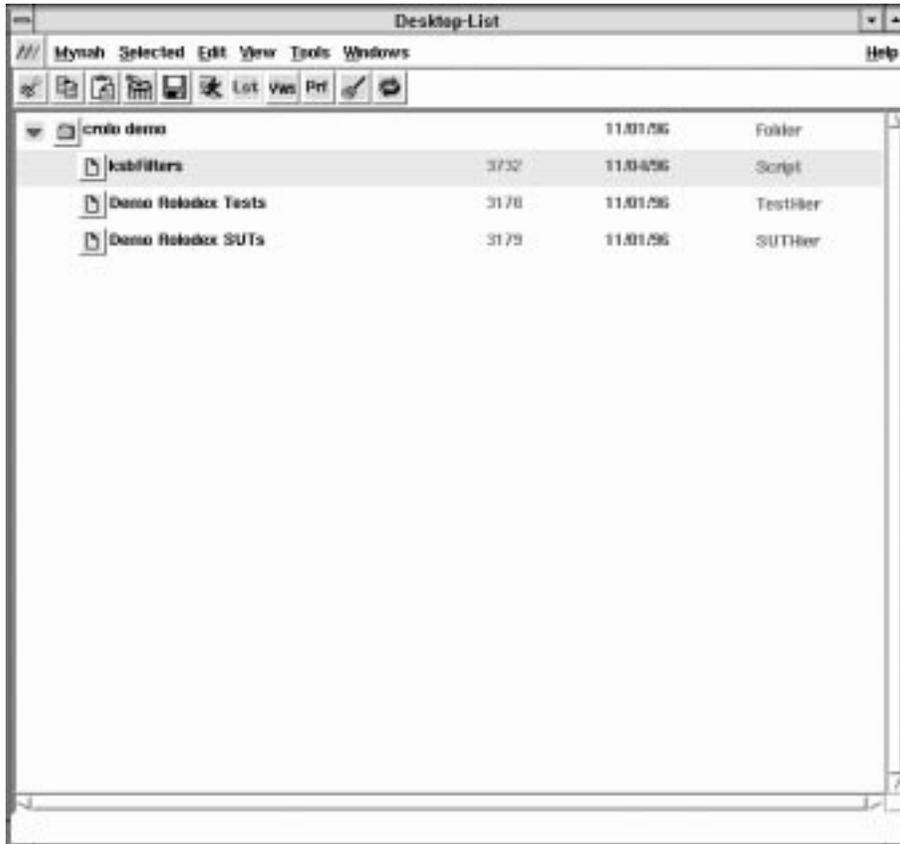
Figure 3-12. MYNAH Desktop - List View

The List view illustrated in Figure 3-12 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 we clicked on the arrow for icon labelled **crolo demo** the MYNAH Desktop List view would expand as shown in Figure 3-13. Note that the arrow next to the icon points down to indicate that this is an expanded display of the object.
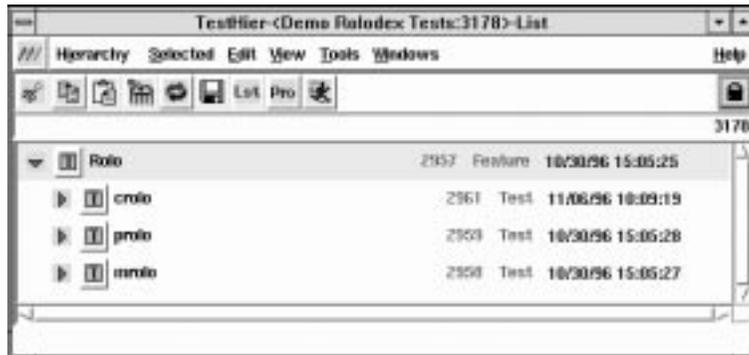
Figure 3-13. List View Expanded

Items that appear on the MYNAH 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

• Click on the **Selected** menu and then on **Expand Fully** and the system will display all
  the levels of a hierarchy, as shown in Figure 3-14.
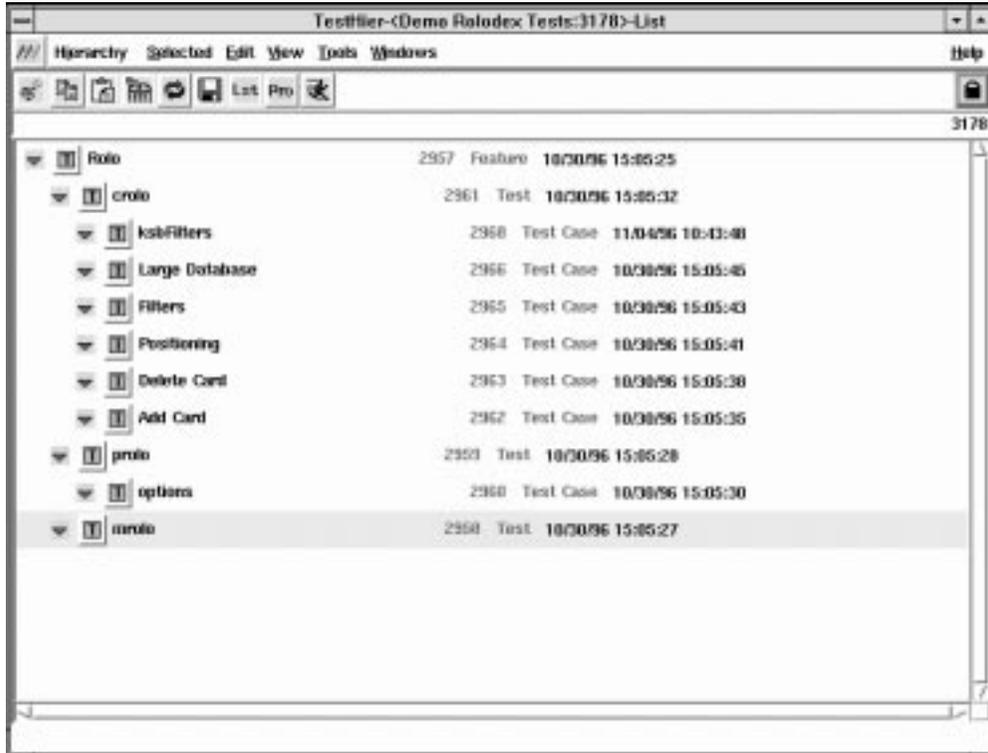
Figure 3-14. List View Expanded Fully

3.7   Preferences View

The MYNAH Desktop Preferences view allows you to set items that affect all objects and processing done during the session. Click on **Prf** from the Toolbar menu to get Desktop-Preferences View window. Figure 3-15 shows the Preferences View.
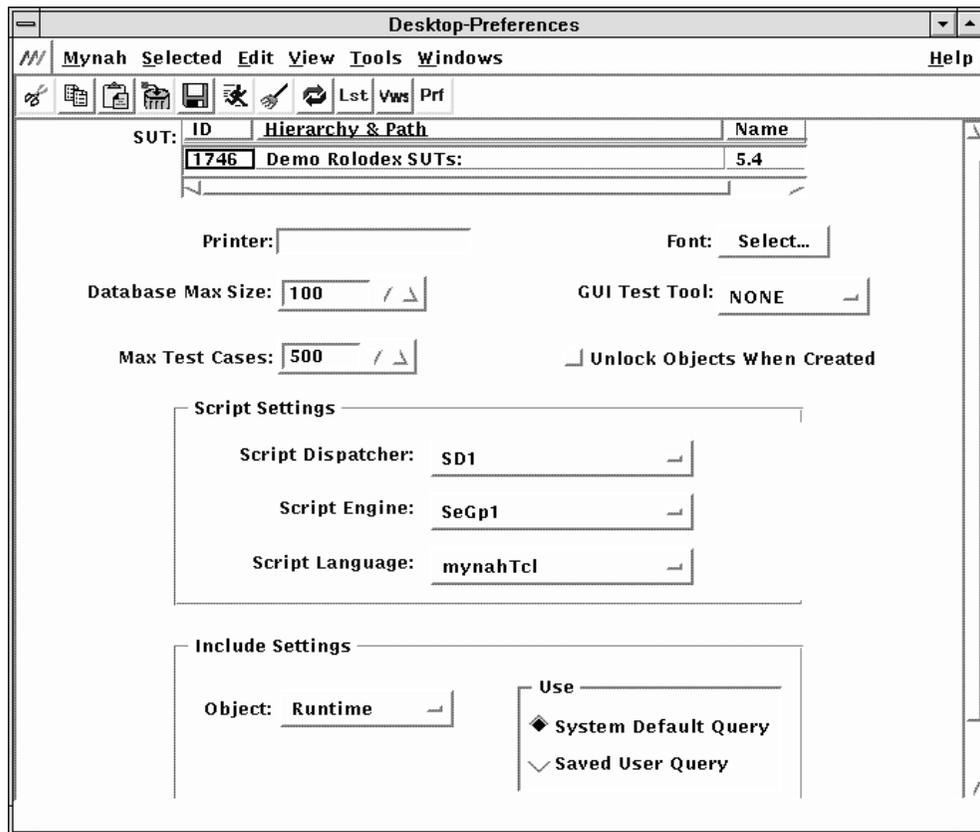
Figure 3-15. MYNAH Desktop - Preferences VIew

The Default Settings are:

SUT	Displays the default SUT which the system will use to pre-populate the Run and Reports dialogs. Alway check your SUT prior to running a script or report. The system will displays the default SUT's **Name** and its **Hierarchy & Path**.
Printer	Sets the MYNAH System printer. This selection will appear automatically on all print dialogs.
Font	Sets the font size for system displays.

Database Max Size	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). Refer to *Section 5: Using MYNAH Objects* for information about the Database Browser.
GUI Test Tool	Defines third-party test tool you are using. Choices are: **None**, **QA Partner**™, **XRunner**™, and **QC/Replay**™. This selection appears on the Tool menu. You can use it to start your GUI test tool.
Max Test Cases	Specifies the number of Test cases that the system will automatically load into a test object Matrix view.
	NOTE -- This option is associated with the AETG System only, see the *AETG Systems Users Guide* for further information.
Unlock Objects When Created	Allows you to set newly created objects automatically to be unlocked by the system when opened for the first time.
Script Dispatcher	Defines the default SD used for the session. The default value is **SD1**, but there can be others. The system uses this value when you are creating new Script objects.
Script Engine	Defines the default SE used for the session. The default value is **SeGp1**, but there can be others. This is a user definable setting so check with your System Administrator for valid choices. The system uses this value when you are creating new Script objects.
Script Language	Defines the scripting language used for the session. Choices are: **mynahTcl**, **4Test**, **TSL**, **ReplayTcl**, and **others**. This will be the setting for any new script you create.
Object	Selects a Runtime or Result object so you can set a system or user defined query to populate the Database Browser and Job Status window. This setting is used in conjunction with the next two settings. We explain more about this in Section *3.7.1*.
System Default Query	Specifies that the system-defined query should be used to populate the Database Browser and Job Status window for Runtime and Result objects.
Saved User Query	Specifies that a saved, user-defined query should be used to populate the Database Browser and Job Status window for Runtime and Result objects.

3.7.1 Selecting System- or User-defined Queries

The MYNAH System uses a default query to populate the Database Browser and Job Status window for Runtime and Result objects. Each time you start the MYNAH System, the default include query determines today's date and the date one week prior to today. It uses these dates as the basis of its query for Runtime and Results objects.

You can select an include query that you yourself have defined. You define Include queries using the Include dialog which you can access from the Database Browser or Job Status window. We explain how to use the Include Dialog in *Section 5: Using the Database Browser.* (The Include dialog operates the same for the Database Browser and Job Status window.)

If you define queries for Runtime and Result objects, the next time the system starts up, it will use your user-defined query. User-defined queries override system-defined queries in this case, but remember, the system-defined query is the only way you can use dynamically defined dates as the basis of an Include query.

If you check **System Default Query** on the Preferences views, the system defined query will override any query you have created. If you want to change back to your query, you can return to the Preferences view and check **Saved User Query**.

3.7.2 Saving Default Settings

You can change default values. As an example, let's go through the default settings as if we were setting them. We won't change the default SUT since that involves using the Database Browser which you will learn about in *Section 4: Using MYNAH Objects*, but we will explain how to set all the other preferences.

1. Position the pointer in the **Printer** data entry area and type in a printer name, e.g., **printer1**.

2. To select a font, see Section 3.7.3.

3. Position the pointer in the **Database Max. Size** data entry area and type in a number, e.g., **100.**

4. Click on the **Gui Test Tool** drop-down list and click on the test tool you want, e.g., **None**.

5. Check **Unlock Objects When Created** if you want the system to automatically open newly created objects. (We checked this.)

6. Click on the **Script Dispatcher** drop-down list and click on the SD you want, e.g., **SD1**.

7. Click on the **Script Engine** drop-down list and click on the SE group you want, e.g., **SeGp1**.

8. Click on the **Script Language** drop-down list and click on the scripting language you want, e.g., **mynahTcl**.

9. Click on the **Object** drop-down list and select **Runtime** or **Result**. (We will leave the default query, so we don't need to select an object.)

10. Check **System Default Query** or **Saved User Query**. (We left **System Default Query** checked.)

If you save the Preferences view, the settings will become the default settings for all subsequent MYNAH 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.

If you want to save these preferences:

- Click on the **Mynah** menu and then on the **Save** menu selection to save the value you just selected.

3.7.3　Selecting The Default Fonts

We skipped over setting the default font in the procedure above. We will cover it here. To change the default font:

1. Click on the **Font** Select dialog which appears in the client area of the Preferences view.

   — The system will display the Font Chooser dialog shown in Figure 3-16. The top panel of the dialog shows the alpha-numeric 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 Select dialog.
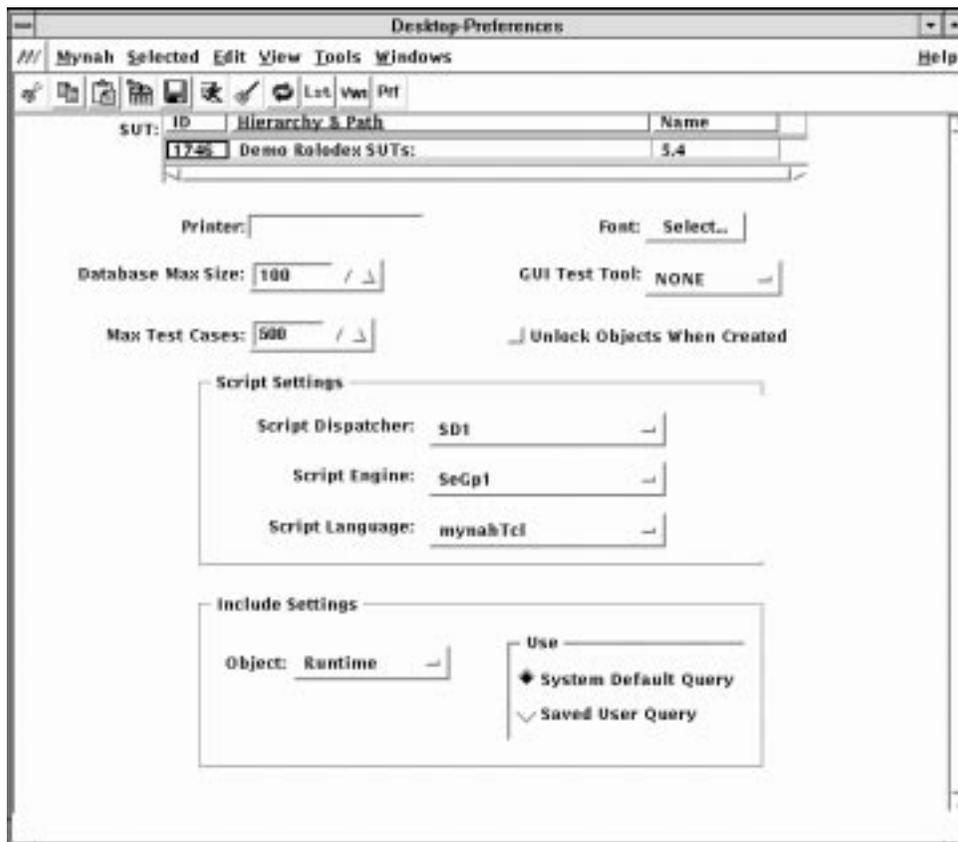
Figure 3-17. Preferences View with Font Change

3.7.4   Using the Default Views

The Default Views, Figure 3-18, allows you to set the default views for objects, folders, and Tools. Default views means that the view you select here will appear when you open an object of the selected type. For example, when you first accessed the MYNAH System, you saw the Mynah List (Desktop) view. This was because the Desktop -List view is the default view for the Mynah System.
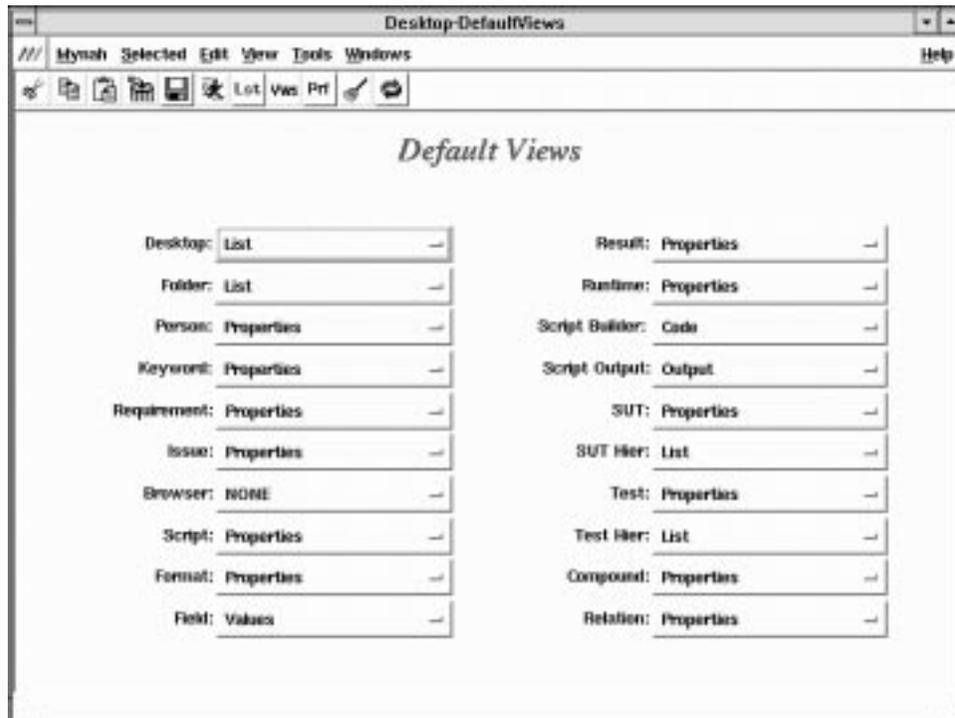
Figure 3-18. MYNAH Default Views

There are reasons why you may want to change these Default views. For example, by default, the Script object Properties view is the first to appear when you open a Script object. However, much of the work you do when developing scripts will be done with the Code view. It may save you a little time to have the Code view as the Default view.

To change the Default view for an object:

1. Click on the **Tools** drop-down list for an object. For example, click on the **Script Builder** drop-down list.

2. Select a view. For example, select **Code**.

3. Repeat Step 1 for each object.

4. To save any changes you make, select the **Save** menu option from the **Mynah** menu.

    — The system will open objects to the Default View you set here.

3.8   Changing a MYNAH View

You can change your view during a session using the **views** menu or by clicking on the **VWS** 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:

*   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 Mynah Desktop.

OR

*   Click on the **View** menu and then on the **Change View** menu selection.

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

3.9   Creating Folders

As we said earlier, you can create Folder Objects and use them to organize your work. We will use our **crolo** application example to illustrate creating a folder. We will create a folder where we will place all the items needed to document, develop, and analyze tests for the **crolo** application. We will name this folder **crolo demo**.

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

1. First to ensure nothing is selected on your Desktip, click on the **Edit** menu and then on the **Deselect All** menu.

2. Click on the MYNAH or **Selected** menu options and then on the **New** menu selections

    — The MYNAH System will display a cascade menu.

3. Select **Folder** from the cascade menu.

— The system will display a dialog requesting a name for the new folder. See Figure 3-19..
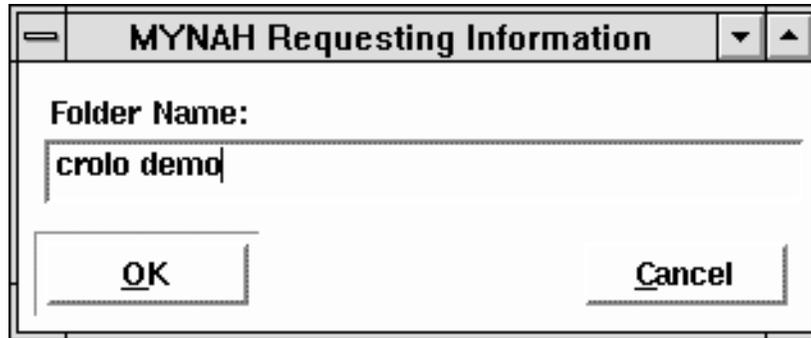
Figure 3-19. Requesting Information Dialog

4. Type in a name, e.g., **crolo demo**, in the **Folder Name** field.

5. Click **OK**.

— The MYNAH System will display the icon for the new folder. See Figure 3-20.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.

New
Folder
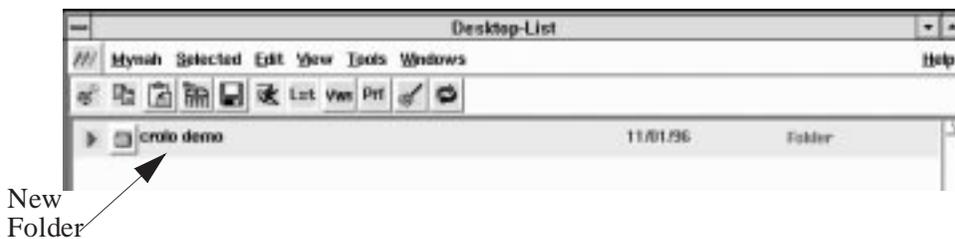
Figure 3-20. New Folder Icon

You can open this folder by double clicking on its icon in the List View.You can create objects from within a folder via the **Tools** menu, or copy objects into it using the **Database Browser**. We explain how to use the Database Browser in the next chapter *Section 4: Using MYNAH Objects.*

> **NOTE —** However, SUT and Test objects cannot appear in ordinary folders. They must appear in their own special folders called hierarchies. Refer to *Section 7: Defining System-Under-Test (SUT)* and *Section 8: Using Test Objects* for more information about this.

3.9.1   Saving Folders

You save folders by clicking on the **Folder** menu and then on the **Save** menu selection.

3.9.2   Deleting Folders

You can delete a folder the same way you delete any other object. We will explain how to do this in *Section 4: Using MYNAH Objects*. However, we want to note here that you must 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 Desktop Properties View. To access this view, perform one of the following:

- Open the Folder by double clicking on the icon to the left of the folder name. The Properties View screen will come up.
  OR

- Click on the **View** menu, then on the **Change View** menu selection and finally, select **Properties.**

- Click on the **Pro** icon.

The system will change the view to the one shown in Figure 3-21.

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.

Figure 3-21. Folder Properties View

For now, to change the name of a folder:

1. Erase the old name.

2. Type in a new name.

3. Click on the **Folder** menu and then on the **Save** selection to save the folder under the new name.

3.10   Printing

The MYNAH 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 Script objects this is the **Script** menu. When you do this, the system will display the dialog shown in Figure 3-22.

Figure 3-22. 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 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 name of the file in the **File** data entry area.

3. After you make your selections, click **OK** to print.

Figure 3-23 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.
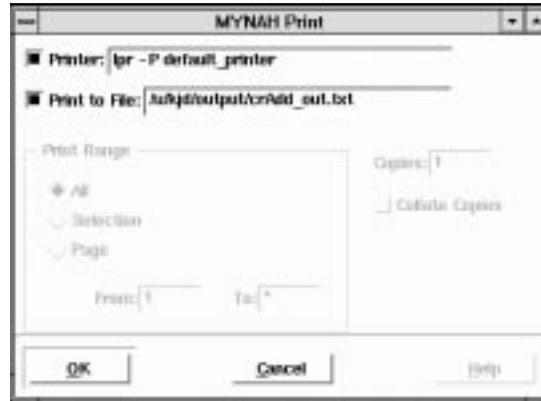
Figure 3-23. Example Print Dialog Box

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 3-24 and 3-25 show the same excerpt of the printout generated by a Test Object. Figure 3-24 shows an example of a printout generated on a printer using fixed width fonts.

```
**TEST ASSOCIATIONS VIEW**

Associated SUTs:
ID          Hierarchy & Path                        Name
----------------------------------------------------------------------
1744        Demo Rolodex SUTs:5.3                   First
1745        Demo Rolodex SUTs:5.3                   Second

Associated Keywords:
ID          Type            Keyword
----------------------------------------------------------------------
1080        General         defaults_db

Associated Issues:
ID          Type            Issue
----------------------------------------------------------------------
```

Figure 3-24. Printout Using Fixed Width Fonts

Figure 3-25 shows an example of the same output generated on a printer using variable width fonts.

```
**TEST ASSOCIATIONS VIEW**

Associated SUTs:
ID       Hierarchy & Path              Name
----------------------------------------------------------------------
1744     Demo Rolodex SUTs:5.3         First
1745     Demo Rolodex SUTs:5.3         Second

Associated Keywords:
ID       Type         Keyword
----------------------------------------------------------------------
1080     General      defaults_db

Associated Issues:
ID       Type         Issue
----------------------------------------------------------------------
```

Figure 3-25. Printout Using Variable Width Fonts

As you can see, the headings and the data in Figure 3-25 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   Where to Go Next

If you have tried the MYNAH System as described in *Section 2: A Quick Tour Through MYNAH* and are comfortable with the Basics of MYNAH. We've covered in this section folders, navigating around the MYNAH System, the GUI controls, and routine tasks, now you can go on to *Section 4: Using MYNAH Objects.*

4. Using MYNAH Objects

We introduced you to the concept of Views and what they are in the MYNAH System in the last section. 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

- Object hierarchies

- Creating new objects

- Opening objects for editing and updating

- Deleting objects

4.1 What Are Objects?

MYNAH objects represent all the entities you need to manage your testing and task-automation efforts. They represent entities such as a requirements, tests, or scripts. Information contained in objects are called attributes. Attributes are pieces of information that define the object and are used by the MYNAH System to manage testing and task automation. 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 it. Table 4-1 lists all the MYNAH objects.

The MYNAH 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.

Objects are like database records that store information. Attributes are like fields within a database record.

Appendix B: MYNAH Objects and Their Attributes and Associations provides a comprehensive list of all objects and their attributes.

Table 4-1. MYNAH Objects

Object	Function
Script	Documents a script. One attribute of a Script object is a reference to a file containing Tcl statements.
Result	Contains the results of running a test.
Runtime	Contains a record of a script's execution.
Issue	Allows you to record issues.
Keyword	Allows you to create user-defined Keywords with which you can reference other objects.
Requirement	Defines test requirements for a SUT.
SUT	Describes a System Under Test.
SUT Hierarchy	Allows you to create and store SUTs.
Test	Defines a test.
Test Hierarchy	Allows you to create and store Tests.
Person	Identifies a MYNAH user.

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 allow you to enter information. A single object usually has several views each of which displays a different set of an object's attributes. Views allow you to enter information into and retrieve information from the object. The MYNAH System processes this information in the course of helping you with your activities.

4.2   MYNAH Hierarchies

SUT and Test objects always appear in their own SUT and Test hierarchies. Object hierarchies are useful for organizing your work. Hierarchies are relationships between 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. Figure 4-1 illustrates this idea.

Object Hierarchy   UNIX Directory and Sub-Directory
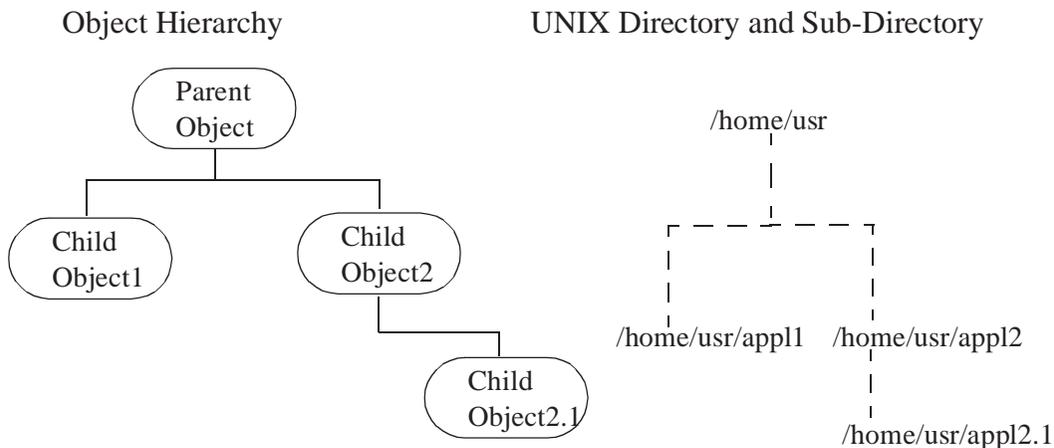
Figure 4-1. Hierarchy Idea

4.2.1   Test and SUT Hierarchies

There are two specialized hierarchies available through the MYNAH System - Test and SUT. We call these specialized because only one type of object can appear in them. Test hierarchies contain only tests and they are the only place in the MYNAH System where you can place test objects. The same is true for SUT hierarchies. Only SUT Objects can appear in SUT hierarchies. You can, however, have several Test or SUT hierarchies and you can place the hierarchies in folders you've created.

Figure 4-2 illustrates a Test object hierarchy. As you can see, hierarchies are most obvious when we look at them in an expanded list view. This is the test plan for the Rolo application. Note how we arranged **crolo**, **prolo**, and **mrolo** as child objects of the parent **Rolo** and how two of these children (**crolo** and **prolo**) have child objects of their own.

We will explain here briefly how to create SUT and Test Hierarchies. We will explain this again in detail in *Section 7: Defining a System-Under-Test (SUT)* and in *Section 8: Using Test Objects*.
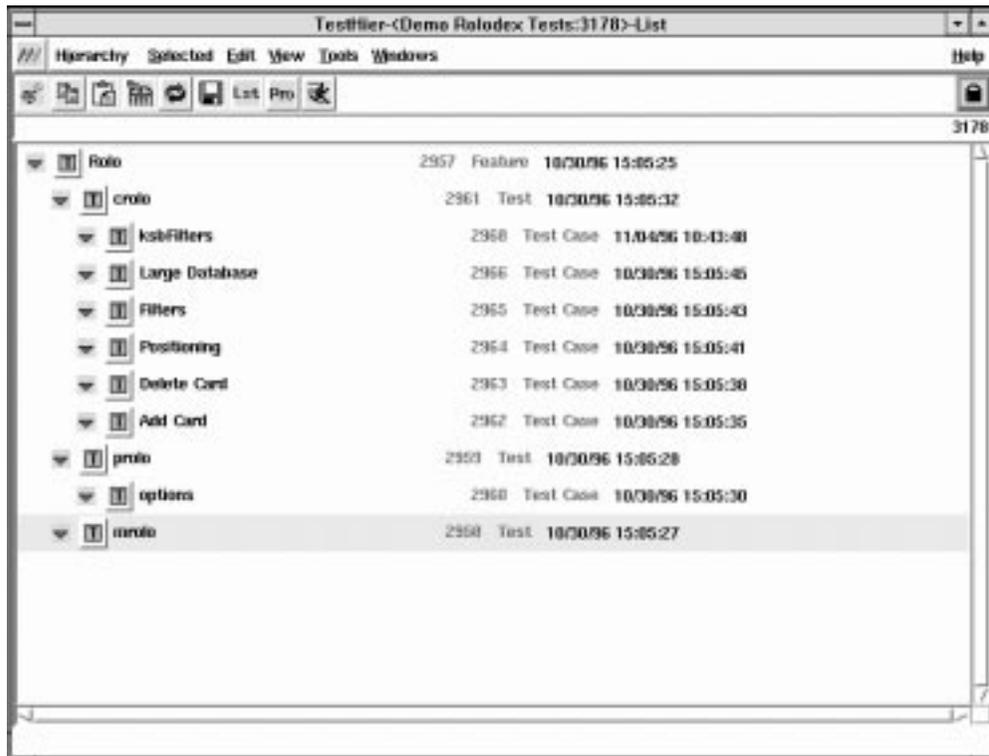
Figure 4-2. Test Object Hierarchy

4.2.2   Creating and Naming Test and SUT Hierarchies

You can create a SUT or Test hierarchy from the MYNAH Desktop-List folder or any other folder you have created. Creating a hierarchy involves creating it from a folder and giving it a name. To create a new hierarchy:

1. Click on the **Mynah** menu option and then on the **New** menu option.

2. After the system displays the **New** menu list, click on the hierarchy you want, e.g., **Test Hierarchy** or **SUT Hierarchy**.

— The system places the new hierarchy in the folder. For example, if we added a SUT hierarchy to the **crolo demo** folder, it would appear similar to the one we show in Figure 4-3.
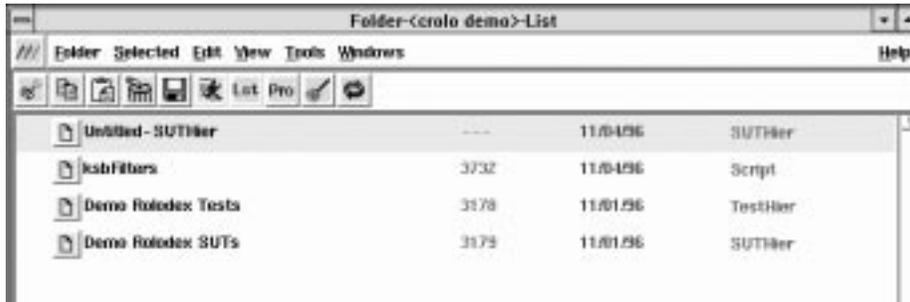
Figure 4-3. New SUT Hierarchy.

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

— The system displays the SUT hierarchy List view.
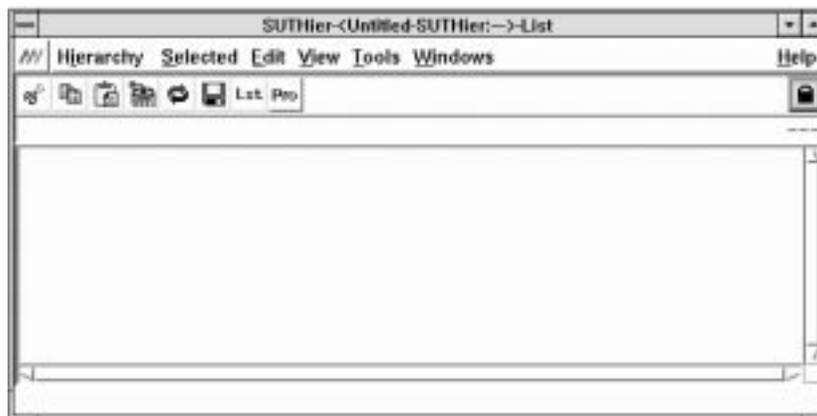
Figure 4-4. SUT Hierarchy List View

4. Click on the Lock icon to unlock the SUT hierarchy.

    — The system displays a Name dialog for the Hierarchy you have created. See
      Figure 4-5.
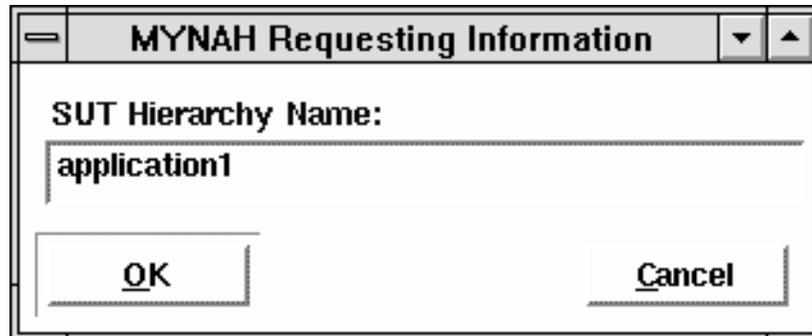
Figure 4-5. Name Hierarchy Dialog.

5. Type in a name for the new SUT hierarchy, e.g., **application1**, and click **OK**.

    — The system will dismiss the Name dialog.

6. To save the named hierarchy, click on the **Hierarchy** menu and the **Save** option.

For now we won't go into the other attributes for SUT hierarchies. We will discuss these in
Section 7: Defining a System-Under-Test (SUT). We discuss adding Test or SUT objects to
a hierarchy in Section 4.3.3.

After you add an object to a Test or SUT hierarchy, you must save the hierarchy before you
can open any newly created objects. The MYNAH System is programmed to react this way
so that other users can access a hierarchy without disrupting its structure.

To save a hierarchy, click on the **Test** or **SUT** menu in the menu bar and then on the **Save**
option.

4.2.3   Deleting Hierarchies

You delete hierarchies the same way you delete any other MYNAH Object. See Section 4.7
for information about deleting hierarchies. The only persons who can delete a hierarchy is
the user who created it or the MYNAH 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 MYNAH Database. No
matter who owns them! The system will always remind you of this with a confirmation
dialog.

4.3   Creating Objects

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

4.3.1   Creating Objects on the MYNAH Desktop

To create an object on the MYNAH Desktop:

- Click on the **MYNAH** menu option, then on the **New** menu selection, and then on the object type you want. For example, on the MYNAH Desktop List view, click **MYNAH**, then **New,** and then choose **Script**.

  — The system will place the new object in the first position on the MYNAH Desktop. See Figure 4-6.
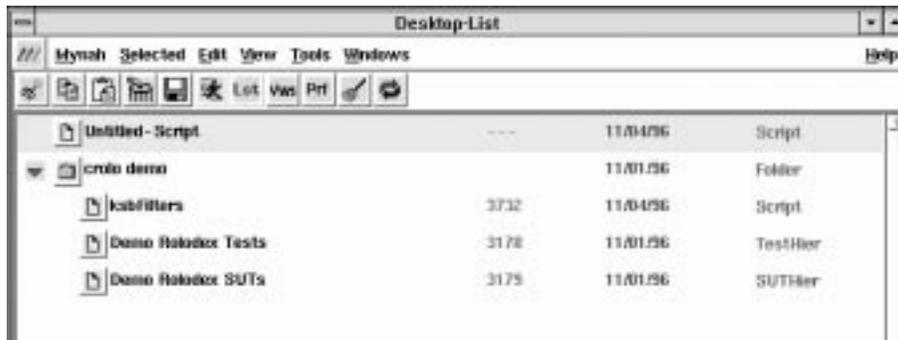
Figure 4-6. New Script Object on MYNAH Desktop

4.3.2   Creating New Objects in Folders

You can create objects in both opened and closed folders.

With *opened* folders:

1. Open the folder by double clicking on its icon.

   — By default the system will display the properties view. Choose the List view to create new objects.

2. Click on the **Folder** menu option, on the **New** menu option, and then the object type you want.

With *closed* folders from MYNAH Desktop-List view:

1. Select the folder from the MYNAH Desktop-List view.

2. Click on either the **MYNAH** or **Selected** menu options choose **New** menu selection, and then on the object type you want.

    — The System will expand the folder's listing on the MYNAH Desktop-List view and place the object in the first postion. See Figure 4-7.
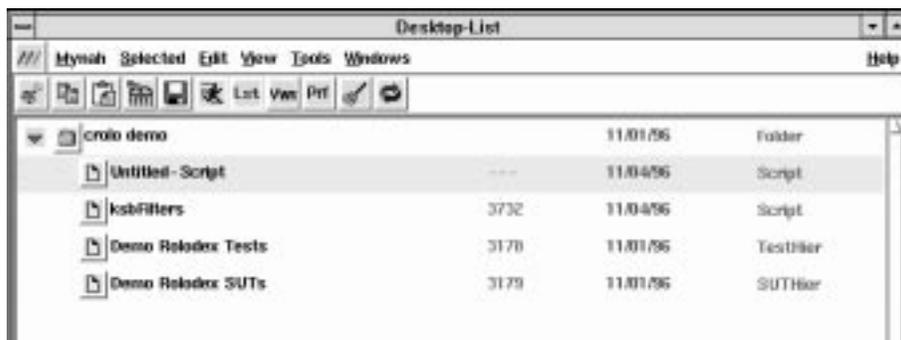
Figure 4-7. New Object Created For a Closed Folder.

4.3.3 Creating Test and SUT Objects in Hierarchies

Test and SUT objects can be created at the first level or at any level below that in a hierarchy. We will continue the example we started in Section 4.2.2 by creating SUT objects in the SUT Hierarchy we created.

> **NOTE —** When creating a New object you have two menu options to choose from. If you want a parent object, you must Deselect All, then choose from the **Menu bar** option (e.g. **Folder, Hierarchy or SUT**) the oject you want. If you want to create a child object select the parent object then choose **Selected** from the menu options the the object you want.

To create an object at the first level of a folder or hierarchy:

1. Open the hierarchy in which you are placing the new object. (We used the hierarchy we created in Section 4.2.2.)

2. Make sure that no other objects are selected. If you need to, select **Deselect-All** from the **Edit** menu to clear all selections. (We didn't have to worry about this because there were no other objects in the hierarchy.)

3. Unlock the hierarchy by clicking on the Lock icon.

4. Click on the **Hierarchy** menu and then on the **New SUT** option.

— The system placed the new Untitled- SUT Object in the Hierarchy. See Figure 4-8.
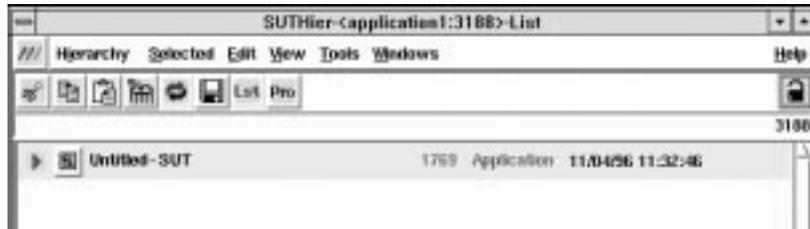
Figure 4-8. New SUT Object Dialog

To place a new object at a level below (as a child) the first level of a folder or hierarchy:

1. Select the object *above* which you want the new object to appear. For our example we selected the object we just created.

2. Click on the **Selected** menu, then on the **New SUT** option.

— The system placed the new object in the SUT Hierarchy below the first SUT Object. Note how it is indented indicating that it is a child object of the other SUT object we created. See Figure 4-9.
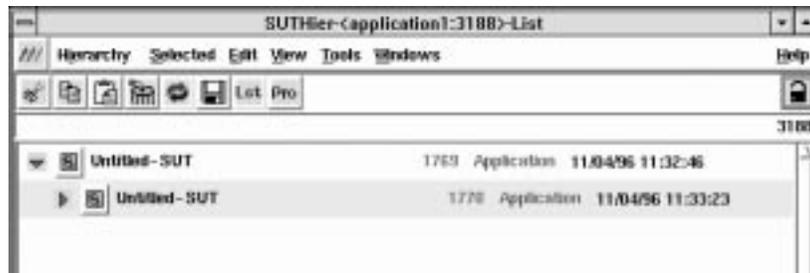
Figure 4-9. New Object at Second Level

If we wanted to place another object at the first level, we would:

1. Make sure that no other objects are selected. If you need to, select **Deselect-All** from the **Edit** menu to clear all selections.

2. Click on the **Hierarchy** menu and then on the **New SUT** option.

   — The system placed a new Untitled- SUT object at the first level in the Hierarchy. See Figure 4-10.
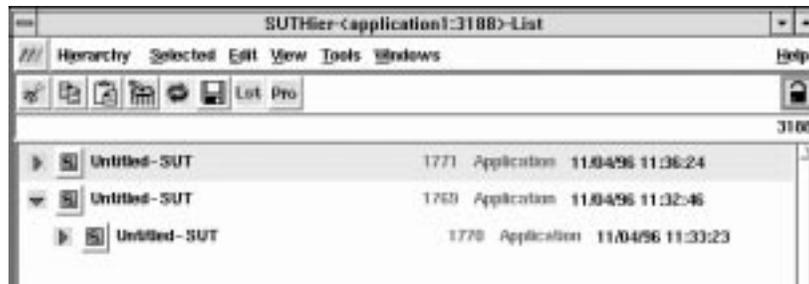
Figure 4-10. Another New Object at First Level

3. Save the hierarchy by clicking on the **Hierarchy** menu and then on the **Save** option. (Normally, you would open the new SUTs and fill in their attributes at this point.)

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, click on the **Selected** menu and then on the **Open** menu selection

- Select the object and press **alt** and *s*, then *o*

The MYNAH System will open the object to a default view unless you have changed the default view with the MYNAH Preferences view, then 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. Click on the **Edit** menu and then on the **Copy** option.

To copy an opened object we would:

 • Click on the **Edit** menu and then on the **Copy object** option

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

> **WARNING —** Copy/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 Test or SUT object within a folder or hierarchy, you must use **Duplicate**. **Duplicate** creates a new object with many of the attributes of the original. It gives the new object the name of **CloneOf <id>** and 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.

3. Click on the **Edit** menu and then on the **Duplicate** option.

 — The system will place an object labelled **CloneOf** in the hierarchy as a sibling of the object you duplicated. Figure 4-11 illustrates this.

Original
Object

New
Duplicated
Object
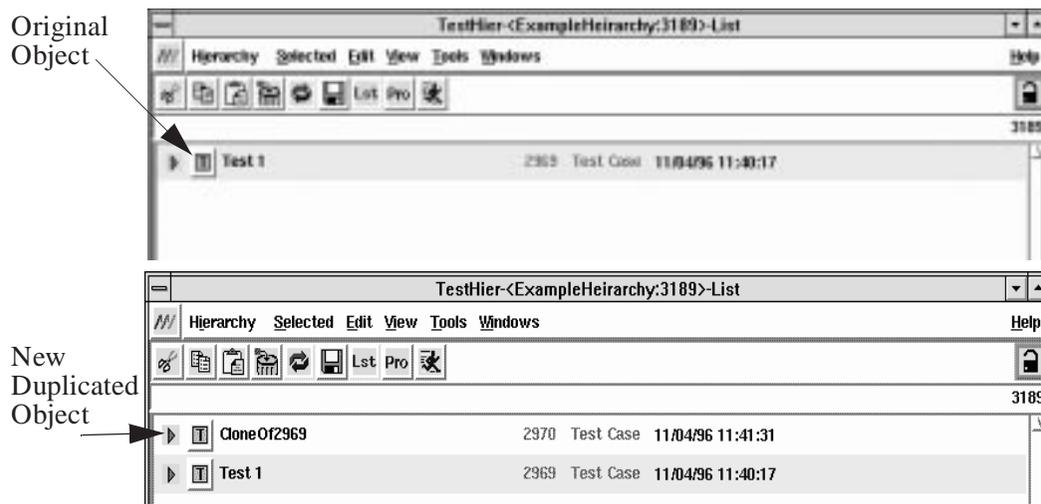
Figure 4-11. Duplicating Objects

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 actually create a new object, They create a link between the existing object and a new location (folder or the MYNAH desktop) so that you can access the object from the new location.

- **Duplicate** actually creates a new object in the MYNAH 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 over to the new object when you use Duplicate.

Table 4-2. Attributes Copied by Duplicate

Object	Attributes
Test	Keyword Associations, SUTs Associations, Test Importance Test Effort, Priority, Type, Description, Dependencies
SUT	Test Associations, Type

4.7 Deleting Objects

When we speak of deleting objects we mean deleting objects from the MYNAH database. You can remove objects from the Mynah 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 or SUT parent oobjectbject, child objects are deleted.

- Any Runtime objects associated with a Script object are deleted.

- Any Results objects associated with a test are deleted.

- All of an object's associations are removed.

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

To delete an object from a MYNAH database:

1. Select the object you want to delete.

B. Click on the **Selected** menu and then on the **Delete** option.

> **NOTE —** 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 MYNAH Database and from the display.

4.8 Removing Objects from the MYNAH GUI Display

As we mentioned earlier, you can remove objects from the GUI or Desktop 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
MYNAH Database. Figure 4-12 shows the Status display area for a Script Object.
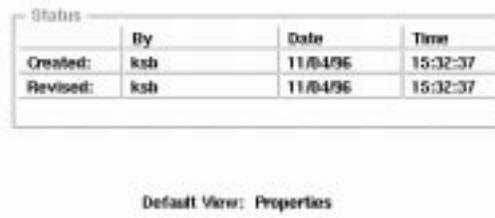
Figure 4-12. Object Status Display and Default View Area

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 MYNAH Desktop's Preference view.

4.10 Modifying Another Person's Objects

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

> **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   Where to go Next

If you have looked at Sections 1, 2, and 3, you should be fairly familiar with the MYNAH Test System. You can go on to the next section, *Section 5: Using the Database Browser* to learn about a very useful tool for managing objects.

5. Using the Database Browser

The MYNAH System provides you with three tools which you access from the **Tools** menu. These are the **Database Browser**, **Job Status** window, and **Script Builder**. We describe the **Script Builder** in its own section *Section 12*. We describe the **Job Status** Window in *Section 10*. Here we will describe the **Database Browser**.

> **NOTE —** In addition to these three tools, the **Tools** menu lets you start a GUI test tool if you have installed one.

In this section we will 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

As you use the MYNAH System, you will create many objects to support your activities. These objects will be stored in the MYNAH Database. The Database Browser provides you a quick and easy way to locate objects.

The Database Browser is important for many of the activities you may engage in while using the system. It is the chief tool for managing objects, since it allows you to display, open, and run all existing objects. It is also the tool you may use to associate objects with other objects.

5.2   Accessing the Database Browser

You access the Database Browser from the **Tools** menu by clicking on the **Database Browser** menu selection. When you do this, the system displays a **Database Browser** main window like the one shown in Figure 5-1.

Figure 5-1. Database Browser Window

5.2.1   Displaying Object Types with the Database Browser

By default the *Keyword* object is the first selected object type the system displays, but you can select any object you want using the **Database Object** option list.

For example, if you wanted to see Script objects, you would simply click on the **Database Object** option list and select **Script**. If you are using the default query for the Database Browser, the system will display the dialog shown in Figure 5-2.
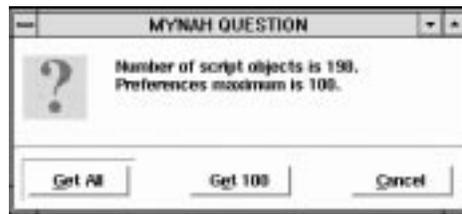
Figure 5-2. MYNAH Question Dialog

You can choose to have the system display all the objects of that object type or display the maximum you specified using the MYHAH Desktop Preferences View. Refer to *Section 3: MYNAH Basics* for information about this setting.

After you respond to the MYNAH Question dialog, the system displays a list of script 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. Refer to *Section 3: MYNAH Basics* for an example of doing this.
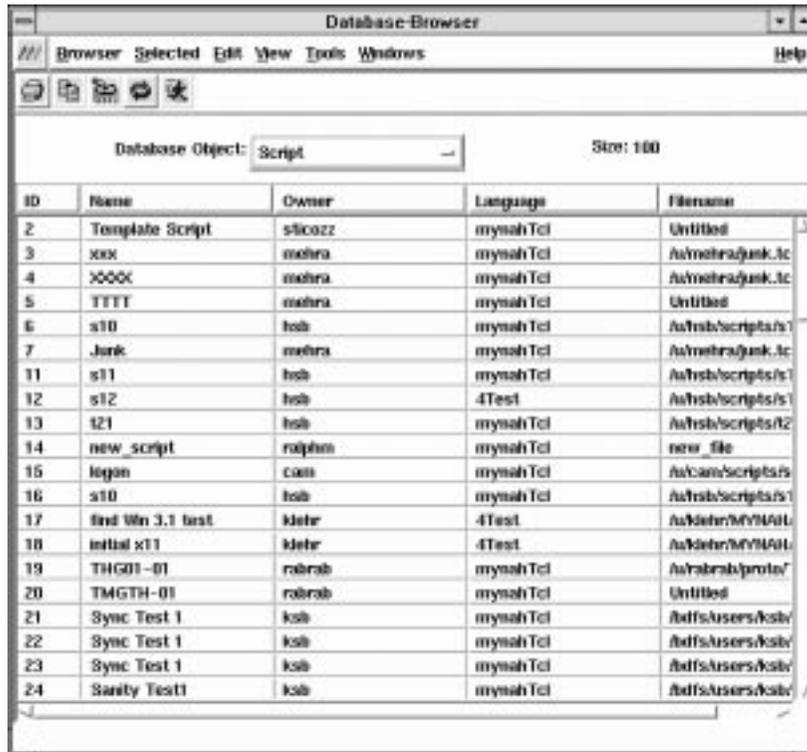
ID	Name	Owner	Language	Filename
2	Template Script	sticozz	mynahTcl	Untitled
3	xxx	mehra	mynahTcl	/u/mehra/junk.tc
4	XXXX	mehra	mynahTcl	/u/mehra/junk.tc
5	TTTT	mehra	mynahTcl	Untitled
6	s10	hsb	mynahTcl	/u/hsb/scripts/s1
7	Junk	mehra	mynahTcl	/u/mehra/junk.tc
11	s11	hsb	mynahTcl	/u/hsb/scripts/s1
12	s12	hsb	4Test	/u/hsb/scripts/s1
13	t21	hsb	mynahTcl	/u/hsb/scripts/t2
14	new_script	ralphm	mynahTcl	new_file
15	logon	cam	mynahTcl	/u/cam/scripts/s
16	s10	hsb	mynahTcl	/u/hsb/scripts/s1
17	find Win 3.1 test	klehr	4Test	/u/klehr/MYNAH/
18	initial x11	klehr	4Test	/u/klehr/MYNAH/
19	THG01-01	rabrab	mynahTcl	/u/rabrab/proto/
20	TMGTH-01	rabrab	mynahTcl	Untitled
21	Sync Test 1	ksb	mynahTcl	/bdfs/users/ksb/
22	Sync Test 1	ksb	mynahTcl	/bdfs/users/ksb/
23	Sync Test 1	ksb	mynahTcl	/bdfs/users/ksb/
24	Sanity Test1	ksb	mynahTcl	/bdfs/users/ksb/

Figure 5-3. Database Browser Window with Scripts Displayed

BR 007-252-006   **MYNAH System Users Guide**
Issue 3, October 1997   **Using the Database Browser**
Release 5.2

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.

> **NOTE —** The Include dialog is also available for Job Status Window (Section 10.7.5) and Result object (Section 14.2).

You access the Include dialog by clicking on the **View** menu and then on the **Include** menu selection. 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 Keywords in Figure 5-5 is **name is Inventory END**. This will search for Keyword named Inventory.
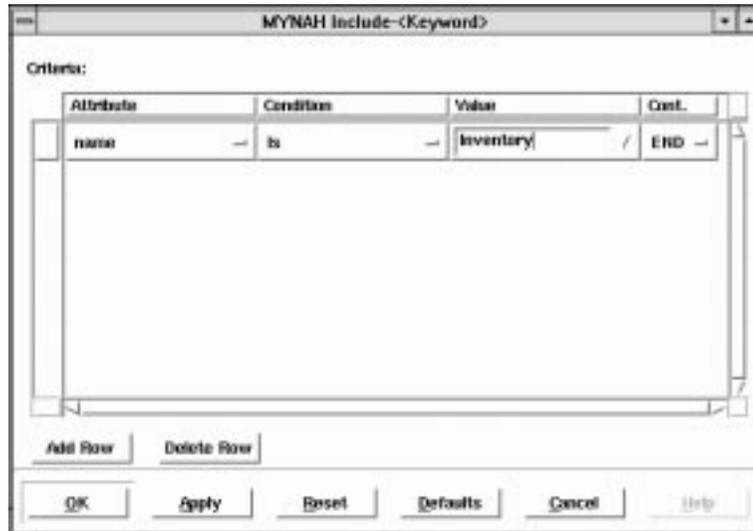
Figure 5-5. Include Query Example

5.2.2.1   Object Attributes and Default Queries

Table 5-1 shows the attributes used with objects and the default include queries

> **NOTE —** The brackets around attributes only in the subsequent table mean that you use an object ID in the *Value* column instead of a string. The user should enter the object's system ID in the *Value* column instead of the object name..

Table 5-1. Include Attribute and Default Query (Sheet 1 of 2)

Object Type	Attributes	Default Query	What Query Does
Issue	id, name, external ID, OwnedBy, State, type	**OwnedBy is <login>**	Displays all Issue objects you own.
Keyword	id, name, type	**None**	Displays all Keyword objects.
Person	id, authority, firstname, keyword, lastname.	**Authority NOT EQUAL "Inactive"**	Displays all Person objects except Inactive.

Table 5-1. Include Attribute and Default Query (Sheet 2 of 2)

Object Type	Attributes	Default Query	What Query Does
Requirement	id, name, type keyword	**None**	Displays all Requirement objects.
Result	id, activity, WhenCreated, Source, ReasonCode, status, <test>	**CreatedBy is <login> AND** **WhenCreated Greater Than <date>AND** **WhenCreated Less Than or Equal To <date>**	Displays all Results objects you own that were created between the defined dates. The dates define the week preceding the current date, including the current date.
Runtime	id, ResultCode, SD, SeBegin, SeEnd, SeGroup, status, <SUT>, WhenInit, Who	**Who is <login>** **Wheninit Greater Than <data>** **Wheninit Less Than or Equal To <data>**	Displays all Runtime objects you created between the defined dates. The dates define the week preceding, including the current date.
Script	id, CreatedBy, filename, keyword, Language, name, OwnedBy, RevisedBy, WhenCreated	**OwnedBy is <login>**	Displays Script objects you own.
SUT	id, name, OwnedBy, status, Type	**None**	Displays all SUT objects.
SUT Hierarchy	id, name, RevisedBy, WhenRevised, CreatedBy	**None**	Displays all SUT Hierarchies.
Test	id, keyword, <SUT>, measureEffort, measureImport, name, OwnedBy, priority, ScriptState, Type	**OwnedBy is <login>**	Displays Test objects you own.
Test Hierarchy	id, name, RevisedBy, WhenRevised, CreatedBy	**None**	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:

is	The attribute has the specified value.
is not	The attribute does not have the specified value.
Greater Than	The attribute has a value greater than the specified value.
Greater Than or Equal To	The attribute has a value greater than or equal to the specified value**.**
Less Than	The attribute has a value less than the specified value**.**
Less Than or Equal To	The attribute has a value less than or equal to the specified value**.**
Contains	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:

END	Indicates that the statement is not continued. There are no other conditions.
and	Indicates that this statement plus the next statement(s) define the query.
or	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 will set as our task finding all the scripts owned by a user named "rachel" that were written in the MYNAH scripting language. We will use two rows in the Include dialog to do this:

- The first row will find all the scripts owned by **`rachel`**

- The second row will find all of rachel's scripts that use **`mynahTcl`**

We will connect these two rows with **`and`** so that the system finds Script object that satisfy both conditions simultaneously. If we selected **`or`**, the system would find all the Scripts objects that satisfied one of the conditions, but not necessarily both. For example, a Script object owned by someone other than rachel, but which used mynahTcl would be included in the Database Browser display.

From the Database Browser with Script objects selected:

1. Click on the **View** menu and then on the **Include** menu selection.

   — This causes the system to display the Include dialog shown in Figure 5-6. Since we selected the Include dialog from the Database Browser with Script Object selected, the dialog will be set to filter for Script objects.

Figure 5-6. Include Dialog

2. To add the first row, click on the **Add Row** pushbutton.

   — The system will add a row with which to set attributes and conditions.

3. To specify the **Attribute**, **Conditions,** and **Values** for the first row:

   A. Select an attribute from the first **Attribute** option 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 "rachel").

   > **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 with which to set attributes and conditions. Its
relation to the first condition will be set to "and" automatically by the MYNAH
System which is what we want.

5. To specify the **Attribute**, **Conditions,** and **Values** for the second row:

A. Selecting an attribute from the first **Attribute** option list. (e.g., **Language**).

B. Selecting a condition from the **Condition** option list. (e.g., **is**).

C. Select a value from the **Value** option list. (e.g., **mynahTcl**).

— After you complete selecting items for the row, the Include dialog would look similar to the one shown in Figure 5-8.
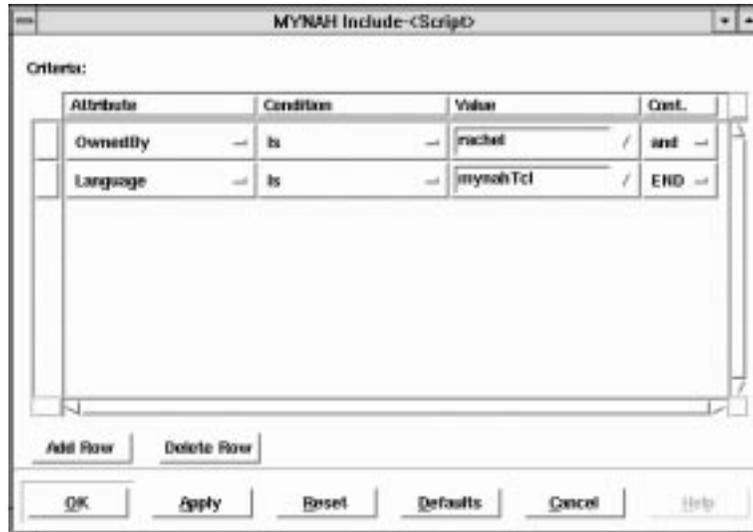
Figure 5-8. Include Dialog with Two Conditions Specified.

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.

2. Click on the Delete Row push button.

— The system will delete the row you checked. Figure 5-9 illustrates this.

1. Check here to select the row.

2. Click here to delete the row.

3. System removes the row.
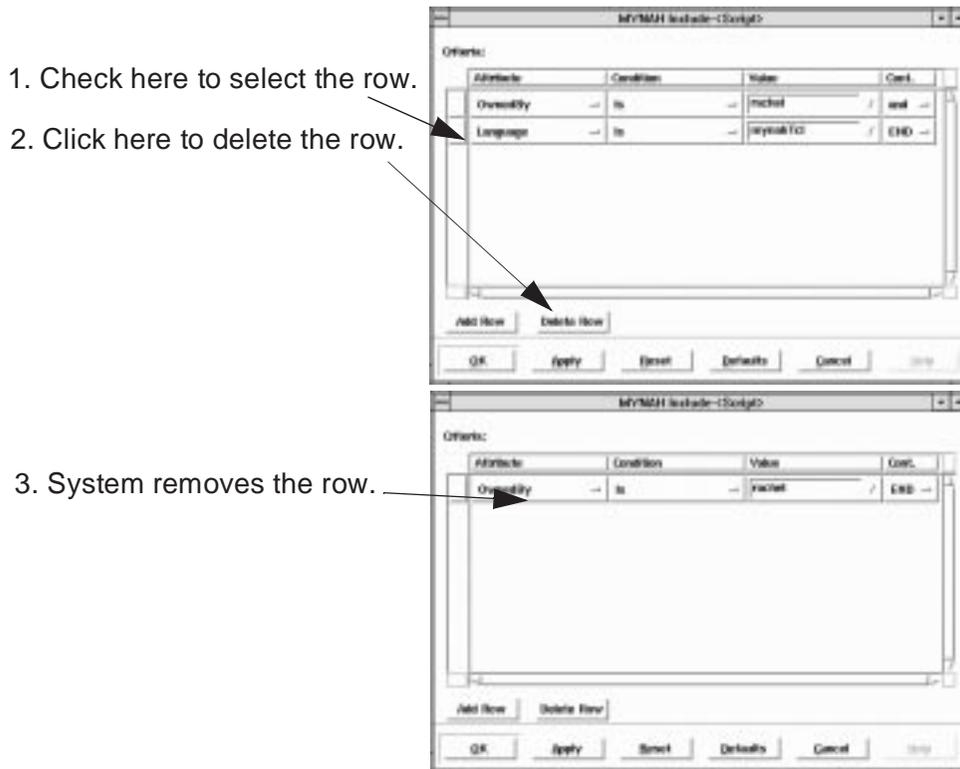
Figure 5-9. Deleting a Row

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 by selecting it.

2. Click on the **Selected** menu and then on the **Open** option.

    — The system will open the object you selected and display its default view.

OR

3. Simply double click on the object row item.

5.2.4   Running an Object From the Database Browser.

You can run scripts or tests in the BEE from the Database Browser window. To do this:

1. Click on the script or test you want to run to select it.

2. Click on the **Selected** menu and then on the **Run** menu selection.

    — The system will open the Run Script dialog. See *Section 12* for more information about running scripts.

5.2.5   Changing an Object's Owner

By default the person who created the object owns the Test or Script object, but you can change this. You can change the owner for one or several objects.

To change a Test or Script object's owner:

1. Select the object or objects.

2. Click on the **Selected** menu and then on **Bulk Change Ownership** menu selection.

    — The system will display the Requesting Information dialog shown in Figure 5-10.
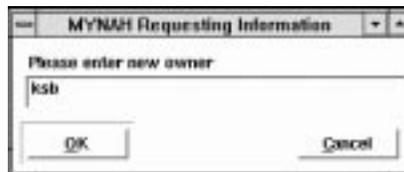
Figure 5-10. Change Owner Dialog.

3. Type in the new owner's name (e.g., ksb) and click OK.

    — The system will change the Test or Script object's owner to the one you specified.

5.3 Associating Objects with One Another

A common task you will need to perform is to associate an object with another object. There are two ways you can do this. You can associate an object with other objects through its Association view; or you can associate objects through the **Database Browser** using the **Bulk Association** menu selection which appears on the **Selected** option.

5.3.1 Associating Objects Using the Association View

Associating objects using the Association view of the object is basically a two-step process: You locate the object you want to associate; and then copy it into another object's Association view. You locate the object using the **Database Browser**. If you know where the object is listed in a ruler display area, you can copy it from there also, but usually it is easier to use the **Database Browser**.

For example, to associate a SUT with an opened Test object:

1. Locate the SUT you want to associate with a test using the **Database Browser** and copy it.

2. Unlock the Test object and position the pointer in it's **Associations** view.

3. Click in the **SUT** panel of the **Associations** view.

4. Click on the **Edit** menu (in the Test Associations View) and then on the **Paste** option.

   — The system will paste the SUT object into the SUT panel of the Associations view. Figure 5-11 illustrates this process.
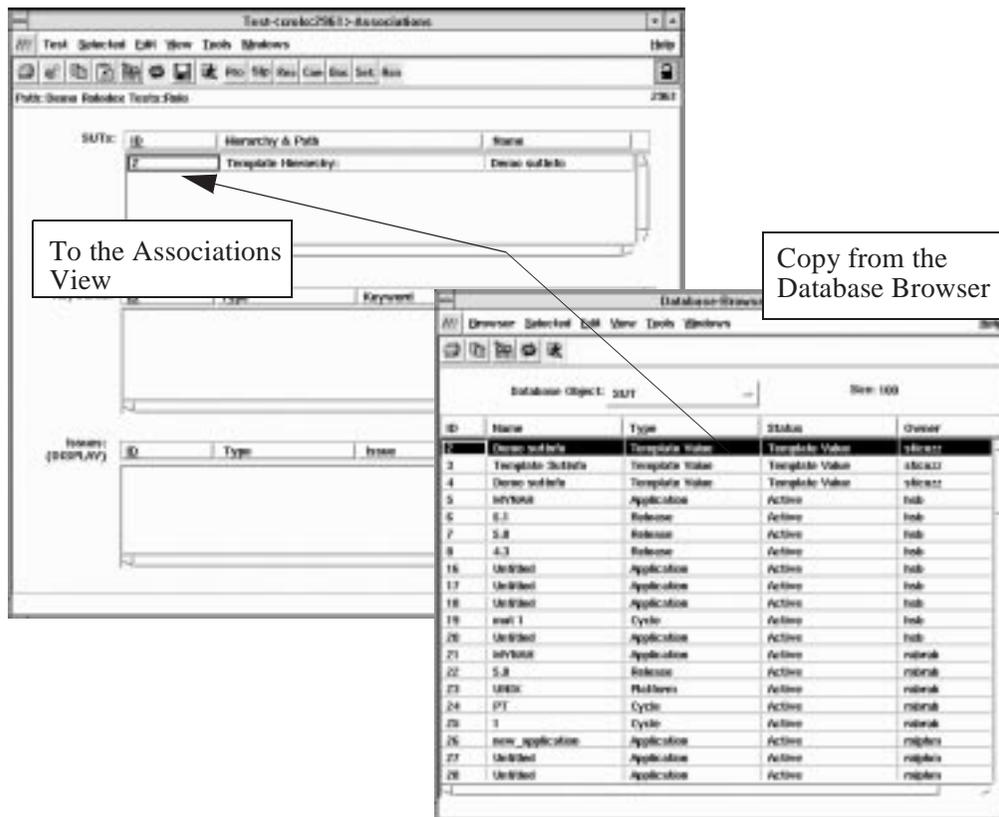
Figure 5-11. Copying a Keyword object into an Association View

5.3.2   Associating Objects Using the Bulk Association Menu Selection

There are three types of objects you can associate with one another using the Selected menu, **Bulk Association** option. These are Keyword, SUT, and Test objects. Associations can be made from:

- Keyword objects to Script and Test objects
- SUT objects to Tests objects
- Test objects to SUT objects

For an example, let's associate a number of Script objects with a Keyword object.

1. Make sure **Keyword** is the **Database Object** type for the Database Browser.

2. Select the Keyword you want to associate with Script objects.

    — Figure 5-11 illustrates this.

3. Select **Edit** menu, then the **Copy** option to put on clipboard.
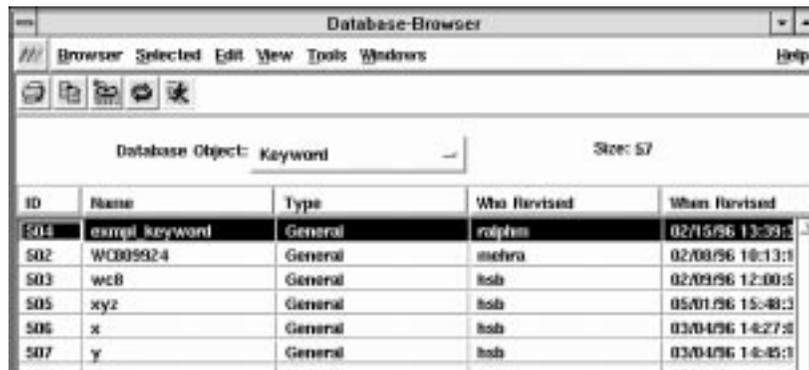
Figure 5-12. Select Object Type and Object

4. Change the **Database Object** type to **Script.**

5. Select the Script(s) you want to associate with the Keyword object.

— Figure 5-13 illustrates this.
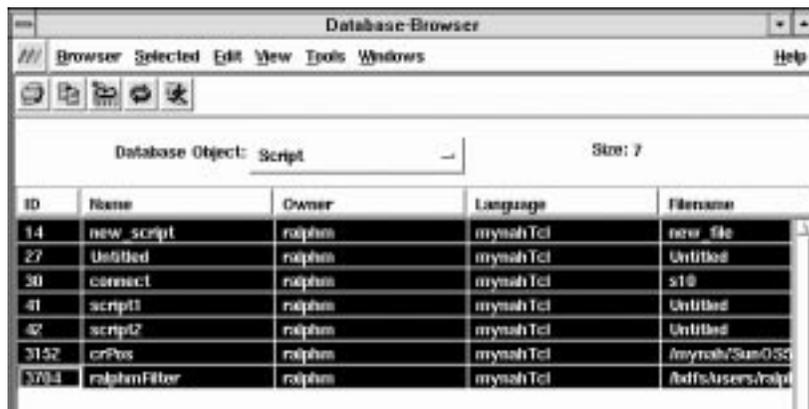
Figure 5-13. Select Object Type and Object(s) To Associate

6. Click on the **Selected** menu and then on the **Bulk Associate** option. (Note that the Bulk Associate menu option names the object type you are associating the selected objects with. In our example it displayed **Keyword** as the object).

— The system will display a dialog asking you to OK the association. See Figure 5-14.

Figure 5-14. Confirm Association

7. Click **OK**.

— The MYNAH System will associate all the keyword to all of objects you selected.

MYNAH System Users Guide   BR 007-252-006
Using the Database Browser   Issue 3, October 1997
Release 5.2

5.4 Using Person Objects

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

When you first logon to the MYNAH 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.

5.4.1 Viewing Person Objects

To view Person Objects:

1. Open the **Database Browser** from the **Tools** menu.

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

   • Clicking on the **Selected** menu and then on the **Open** option

      — Figure 5-15 shows a Person object Properties view we opened. The person object we opened happens to be for a user with Administrative privileges.
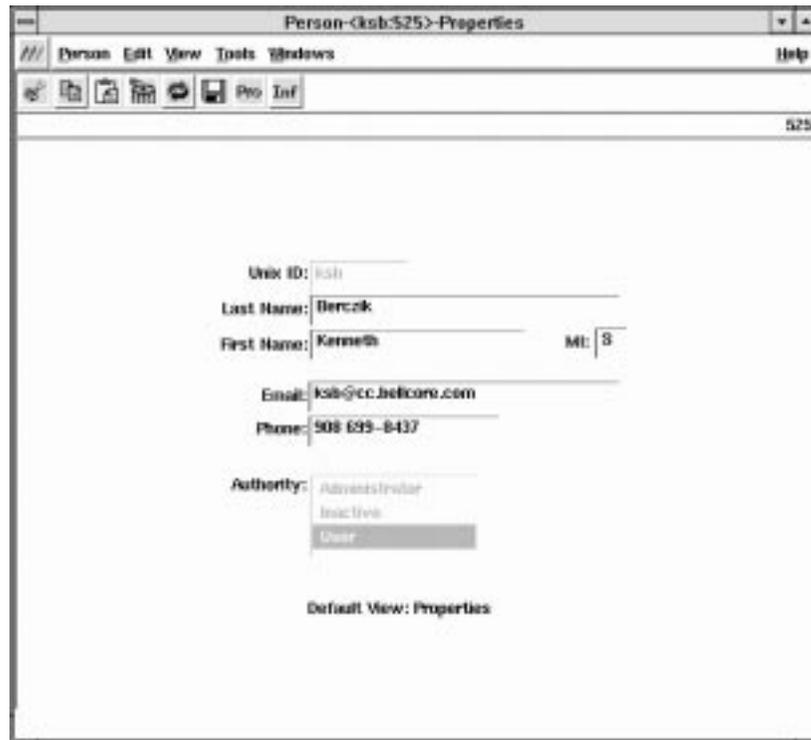
Figure 5-15. Person Object Properties View

> **NOTE —** 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 5-15 each Person object lists:

User ID   The UNIX user ID.

Last Name   The user's last name.

First Name   The user's first name.

MI   The user's middle initial.

Email   The user's email address.

Phone   The user's phone number.

Authority   System authority granted the user.

Default View The currently specified default view.

5.4.2   Changing Data in Your Person Object.

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

To change information in your person object:

1. Open the **Database Browser** from the **Tool** menu.

2. Select **Person** as the Database object Type.

3. Select your Person object.

4. Open it by performing one of the following:

    • Double clicking on it

    • Clicking on the **Selected** menu and then on the **Open** option

        — Figure 5-16 shows a Person object Properties view we opened.
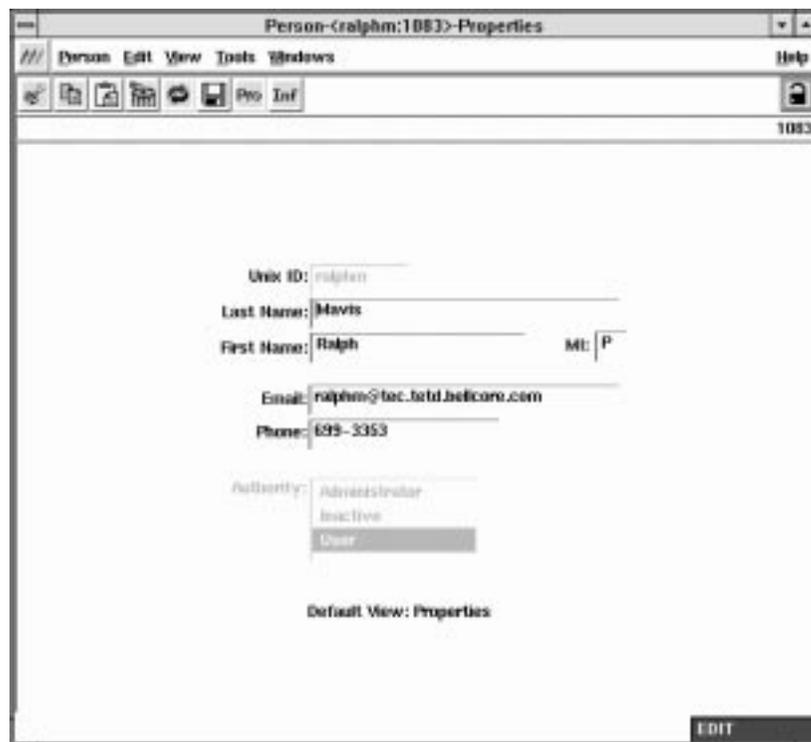
Figure 5-16. Person Object Opened for Editing

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. Click on the **Person** menu and then on the **Save** option to save your changes.

5.4.3   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 5-17.
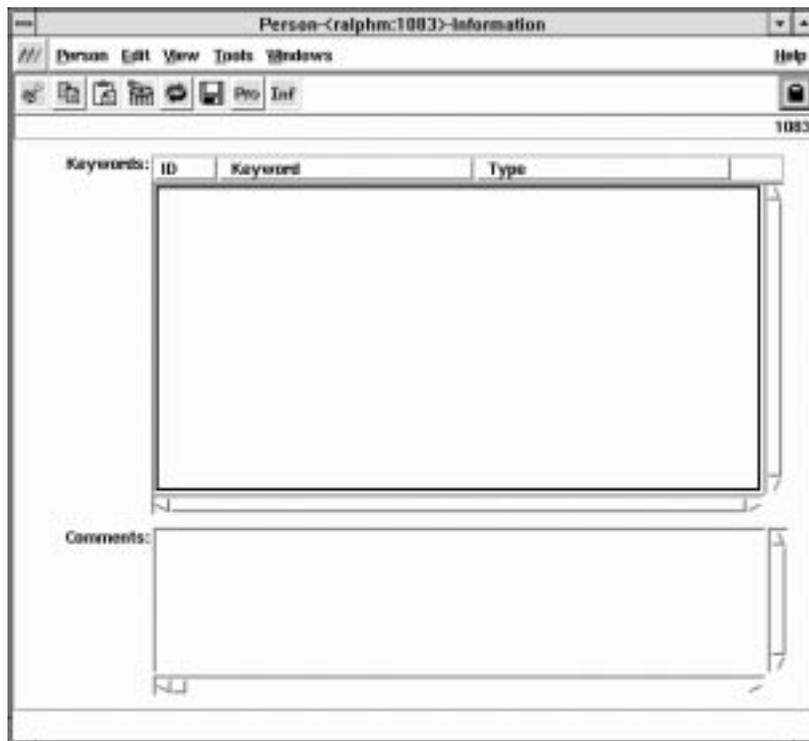
Figure 5-17. Person Object Information View

You can see all the Keyword objects associated with a user in the **Keywords** scrolling list. If this is your Person object, you can copy Keyword objects into this scrolling list. See Section *5.3* for information on doing this.

You can also 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, click on the **Person** menu and then on the **Save** menu selection.

5.5   Where to go Next

If you have looked at Sections 1, 2, 3, and 4 you should be fairly familiar with the MYNAH System. The next section which may be of interest to you is, *Section 6*. Here you can begin developing tests by defining what you are testing.

6. Using Requirement Objects

Before creating tests and scripts you may want to create Requirement objects (although you can create them any time you want). Requirement objects are optional so far as the MYNAH System is concerned. You can create tests and develop scripts without Requirement objects.

In this section we will explain how to

- Create Requirement objects
- Associate them with tests and other objects.

6.1 How Requirements Objects Help You with Testing and Scripting

Requirements are the basis of most testing. By requirements we mean the requirements that define how an application is supposed to behave. You will develop test cases based on requirements.

Requirement objects allow you to develop and describe requirements and associate them with Test objects through the MYNAH System. The system also uses Requirement objects for Requirement Traceability reports which you can generate.

6.2 Creating Requirement Objects Example

You create Requirement objects in the same way you would any other object. You access the Requirements object by opening the Test or SUT you want to create requirements for. To creat a requirement object select the **Folder** menu, the **New** menu option, then choose the **Requirement** option. We created a new Requirement object in the **crolo demo** which appears in the **crolo demo** display as **"Untitled Requirement"**. See Figure 6-1.
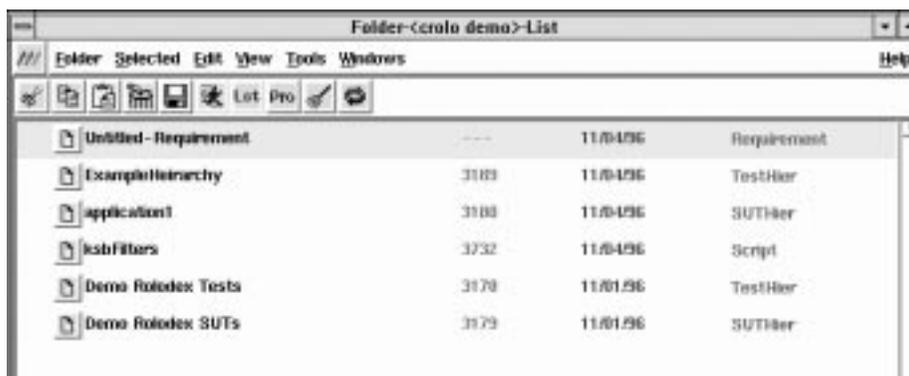
Figure 6-1. New Requirement Object

6.3   Specifying Properties for a Requirements Object

After you create a new Requirement object, you have to specify its properties. You do this
with the Requirements Property view shown in Figure 6-2. You access this view by double
clicking on the icon for the new "Untitled- Requirements" object.

Figure 6-2. Requirement Object Properties View.

The Properties view allows you to specify the basic properties of an Requirement object.
There are three properties you have to specify:

Name   Is the name you give the object.

Type   Refers to the type of Requirement. We used one of the default values
delivered with the system, **crolo demo**. This is the type for the demo
requirements**.** Your System Administrator can replace these values or add
others to them.

Description   A free form data entry area where you type in a description of the
requirement.

6.3.1   Entering Properties Information

We will show you how to enter properties using the **R1-Add** Requirement object from our
ongoing example **crolo**. You can see this example Requirement object using the Database
Browser as we explained in *Section 5: Using the Database Browser*. To enter the properties
for a Requirement object:

1. Click on the lock icon to unlock the object.

2. Position the pointer in the **Name** data entry field and type in a name for the object. (We
    typed in `R1-Add`.)

3. Select a type for the object from the **Type** option list. (We used the default type **crolo)**.

4. Position the pointer in the first available line of the **Description** data entry area and
    type in a description for the Requirement object. Typically you would enter some of
    the actual text for the requirement as we did.

5. We won't save the object yet because we are going to describe another view. If you
    want to save the object, click on the **Requirement** menu and the **Save** selection.

    — Figure 6-3 is our finished Properties view.
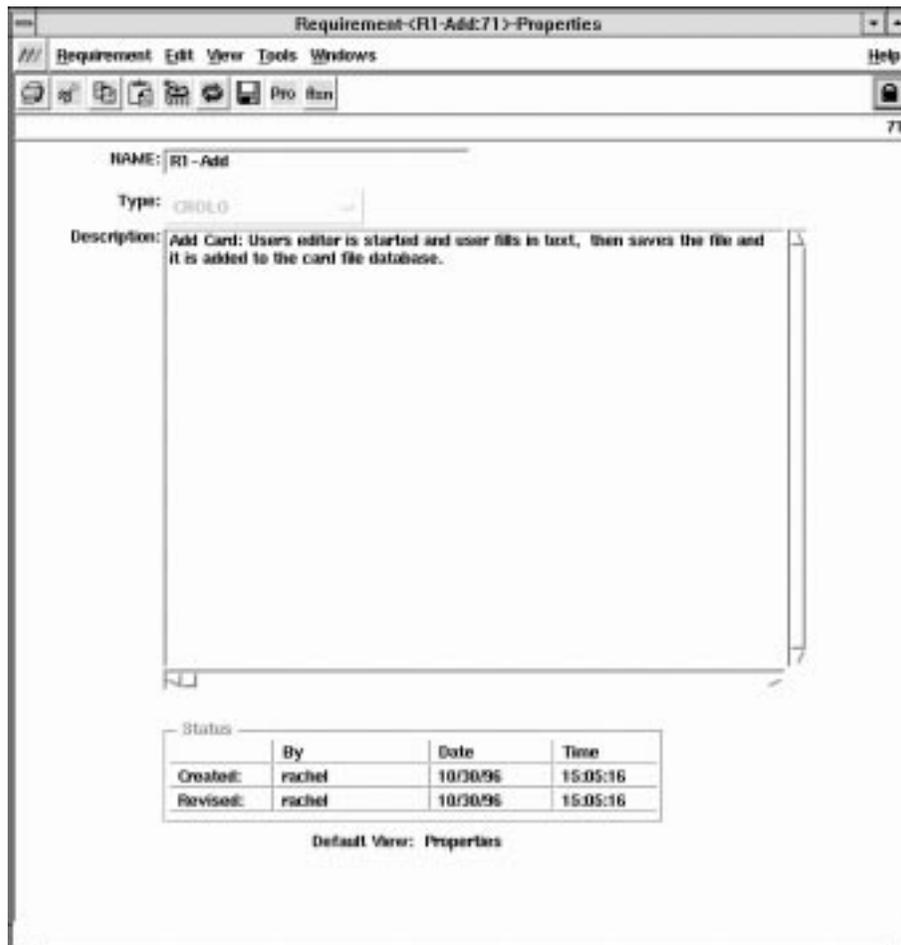
Figure 6-3. Finished Requirement Object Properties View.

6.3.2   Displaying Requirement Object Associations

Requirement objects can be associated with one or more Test objects. Sometimes a single requirement needs several tests to validate it. Associating Requirements objects with Test objects will give you the ability to trace test efforts. For example, if no Test objects are associated with a Requirements object, you know that the requirement isn't covered by your test plan. Also, the MYNAH System can generate a report (based on Test objects associated with a Requirements object having been run) that indicates which requirements have been tested.

You actually make the association between requirements and tests through the Test object, but you can view associations through the Requirements object. After you select the Association view, the system will display a window similar to the one shown in Figure 6-4.
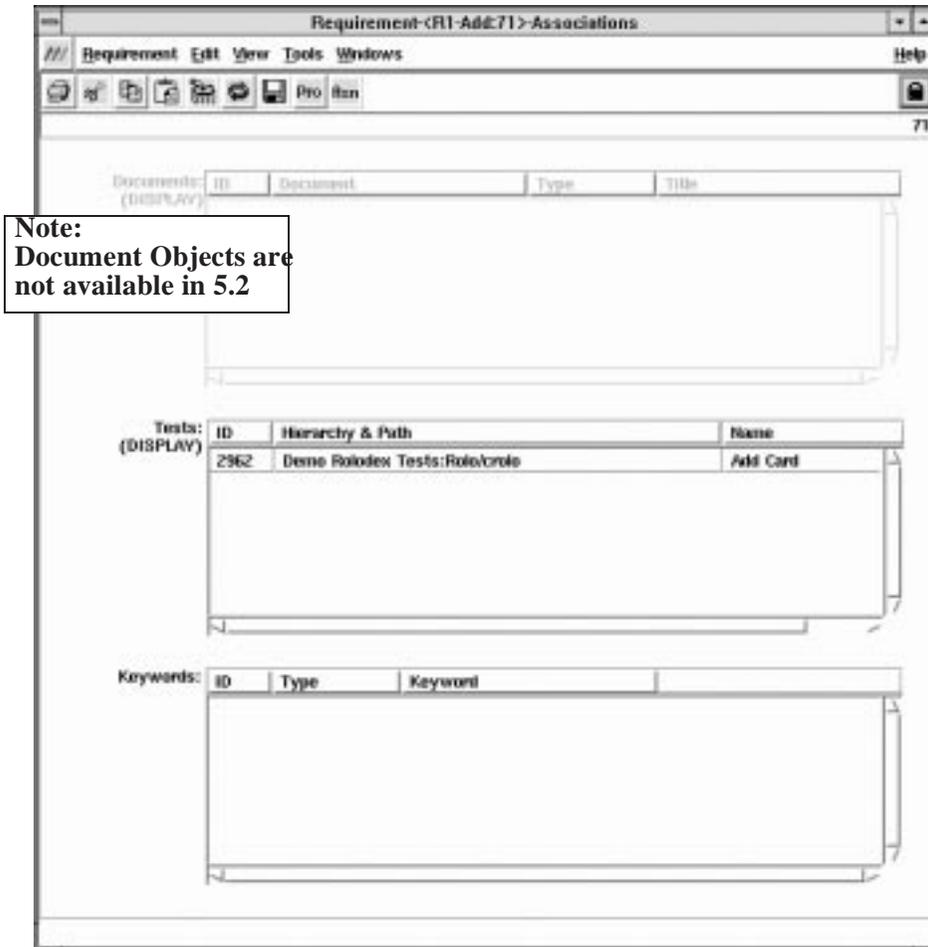
Figure 6-4. Associations View.

The Test objects associated with the currently opened Requirement object appear in the tests scrolling list. Note that the Test object, **Add Card**, is associated with this requirement. You can scroll through this list in the same way you would scroll through any other list. You can open Test objects from this view, by double clicking on the line item for the object.

6.3.3   Associating Keywords with a Requirements Object

You can associate a Requirement object with Keyword objects through the Requirement object Association view. Keyword objects help you to organize your testing efforts. Refer to *Section 11: Using Keywords* for more information about how Keyword objects can help you.

You associate the currently opened Requirement object with Keyword objects using the Database Browser. We explained how to do this in *Section 5: Using the Database Browser*. Refer to this section if you need to refresh your memory on how to do this.

BR 007-252-006   **MYNAH System Users Guide**
Issue 3, October 1997   **Defining a System Under Test (SUT)**
                                                      Release 5.2

7. Defining a System Under Test (SUT)

The first step in using the MYNAH System is to define a System Under Test (SUT). All other development work you do will be based on a defined SUT. A SUT object can represent a software application, platform, release, cycle or anything else you plan to test. It is simply the target of your testing efforts.

A SUT object has attributes that will help you keep track of it. In addition, you can document testing schedules and see other objects associated with the SUT object.

In this section we will explain how to:

- Specify a SUT's properties
- Document SUT schedule
- Display SUT associations with other objects.

7.1 How SUTs and SUT Hierarchies Help You

Often one person in your organization, perhaps the Test coordinator, is responsible for creating and maintaining SUT objects. Both you and your test coordinator have probably faced the problem of keeping track of testing for several different versions of software. One of the solutions the MYNAH System offers for this type of problem is the SUT object and SUT Hierarchy.

Let's use a new example to illustrate how SUT objects and hierarchies can help you. We will return to our ongoing **crolo** example when we describe defining a SUT object's properties.

Suppose we have "Application A" that runs on two different platforms — UNIX workstations and PCs. We want to test the application on each of these platforms. We also want to keep track of two different releases of our application. The MYNAH System can help you with SUT Hierarchies. We can have one hierarchy that contains both representations of releases.

Figure 7-1 Illustrates how SUT objects and hierarchies support this.

SUT Hierarchy for Application A

```
┌─────────────────────────────────────────────┐
│  ( SUT Object: Release 1.0 )                  │
│      ( SUT Object: PC )                        │
│        ( SUT Object: UNIX Workstation )        │
│                                               │
│  ( SUT Object: Release 2.0 )                  │
│      ( SUT Object: PC )                        │
│        ( SUT Object: UNIX Workstation )        │
└─────────────────────────────────────────────┘
```
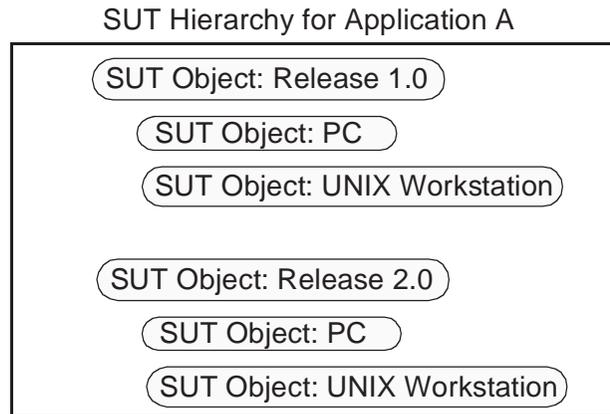
Figure 7-1. How SUT Objects and Hierarchies Help You.

SUT objects document each release platform, notice our example "Application A", SUT Hierarchy add to the convenience of doing this in a fairly simple way. Releases and platforms for the same basic application tend to be similar. To create and then re-create paper documentation for different versions of the same software can be time consuming. With the MYNAH System all you need to do is duplicate the SUT object that represents one release or platform and modify it to fit the another release or platform.

The MYNAH System also helps you keep track of your testing efforts using SUT objects. Every time you run a test or script, the MYNAH System requires you to specify the SUT against which you are running the test or script. The system keeps a record of the scripts and tests run against a SUT. You can request a Testing Progress report which includes this information. See *Section 14 Examing Results and Reporting on Test Activities,* for information about this report.

7.2   Creating SUT Hierarchies and SUT Objects

In this sub-section we will take the example we just described and represent it with a SUT Hierarchy and SUT objects. First we will create the hierarchy in our **crolo demo** folder and then we will place SUT objects in it.

7.2.1   Creating a SUT Hierarchy

We will create our example SUT Hierarchy in the **crolo demo** folder which we created in
Section 3 MYNAH Basics. SUT objects can only appear in SUT Hierarchies. To do this:

1. Click on the **Edit** menu and then on the **Deselect All** option to make sure nothing is
   selected.

2. Click on the **Folder** menu selection, then the **New** menu selection, and then on the **SUT
   Hierarchy** option

   — The system will place a new "Untitled" SUT Hierarchy in the folder. See
     Figure 7-2.

Figure 7-2. New SUT Hierarchy

3. Double click on the new SUT Hierarchy's icon to open it.

4. Click on the Lock icon to unlock it.

   — The system will display a **Requesting Information** dialog like the one in
     Figure 7-3.
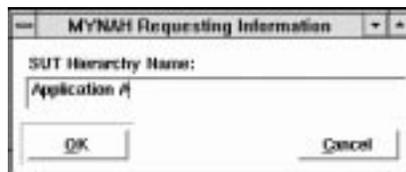
Figure 7-3. Naming a SUT Test Hierarchy

5. Type in a name for the new SUT Hierarchy, and click **OK**. (We typed in
   Application A.)

   — After we clicked OK, the system assigned the new name to the SUT Hierarchy, but
     left it in edit mode. See Figure 7-4.

Figure 7-4. New SUT Hierarchy Named

7.2.2   Creating SUT Objects in a SUT Hierarchy.

Now we have a SUT Hierarchy to place our SUT objects in. We will create all the SUT objects we need to represent our example before naming them. In effect we will be creating the structure of our SUT Hierarchy before we specify any of the SUT object's attributes.

There is a reason we create objects in this way. If we created a SUT object and then tried to open it, we would have to save the hierarchy. It is easier to create all the objects while the hierarchy is in Edit mode, save the hierarchy and then go back to specify the objects' attributes.

If this were a saved SUT Hierarchy, we would have to unlock it by clicking on the Lock icon.

> **NOTE —** When creating **New SUTs** you have multiple options to choose from. For example if you want a parent SUT, Deselect All then choose **Hierarchy** then New SUT. If you want a child SUT select the parent SUT, choose **Selected** then New SUT.

To create SUT objects for our example Hierarchy:

1. Click on the **Hierarchy** menu and then on the **New SUT** option.

   — The system places the first untitled SUT object in the hierarchy at the highest level. See Figure 7-5. This object will represent Release 1.0.
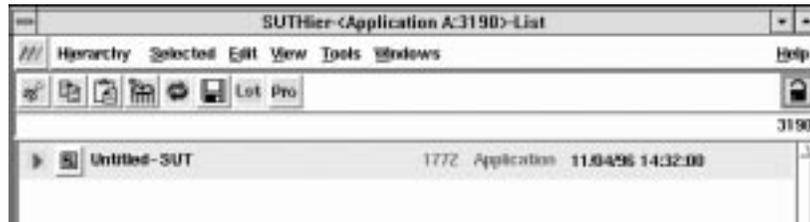
Figure 7-5. SUT Object Representing Release 1.0.

2. Click on the **Edit** menu and then on **Deselect All** so that the next SUT we created appears at the highest level. (Or **Esc** to clear any selected object since at the time there was only one selected object.)

3. Click on the **Hierarchy** menu and then on the **New SUT** option again.

   — The system will create another Untitled SUT object at the first level of the hierarchy. See Figure 7-6. This SUT object will document Release 2.0.
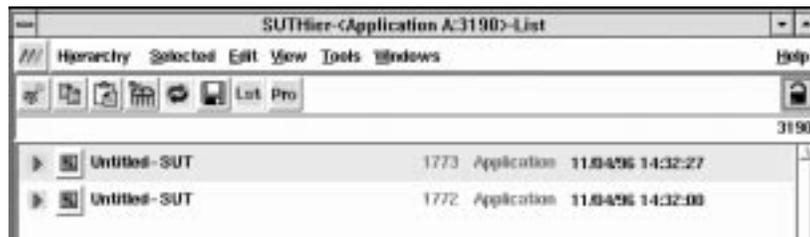
Figure 7-6. SUT Object Representing Release 2.0

4. Click on the **Selected** menu and then on the **New SUT** option again.

— The system places an Untitled SUT object in the hierarchy as a child of the first level we just created. This SUT object will represent the UNIX platform. See Figure 7-7.
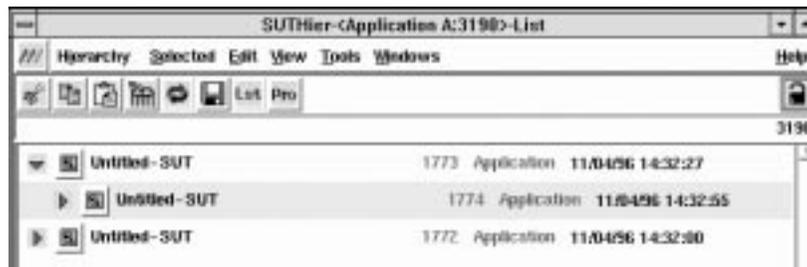
Figure 7-7. SUT Object Representing The Release 1.0 UNIX Platform

5. Select the first SUT object at the highest level and then click on the **Selected** menu and the **New SUT** option.

— The system places another SUT object at the second level as a sibling of the first. This SUT object will represent the PC platform. See Figure 7-8.
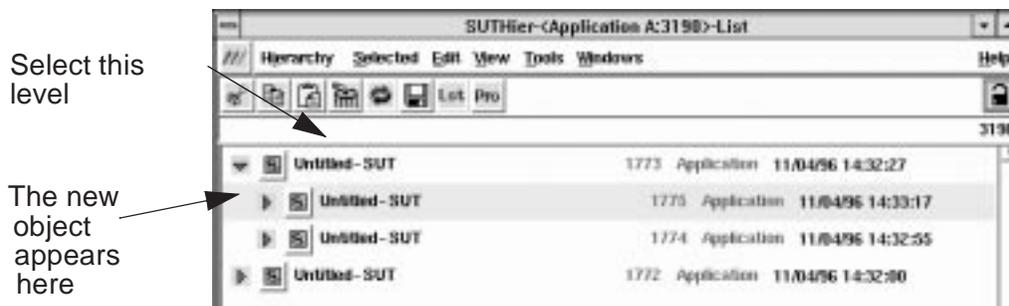
Figure 7-8. SUT Object Representing The Release 1.0 PC Platform

> **NOTE —** The system created the new SUT object *above* its sibling.

6. Select the second, highest-level SUT object and then click on the **Selected** menu and the **New SUT** option.

— The system places a new SUT object at the second level of the second highest level SUT object. This is the SUT object for the Release 2.0 UNIX platform. See Figure 7-9.
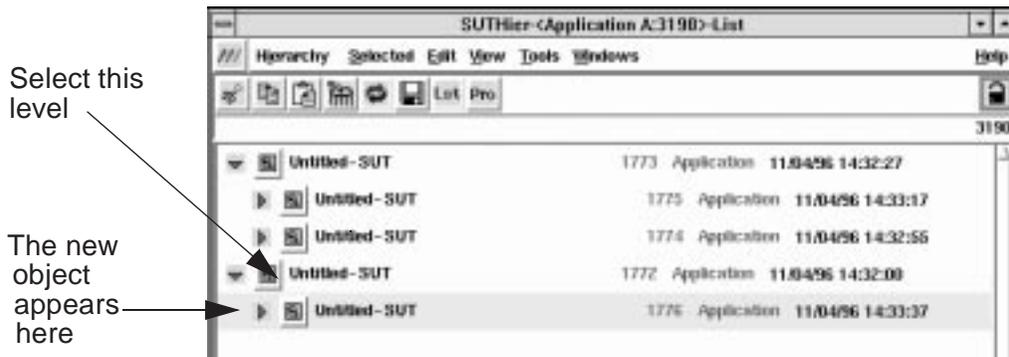
Figure 7-9. SUT Object Representing the Release 2.0 UNIX Platform

7. Again, select the second SUT object at the highest level and then click on the **Selected** menu and the **New SUT** option.

— The system places another SUT object at the second level as a sibling of the first. This is the SUT object for the Release 2.0 PC platform. See Figure 7-10.
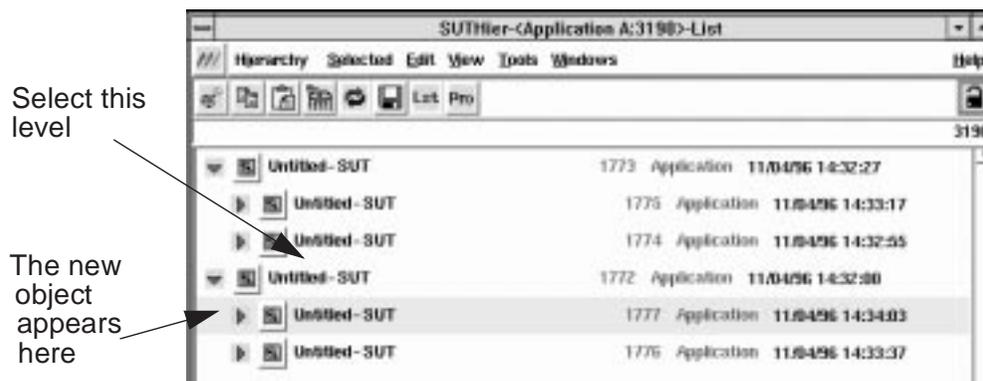
Figure 7-10. SUT Object Representing the Release 2.0 PC Platform

8. Now that we have added all the SUT objects we need, click on the **Hierarchy** menu and then on the **Save** option.

We will explain how to specify the name and the other attributes of a SUT object in the next section. We went ahead and named all the objects we created in this hierarchy and when we were finished the SUT Hierarchy looked like Figure 7-11.
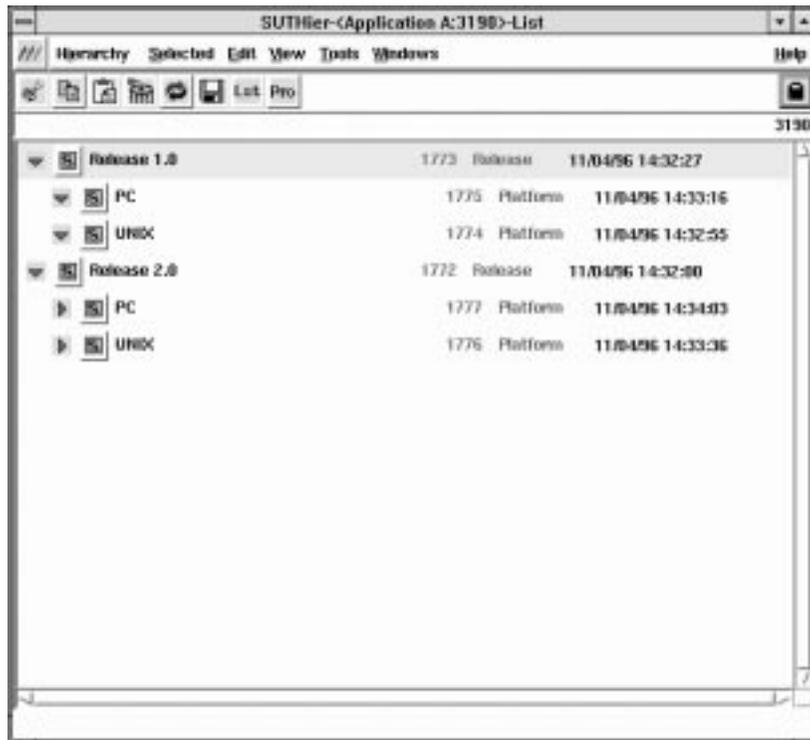
Figure 7-11. Completed SUT Hierarchy

7.3   Specifying SUT Properties

We will return to our ongoing **crolo** example to show you how to specify a SUT object's properties. In this cycle, let's suppose that we are defining the third release (5.5) of our **crolo** example as the SUT.

We placed our example SUT hierarchy in the MYNAH folder we created in *Section 3: MYNAH Basics*, **crolo demo**, but we could have placed it in any other folder we created. We copied the SUT hierarchy, **Demo Rolodex SUTs**, using the Database Browser tool.

If you double click on the icon for **Demo Rolodex SUTs** in the **crolo demo** folder, you will see that there are two SUTs in this hierarchy, **5.4** and **5.3**. Figure 7-12 shows the SUT Hierarchy List view for our example.

Figure 7-12. SUT Hierarchy List View in the Example Folder.

Now we will create a new SUT object to represent our new release. To do this:

1. Click on the **Edit** menu and then on **Deselect All**, if you need to clear selections.

2. Unlock the object by clicking on the Lock icon.

3. Click on the **Selected** menu and then on the **New SUT** option.

    — The system places an untitled SUT object in the hierarchy at the highest level. See Figure 7-13. This object will represent the new release.
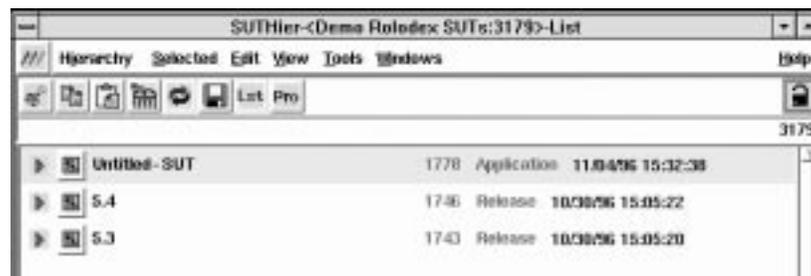
Figure 7-13. SUT Object To represent Our New Release.

4. Click on the **Hierarchy** menu and then on the **Save** option.

Once you have created a SUT object, you will have to define its basic properties. This is done with the SUT object Properties view shown in Figure 7-14. To access the new SUT object double click on the new SUT's icon; or select the new SUT object, click on the **Selected** menu and then on the **Open** option.
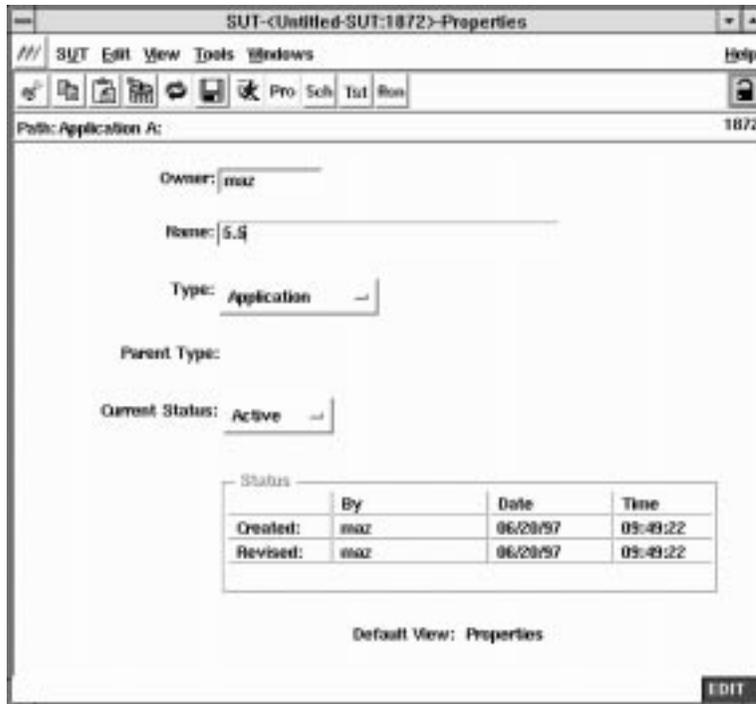
Figure 7-14. SUT Object Properties View

The path for the SUT object (**Demo Rolodex SUTs)** appears below the Tool Bar. The system-assigned **ID** appears to the right below the Lock icon.

The Properties view has the following items on it:

Owner   Refers to the owner of the SUT object. This is the UNIX id of the person who created the object for new objects, but you can change this. Refer to *Section 4: Using MYNAH Objects* for information about doing this.

Name   Is the name you assign to this object.

Type   Refers to the type of SUT the currently opened object represents. We provide default values which are **Application**, **Release**, **Platform**, and **Cycle** your system administrator can add to these.

Parent Type   Displays the type setting for parent of the currently opened object.

Current Status   Refers to whether or not this SUT is available for testing. The possible values are **Active** or **Idle**.

7.3.1   Entering Property Information

We will explain how to enter property information using our example.

To specify a SUT object:

1. If the SUT object is locked, unlock it by clicking on the Lock icon.

2. Position the pointer in the **Name** data entry field and type in the name of the new SUT object (e.g., `5.5`).

3. Select a SUT object type from the **Type** drop-down menu (e.g., select **Release**).

   — The system displays the **Parent Type** automatically. Our example has no parent object, so this area is blank.

4. Select a status from the **Current Status** option list (e.g., select **Active**).

5. We will go on to describe the next view, so we won't save the SUT object yet. If you want to save the object, click on the **SUT** menu and then on the **Save** option.

When we complete the Properties view it looks like Figure 7-15.

Figure 7-15. Completed SUT Properties View.

7.4   Documenting SUT Schedules

The MYNAH System provides you with a way of documenting test schedules with the SUT object Schedule view. You access this view from the **View** menu, click on the **Change View** menu then on **Schedule** option. After you select the Schedule view, the MYNAH System displays a window similar to Figure 7-16.
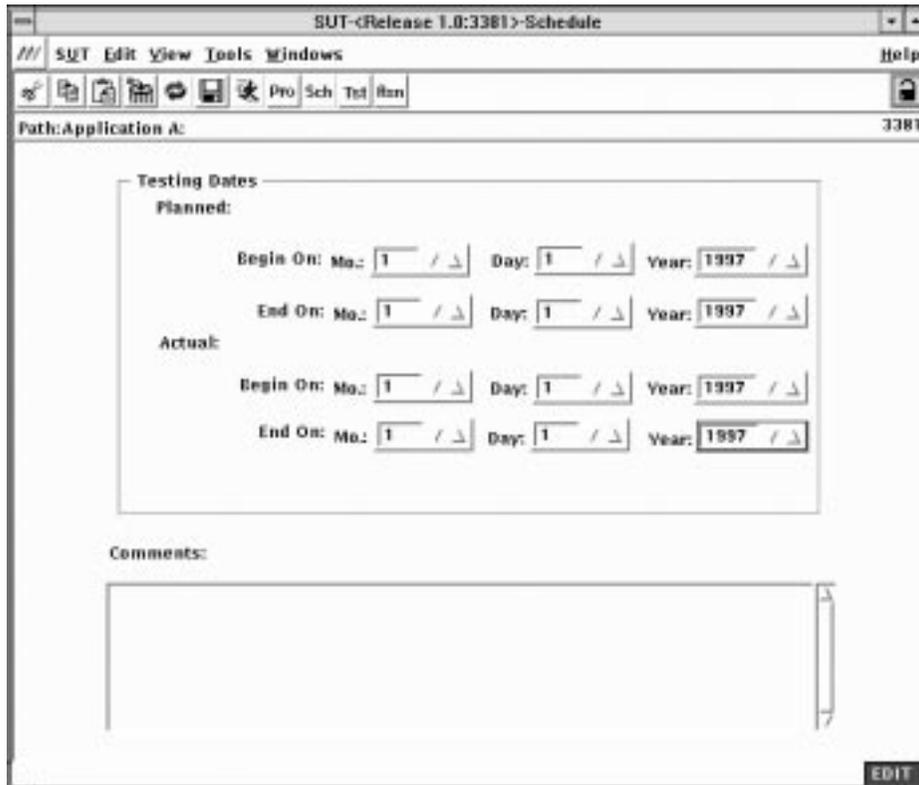
Figure 7-16. SUT Object Schedule View

You can use a series of spin buttons to set the date (month, day, and year) for a SUT test. You can set the planned test date and the actual test date. The **Planned** test date is the date on which you want to begin testing; the **Actual** test date is the date the test actually started.

In addition to the dates, the Schedule view provides you with a Comments data entry area for entering comments or reviewing any existing comments about the schedule.

To document the schedule for our example:

1. If you need to, unlock the object by clicking on the Lock icon.

2. Set the object's test date using one of the following methods:

    - Enter the date directly by positioning the cursor in the date fields (**Mo: Day: Year:**) and typing in a number.

    - Enter the date using the spin buttons using the following method:

        A. Click on the up or down arrow of the **Mo**: (month) number wheel until the month you want appears. Values are 1 to 12.

        B. Click on the up or down arrow of the **Day:** number wheel until the day you want appears. Values are 1 to 31.

        C. Click on the up or down arrow of the **Year:** number wheel until the year you want appears. The year must be a 4 digit value, e.g. 1996.

        D. There must be a value in these fields. Make sure that there are zeroes if you don't want to set a date.

3. If you want to add comments to the schedule, position the pointer in the **Comments** data entry area and type in your comments.

4. We will go on to describe the next view, so we won't save the SUT object yet. If you want to save the object, click on the **SUT** menu and then on the **Save** option.

7.5   Displaying SUT Object Associations

The SUT object's associations view is provided to document, Issue and formats objects. The association between the SUT object and these other objects are created through the other objects.

You access the Associations view from the **View** menu or by clicking on the **Asn** icon. When you do this the system displays an **Associations** view similar to Figure 7-17. Note that all the scrolling lists on this view are marked **(DISPLAY)** which means that these are display only lists. This means that you cannot add to or delete from these lists. However, you can select a list item and copy it to the clipboard.
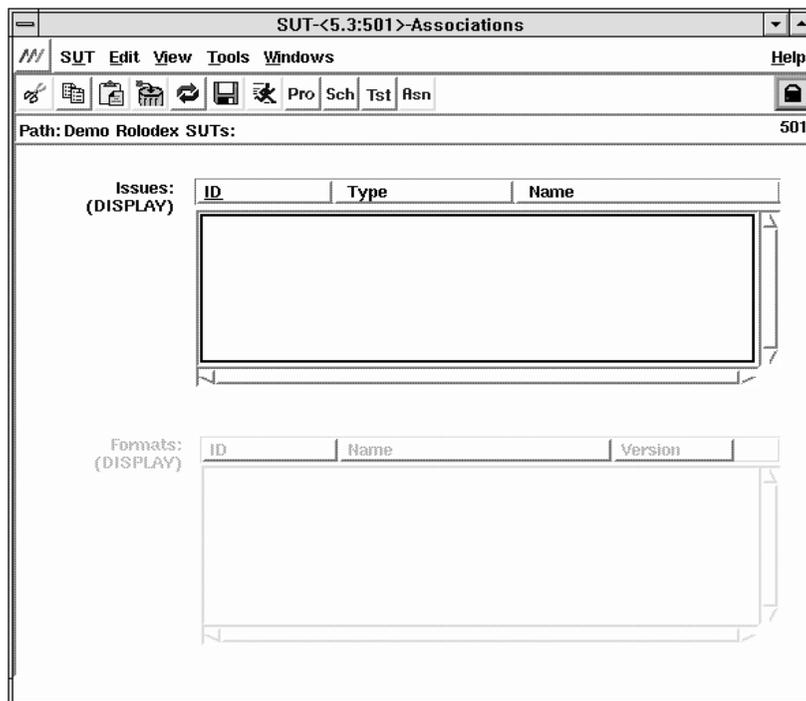
Figure 7-17. SUT Object Association View

The scrolling lists on this view are:

Issues   Displays all the Issue objects associated with the currently selected SUT object and the Issue's type.

Formats The Format parameter on the SUT object Association view is part of the AETG System, and is only available if you license the AETG System with the MYNAH System. For information on the Format parameter, see the *AETG System Users Guide.*

Once you have defined the SUT objects you will be using for testing, you can go on to define Requirement, Test, Issue, and Keyword objects.

7.6   Displaying SUT Test View

You access the **Test** view from the **View** menu or by clicking on the **Tst** icon. The system displays a **Test** view similar to Figure 7-18. This SUT Test view are for display only list.
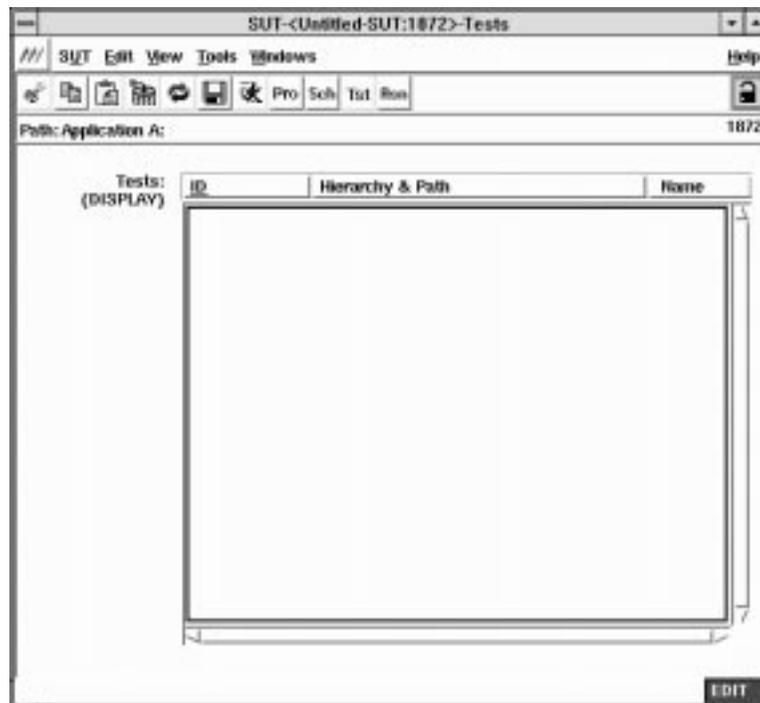
Figure 7-18. SUT Object Test View

The scrolling lists on this view are:

  Tests   Displays all the Test objects associated with the currently selected SUT object. Each listing shows the test hierarchy the Test object belongs to, and the path to it.

8.   Using Test Objects

Test objects document your test plans and tests. They allow you to document every level of your test plan from the highest down to test cases. In addition, Test objects allow you to link tests to the scripts that implement them. You can also associate Test objects to SUT objects and keywords.

Test objects can document both automated tests, those that can be implemented with Tcl code in a script or other scripting languages, and tests that can not be automated.

In the remainder of this section we will explain how to define test objects and associate them with other objects needed for test operations. Creating and specifying Test objects and associating them with other objects is a multi-step process that includes:

- Organizing Test objects in a Hierarchy

- Defining a Test object properties

- Associating the object with SUTs and keywords

- Displaying associations with other objects

- Specifying Test settings

- Running Tests

- Adding comments to a Test object.

8.1   How Test Objects Automate Your Test Plan

Rather than relying totally on bulky and cumbersome paper documents to support your test plan, you can use electronically stored graphical objects to keep track of your testing activities. You can easily copy and duplicate these objects as your test plan grows or as you develop new test plans.

Test objects help you to organize tests to meet your needs and priorities. You can document each level of testing you will do using Test Hierarchies and Test objects. Figure 8-1 shows the Test Hierarchy for our example application, **crolo demo** folder. Notice how each level of our test plan is documented in the plan (**Feature**, **Test** and **Test Case**) and how at a glance we can see the entire plan schematically with the Test Hierarchy List view.
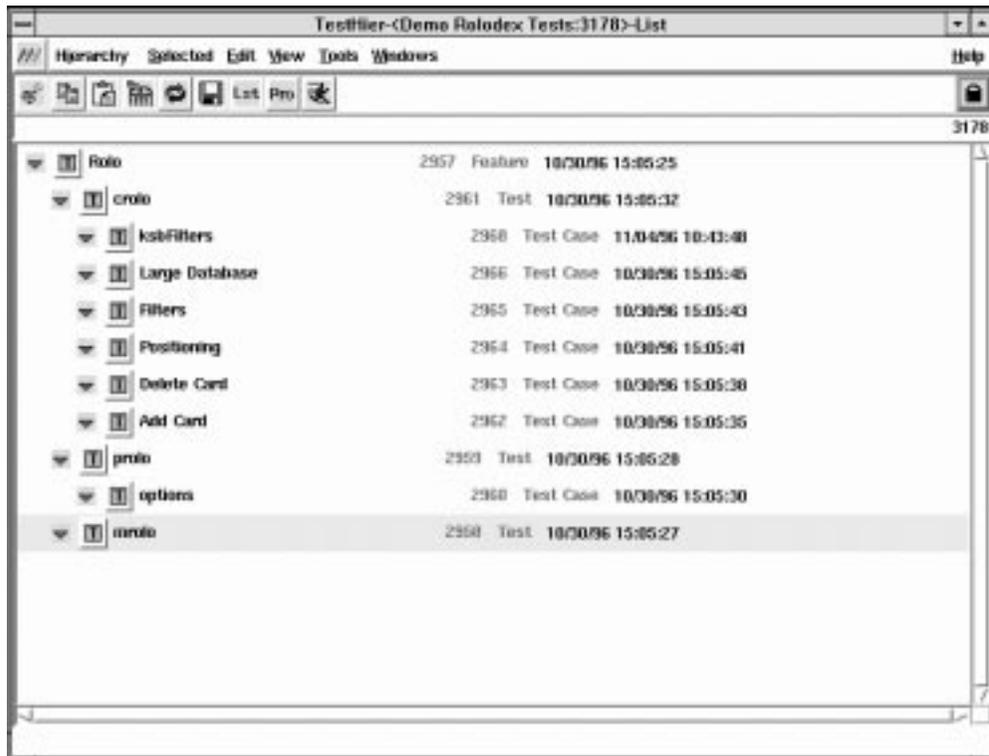
Figure 8-1. Test Hierarchy For the Example Application

8.2   Creating Test Hierarchies and Test Objects

Our example Test Hierarchy is already set up. We will create a new Test Hierarchy and add objects to it so that you can see how you can do this. We'll use an example of utilities. We will use File as our example. Our testing will cover some basic commands:

- **ls -l** which displays a list of files in the current directory

- **cat <filename>** which displays the contents of the specified file

- **lp <filename>** which sends a file to the default printer

Figure 8-2 illustrates this example and how it fits with a Test Hierarchy and Test objects. We won't actually define all these objects; we will simply set up the structure in a hierarchy.

<u>**Application**</u>   <u>**Test Objects**</u>

Utilities   (**Test Object 2.4**)

    File   (**T.O 2.4.1**)

                             ⟷   (**T.O. 2.4.1.1**)

Test ls -l   (**T.O. 2.4.1.2**)

Test cat <filename>   (**T.O. 2.4.1.3**)
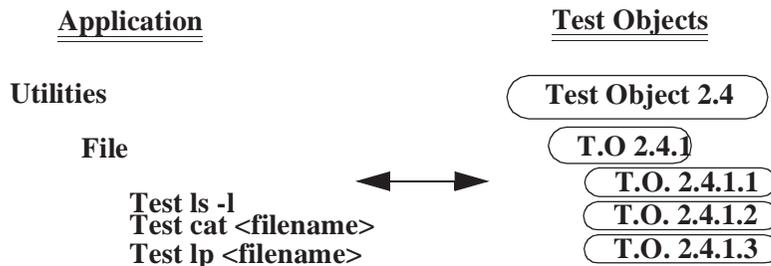
Test lp <filename>

Figure 8-2. Test Objects Document a Test Plan

We can use Test objects to document each level in a Test Plan:

- At the highest level, **Utilities** represents an area of testing.

- At the second level, **File** represents a specific feature for testing.

- At the third level, **ls -l**, **cat**, and **lp** represent the specific test cases for that feature.

8.2.1   Creating a Test Hierarchy

We will create our example Test Hierarchy in the **crolo demo** folder which we created already. (Refer to *Section 3: MYNAH Basics*). We do this by:

1. Click on the **Edit** menu and then on the **Deselect All** option to make sure nothing else is selected.

2. Click on the **MYNAH** menu, the **New** selection, and then on the **Test Hierarchy** selection.

— The system will place a new "Untitled" Test Hierarchy in the folder. See Figure 8-3.
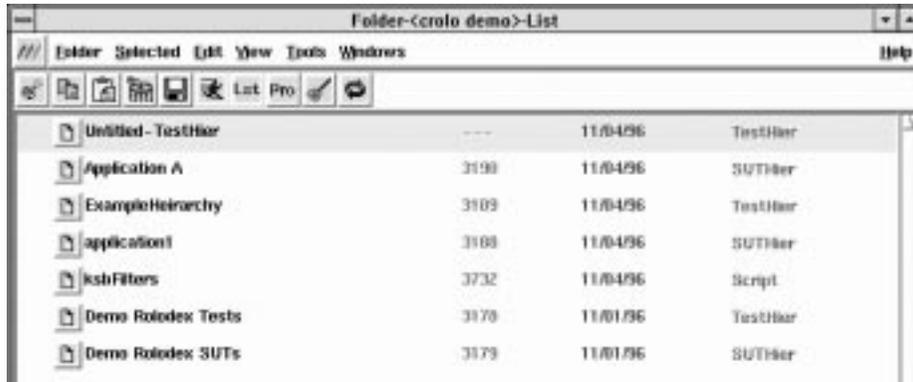
Figure 8-3. New Test Hierarchy

3. Double click on the new Test Hierarchy icon to open it.

4. Click on the Lock icon to unlock it.

— The system will display a **Requesting Information** dialog like the one in Figure 8-4.
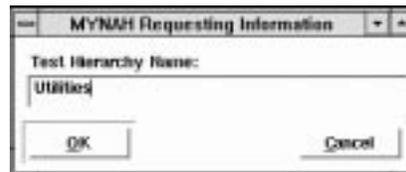
Figure 8-4. Naming a New Test Hierarchy

5. Type in a name for the new Test Hierarchy, and click **OK**. (We typed in **Utilities**.)

— After we click OK, the system assigned the new name to the Test Hierarchy, but left it in edit mode (Figure 8-5).
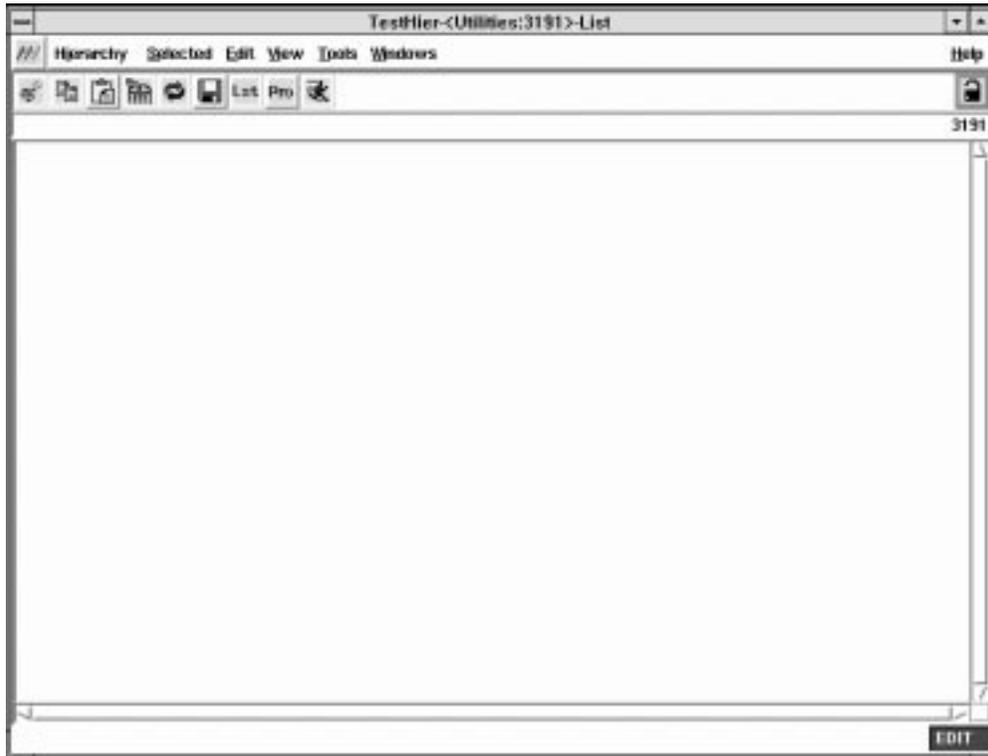
Figure 8-5. New Test Hierarchy Named.

8.2.2 Creating Test Objects in a Test Hierarchy

Now we have a Test Hierarchy to place our Test objects in. We will create test objects to represent the Test Plan we illustrated in Figure 8-2.

We will create all the Test objects we need before naming them. In effect we will be creating the structure of our Test Hierarchy before we specify any attributes of the Test objects themselves.

To create Test objects for our example click on the Lock icon to unlock the Hierarchy:

1. Click on the **Hierarchy** menu and then on the **New Test** option.

   — The system places the first Untitled Test object in the hierarchy at the highest level. See Figure 8-6. This object will represent the Operating System.
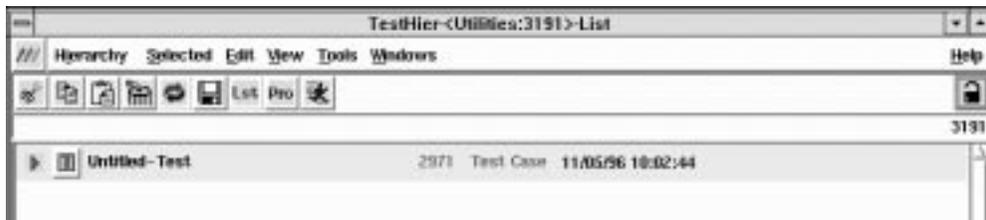
Figure 8-6. Test Object at the First Level of a Hierarchy.

2. Click on the **Selected** menu and then on the **New Test** menu option again.

   — The system will create another Untitled Test object as a child of the first object. See Figure 8-7. This Test object will represent a feature.
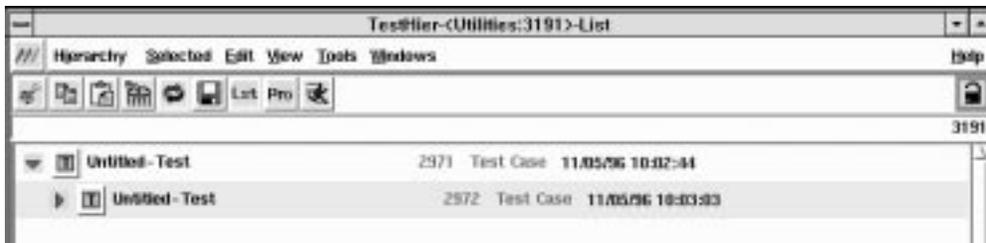
Figure 8-7. Child Test Object at the Second Level

3.　Click on the **Selected** menu and then on the **New Test** menu option.

　　— The system places another Untitled Test object in the hierarchy as a child of the second level we just created. This Test object will document our first test case. See Figure 8-8.
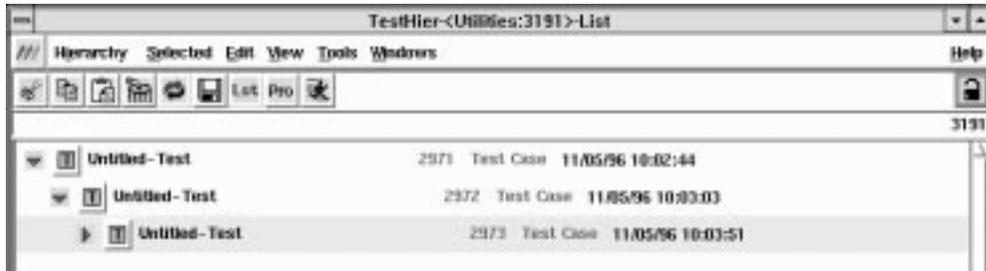
Figure 8-8.　Child Test Object at the Third Level

4.　Select the Test object at the second level and then click on the **Selected** menu and the **New Test** option.

　　— The system places a new Test object in the hierarchy as a "sibling" (at the same level) as the last one you created. Figure 8-9 illustrates this.

> **NOTE —** See in our example Figure 8-9 that the system created the new Test object *above* its sibling. For example, following our example in Figure 8-2, Test object 2.4.1.1. would appear *below Test object* 2.4.1.2.
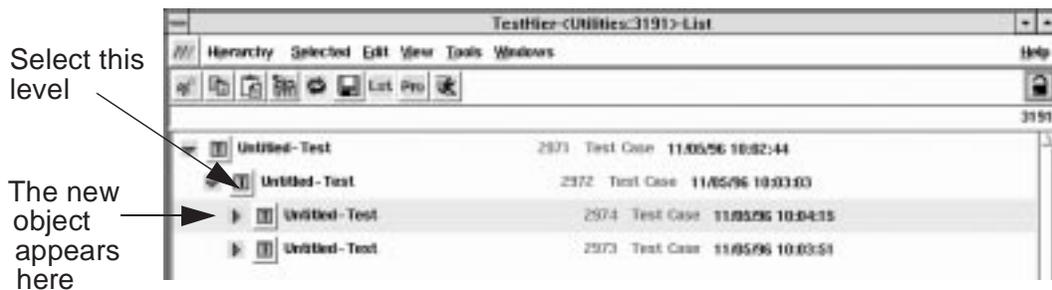
Figure 8-9.　First Sibling Test Object at the Third Level

5. Select the Test object at the second level and then click on the **Selected** menu and the
   New Test option.

   — The system places a new Test object in the hierarchy as a "sibling" (at the same
     level) as the last two you created. See Figure 8-10.
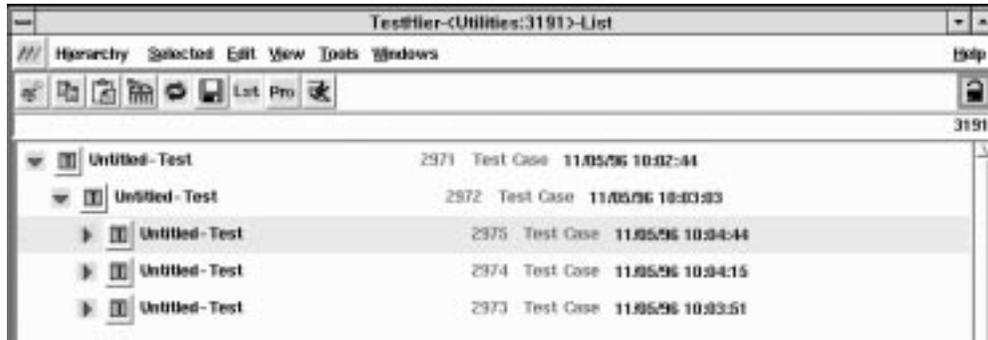
Figure 8-10. Second Sibling Test Object at the Third Level

6. Now that we have added all the Test objects we need, click on the **Hierarchy** menu
   and then on the **Save** option.

We will explain how to specify the name and the other attributes of a Test object in the next
section. We went ahead and named all the objects we created in this hierarchy and when we
were finished the Test Hierarchy looked like Figure 8-11.
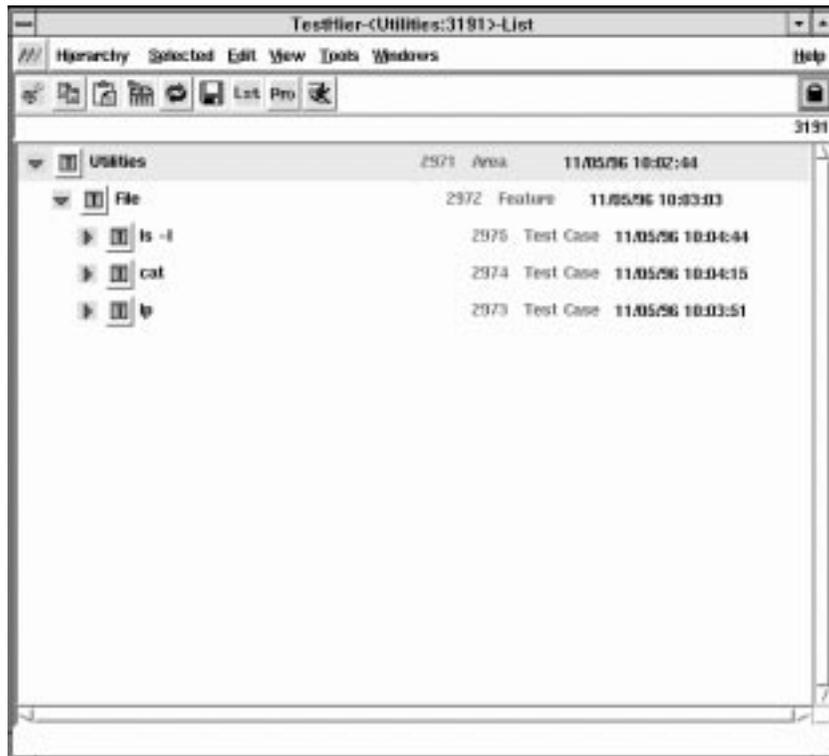
Figure 8-11. Completed Test Hierarchy

8.3   Specifying a Test Object's Attributes

We will return to our **crolo** example to explain how to specify an object's attributes. Remember, when we say attributes we are referring to attributes and what we call properties. Attributes and properties identify a Test object to the system and provide information the system can use to help you manage your test plan.

We copied the **Demo Rolodex Tests** Test Hierarchy into the **crolo demo** folder using the Database Browser.

For our example, let's suppose that we want to create a new test case under the **crolo** test. This new Test object will document the testing of a small database of names. To create a new Test object in the example hierarchy:

1. Double click on the icon for the **Demo Rolodex** Test Hierarchy in the **crolo demo** folder.

2. Click on the Lock icon of the **Demo Rolodex** Test Hierarchy to unlock the hierarchy. (If you haven't unlocked it already.)

3. Select the **crolo** Test object. (We want the new object to appear as a child object of **crolo**.)

4. Click on the **Selected** menu and **New Test** option.

   — The system will place a new untitled Test object in the hierarchy. See Figure 8-12.
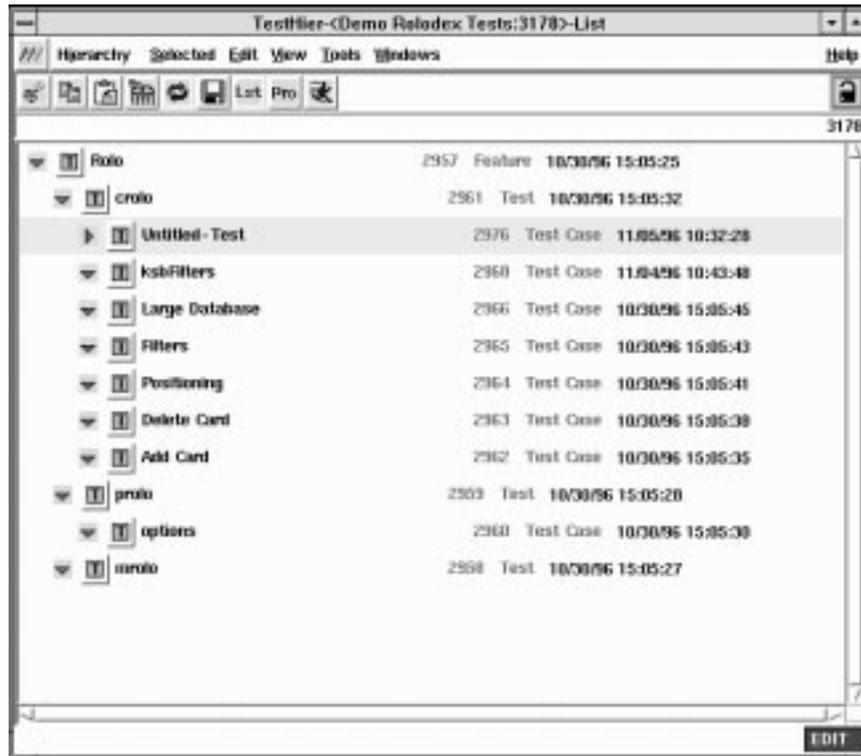
Figure 8-12. New Test Object in Hierarchy

5. Click on the **Hierarchy** menu and then on the **Save** menu selection.

Once you have created a Test object, you will have to define its basic properties. This is done with the Test object Properties view shown in Figure 8-13. Double click on the Test object's icon to open it to the Properties view.
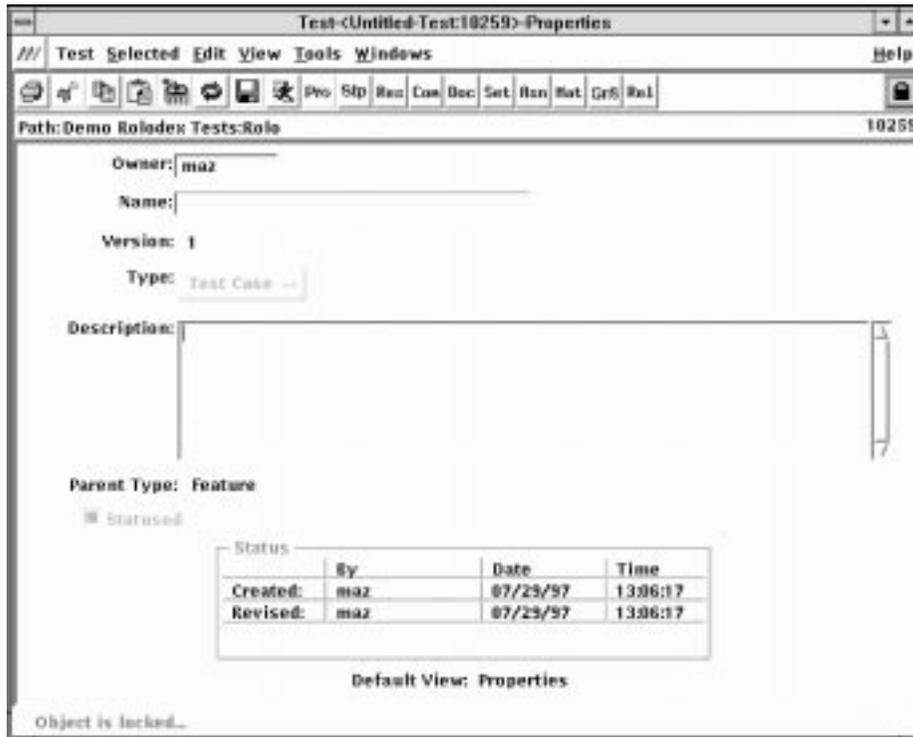
Figure 8-13. Test Object Properties View

A Test object's path (**Path:Demo Rolodex Tests:Rolo**) appears below the Tool Bar. It's **ID** appears to the right, below the Lock icon.

The Properties view has the following items on it:

Owner	Is the owner of the Test object. This starts out with the login ID of the person who created the object. (e.g. **maz**) You can change the owner of an object.
Name	Is the name you give to a Test object.
Version	Is the version of the Test object. Note that, in the MYNAH System, this will always be set to 1.
Type	Is a type provided for your convenience so you can organize tests. We provide default values of **Area**, **Feature**, **Test** or **Test Case**, but your System Administrator can add to these values.
Description	Is a free-form data entry area that allows you to enter a description of the test or view the description for existing tests.
Parent Type	Displays the parent Test object type for the current object.

Statused Statused means that the object will be included in a Testing Progress Report. It is a checkbox that allows you to select whether or not this Test object will be statused.

8.3.1 When Should a Test Object be "Statused"

Typically you will want only the lowest level tests statused since these are the tests the system will count for Testing Progress Reports. (Refer to *Section 14: Examining Results and Reporting on Test Activities* for more information about this report). In our **crolo demo** example, we didn't status **rolo**, **crolo**, **prolo**, and **mrolo** since they are used to organize Test objects rather than represent actual test cases. We did status the **Large Database**, **Filters**, **Positioning**, **Add Card, Delete Card**, and **Options** test objects since they represent test cases.

8.3.2 Entering Basic Properties

We will explain how to enter basic properties using our example. Remember, we are defining a Test object for the Small Database test.

To enter a Test object's basic properties:

1. Position the pointer in the **Name** data entry field and type in the name of the new Test object (for our example, **small database**).

   — The system automatically displays the **Version**: as **1**.

2. Select a type from the **Type** option-list (for our example, select **Test Case**).

3. Position the pointer in the **Description** data entry field and type in a description for the new test. We typed in a description of what the test case does.

   — The system sets the **Parent Type** which for our example is **Test**.

4. Click the **Statused** checkbox if you want to include this test object in Testing Progress Reports.

5. We will go on to describe the next view, so we won't save the Test object yet. If you want to save the Test object, click on the **Test** menu and then on the **Save** option.

— The MYNAH System will save the new object. Figure 8-14 shows the Properties view after we entered information but before we saved it. You can see that it is still in Edit mode.
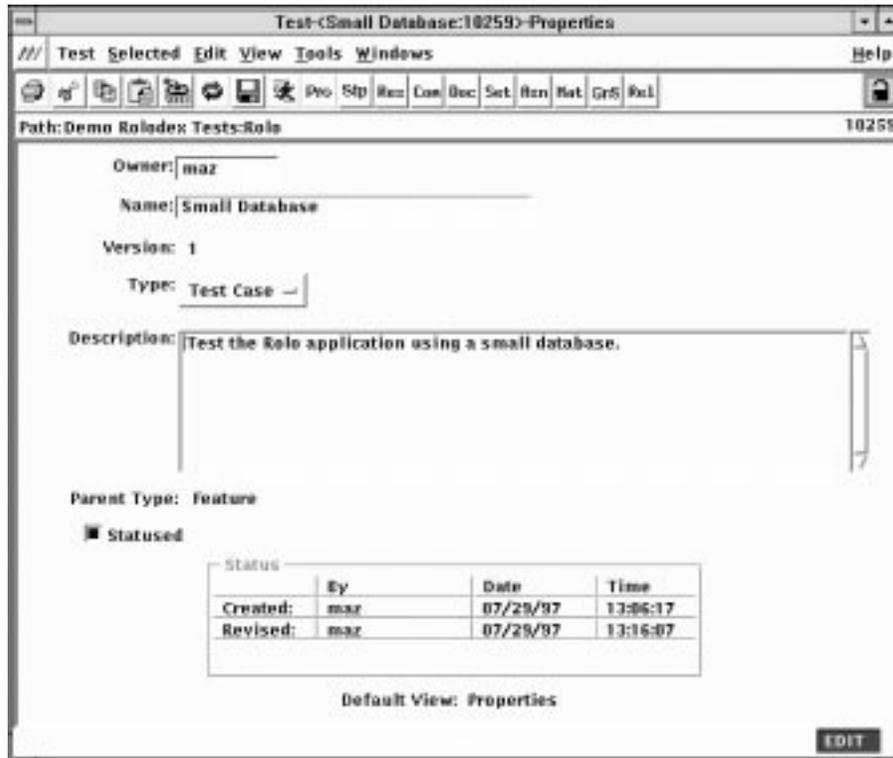
Figure 8-14. Completed Test Object Properties View

8.4   Specifying Steps in a Test

You can lay out the steps it will take to implement a test using the StepList view. This is especially useful if you are the one who is creating the test plan that someone else will implement. They can refer to this view while developing scripts for example.

From Test<- -->Properties view window you can access the StepList view from the **View** menu option or by clicking the **Stp** icon. Figure 8-15 shows the StepList view for the test case we are defining.
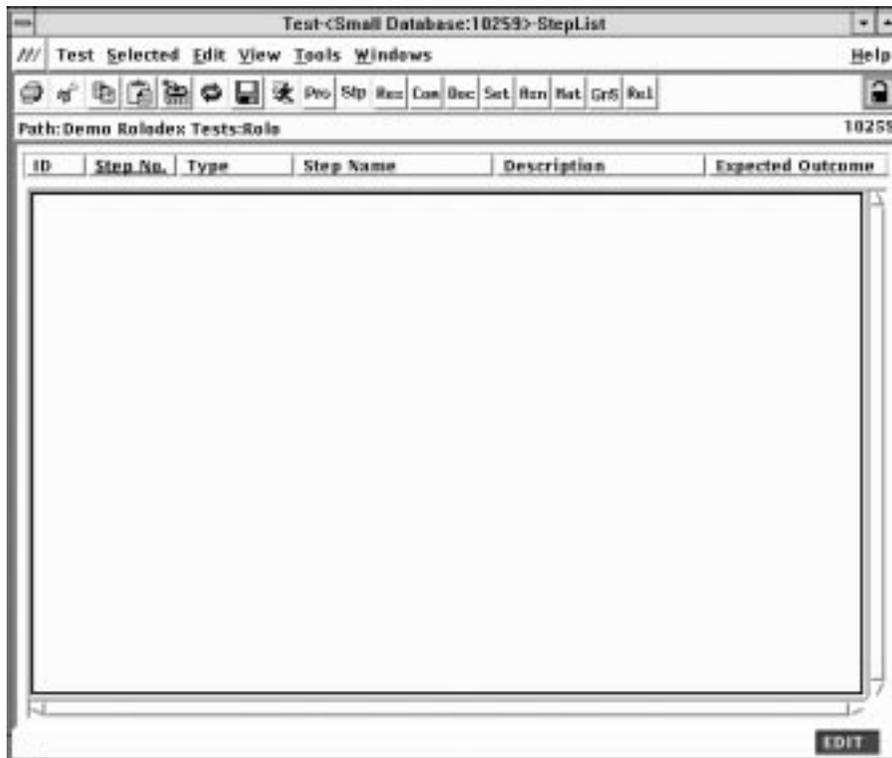
Figure 8-15. StepList View

Steps will appear in the client area as you create them. Among the attributes displayed here are:

Id   A system assigned ID number used to identify the step in the database.

Step No.   A sequential step number assigned by the system.

We will describe the other attributes, **Type**, **Step Name**, **Description**, and **Expected Outcome**, when we describe the StepList Properties view in Section 8.4.1.

8.4.1   Adding Steps

Let's suppose that there are three steps for the test case we are creating:

 • Logon to the remote system

 • Start the **crolo** application

 • Verify that the crolo screen comes up

To add steps (from the Test<---> Steplist view):

 1. Click on the **Test** menu and the **New** menu then select the **Step** option .

    — The system will display a Requesting Information dialog like Figure 8-16.
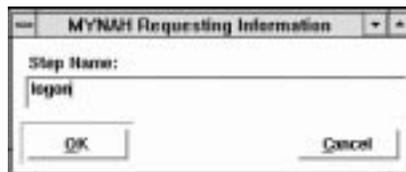
Figure 8-16. Naming a Step

 2. Type in a name for the step and click **OK**. (We typed in `logon` as the name of our first
    step.)

    — The system will place a new step in the List view. See Figure 8-17.

Double
Click
anywhere
on listing
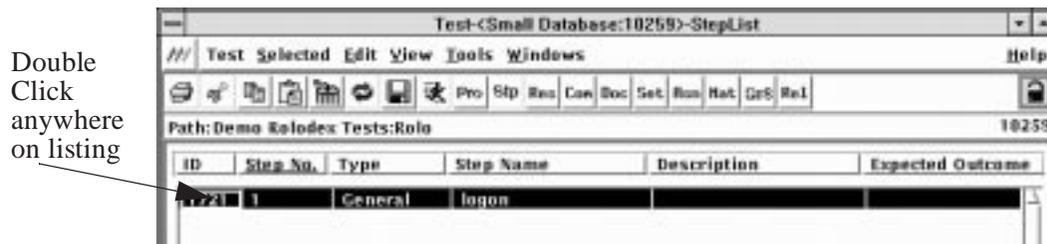
Figure 8-17. New Step in StepList View

3. Double click anywhere in the step listing.

— The system will display the Step Object Properties view shown in Figure 8-18.
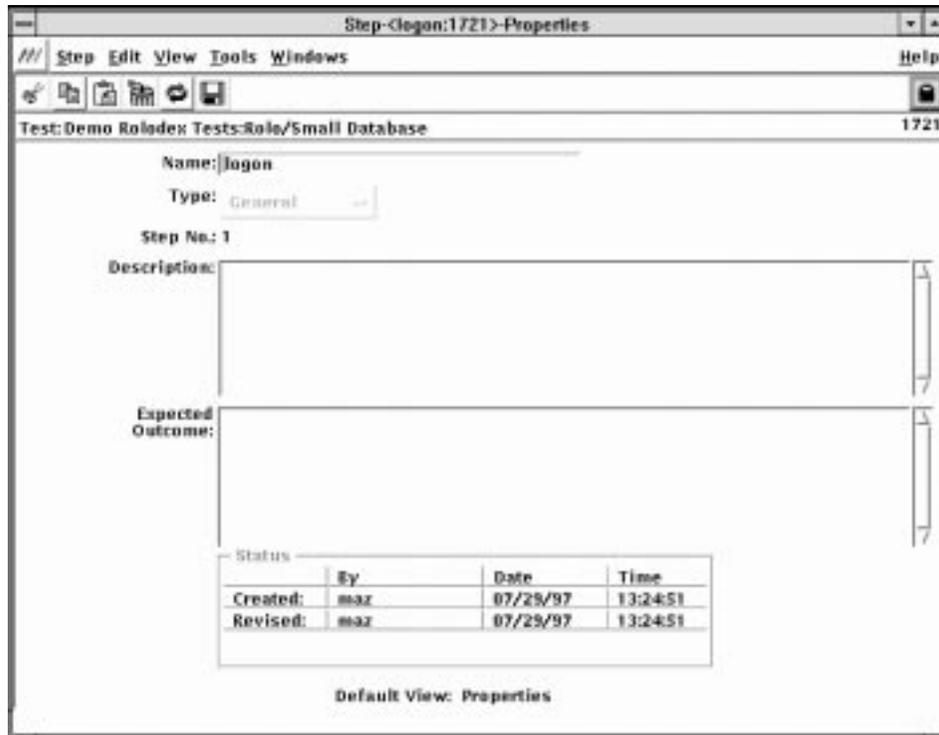
Figure 8-18. Step Properties View

The Step object Properties view allows you to define what the step is and what the expected outcome will be. Entries include:

Name	The name you give the step.
Type	The type of the step. Choices are: **General**, **Initialization**, **Validation**, and **Cleanup**. We supply these values, but your System Administrator can add to them.
Step No.	The sequential position of this step in a list of steps.
Description	A free form data entry area where you can type in a description of the step.
Expected Outcome	A free form data entry area where you can type in a description of the expected outcome of the step. (This field is limited to 200 characters).

8.4.1.1   Example of Defining a Step Properties

Following our example, let's add the first step for the **small database** Test object.

1. If the Step view is locked, click on the Lock icon to unlock it.

2. Position the pointer in the **Name** field if you want to change the name of the Step. (We accepted the name we already entered).

3. Click on the **Type** menu option list and select the type of step this is. (We selected **Initialization**).

4. Position the pointer in the **Description** field and type in a description of the step. See Figure 8-19 for the description we entered.

5. Position the pointer in the **Expected Outcome** field and type in what you expect to be the outcome of a successful execution of this step. See Figure 8-19 for the outcome we entered.

— The Properties view looks like Figure 8-19 when we finish.

Figure 8-19. Completed Step Properties View

6. To save the Step object, click on the **Step** menu and then on the **Save** option.

    — After you save the first step it will appear in the StepList View.

We repeated Steps 1 through 6 of this procedure to define the remaining steps for this test case. When we were through the StepList View will look like Figure 8-20.
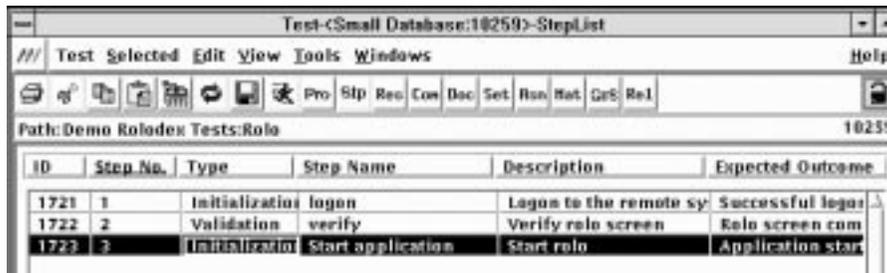
Figure 8-20. StepList with Example List

8.4.1.2   Re-Arranging Steps

If you look closely at Figure 8-20, you will see that our steps are out of order. We want the step named **Start Application** before **Verify**.

We can re-arrange steps using a simple drag and drop method. We will select the Step and drag it to a new position using the **Middle** mouse button. To rearrange steps:

1. If the Test object is locked, click on the Lock icon to unlock it.

2. Position the pointer on the step you want to move.

3. Press down and hold the **Middle** mouse button while dragging the step to a new position.

    — Notice that the step you select is boxed, i.e., the lines surrounding the step are darkened.

4. Release the **Middle** mouse bottom when the step is in the position you want.

    — The system moves the step to the new location and re-numbers the steps. Figure 8-21 shows the StepList view with the new order.
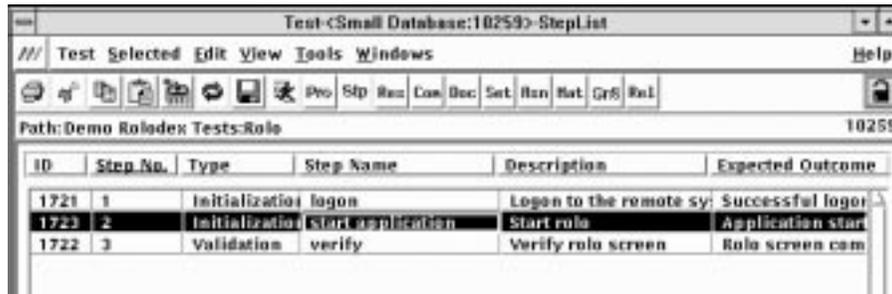
Figure 8-21. StepList with New Step Order

8.4.1.3   Viewing a Step List

As you can see in Figure 8-21, sometimes information in the **Description** and **Expected Outcome** fields of the Step List view is truncated. This will be inconvenient if you are following the Step List to implement a test since you will have to scroll the display or open the Step to its Property view to see all the information.

You can change the size of the **Description** and **Expected Outcome** fields, or any other field on this view. To do this:

• Resize the window

• Drag the edge of the column heading to resize it

Figure 8-22 illustrates this.
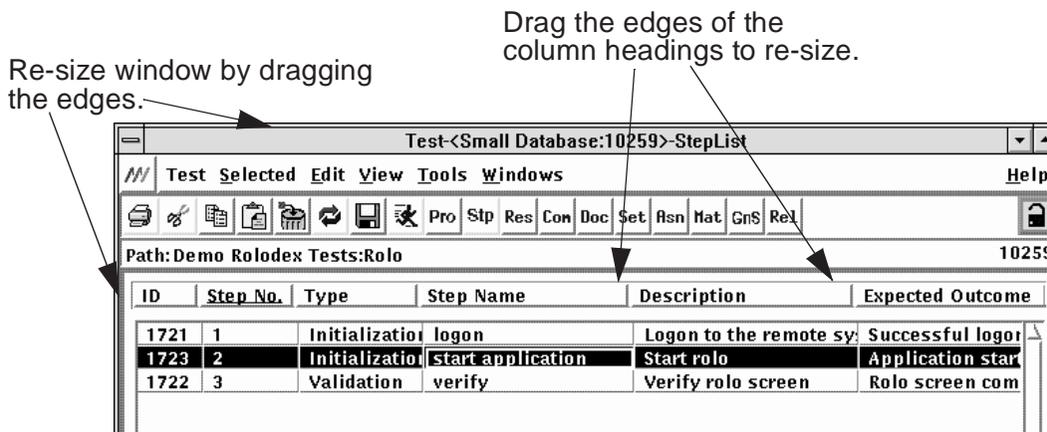
Figure 8-22. Resizing a Step Display

8.5   Associating Test Objects with SUTs and Keywords

You can associate tests with SUTs and Keywords through a Test object. You can also display a Test object's associations with Issue objects through a test object, although the actual associations between Test and Issue objects are made through the Issue object.

You perform these activities using the Test object Associations view which you can access from the **View** menu or by clicking on the **Asn** icon. The Associations view is shown in Figure 8-23.
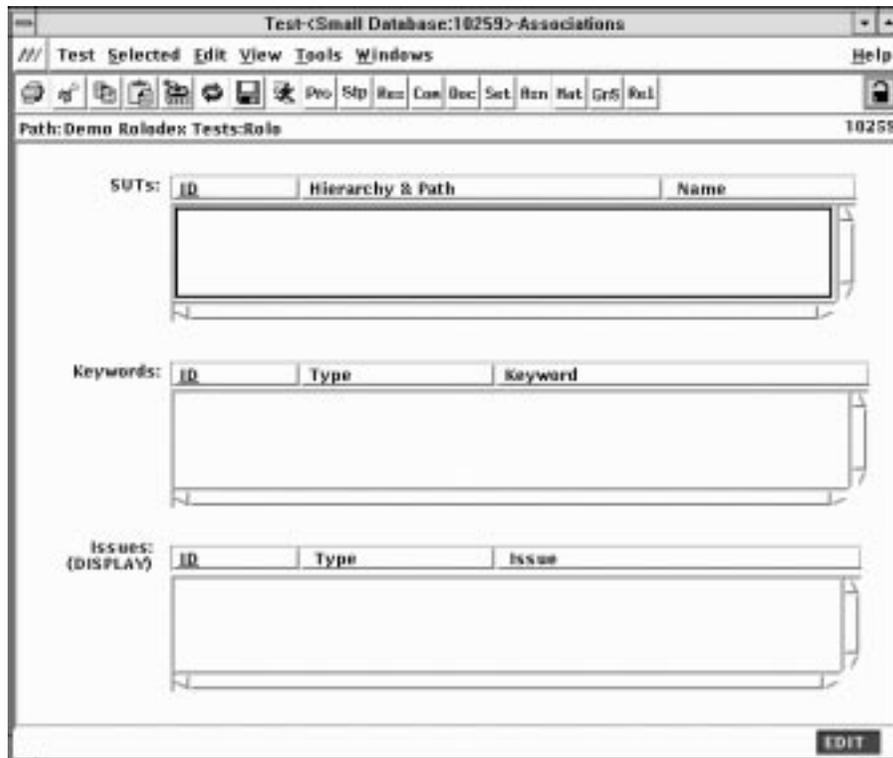
Figure 8-23. Test Object Association View

The scrolling lists from this view will Display the following: :

SUTs	The SUTs associated with the Test object. You can make association to SUTs through this list.
Keywords	The Keywords associated with the Test object. You can make association to Keywords through this list.
Issues	The Issues associated with the Test object. This is a display only list; associations are made through the Issue object.

Associating SUT and Keyword objects with a Test object is a two step process:

1. Use the Database Browser to locate the SUT or Keyword object you want to associate with the current test object;

2. Copy it from the Database Browser into the scrolling list on the Test object Associations view.

You can also copy SUT and Keyword objects from a folder, any other list or a ruler display. We explained how to do this in *Section 4: Using MYNAH Objects.*

8.6   Displaying Documentation and Requirements

The Documentation view provides you with a way to display documents and requirements associated with the currently opened Test object. Like the Associations view, you can access the Documentation view from the **View** menu or by clicking on the **Doc** icon.

After you select the Documentation view, the MYNAH System displays the Documentation view shown in Figure 8-24.
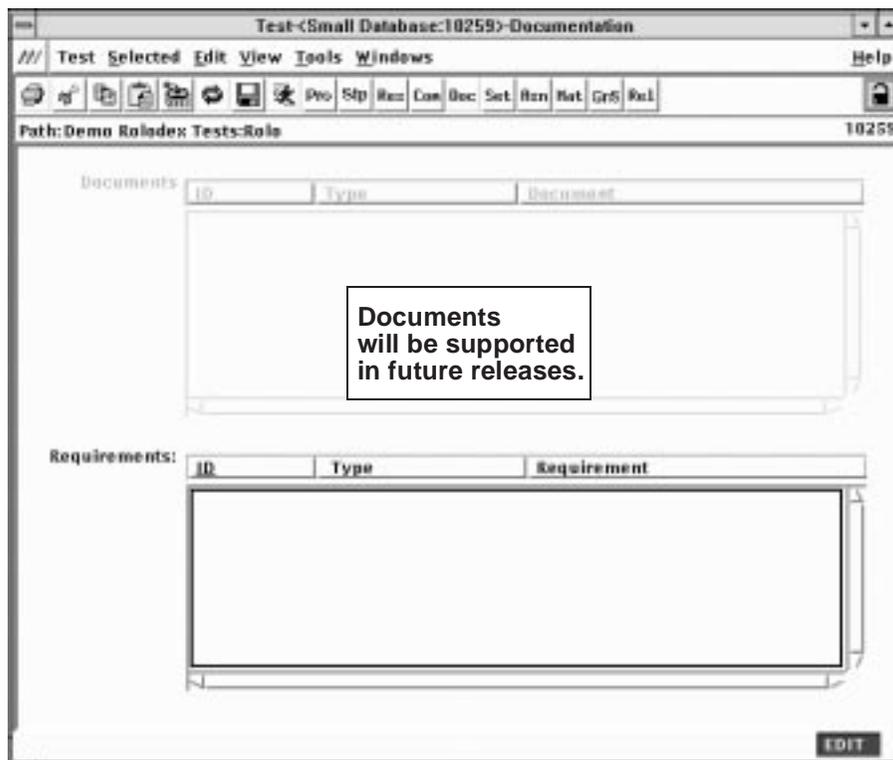
Figure 8-24. Test Object Documentation View

You can see that there are two scrolling lists:

Documents   List shows the name of the document and the type of document it is.

Requirements List shows the requirement and the type of requirement it is.

Both requirements and documents are assigned ID numbers by the MYNAH System.

> **NOTE —** Document objects will be supported in future releases.

The Requirements list is populated with requirements associated with the currently opened Test object. The Requirement objects that appear in this view should be the conditions that this test validates.

You associate Requirement objects with the currently opened Test object through this view. You make the association in the same way you would make any other association. We explained how to do this in *Section 4: Using MYNAH Objects.*

8.7   Specifying Test Settings

The Test object Settings view allows you to specify ranking and automation information. It also allows you to identify the script that implements the test. You access the Settings view from the **View** menu or by clicking on the **Set** icon. The Test object Settings view appears in Figure 8-25.
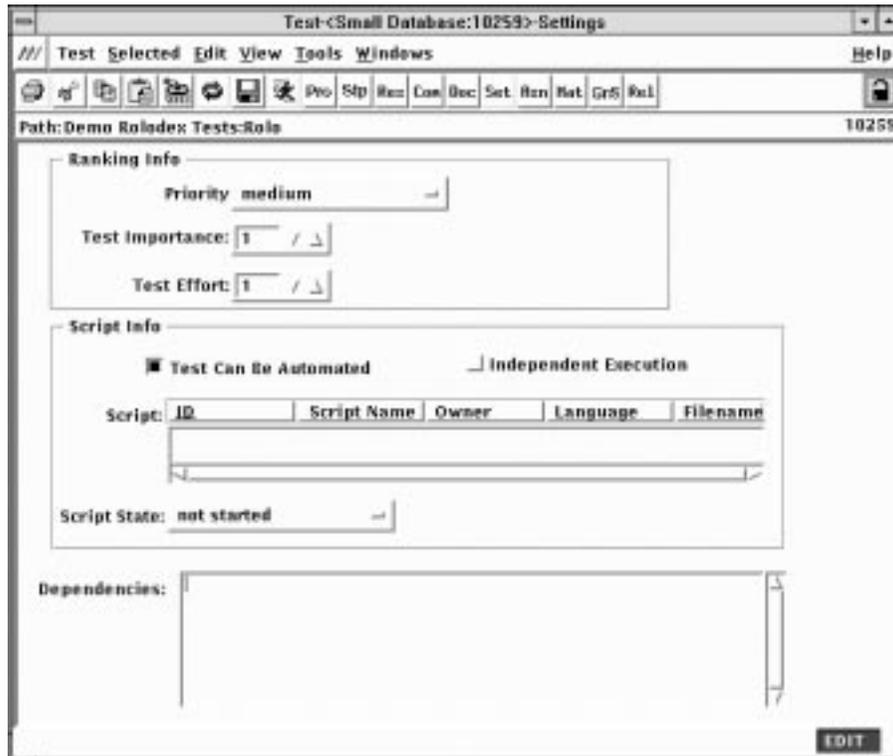
Figure 8-25. Test Object Settings View

8.7.1   Ranking a Test Relative to Other Tests

The **Ranking Info** data entry/display area is important for coordinating the current test with your other test efforts. The information you enter here includes:

 Priority   Refers to the relative importance of performing this test. The Priority
                           can be **High**, **Medium,** or **Low**.

Test Importance	Refers to how important the test is to your test efforts relative to other tests. Test Importance is a number value which runs from **1** to **10**. Ten (10) indicates the highest importance; one (1) indicates the lowest.
	This value can be used in the **Testing Progress Report**. Refer to *Section 14:Examining Results and Reporting on Test Activities* for more information about this report.
Test Effort	Refers to the amount of effort you will have to expend to perform this test. Again this is a number value running from **1** to **10**.
	This value is used in the **Testing Progress Report**. Refer to *Section 14:Examining Results and Reporting on Test Activities* for more information about this report.

You derive **Priority** by combining the values of **Test Importance** and **Test Effort**. For example, if **Test Importance** was high (8-10) but **Test Effort** was also high (8-10) you could decide that **Priority** is **Medium.**

You could then use the **Priority** value to select which tests to run if you were under a time constraint. For example, if you had only a short time to finish a test cycle, you could look for tests that have a high priority, i.e., they are important but they don't take too much effort to run. This assumes that these tests will give you the most coverage with the least effort.

The meanings of these values are not fixed by the MYNAH System. You define their meaning to fit your business and testing needs.

8.7.2   Identifying the Script That Implements a Test

The **Script Info** data entry area allows you to identify a script that implements the current test. Test objects document tests. The code that actually implements the test is referenced by a Script object. In some cases, you may not be able to automate a test --- turning on and off a computer for example.

The information here allows you to specify the following:

Test Can be Automated	Whether or not the current test can be automated using a scripting language. (If you check this, the remaining fields in the Script Info data entry area enabled. If it's not checked the fields are disabled.)
Independent Execution	Whether or not the script can be run independently of other scripts. In most cases this should be true. However, if you are using a parent script, the child script may not run independently of the parent.

Script	Displays the name of the Script object which contains the code for this test. You can copy a Script object into this area from the Database Browser or a folder.
Script State	Can be set to describe how much of the test has been automated in a script. The values are **complete**, **partial**, and **not started**.

8.7.3   Entering Dependencies

Finally at the bottom of the Test object Settings view there is a free form data entry area where you can enter comments about any dependencies the test has.

8.7.4   Using the Test Object Settings View with Our Example

Let's place the Test object Settings view in the context of our example. The test case we are documenting with this Test object is **small database.** The test is important and the it has a relatively long run time. Our test can be automated and can be executed independently because it is in a stand-alone script. These are the basic parameters we will need to specify our Test Settings.

To specify test settings:

1. Select the test's priority relative to other tests by selecting a priority from the **Priority** drop-down list (for our example, we would select **Medium**).

2. Click on the **Test Importance** spin button to specify the test's importance relative to other tests. (Again we would make this a high priority by selecting a high number from the number wheel, e.g., "8.")

3. Click on the **Test Effort** spin button to specify the level of test effort you want for the test. (The test will take a long time to run, so we set the **Test Effort** high, e.g., "8." Remember, with **Test Importance** and **Test Effort** high we rated the test as a Medium priority in our scheme.)

4. If this test can be automated, click the **Test Can be Automated** check box. (The test can be automated.)

5. If the test can be executed by itself, click on the **Independent Execution** check box. (Our example script can be executed by itself.)

6. Copy in a Script object from the Database Browser or a Folder. (We assume that a script has been created.)

7. Select a state from the **Script State** drop-down list. (Our example is fully automated, so we would select **completed** to indicate that the script is finished.)

8. To enter information about any dependencies the script has, position the pointer in the **Dependencies** data entry field and type in your comments.

9. We will go on to describe the next view, so we won't save the Test object yet. If you want to save the Test object, click on the **Test** menu and then on the **Save** option.

— The Test object Settings view looked like Figure 8-26 just before we saved it.
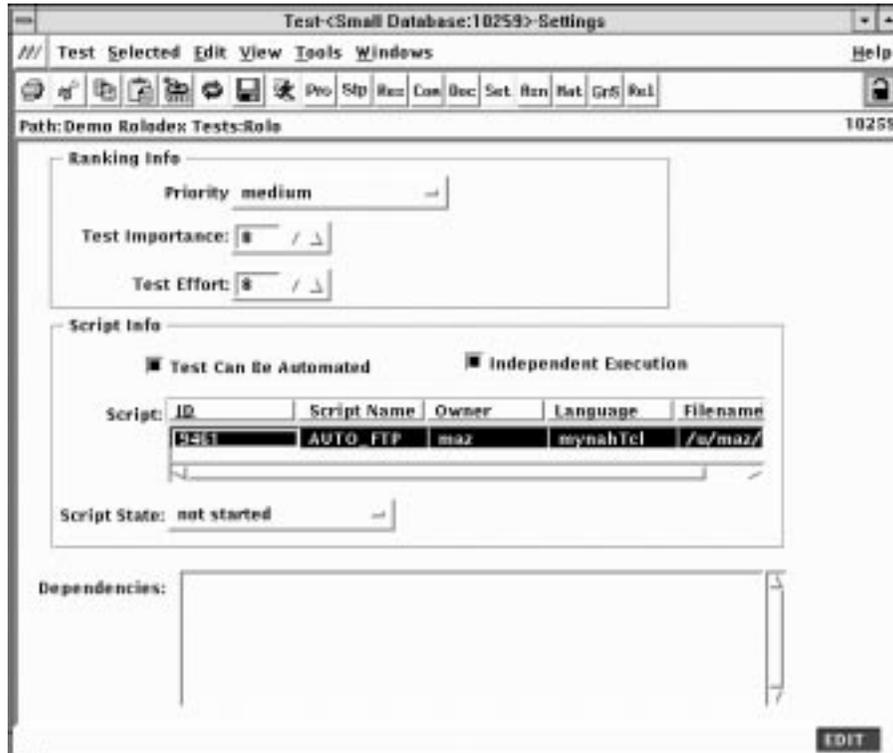
Figure 8-26. Completed Example Test Object Settings View

8.8   Running Automated Tests

You can run tests directly from a Test object or from a Test Hierarchy. More accurately, you can run the script that implements a test directly from a Test object. When you run a script from a Test object, the MYNAH System runs the test in the BEE.

When you run a test the MYNAH System will check the selected Test object, all of its children, and all its children's children for associated scripts. The system will run those scripts if:

- The test is an automated test (**Test Can be Automated** is checked).

- The test is associated with a script or scripts.

- Any scripts associated with the test are complete and capable of being run independently.

To run a script from a test object:

1. Save the test, if you haven't already done so, by clicking on the **Test** menu and then on the **Save** option.

2. Click on the **Test** menu and then on the **Run** menu selection.

   — The system will display the Run Script dialog. See Figure 8-27.

> **NOTE —** The script for our test already appears in the **Input Settings** area of the dialog. Refer to *Section 10: Using Script Objects* for more information about the settings available on this dialog. You can simply run the test with the default settings.
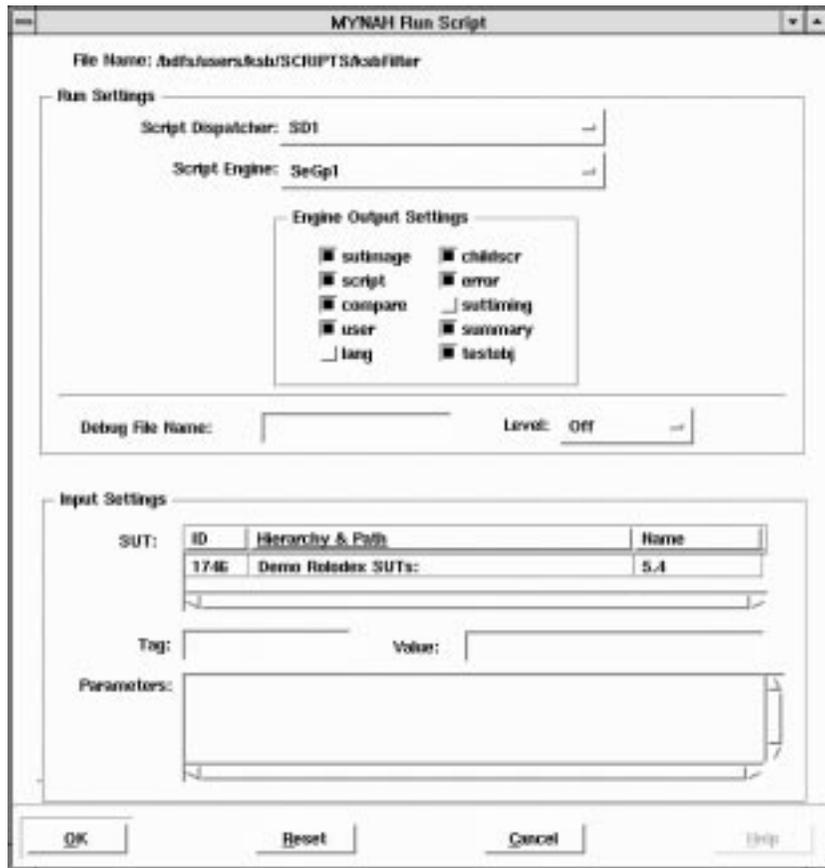
Figure 8-27. Run Script Dialog

3. Click **OK** to run the test.

8.9 Recording Manually Run Tests

There are tests that you can not automate. These are tests like turning a computer off and then on to see an application's reaction. You can document tests such as these with a Test object and then record when the test was run and what it's results were through the MYNAH System.

You do this using the Manual Run dialog shown in Figure 8-28. You access this dialog from the **Test** menu and **Record Manual Run** menu option.

Figure 8-28. Record Manual Run Dialog

We describe the attributes on this dialog in *Section 14: Examining Results and Reporting on Test Activities*. Refer to this section for more information about the results you can define for a manual run.

8.10 Viewing a Test Object's Results

Under certain circumstances, when you run a test, the MYNAH System will produce a Result object. We will discuss this in detail in *Section 14: Examining Results and Reporting on Test Activities*. For now, note that you can view a test's results from a Test object.

You can easily review the results created for a Test object using the Results view. You access this view from the **View** menu or by clicking on the **Res** icon. See Figure 8-29. The MYNAH System lists the **Date/Time** a Results object was created for the test, its **ID**, the **Activity Code**, **Status** and the **SUT** against which it was run. This scrolling list is display only.
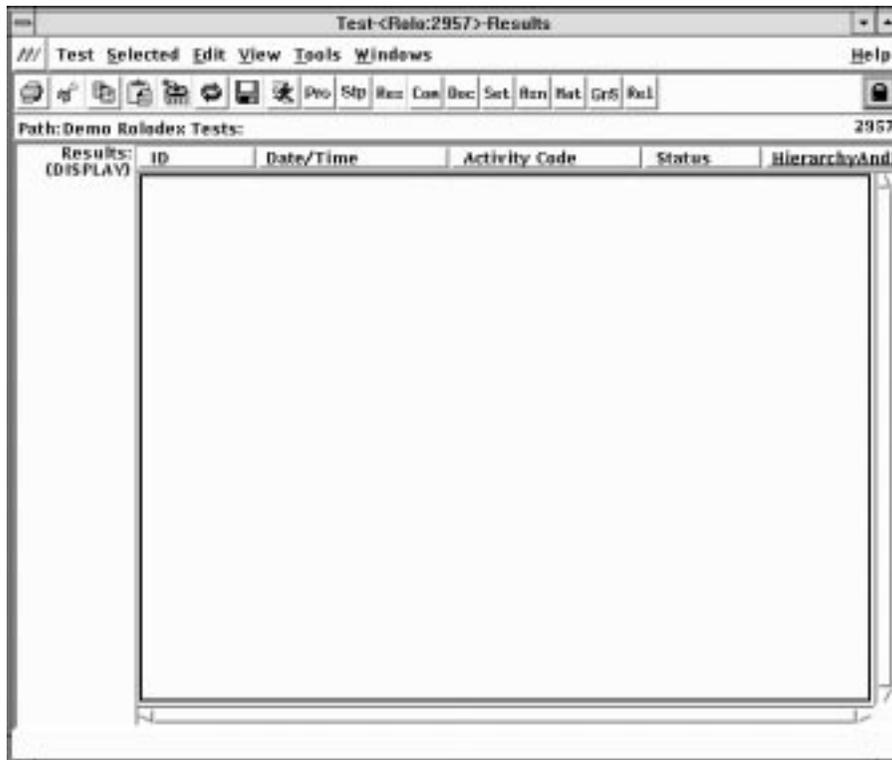
Figure 8-29. Results View

8.11 Adding Comments to a Test Object

The MYNAH System provides you with a convenient way to add comments to a test object. This is done with the Comments view. Like other test views you access it from the **View** menu or by clicking on the **Com** icon.

After you select the Comments view, the MYNAH System displays a window similar to the one shown in Figure 8-30. You see that this is a text view which operates like any other text view.
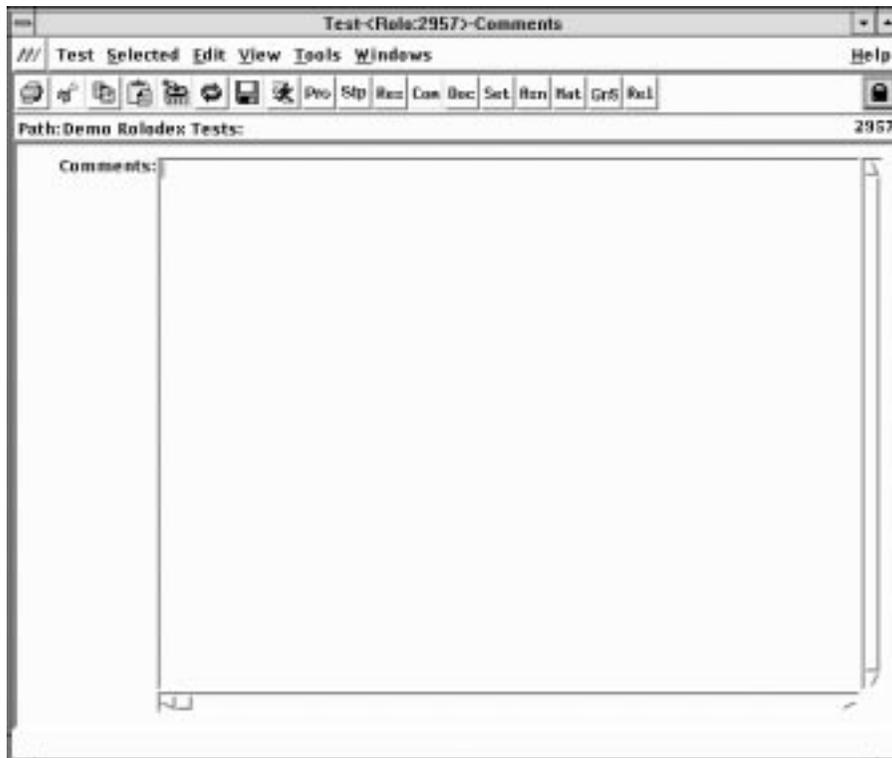
Figure 8-30. Test Object Comments View

To enter comments:

1. Position the cursor in the first empty line of the **Comments** data entry area and begin typing in your comments.

2. This is the last view for our example Test object, so we clicked on the **Test** menu and the **Save** selection to save the Test object.Test Object

8.12   Matrix View

The Test object Matrix view (Figure 8-31) is part of the AETG System, and is only available if you license the AETG Systemwith the MYNAH System. For information on the Matrix view, see the *AETG System Users Guide*.
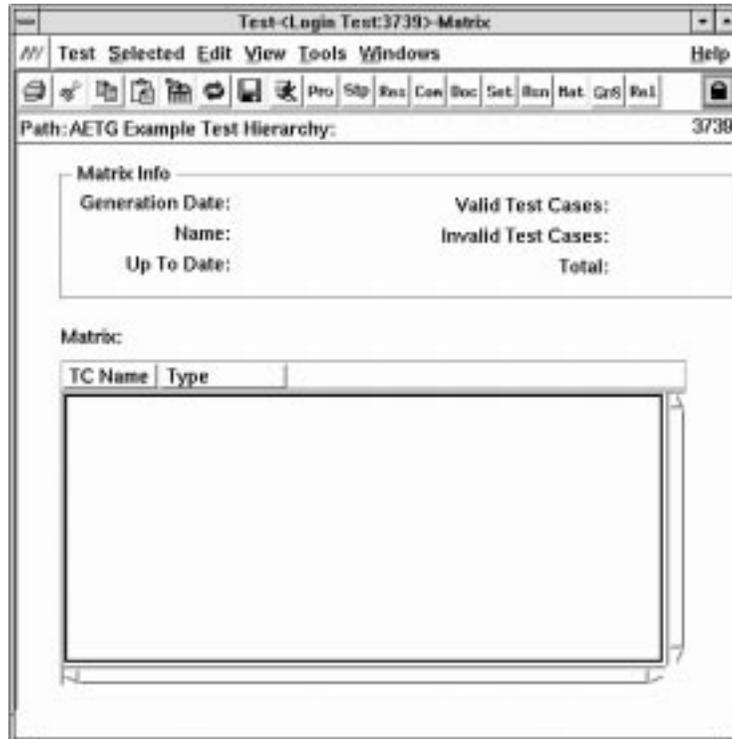
Figure 8-31. AETG Test Object Matrix View

8.13   GenScript View

The Test object GenScript view (Figure 8-32) is part of the AETG System, and is only available if you license the AETG Systemwith the MYNAH System. For information on the GenScript view, see the *AETG System Users Guide*.
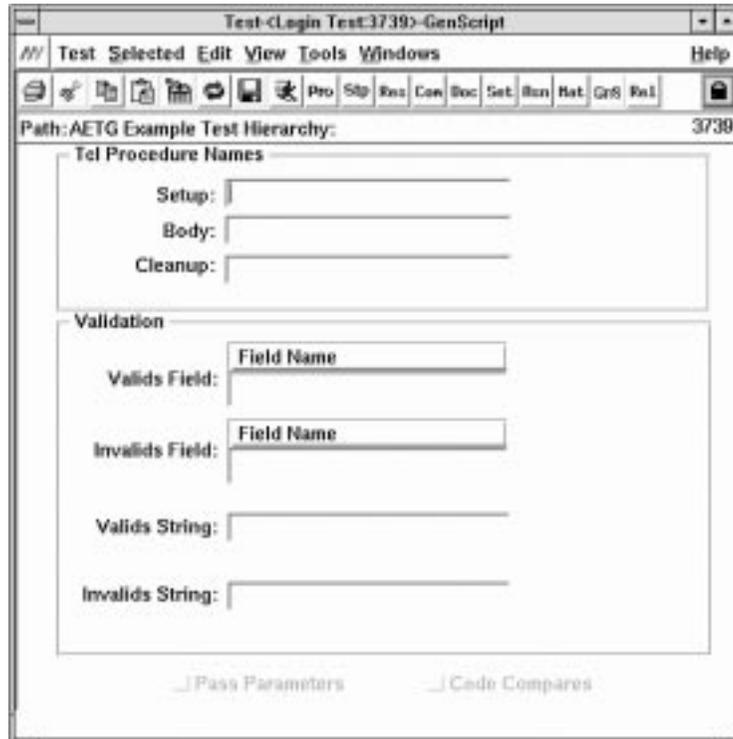
Figure 8-32. AETG Test Object GenScript View

8.14   Relations View

The Test object Relations view (Figure 8-33) is part of the AETG System, and is only available if you license the AETG Systemwith the MYNAH System. For information on the Relations view, see the *AETG System Users Guide*.

Figure 8-33. AETG Test Object Relations View

9. Using Issue Objects

Issue objects allow you to track ongoing issues while you are testing. They contain information about problems or failures found while testing or scripting.

In this section we will explain how to:

- Create Issue objects

- Associate Issue objects with tests and other documents.

9.1 How Issue Objects Help You

Issue objects provide you with a way to track any problems you encounter while testing or scripting. The problem could be with a SUT or the environment. You may find, for example, that a test doesn't quite cover the requirements and that you need to run additional tests or that functionality has been added to a SUT but not to the Test Plan. These and other issues can be recorded in an **Issues** object.

9.1.1 Creating Issue Objects

You create **Issue** objects in a Folder or on the Desktop the same way you would create other objects in the MYNAH System. We covered this in *Section 4: Using MYNAH Objects.* Refer to this if you need to.

We created an Issue Object and added it to our **crolo demo** folder, the system displayed it as a line item in the Folder<---> List view. See Figure 9-1.
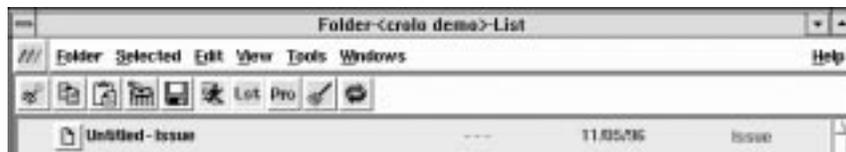
Figure 9-1. New Issue Object

9.1.2 Specifying an Issues Object's Properties

You open an **Issue** object by double clicking on its icon or by selecting it and then selecting **Open** from the **Selected** menu. After you do this, the system will display the Issues Properties view shown in Figure 9-2. We clicked on the Lock icon to unlock it for editing.
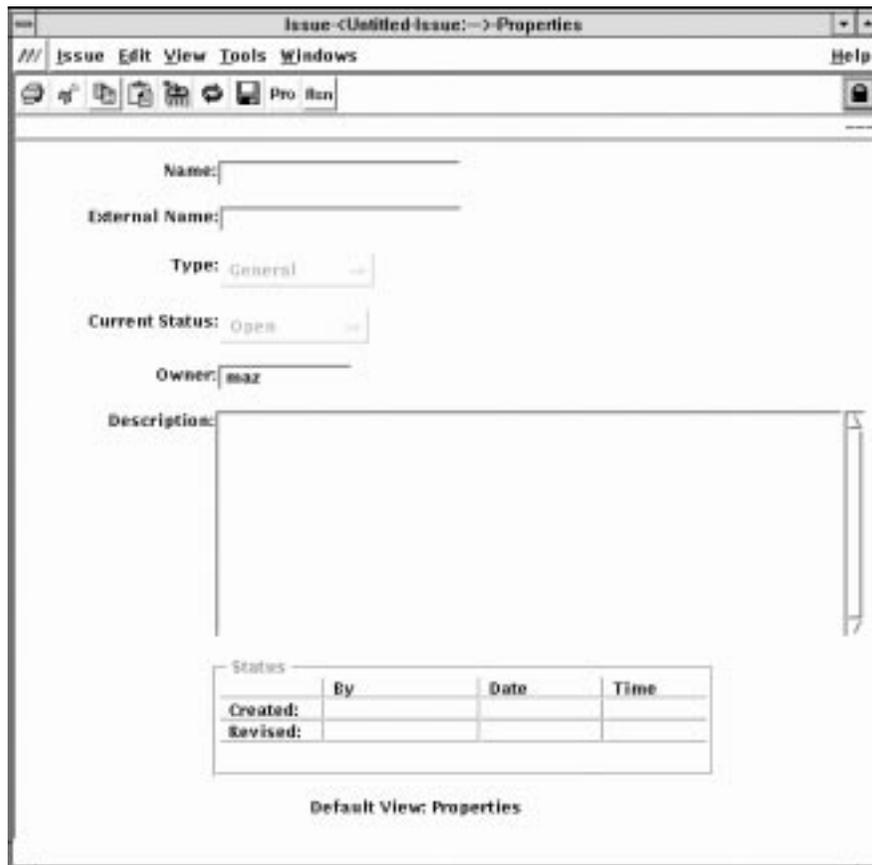
Figure 9-2. Issue Object Properties View

The attributes on this view are:

Name	A name you give to the issue.
External Name	An optional identifier used by other systems. It is a character field in which you can enter up to 30 characters. For example, if you used an external defect tracking system, you could enter the ID for this Issue here.
Type	The type of issues. Values are: **General, Environment,** or **Software.** Your System Administrator can add additional values to this list.
Current Status	The status of the issue. The status can be either **Open** or **Closed**.
Description	Free form data entry area where you enter a description of the issue.
Owner	The user who is currently responsible for this Issue object. By default this is the person who created the object, but the owner can be changed.

9.1.2.1   Entering Issue Object Properties Example

Let's take for our example a situation where the requirements do not quite cover the functionality of a SUT and so we want to log the fact that there are missing requirements.

To specify this issue we would:

1. Click on the Lock icon to unlock the Issue object.

2. Type in a **Name** for the issue. (We typed in `new requiremen`).

3. Type in an **External Name** for the issue. (We left this blank).

4. Select the object type from the **Type** option list. (We used the default **General**)

5. Select the current issue **Current Status** (We selected **Open**).

   — The system assigned an **Owner** who is the person who created the object. (We accepted this default).

6. Position the pointer in the **Description** data entry area and type in a description of the issues. (We typed in a description explaining the issue).

7. We won't save the object yet because we are going on to describe another view. If you want to save the object, click on the **Issue** menu and the on the **Save** option.

   — Our completed Issues Property view looked like Figure 9-3.
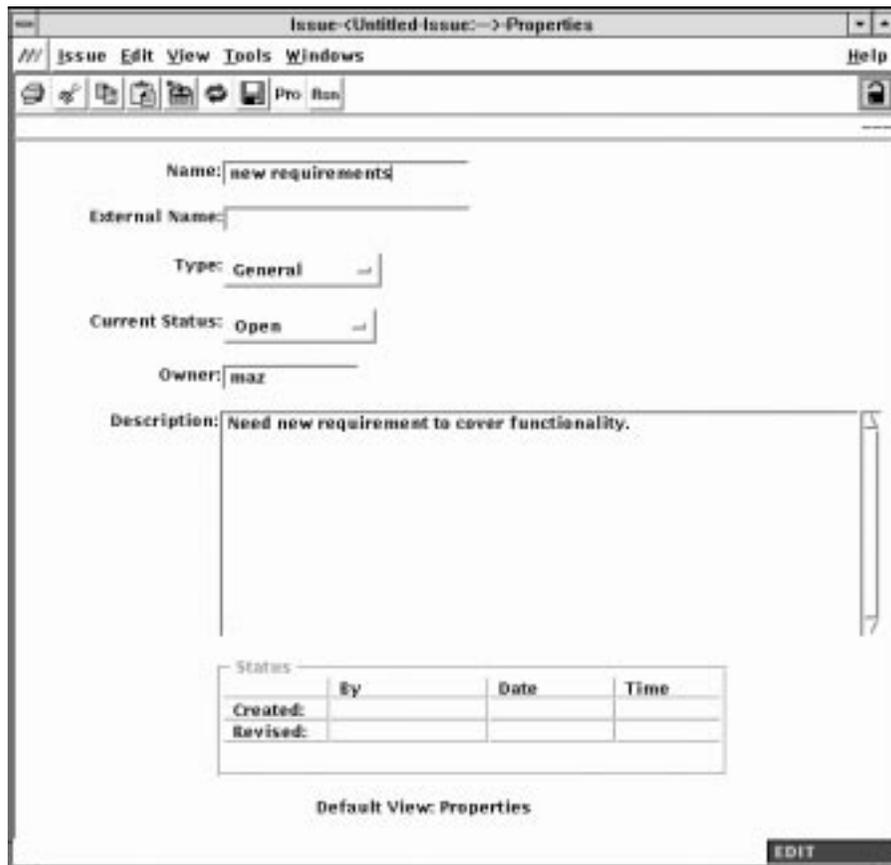
Figure 9-3. Completed Example Issues Properties View

9.1.3   Associating Issues with Result, SUT, and Test Objects

Once you have identified an **Issue** and have defined its properties, you may want to associate the **Issue** with **Result**, **SUT**, and **Test** objects. You can do this using the Association view for Issue object . You can access this view from the **View** menu or by clicking on **Asn** icon. Figure 9-4 shows the Association view.
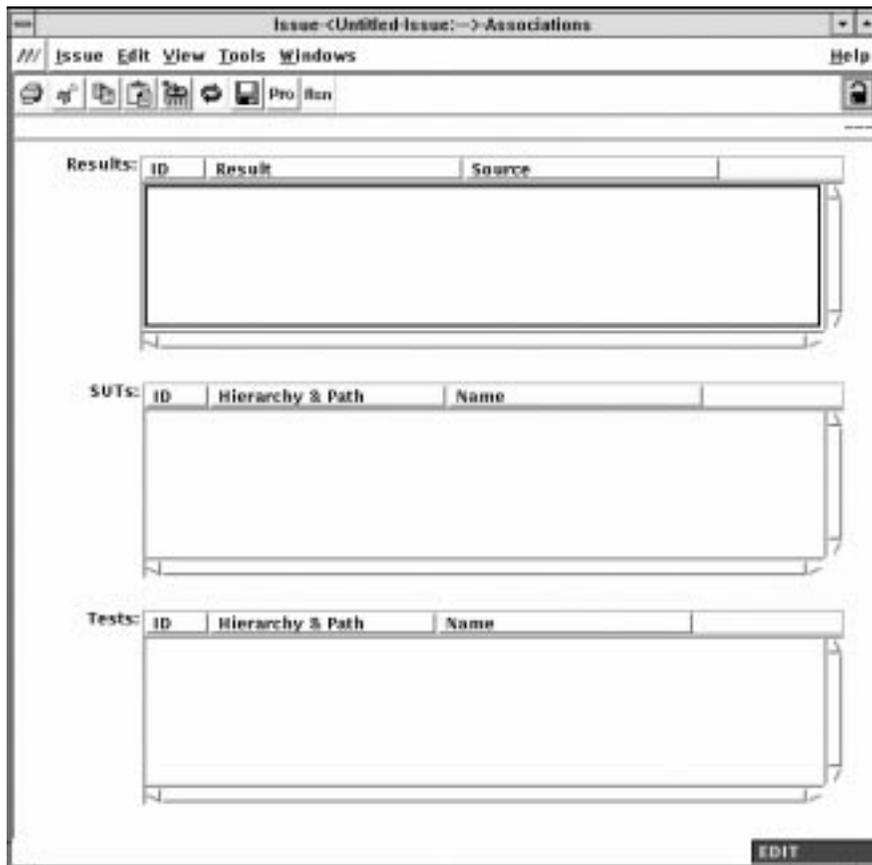
Figure 9-4. Issue Object Associations VIew

There are three modifiable scrolling lists on the Associations view, **Results**, **SUTs**, and **Tests** each associated with the current Issue.

9.1.3.1   Associating SUT with an Issue Object

The **Issues** you encounter may be related to a SUT, perhaps a new feature has been added to the SUT. In this case you want to relate the Issue object to the SUT.

> **NOTE —** You associate an Issue object to a SUT object
> in the same way you make any other association. If you
> need to refresh your memory on how to do this, refer to
> *Section 5: Using the Database Browser*.

9.1.3.2　　Associating Results with an Issue Object

In examining a result you may find an issue that needs to be documented. You can relate the issue to the Result object that raised the issue through the Associations view.

9.1.3.3　　Associating Tests with an Issue Object

Running a specific test may have uncovered an issue. The Issue object Associations view allows you to associate an issue with a test.

10. Using Script Objects

Script objects help you to document and maintain scripts that implement tests. Scripts themselves are test actions encoded using one of the scripting languages compatible with the MYNAH System. The code for a script resides in a file which the Script object references.

Script objects also allow you to maintain script code through the MYNAH System. You can open an existing script and edit the code for that script through a Script object's Code view.

If you want to develop scripts, we recommend that you use the Script Builder which we describe in *Section 12*. While Script objects provide you with many features that you can use to document scripts, the Script Builder is the primary method of developing script code.

In this section we will describe how to:

- Specify a script object's Properties
- Associate script objects with other objects
- Enter the Tcl code in a script object
- Set run parameters for a script object
- Run a script in the BEE from a script object
- Use Runtime objects.

10.1 How Script Objects Help You

Script objects help you to keep track of scripts that implement your tests and test plans. It is one more MYNAH feature that helps you track the complex process of documenting and implementing your test plan.

You can document the scripts that contain the code that perform tests and associate them to Test and Keyword objects. This too will help you with your test plan.

10.2 Script Object Example

We will use a script called **crAdd.tcl** which will add names to the a demo database. This Script object already exists in the demo, so we copied it into our folder using the **Database Browser** tool. Figure 10-1 shows the **crolo demo** folder with the Script object highlighted.
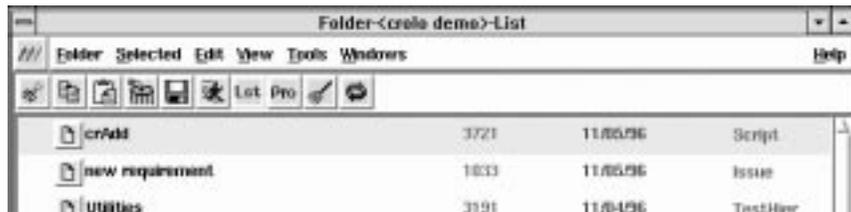
Figure 10-1. Example Script in Demo Folder

If you double click on the icon for the crAdd Script object, the system displays the Script
Object Properties view shown in Figure 10-2. The Script object is unlocked. The status line
shows that it is in **EDIT** mode.

Figure 10-2. Example Script Object Properties View.

10.3   Script Object Properties

The Script object properties are:

Owner	Refers to the person who is responsible for the script. By default this is the creator. In this case the owner's UNIX login is rachel. Your UNIX loginwould appear here if you created the script object.
Logical Name	Refers to the name you give to a Script object.
File Name	Refers to the name of the UNIX file that contains the script code. You must enter the complete path to the file.
?	Causes the system to display a File Choose dialog. The File Chooser dialog allows you to choose another file for this script object. We explain how to use the File Chooser dialog in *Section 12*.
Description	Is a free from data entry area in which you can type in any description you wish or review descriptions made by others.
Script Language	Identifies the language the script is written in. Your choices are:

- **MYNAH Tcl** refers to the MYNAH extension to the Tcl scripting language

- **4Test** refers to the QA Partner scripting language

- **TSL** refers to the X Runners scripting language

- **Replay Tcl** refers to the QC Replay scripting language

- **Other** refers to any other scripting language you may use locally

10.3.1   Entering Properties

Here we will explain how to specify a Script object's properties using entries for our example. The properties are already entered as you can see, but we will go through them as an example.

To specify the basic properties for a script object, unlock the Properties View:

1. Type in a **Logical Name** for the object in the Logical Name data entry field (e.g., **crAdd**).

2. Type in a **File Name** for the object in the File Name data entry field (e.g,, `/mynah/SunOS5.4/5.0/mut/demo/scripts/cdAdd.tcl`).

3. Type in a description of the object in the **Description** data entry field (for our example we noted what the script did).

4. Select the **Script Language** in which the script will be written from the Language drop-down list. (Default is **MYNAH Tcl**).

5. We will go on to describe the next view, if you want to **Save** the Script object you can do so at this time.

   — When we finished entering these properties, the Script object Properties view looked like the one in Figure 10-2.

10.4   Script Object Associations

You can associate Script objects with Keyword and Test objects. The association between Keyword object and Script object is done here through the Script object. Associations between Test object and Script object is done through the Test object, but you can view the associations through the Script object.

You make associations with the Association view which you access from the **Views** menu or by clicking on the **Asn** icon. After you do this, the system will display the Associations view shown to Figure 10-3.
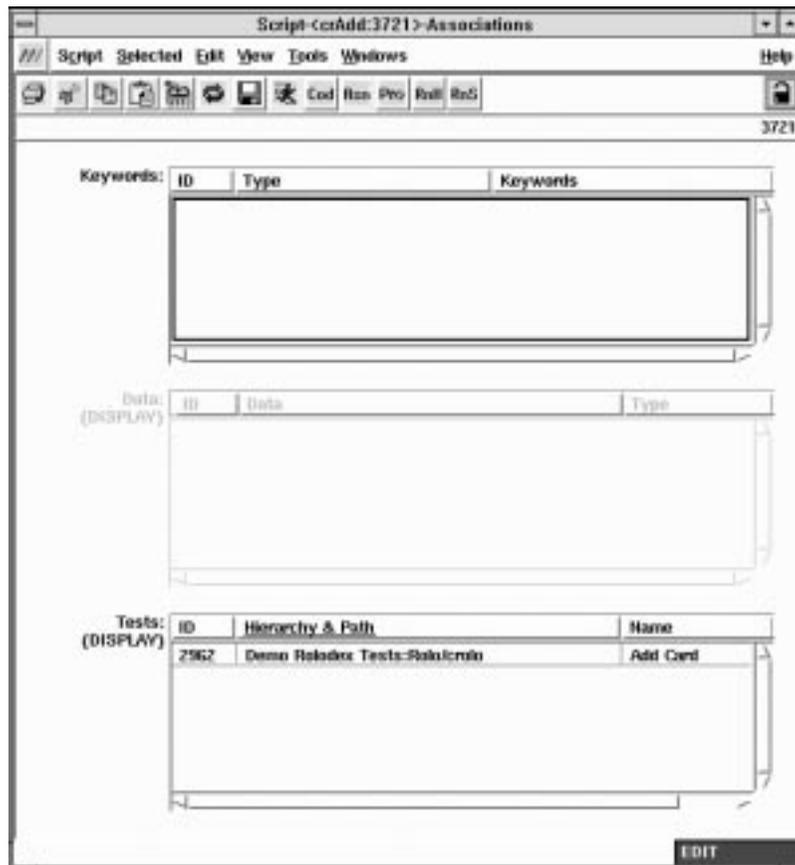
Figure 10-3. Script Object Association View.

There are three scrolling lists in this view.

Keyword   Lists Keyword objects associated with the current Script object.

Data   Data Objects are *not* available at this time, but will be in a future release of
          the MYNAH System.

Test   Lists Test object**s** associated with this Script object. (Notice in Figure 10-3
          there is one Test, named **Add Card,** associated with this Script object.)

10.4.1   Associating Keywords with a Script Object

The Keyword object scrolling lists is a selectable scrolling lists. This means that you can
associate Keywords with the Script object using this list area. Doing this is a two step
process: first you locate the object you want to associate with the Script object. (You can
use the **Database Browser** for this.) You can then copy the object from the **Database**

Browser or a Ruler display area into the scrolling list on the Script Associations view. We explained how to do this in *Section 5*. Refer to *Section 5* for more details on doing this.

10.4.2   Viewing Tests Associated with a Script

You can view Test Objects associated with the currently opened Script object with the Association View. The associations between Test and Script objects are done through the Test object. (Refer to *Section 8* for information on doing this*).*

The **Test** scrolling list is a display only scrolling list which shows all the Test objects associated with the Script.

You can open any Test objects associated with the Script object by double clicking on it's listing in the Associations view.

10.5   Specifying Run Settings

The Run Settings view allows you to specify the resources used for background execution of scripts. You can indicate the length of time a script is expected to run, and specify the Script Dispatcher, Script Engine, the percentage of system resources it will use. All the settings here affect the BEE, not the interactive environment of the GUI.

You access the Run Settings view from the View menu or by clicking on the **RnS** icon. When you do this the system will display a view similar to the one shown in Figure 10-4.

Figure 10-4. Script Object Run Setting View.

The Run Settings are:

Expected Duration	Indicates how long you think the script might run.
Script Dispatcher	Identifies the SD used to run the script.
Script Engine	Identifies the SE used to run the script. The system uses information from the SE when determining test results. See *Section 14* for information about this.
Resources	Resources are not available in future releases of the MYNAH System.

**Post Execution
Activity Required**   Indicates that a person will have to look a the test results to determine the outcome of the test, i.e., whether it passed or failed. What you choose to do here has implications for Results object. Refer to *Section 14* for information about this.

10.5.1   Specifying Run Settings Example

Our example script is a relatively short script and should not run very long. We expect no more than one minute and 40 seconds. We will use the default Script Dispatcher and the default Script Engine Group.

A script that completes successfully will require that someone examine the results. We checked **Post Execution Activity Required.**

To specify run settings:

1. To set the **Run Duration**, perform one of the following:

   • Click on the hours (HH), minutes (MM), and seconds (SS) number spin wheels to set the Expected Duration.

   • Position the pointer in the (HH), minutes (MM), and seconds (SS) number wheel data entry area and type in the values you want.

2. Select an SD from the **Script Dispatcher** drop-down list (leave the default for our example).

3. Select an SeGp from the **Script Engine** drop-down list. (leave the default for our example).

4. If the script you are running requires post execution activity, click on the **Post Execution Activity Required** radio button.

5. You can **Save** the Script object.

10.6 Editing Code Through a Script Object

A script object points to a file that contains Tcl code (or other types of code compatible with the MYNAH System). You can edit Tcl code in these files through a script object using the Code view.

You can also enter code, but we suggest that you use the Script Builder to enter code. The Script Builder provides you with tools that makes developing code easier. For example, with the Script Builder you can open a synchronous or asynchronous connection and run all or some of the code on a SUT across the connection. This will make debugging and code testing much easier.

Also the Script Builder runs code interactively. You will be able to see just how the code executes. When you run code through a Script object it runs in the background, so you won't be able to see exactly how the code executes. You will only know if it completes or fails to complete execution. You can look at output files produced by a script run to see how it executed.

You access a Script objects Code view from the **View** menu or by clicking the **Cod** icon. Figure 10-5 shows a Code view.

There are three ways you can enter or edit code into a script object:

- Typing code directly into the Code view client are.
- Accessing an editor from the Code view and entering code using this editor.
- Importing a Template.

You will also be able to enter comments in the Parameter Comment data entry area located near the bottom of the Code view. This a free form text field in which you can describe any input parameters the script may require. The MYNAH System doesn't do anything with this field. It is for documentation purposes only.

Figure 10-5. Script Object Code View

10.6.1   Entering and Editing Code DIrectly in the Code View

To enter code directly into the code view:

1. Unlock the code view object.

1. Position the pointer in the first available empty line in the Code view.

2. Type in Tcl statements.

3. When you are finished typing in code, click on the **Script** menu and then on the **Save** option.

   — The system will save the code to the file you specified with the Properties View.

To edit existing code:

1. Position the pointer on the line where you want to add or change code.

2. To delete code, highlight it by dragging the pointer, and press the **Delete** key, or click on the **Selected** menu and then on the **Delete** option.

3. When you are through editing, then you can **Save** the Code.

   — The system will save the code to the file you specified with the Properties View.

10.6.2   Using an External Editor to Enter Code

You can use your own editor of choice to enter code into a Script object. The editor you use must be specified in your UNIX *.profile*, in the environment variable EDITOR, e.g., EDITOR=vi. If you didn't specify an editor, the MYNAH System will default to the VI editor.

To use an external editor to enter Tcl code:

1. Click on the **Edit** menu, and then on the **External Editor** menu selection

   — The system will display a window for the external editor. We show a VI screen in Figure 10-6.

Figure 10-6. VI External Editor Window

2. Type in code and save it under your own editor.

3. Exit your editor.

   — The code you enter using your own editor will appear in the Code view.

4. Click on the **Script** menu and then on the **Save** option.

   — The system will save the code to the file you specified with the Properties View.

10.6.3   Editing Scripts with the Insert Template Dialog

The MYNAH System provides you with an easy way to create or make additions to scripts with the Insert Template dialog. You received a template for each of the MYNAH Tcl extension statements and a template for each of the delivered MYNAH Procedures. Usually you use Insert Template to create a script in the Script Builder, but if you are editing a script you may want to add or delete code.

To access the Insert Template dialog from the Code view, click on the **Edit** menu and the **Insert Template** menu option. The system will display the dialog shown in Figure 10-7.

Figure 10-7. Insert Template Dialog

The scrolling list that appears below the **Type** and **Package** menus displays all the commands or procedure for the type and package.

When you select a command or procedure, the system populates the **Template** and **Example** display areas. The **Template** display area shows the template itself while the **Example** display area shows an example of the command or procedure's usage.

10.6.3.1   Selecting a Template

You begin inserting a template by selecting a template **Type** and then, optionally, selecting
a **Package**.

10.6.3.1.1   *Selecting a Template Type*

There are two types of templates available with the system:

Commands   are MYNAH Tcl commands.

Procedures   are statements written in MYNAH Tcl that perform often-used functions.

10.6.3.1.2   *Selecting a Template Package*

Once you select a **Type**, you can choose a **Package**. Packages are subsets of templates. The
default for **Packages** is **all** which means the system displays all templates of that type.
Table 10-1 shows the package selections for each **Type** of template.

Table 10-1. Template Type and Package Selections

Type	Package	Package Description
Commands		
	3270	3270 terminal emulation commands
	Async	Async terminal emulation commands
	DCE	DCE commands
	ChildScript	Parent/child script commands
	Prt3270	3270 printer emulation commands
	General	General MYNAH extensions
	Tcl	Tcl commands
	TclIX	Tcl TclX extension commands
	All	All available commands
	TOP	App-to-app commands used with TOP
Procedures		
	Batch	Batch procedures
	General	General purpose procedures
	All	All available procedures
	CONV-3270	Conversion procedures to convert scripts developed with earlier versions of the MYNAH System for use with the MYNAH System.

Table 10-1. Template Type and Package Selections

Type	Package	Package Description
	CONV-Async	Conversion procedures to convert scripts developed with earlier versions of the MYNAH System for use with the MYNAH System.
	CONV-Lma	Conversion procedures to convert scripts developed with earlier versions of the MYNAH System for use with the MYNAH System.
	CONV-Tsf	Conversion procedures to convert scripts developed with earlier versions of the MYNAH System for use with the MYNAH System.

We provide examples in Section 12.7.2.

10.7   Running a Script Object

You can run code directly from the script object. You have two choices when it comes to running scripts: you can run a script in the background through the BEE; or you can run a script interactively with the Script Builder.

Typically, when you are developing scripts for the first time, you will use the interactive capability of the Script Builder.

We will describe running a script in the background using the BEE. You do this with the Run Script dialog. You access the Run Script dialog from the **Script** menu or with the **Run** icon. Once you select **Run**, the system displays the Run Script dialog shown in Figure 10-8.
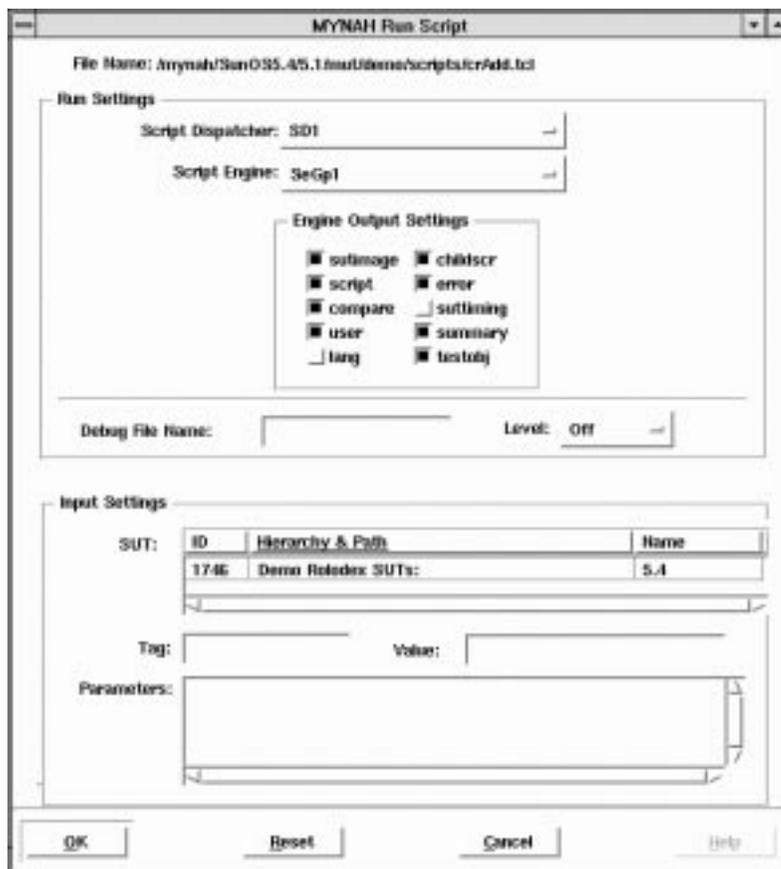
Figure 10-8. Run Script Dialog

10.7.1   Specifying Run Settings

The top panel of the Run Script dialog displays information about the run settings for the script and allows you to name a Debug file and set the debug level.

Script Dispatcher	The Script Dispatcher you specified on the Run Setting view. Your System Administrator sets the default for this.
Script Engine	The Script Engine or Script Engine Group you specified on the Run Settings view. Your System Administrator sets the default for this.
Debug File Name	The name of the file that will contain debug messages. Typically, this is used by Administrators for identifying system problems.
Level	How detailed the debug message will be. Values are **High, Low**, or **Off**.

10.7.1.1   Specifying Output Settings

Output Settings refer to the various types of output the Script Engine will produce for the script you are executing. Your System Administrator sets this in the MYNAH configuration file. These settings are pre-populated with the settings for the SE Group., but you can change them here. You can also select out put types. You can deselect one, a few, or all the output types.

By default, output files are placed in a sub-directory below the level of the script and given the suffix *.out.* and a time stamp, e.g., */home/scripts/script1.out.<timestamp>* .

The output recording selections are:

sutimage	Screen image in the sutimage file.
script	Script events in an output file.
compare	The results of compares in a compares file.
user	User defined events in an output file.
lang	Language events in an output file.
childscr	Child script events in an output file.
error	Error events in an output file.
suttiming	SUT timing events in an output file.
summary	Summary events in an output file.

See the *MYNAH System Scripting Guide*, for more information about these output files.

> **NOTE —** We recommend that you do not choose **lang**
> when running scripts in the BEE. Choosing this will
> produce a very large volume of output.

10.7.2   Specifying Input Settings

Input settings refer to the SUT and any parameters you have set for the script. The input settings for SUT fields are pre-populated with the values you specified with the MYNAH Preferences view. This item appears in a ruler display area from which you can copy it. If you want to change the SUT do so with the MYNAH Preferences view. See *Section 3* for information on how to do this.

SUT settings are very important. They are used by the MYNAH System to keep records about which tests have been run in a test cycle.

The following information appears below the SUT identification information:

Tag   Refers to the parameters tag

Value   Refers to the value assigned to the parameter.

Parameters   Displays the parameters.

The script must be written to make use of these values.

If the SUT displayed in the Run Script Dialog is not the one you want, you must:

1. Cancel the Run Script dialog.

2. Return to the MYNAH Preferences view.

3. Set the SUT to the one you want.

4. Start the script run again.

10.7.3   How Scripts Run From a Script Object

When you run a script from a Script object, as we are doing, the script will run in the background. This means that you will not see much information about how the script executes. After a script has run to completion or aborted, you can look at any output files created by the script.

The MYNAH System will run the script and perform its actions. It will also:

- Create a Runtime object to represent the script run

- Pop-up the Job Status window (See *Section 10.7.5* for more information about the Job Status window.)

- Place any results in a Result object

10.7.4   Entering Information in the Run Script Dialog

We will use our example **crADD** script to illustrate how you run a script from a Script object.

If the Script object is in EDIT mode, "unlocked", you have to save it before you run it, i.e., Lock it again. To run a script:

- You can simply accept all the default settings on the Run Script dialog

- And then click the **Run** push button to execute a script

We could do this for our example script. However, we will go through all the settings in the procedure below as an example.

1. Click on the **Script** menu and then on the **Run** menu selection.

   — This will cause the system to display the **Run Script** dialog shown in Figure 10-8.

2. Select an SD from the **Script Dispatcher** option list. (We used the default for our example script).

3. Select a SE from the **Script Engine Group** option list. (We used the default for our example script).

4. Select output files by clicking on the radio button(s) that corresponds to the type of output file you do want/don't want , e.g., **sutimage**. (We accepted the default).

5. Position the cursor in the **Debug File Name** and type in a name. (We didn't use this).

   — The system populates the **SUT** field with the values for the default SUT specified on the MYNAH Preferences view.

6. To change the default SUT, you must return to the MYNAH Preferences view and re-set the default.

7. Position the pointer in the **Tag** data entry area and type in a Parameter Tag. (We didn't use this for our example).

8. Position the pointer in the **Value** data entry area and type in a value for the Parameter Tag. (We didn't use this for our example).

9. Click the **Run** icon .

— The system will run the script. It will also create a Runtime object for the run and pop-up the Job Status window. See *Section 10.7.5*.

10.7.4.1 Running Multiple Scripts

You can run more than one script at time with the MYNAH System using the **Database Browser**. We will use the example of running several scripts created for our **crolo demo** application.

Figure 10-9 shows the **Database Browser** with the scripts for our application displayed. (We used a query to filter all available Script object so that only the ones for our demo appeared in the **Database Browser**). We also selected them by dragging the pointer across them.

Figure 10-9. Script Objects For the Demo Application

To run these Script objects, perform one of the following:

• Click on the **Browser** menu and on the **Run** menu option

• Click on the **Runner** icon

The system will display the Run Script dialog. We show the top portion of the Run Script dialog in Figure 10-10, so you can see that the **File Name** when you run multiple scripts is **Various**.

Figure 10-10. File Name When Running Multiple Scripts

The choices you make on the Run Script dialog are for all the Script objects you have selected to run. You make choices for a multiple script run in the same way as you do for a single run. Refer to *Section 10.5* if you need a refresher on what the choices are.

The system will create a Runtime object for each Script object you run and will pop-up the Job Status window as it did for the single Script object we ran earlier.

Remember, you can also run individual Script objects from the **Database Browser**. We covered this in *Section 5*.

10.7.4.2   Running Scripts from a Folder

You can run Script objects from a folder. To do this:

1. Select the Script object you want to run.

2. Click on the **Runner** Icon.

3. After the system displays the Run Script dialog, set any selections you want and click **OK**.

   — The system will run the Script object, create a Runtime object for the run and pop-up the **Job Status** window.

10.7.5   Using the Job Status Window

As we mentioned earlier, the MYNAH System provides you with a way to see whether or not a script executed. The **Job Status** window keeps a record of every script executed in the BEE and is dynamically updated as scripts execute.

You access the **Job Status** window from the **Tools** menu by clicking the **Job Status** menu selection. When you do this the system displays a window like the one in Figure 10-11. When you run a script, the system will automatically display this window..

Figure 10-11. Job Status Window

> **NOTE —** You can control how many objects appear in the **Job Status** window using the Include dialog. The Job status container displays all requested jobs plus any new jobs submitted by logids you're interested in. See Section 5.2.2 for information on using the Include Dialog.

The illustration shows the status for a number of scripts. The system provides the following information for each job executed.

Job Name	The name of the script you submitted and the ID assigned by the system.
RunTime ID	The ID for the runtime object created by the system for this script run.
Status	The current execution status of the script. The status can be: **submitted**, **queued**, **executing, or complete**.

Submitted	The date and time the script was submitted for execution.
Submitted By	The login ID of the person who submitted the script for execution.
Dispatcher	The SD that handled the script execution.
Engine	The SE group that handled the script execution.
SUT Name	The name of the SUT against which you ran the script.
Run Summary	The result of the last Tcl statement executed in the script. For example if the last statement was "xmyExit success" the **Run Summary** would be "success".

10.7.5.1   Operating Scripts through the Job Status Window

You can pause, cancel, or resume the execution of a script through the **Job Status** window. You do this from the **Edit** menu by selecting **Set Status**. When you do this, the MYNAH System will display a sub-menu with the following selections.

Pause   Causes the selected script or scripts to suspend execution.

Resume   Causes the selected script or scripts to re-start execution at the point the script(s) was paused.

Cancel   Stops execution of the selected script or scripts.

To use these selections, select the script or scripts you want, and then select **Edit**, **Set Status** and then **Pause, Resume,** or **Cancel**.

10.7.5.2   Forcing Updates to the Job Status Window

When dynamic updates appear to be taking a long time, you can force all information in the Job Status Window to be updated by clicking on the **Refresh** button in the **Toolbar**. If dynamic updates continue to be slow, notify your administrator.

10.8 Viewing the Run History for a Script Object

The MYNAH System keeps a record of the number of times a script was run. This record appears in the Run History view which you can access from the **View** menu or by clicking on the **RnH** icon. When you access it, the MYNAH System will display a view similar to the one shown in Figure 10-12.

Figure 10-12. Run History View

The Run History view displays:

Run ID   The number assigned by the system at the time of the run.

Status   The current status of the run. The status can be **submitted**, **aborted**, **queued**, **executing,** or **completed**.

Submitted   Time the script was submitted to the Script Engine for processing.

Ended   When the script execution ended.

10.9   Script Runtime object

When you execute a script, the MYNAH System automatically creates a Runtime object. A Runtime object stores information about the execution of a script. It is created after you initiate script execution, and may be available before a script has ended execution.

Each entry in the Run History view represents a Runtime object. You can open a Runtime object by either selecting it and then clicking on the **Script** menu, then the **Open** option, or double clicking on a row entry.

If you run the a script while it is open, you will have to click on the **View** menu and then on the **Refresh** menu selection to see it in the Run History view.

We selected the Runtime object for **crAdd** script and the system displayed the window shown in Figure 10-13.

Figure 10-13. Runtime Properties View

10.9.1 Viewing the Properties of a Runtime Object

The Runtime Properties view displays basic information about the script run. Most of the information displayed in this view comes from information you entered on the Run Script dialog. This information includes:

Script   Identifies the Script object the Runtime object is for.

Status   Lists who ran the script (**Created By**), when it was run (**Executed**), and when it stopped executing (**Ended**).

Settings   Lists the SUT, SD, SE, and parameters specified on the Run Script dialog.

10.9.2   Runtime Results View

You can change the view of the Runtime object from Properties to Results using the **View** menu or by clicking on the **Res** icon. This will display the Results view for the Runtime object. The Results view for our example script appears in Figure 10-14.
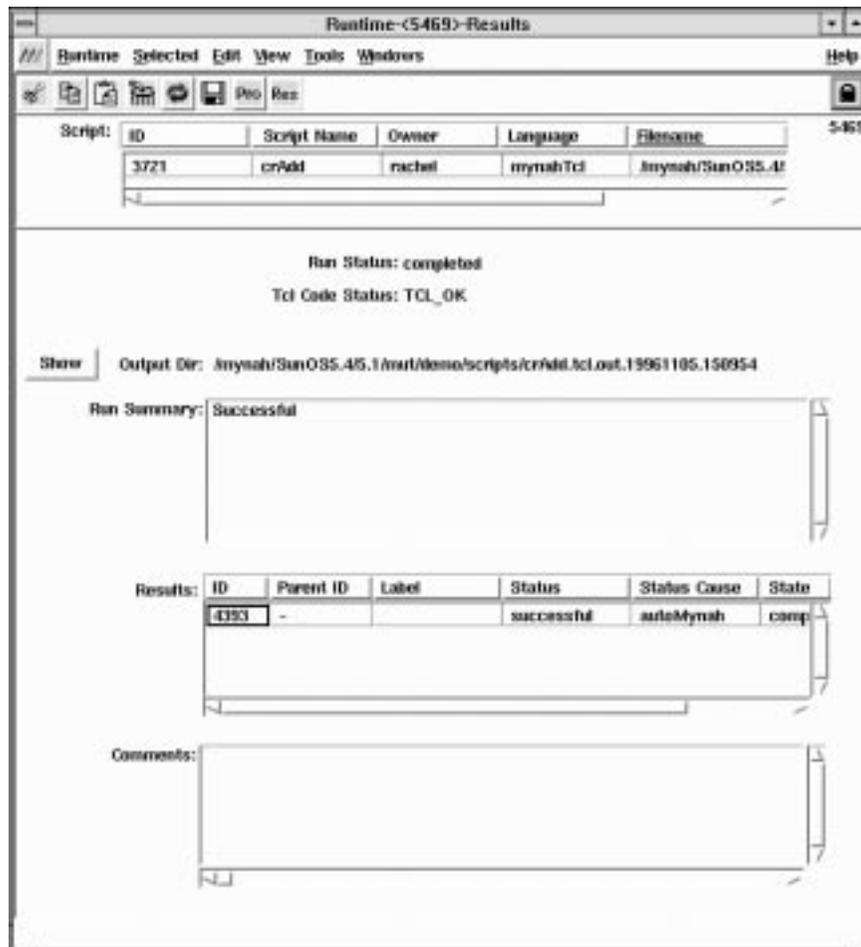
Figure 10-14. Runtime Object Results View

You should not confuse the Results view with the Results object. The Script object Results view contains information about the immediate results of a script's execution. This includes:

Run Status   Displays the current status of the script. The status for an active script can be **submitted**, **queued**, **active,** or **paused**. The status for an ended script can be **cancelled**, **completed,** or **aborted**.

Tcl Code Status	Tcl interpreter status at the end of the run. It can be either **TCL OK** or **TCL-ERROR**.
Show	We will explain the operation of the **Show** icon in *Section 10.9.2.1*.
Output Directory	Displays the directory which contains the output files for the script.
Run Summary	Displays the result of a script based on Tcl statements in the script. For example, you can code a script to set up certain conditions that trigger an **xmyExit success** statement. In this case the **Run Summary** would be displayed **success**.
Results	Lists Results objects created by the script's execution.
Comments	A free form data entry area where you can enter comments.

10.9.2.1   Viewing Output Files

If you specified output files for the Script object with the Run Script dialog, you can see the output using the **Show** icon which appears on the Runtime Results Properties view. If you click on this pushbutton, the system will display the view similar to the one shown in Figure 10-15. This view displays output information for the current object.

The MYNAH Output file is presented with the information from all specified files merged in at appropriate places. For example, a compare block from the Compares file will appear below its compare event in the Output file. Similarly, a SUT screen image will appear in the Output file immediately following the SUT images event for that screen.

> **NOTE —** For more information about output files, refer to the *MYNAH System Scripting Guide*.

Figure 10-15. Script Output View

You can see what output files were selected and the name of the SUT image file created for this script (if any) by performing one of the following:

- Clicking on the **View** menu, selecting **Change View** and then selecting **Properties**

- Clicking on the **Pro** icon

When you do this the system will display the Properties view similar to the one shown in Figure 10-16.
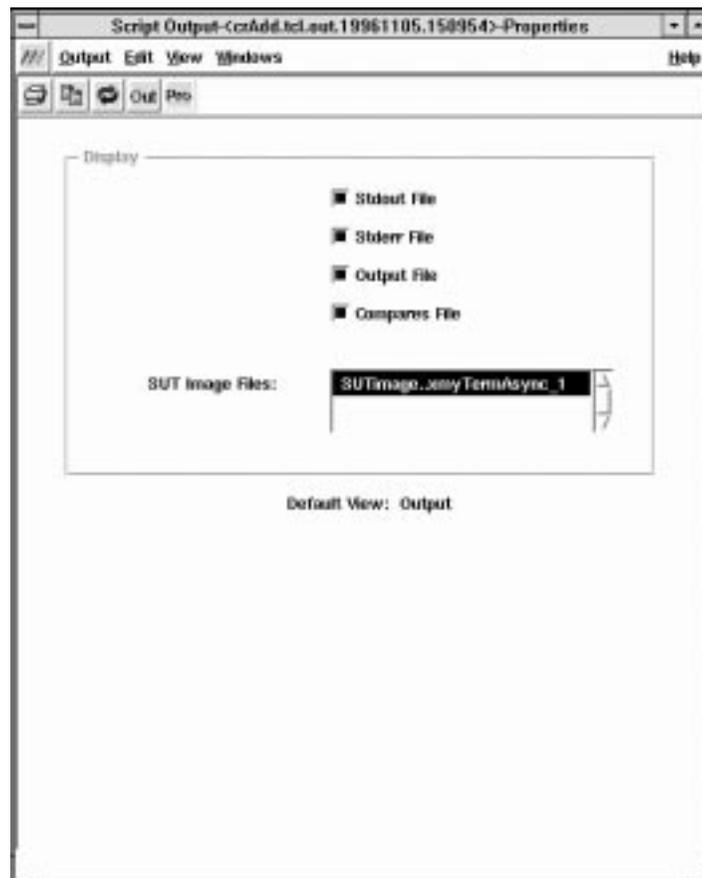
Figure 10-16. Script Output Properties View

The information here includes:

Display   Lists the output files selected on the Run dialog. (**Stdout File**, **Stderr File**, **Output File**, and **Compares File)** and the name of any **SUT Image Files.**

10.9.3   Opening Result Objects from the Results View

Scripts may also produce Result objects. Result objects contain information about test results based on the action a script took on the SUT. The system will produce results for a script under two conditions:

• You associated the script with one and only one test object

• The script uses **xmyBegin** or **xmyEnd** commands.

We cover Results Objects in *Section 14*. Refer to *Section 14* for more about examining results.

You can open a Results object from the Results view by:

- Selecting a Results object in the **Results** scrolling list, clicking on the **Edit** menu and then on the **Open** option

- Double clicking on a Results object listing in the **Results** scrolling list.

Refer to *Section 14* for information about using Results objects.

10.10 Creating Script Objects From The Command Line

You can automatically create Script objects to represent any existing script files from the UNIX command line using the **xmyCreateScriptObject** command. We explain this command in *Section 17.2.13*.

11. Using Keyword Objects

Keyword objects provide you with a way to reference a number of different objects needed for a specific task.

In this section we will explain how to:

- Create Keyword objects

- Associate them with tests and other documents.

11.1 How Keyword Objects Help You

Keyword objects provide you with a convenient way to reference a number of objects used for a particular purpose. Suppose you wanted to identify all the scripts that used a particular system function, for example the **cat** command. You could create a Keyword object that represented this command and associate it with all Script objects that used **cat**. You would then be able to identify all the scripts that used this command whenever you needed to by referring to a Keyword object.

You can associate keywords with:

- Requirements

- Scripts

- Tests

- Users (Person objects).

11.1.1 Creating Keyword Objects

You create keywords objects in the same way you would any other object. Refer to *Section 4: Using MYNAH Objects* if you need to refresh your memory on creating objects.

The system labels newly created objects "untitled-keywords" and places them below the last object you selected or at the top of the list if no objects were selected.

For our example, we clicked on the **Edit** menu and then on the **Deselect All** menu option to clear all selections and then created the Keyword object. The system created the object and placed it at the top of the list view in our **crolo demo** folder. See Figure 11-1.
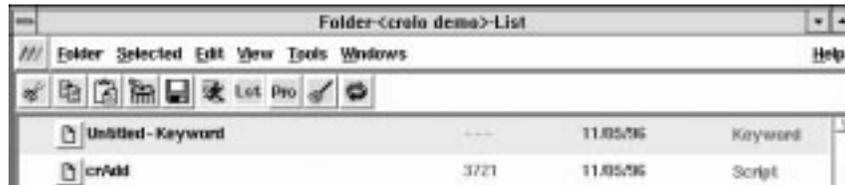
Figure 11-1. List View with New Keyword Object.

11.1.2 Using Keyword Object Properties View

If you double click on a Keyword object icon, the system will open the object and then display a Properties view similar to the one shown in Figure 11-2.

The Properties view allows you to specify the basic properties of a keyword object. These include:

Name   A unique name of your own choosing.

Type   Refers to the type of Keyword the currently opened object represents. We provide the default value **General** but your system administrator can add others.

Description A free form data entry.

Figure 11-2. Keywords Object Properties View

11.1.3   Entering Keyword Properties

We will create a Keyword object called **default_db** and use it to reference all Test objects
that use the default Rolo database delivered with the MYNAH System.

To set the properties for a keyword object:

1. Position the pointer in the **Name** data entry field and type in a name for the object.
   (**default_db** for our example).

2. Select a type for the object from the **Type** option list. (Use the default value **General**.)

3. Position the pointer in the first available line of the **Description** data entry area and
   type in a description for the keyword. (We typed in a description telling what the
   keyword was for.)

4. Click on the **Keyword** menu and the **Save** option to save the properties for this object.

   — After we finished entering the information the Properties view appeared like Figure 11-3.

Figure 11-3. Completed Keywords Object Properties View

11.2   Using the Keyword Object Associations View

Keywords objects can be associated with Requirement Script, Test, and Person (User) objects. You can display objects related to the currently opened keyword but the associations themselves are made through the other objects.

You display objects associated with a Keyword using the Association view which you can access from the **View** menu or by clicking on the **Asn** icon. After you select the Association view, the system displays a window similar to the one shown in Figure 11-4.
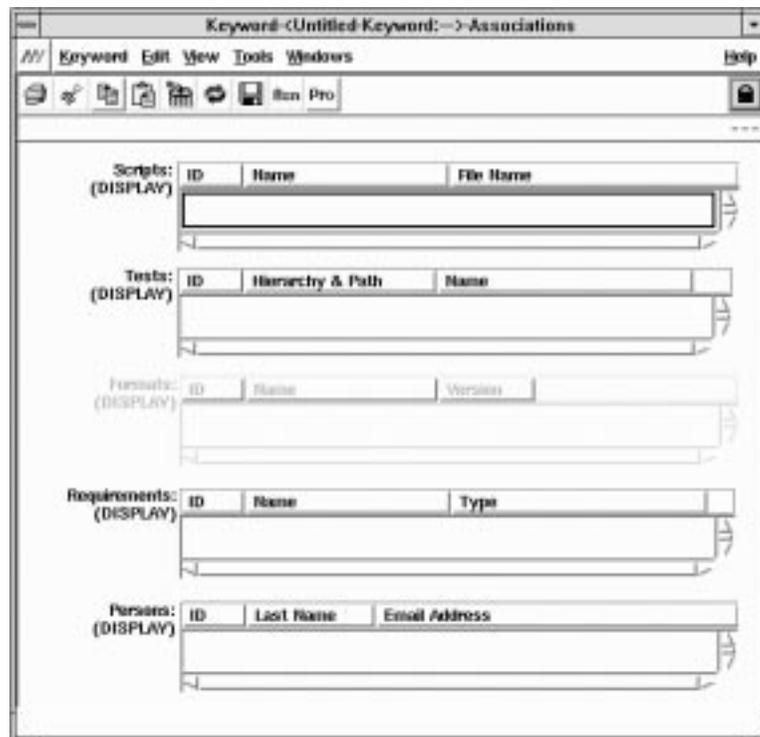
Figure 11-4. Associations View

The scrolling lists on this view are:

Script   Lists the Script objects associated with the currently opened Keyword.

Test   Lists the Test objects associated with the currently opened Keyword.

Formats   Lists the Format objects associated with the currently opened Keyword.

                NOTE — The Formats parameter is part of the AETG System, and is only available if you license the AETG System with the MYNAH System. For more information, see the *AETG System Users Guide.*

Requirement   Lists the Requirement objects associated with the currently open Keyword.

Person   Lists the Person objects associated with the currently open Keyword.

You operate these scrolling list the way you would operate any other scrolling list. Refer to *Section 3: MYNAH Basics* if you need instructions on using these scrolling lists.

11.3   Referencing Test Objects with Our Example Keyword

As we mentioned earlier, we want to use the Keyword object we created to reference tests that use the default names database. This will come in handy if we decide to change records in the default database. With this Keyword object, we can find any tests that may be affected by changes to the default database.

We made the association between the **Common** Keyword and Script objects, through the Test objects' Association views. After we made the associations, we returned to our Keyword object and opened it to the Association view. (See Figure 11-5.)

As you can see, we now had an easy way to find tests affected by any changes to the default database. The Scripts display area is populated with all the Scripts associated with this Keyword. If we wanted to make changes to any of these Script objects, we could open them from this view.
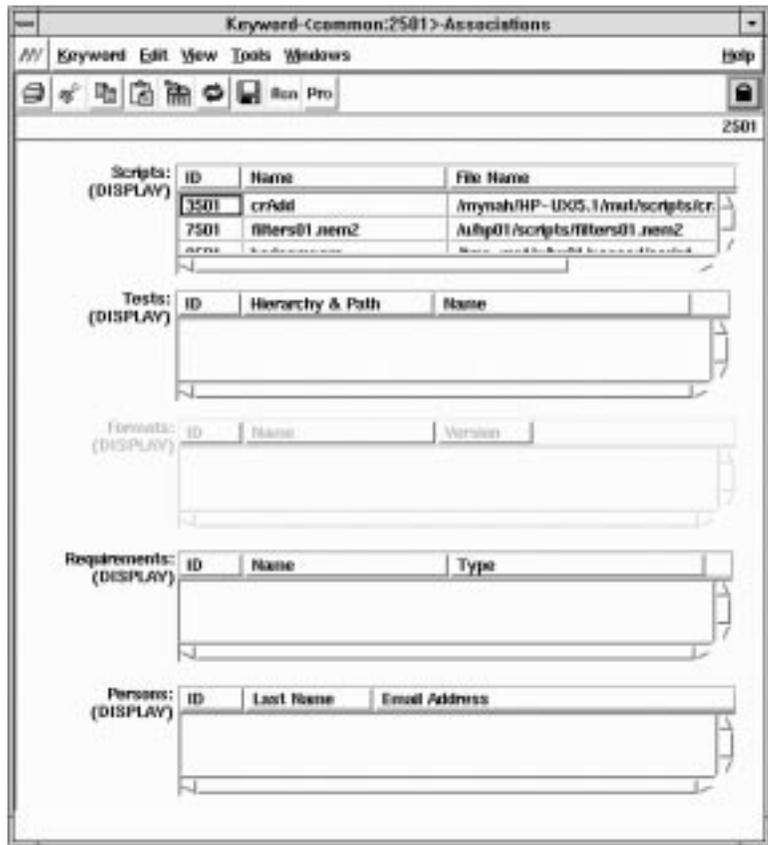
Figure 11-5. Example Associations View

12. Developing Script Code with the Script Builder Feature

The MYNAH System provides you with an easy way to develop scripts with the Script Builder feature. The Script Builder is one of the tools available from the **Tools** menu.

In this section we will explain how to:

- Use Script Builder views

- Enter Tcl code into the Script Builder Code view

- Connect to SUTs via the Script Builder using 3270 and asynchronous connections and record keystrokes and SUT events

- Use other domain packages with the Code view

- Execute code and observe its execution.

The Script Builder is a large feature with many menus and menu options. Refer to Appendix A for a listing of all the menus associated with the Script Builder.

Remember, you can often select menus and menu options using accelerator and mnemonic keys. *Section 3* lists accelerator keys and Appendix A lists the mnemonic keys.

12.1 How The Script Builder Helps You Develop Scripts

The Script Builder makes it easy to develop code. It is comprised of a number of windows and dialogs which allow you to develop code, edit code, run code interactively and then save the code to a script.

The Script Builder also provides a way to select Tcl statements, insert them in your script and edit templates and examples to meet your needs.

You can also record events as they occur on a remote SUT using a 3270 or asynchronous connection and add these to the code you are developing. For example, defining a compare is as easy as dragging the pointer and clicking on an icon.

You will also be able to execute code you develop interactively directly from the Script Builder and observe how it executes.

All in all, the Script Builder is a complete workbench for creating, editing and executing code that can be used in scripts.

12.2 Accessing the Script Builder

You open the Script Builder by clicking on the **Tools** menu and then on the **Script Builder** menu options. By default, the Code view will be displayed (See Figure 12-7).

All views have three main areas:

The Menu and Tool Bars Display of Menus and Tools appropriate for the Script Builder.

The View information This area displays the text areas appropriate for the specified view (e.g., code, preferences, and log).

The Status Bar This area contains two areas. On the left side, system messages will be displayed pertaining to the activity you are performing (e.g., opening a connection, running code, etc.). On the right side, the Execution Mode will be displayed (e.g., Running, Paused, etc.).

12.3 Checking Status Messages

While you are working in a Script Builder session, you should look at the Status Bar for system messages. Messages will appear that will indicate whether the action you performed succeeded or failed. Some messages will inform you about conditions in which you should perform corrective actions.

Messages that do not fit in the Status Bar will appear in the *Run Status* area in the Code view.

For example, Figure 12-1 shows the Status Bar and *Run Status* area for a script that failed. The Status Bar displays the message **run failed (TCL ERROR)**, while the *Run Status* area displays a more descriptive message explaining that the execution timed out. In addition, the Run Status area displays the line of code where the error occured. (In this example, you would see this if you scroll to the right.)

Figure 12-1. Script Builder Status Bar and Run Status Area

12.4   Using the Script Builder Preferences View

The Preferences view (Figure 12-2) lets you set default values for a number of parameters
that you will use during a Script Builder session. These parameters include:

Run Input	How much code you want to run.
Termination Setting	When code will stop executing.
Run Settings	How a script executes while running and where output will go.
Font Setting	Defines the fonts for the Script Builder.
Default Connection Setting	Defines the default connection type, asynchronous or 3270, for the session.

Figure 12-2. Script Builder Preferences View

The following sub-sections explain what the settings are. In Section 12.8 we will explain
how to run code using the Run dialog.

12.4.1   Selecting Run Input

You can select the **Run Input** which means you can choose how much of the code to run. These are exclusive settings. However, you can only choose one.

All Code	Runs all the code currently in the Code view.
Selected Code	Runs only code you have selected. You select code in the Code view by dragging the pointer to highlight it.
All Code After the Cursor	Runs all the code that appears after the current cursor position. For example, if the cursor is at the beginning of the fifth line, the system will run the code from that point to the end of the code.
	Similarly, if the cursor is in the middle of the fifth line, the system will run all the code from that point to the end of the code.

> **CAUTION —** Be careful to position the cursor only at the beginning of valid Tcl commands.

12.4.2   Selecting When Execution will Stop

You can also select when the code will stop executing. You do this with the **Termination Setting**. These are exclusive settings which means you can only choose one. Your choices are:

End of Code	Stops executing when all the code finishes executing or when the last statement of selected code completes.
Breakpoint	Pauses execution when the code reaches an **xmyBreakpoint** statement.
Compare Failure	Pauses execution when the code encounters the first failed **compare**.

12.4.3   Specifying Run Settings

You can specify how the script will execute and where the output from the execution will go. This is done in the Run Settings area. Your choices are:

Execute Step By Step	Executes the script one statement at a time pausing at each step until you indicate that the next statement should be executed, or canceled, etc.

Display Remote Session Connections	Allows you to see the remote connections and observe actions which take place on the remote SUT as a result of code execution.
Generate Script Output	Causes the system to generate standard MYNAH script output. If you select this option, you must specify the name of the directory that will contain the script output directory, and its path in the **Directory Name** data entry field.
Directory Name	Indicates where in the UNIX file system the MYNAH System will place the output.

We will run the code we recorded in a number of different ways in Section *12.8* to show you the versatility of the Run Code dialog. We will also discuss the **Run Progress** dialog in that section.

12.4.4   Setting the Font For the Script Builder

You can set the font and its size for the Script Builder. You do this using the same Font Changer dialog you used when you set the font for the system as a whole. If you need a refresher on how to do this, refer to Section 3.7.3 where we explain how to change fonts.

> **NOTE —** We recommend that you use a fixed width font so that window text lines up properly.

12.4.5   Setting Default Terminal Emulation.

Near the bottom of the Properties view is the **Default Connection Setting** area. Here you can set the default terminal emulation preferences for the session by selecting the terminal type you want. Choices are:

3270   3270 synchronous terminal emulation (See Section 12.4.6).

Async   Asynchronous terminal emulation (See Section 12.4.7).

The following buttons appear at the bottom of the 3270 Terminal Settings (Section 12.4.6) and Async Terminal Settings (Section 12.4.7) dialogs:

OK   Clicking on this button closes the dialog. All of your changes are saved.

Cancel   Clicking in this button closes the dialog. All of your changes are discarded and the default settings that were current when you opened the dialog are restored.

Defaults Clicking on this button restores all settings on the dialog to the state they were when you opened the dialog, i.e., the default settings are restored.

12.4.6   Setting the 3270 Terminal Emulator

When you select **3270** from the **Default Terminal Setting** option list, the MYNAH System
will display the 3270 Terminal Settings dialog shown in Figure 12-3. Some of the settings
on this dialog may be pre-set by your System Administrator but you can override them here.

> **NOTE —** If you want to use these settings in your
> connection windows, you must specify the values on this
> dialog. You may then deselect these settings within
> individual connection windows. For example, if you want
> to use "default compares", you must specify the
> parameters on this dialog.
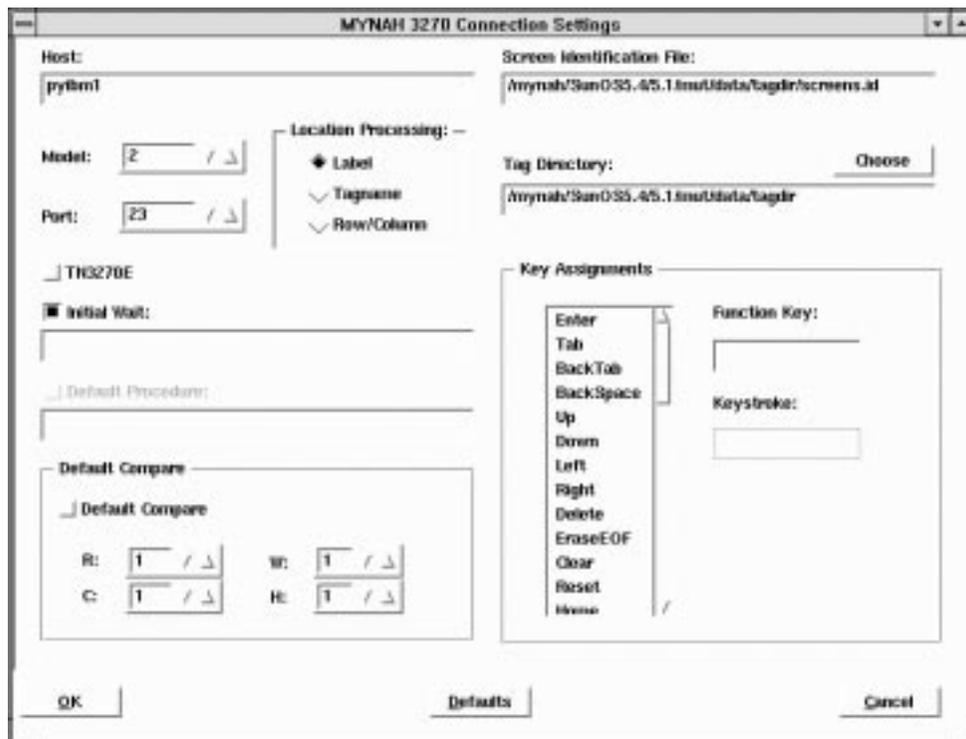
Figure 12-3. 3270 Terminal Setting Dialog

12.4.6.1   Host Settings

You can identify the default host system with this dialog. Settings include:

Host   Identifies the host system. Your System Administrator can set these values.

Model   Identifies the model for the 3270 terminal emulation. The choices are **2, 3, 4** and **5.** Your System Administrator can set these values.

Port   Identifies the connection port. This is a numeric value that represents the port numbers for your system. Check with your System Administrator to find out what ports are available.

Your System Administrator can set these values.

12.4.6.2   Location Processing

Location Processing specifies the method by which the MYNAH System will determine the location of the cursor position as you interact with your SUT and thus the format of the recorded statements. The selections are mutually exclusive. You can choose only one from this group. Your choices are:

Label   Records the **field label** for the field and includes this information in the script you are creating. When the script is executed, it will use the label recorded here to locate the field on the 3270 display.

                  WARNING — Label is any string on the screen contained in a protected field. The MYNAH Script Builder has the ability to generate labels for the various Term3270 commands, if Label processing option is selected. It is easier to let the Script Builder generate the labels. This way, the generated scripts would get executed without any errors. If there are any errors in execution of the generated code, please contact MYNAH Support. You may choose a label manually and code it in the script, but there exists a risk of the label getting rejected in a few cases**.**

Tagname   Records the **tagname** associated with the 3270 display location and includes it in the script you are creating. When the script is executed, the tag name is cross-referenced with a tag name file associated with the display to determine the location to be processed.

> **NOTE —** This method requires "tagname" tables in the default "tagdir" directory or the *tagdir* directory you specify when you establish the connection. The name of the tag name file must be the name of the format/screen the tagname table describes.

Row/ Column   Records the **row and column** location information on the 3270 display and includes this information in the script you are creating. When you execute the script, the system uses this information to determine the location to be processed. This is the simplest form of location processing.

See Section 9.3 of the *MYNAH System Scripting Guide*, for information about location processing.

12.4.6.3   TN3270E Protocol

If your host uses TN3270E protocol for 3270 Terminal Emulation, click on the **TN3270E** radio button. You can only use this if TN3270E protocol is supported by your host.

12.4.6.4   Initial Wait

Initial wait refers to the time the MYNAH System should wait before doing any initial processing.

Initial Wait   Refers to special processing the system will do before executing the first line of code. The system waits for a specified text string to appear on the SUT. After you click on the toggle button, the system will enable the data field below it. Here you enter the expected text string for initial wait. If this setting is set ON, the system will wait for the specified text string. If it is set OFF, the system will not wait. Note that you may specify a text string without setting this option to ON.

For example, if the text string "ENTER APPLICATION" appears near the bottom of the initial IMS screen, you can use this string for **Initial Wait**. You would thus ensure that no Tcl statements were executed before this string appeared.

12.4.6.5   Default Procedure

Default Procedures are procedures contained in a script which you want the system to execute for each new SUT screen appearance. We will explain more about Default Procedures in *Section 12.10.6.3*.

Here you enable default procedures for the Script Builder.

Default Procedure   Specifies the procedure name. For example, if your procedure name is "checktitle", you can simply enter "checktitle'.

12.4.6.6   Default Compares

Default compares allow scripts to perform compares between expected screen values and actual screen values. If you enable this setting, the MYNAH System will record a compare statement for the region you specify after every new application screen is received.

If you know that every screen contain text on a certain row, and it makes sense to compare for that text, enabling default compare for that row eliminates the need to record a compare manually every time you reach a new screen.

For example, if row 1, columns 1 through 10 contain the format name, and this information is useful during script execution, you could specify that region for the default compare (row 1, columns 1, width 10, and height 1.)

This feature is useful even when you do not want a literal default compare statement. For example:

- If you know that for most screens you want compare statements but for some you don't, you can let the MYNAH System generate compare statements for all screens, and delete the statements you don't want in the Code view after the system generates them.

- If you want the compare statements generated, but you know that you will want to change a few characters, let the MYNAH System generate the statements and you can edit them in the Code view.

Your choices for **Default Compares** are:

Row   Starting row coordinate for a compare, e.g., if area begins on row 1, the row coordinate is 1.

Column   Starting column coordinate for a compare, e.g., if area begins in column 1, the column coordinate is 1.

Width   Width of compare area, e.g., if area is 10 characters wide, width is 10.

Height   Height of compare area, e.g., if area spans 1 row, height is 1.

12.4.6.7   Screen Identification File

This setting lets you specify the location of an ASCII file called a *Screen Identification* file, which the GUI uses to identify each format/screen in the applications(s) that you test.

The *Screen Identification* file makes it possible to generate format/screen names automatically. By entering the location of *Screen Identification* as the parameter for the **ScreenIdentificationFile** parameter of a **Term3270** entry in the *xmyConfig* file, the specified *Screen Identification* file is automatically loaded when you establish a 3270 connection.

The *Screen Identification* file contains regular expression patterns (as defined for the **regex(3X)** program) in conjunction with screen location. As a user navigates through an application, the *Screen Identification* file determines if these two conditions exist:

- The format/screen name exists on the format/screen being displayed.

- A regular expression can be defined to uniquely describe the format/screen name.

Enter the full path for the location of the *Screen Identification* file. The field is prepopulated with the location of the *Screen Identification* file specified in the *xmyConfig* file.

12.4.6.8   Tag Directory

This setting lets you specify the *tagdir*, the directory containing **Tag Name** files, which are user-defined labels called **tags** that reference locations on a screen in place of row and column integer values.

> **NOTE —** See Section 9.3.3 of the *MYNAH System Scripting Guide* for more information using tag names to describe locations on a 3270 screen.

Specify the *tagdir* by either entering the full path for the location of the *tagdir* or by clicking on the **Chooser** button. If you click on the **Chooser** button, the *Tagdir* Selector dialog in Figure 12-4 will appear.
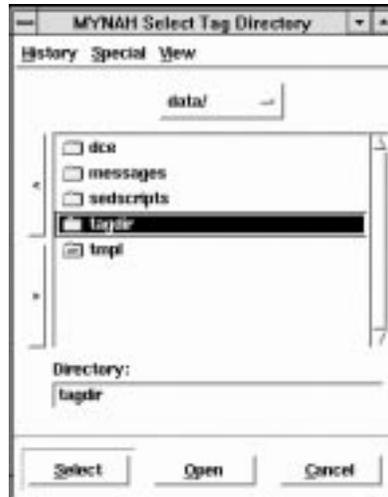
Figure 12-4. Tagdir Selector Dialog

Three menus selection appear on this dialog:

History   Shows a history of the directories you have selected in the order they were selected.

Special   The sort criteria ar by: **Home**, **Mark**, and **Unmark**. This menu selection witches to your **Home** directory automatically. **Mark** and **Unmark** marks and unmarks a file. Marking a directory causes it to appear in the lower half of the **History** list so that it is easier to access.

View   The sort criteria are by: **Name**, **Date Modified**, and **Date Created**. This sort allows you to change the order in which the system displays items in the scrolling list.

By default the *tagdir* specified in the *xmyConfig* file is entered. Only the name of the selected directory appears in the Directory field. If you want to see the complete path for

the directory, you can click on the option list at the top of the dialog, as shown in
Figure 12-5.

Figure 12-5. Tagdir Selector Directory Option List

The Tagdir Selector dialog is a ***directory*** selector. Only directories are displayed in the
dialog, unlike the other dialogs, such as the Save to File dialog (Figures 2-33 and 12-22)
and Insert Code From File dialog (Figure 12-14), that display files and directories.

In addition to the **Cancel** button, the *Tagdir* Selector Dialog contains the following buttons:

Select   Selects the highlighted directory as the *tagdir*. When you click on a directory
                name, the name appears in the Directory field.

Open   Opens the highlighted directory, displaying any subdirectories in the
                directory. If you click on **Open** in an empty directory (a directory that does
                not contain any subdirectories), the system will sound a beep.

                Double clicking on a directory name also opens the directory.

To select a directory as the *tagdir*, click on the directory and click on **Select**. The dialog
will disappear and the selected directory will be entered in the Directory field on the 3270
Terminal Settings dialog.

12.4.6.9   Key Assignments

The **Key Assignments** data display area shows you how Function keys and other keys that appear on an IBM 3270 keyboard are mapped to your workstation's keyboard. To see the assignment for a 3270 key:

- Scroll through the list of PF keys until the one you want is highlighted. (It will also appear in the **Function Key** display area.)

- Look at the **Key Stroke** data display field. The value in this field indicates the key or keys that you can press to perform the action. For example, "erase end of field" function is mapped to **Shift** + **Backspace**. If you press **Shift** and then **Backspace,** you will clear the "end of field".

> **NOTE —** Changing keyboard assignments does not change the performance of the function keys that appear on a connection window. (See Section 12.10.5.)

12.4.6.9.1   Changing Key Assignments

You can change the mapping of 3270 function keys. To do this:

1. Select a key in the **Function Key** scrolling list.

   — The system will display the key name in the **Function Key** display area.

2. Click on the **Key Stroke** data display field.

   — The box that bounds this field will darken to show it is the focus of your keyboard actions.

3. Press the key or key sequence you want mapped to the 3270 function key.

   — The new sequence will appear in the **Key Stroke** field.

12.4.6.10   Setting the 3270 Connection Values

After you make your selections for the 3270, you set them by:

- Clicking on **OK**

- If you want to change back to the default settings, click **Default.**

If you OK the new 3270 Settings, the system will use them for subsequent connections.

12.4.7 Setting the Asynchronous Terminal Emulator

When you select **Async** from the **Default Terminal Setting** option list, the system displays
the Async Terminal Setting dialog shown in Figure 12-6.
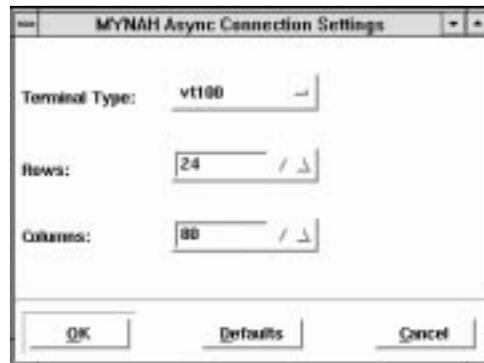
Figure 12-6. Async Terminal Setting DIalog

12.4.7.1 Selecting Async Terminal Emulation Type

This drop-down list allows you to set the type of asynchronous terminal you will use.
Currently, the MYNAH System supports VT100 Terminal emulation.

12.4.7.2 Setting the Display Size

You can set the size of the display for Asynchronous Terminal Emulation. The choices are:

Row   Sets the row count for the display. (Range from 24 to 50).

Column   Sets the column count for the display. (Range from 80 to 99).

You set these by:

1. Clicking on the up or down arrows on the number wheel until the value you want
   appears.

OR

1. You can position the pointer in the number display area and type in a number.

2. To set the Async Terminal values, click **OK**.

12.5   Saving Preferences

To save the preferences you set:

• Click on the **Builder** menu and then on the **Save Preferences** menu option.

Your Preferences settings will be saved to the *.xmyMYNAHrc* file in your HOME directory. They will be in effect until you change them again or reset them to the default settings with the **Defaults** pushbutton.

The **Default Connection Setting** you specified here will remain in effect until you change it. This means that when you open the **Open Connection** dialog, it will be set to the connection type you specified with the Preferences view. (See Section 12.10 for more information about the **Open Connection** dialog.) If you didn't change the default connection, the **Open Connection** dialog will be set to the system default of 3270.

12.6   Using the Code View

The Code View is the central location for building scripts. Here you can enter Tcl code, insert templates that contain Tcl code, add code generated through the system's interaction over a connection to a SUT, and add in key stokes and other events recorded during a remote session. You can insert existing script code files and edit these. You can think of it as an empty work table where you will assemble code that can become a script if you decide to save the code.

You access the Code view from the **View** menu or by clicking on the **Cod** icon. The system will display a Code view window like Figure 12-7.
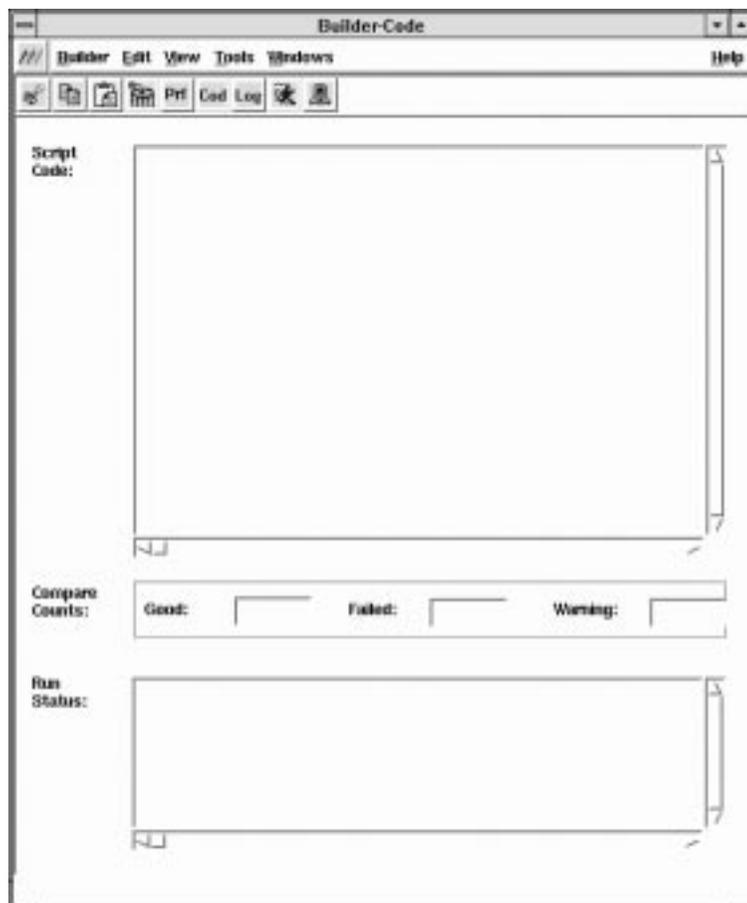
Figure 12-7. Code View

There are three screen areas on the Code view:

Script Code   This is where you will build code you can save in scripts.

Compare Counts   This displays the number of **Good** compares, **Failed** compares and **Warning** compares for code you execute that contains compares.

Run Status   This is an extended status area. When you execute scripts, the MYNAH System displays status messages here.

You should be familiar with most of the menus and menu options available on the Code view. There are some new menus you are not familiar with that we will explain as we go along. For a list of all the menu and menu options available for the Code view see Appendix A.

There are two icons added to the tool bar. (See Figure 12-8.)

- **Run** icon causes the system to display a run dialog. We will explain how to use this in Section *12.8*.

- **Connect** icon causes the system to display a dialog from which you can establish connections to a remote SUT. We will explain how to use this in Section *12.10*.
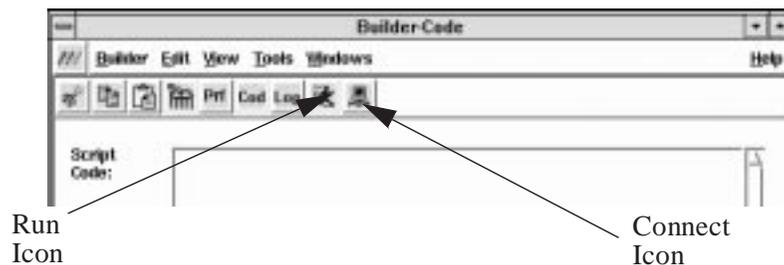
Figure 12-8. Dialog Icon

12.7   Entering Code into the Code View

When you first open the Code view it will be empty. The ways you can enter code are.

- Type code in directly (See Section •.)

- Use your own editor to enter code (See Section 12.7.1.)

- Insert a Template and modify it (See Section 12.7.2.)

- Insert Code from a File (See Section 12.7.3.)

- Record keystrokes or compare regions through a connection. (We will explain how to do this in Section *12.11.)*

You can type Tcl code directly into the Code view. You must be familiar with Tcl to do this. See the *MYNAH System Scripting Guide*, for information about using Tcl.

As an example, let's type in a Tcl statement that sets a value for **x**.

```
set x 10
```

To enter this statement:

1. Position the pointer in the **Script Code** area.

2. Type in the statement.

After we do this, the Code view looks like Figure 12-9.
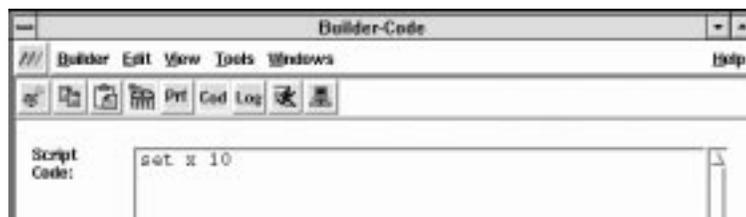
Figure 12-9. Code View with Statement Entered

12.7.1   Using Your Own Editor to Enter Code

You can use your own editor to enter code into Code View. The editor you use must be specified in your EDITOR environment variable. If there is no editor specified for your environment, the MYNAH System will start **vi**.

To use an external editor to enter code:

1. Click on the **Edit** menu, and then on the **External Editor** menu selection.

   — This will cause the system to display a window for the external editor. We show a VI screen in Figure 12-10. Note that since we opened the external editor while there was a Tcl statement in the Code View, the system automatically included this statement in the VI file it opened.

   > **NOTE —** Use of the Code View in MYNAH is inhibited while the external editor is being used.

Figure 12-10. VI External Editor Window

2. Type in code and **Save** it.

3. Exit your editor.

   — The entire contents of the editor at the time you terminated the editor, will replace any code that was in the Code view.

12.7.2 Inserting Tcl Statements and Procedures From Templates

The MYNAH System provides you with templates to make developing Tcl code easier. Templates are Tcl statements which you can copy and/or modify to meet your specific needs. We supply a complete set of templates, one for each Tcl statement, with the delivered MYNAH System. You insert templates into the Code view using the Insert Template dialog.

To access the Insert Template dialog from the Code view, click on the **Edit** menu and the
Insert Template menu option. The system will display a dialog like Figure 12-11.

Figure 12-11. Insert Template Dialog

The scrolling list that appears below the **Type** and **Package** menus display all the
commands or procedure for the type and package.

When you select a command or procedure, the system populates the **Template** and
Example display areas. The **Template** display area shows the command and all options
while the **Example** display area shows an example of the command or procedure's usage.

12.7.2.1   Selecting a Template

You begin inserting a template by selecting a template **Type** and then, optionally, selecting a **Package**.

12.7.2.1.1   Selecting a Template Type

There are two types of templates available with the system they are:

Commands   MYNAH Tcl commands

Procedures   Statements written in MYNAH Tcl that perform often-used functions.

12.7.2.1.2   Selecting a Template Package

Once you select a **Type**, you can choose a **Package**. Packages are subsets of templates. The default for template packages is **ALL** which means the system displays all templates of the specified type.

The Package selections for **Commands** are:

3270	3270 Terminal Emulation commands.
AppApp	
Async	Async Terminal Emulation commands.
ChildScript	Child script commands.
Dce	DCE commands.
General	Mynah Tcl general commands.
Prt3270	3270 Printer Emulation commands.
Tcl	Standard Tcl commands.
TclX	Extended Tcl commands.
Tcp	
TOP	Top commands.
X25	
All	All available commands.

The Package selections for **Procedures** are:

3270	3270 procedures.
Batch	Batch procedures.
General	General purpose procedures.
Top	TOP procedures

CONV-3270	3270 conversion procedures for converting scripts written using earlier versions of the MYNAH System.
CONV-Async	Async conversion procedures for converting scripts written using earlier versions of the MYNAH System.
CONV-Lma	LMA conversion procedures for converting scripts written using earlier versions of the MYNAH System.
CONV-Tsf	TSF (Test Specification File) conversion procedures for converting scripts written using earlier versions of the MYNAH System.
All	All available procedures.

12.7.2.2   Inserting a Command Example

We will illustrate using the **Insert Template** by inserting a `wait` command so that our script will wait during initial processing.

1. With the Code view displayed and the cursor positioned where you want to insert the command, click on the **Edit** menu and then on the **Insert Template** menu selection.

   — This causes the system to display the Insert Template dialog (Figure 12-11) .

2. Choose **Command** or **Procedure** by clicking on the **Type** option list. (For our example we would select **Command**).

3. Choose a package from the **Package** drop-down list. (For our example we left the default **3270**).

4. Select a template from the Template scrolling list by highlighting it. (For our example we selected `xmyTerm3270<Handle> wait`).

   — When you select a template, the system will automatically display it in the **Template** display area and an example of it in the **Example** display area.

5. To use a **Template**:

   A. Click on the **Apply** push button to insert the template into the Code View at the current cursor position.

   > **NOTE —** It is not necessary to apply the entire template. You can select part of the template by dragging the pointer across the portion of the **Template** to highlight it and click **Apply**.

B. Modify the inserted code in the Code View to meet you needs.

— Figure 12-12 illustrates this process.
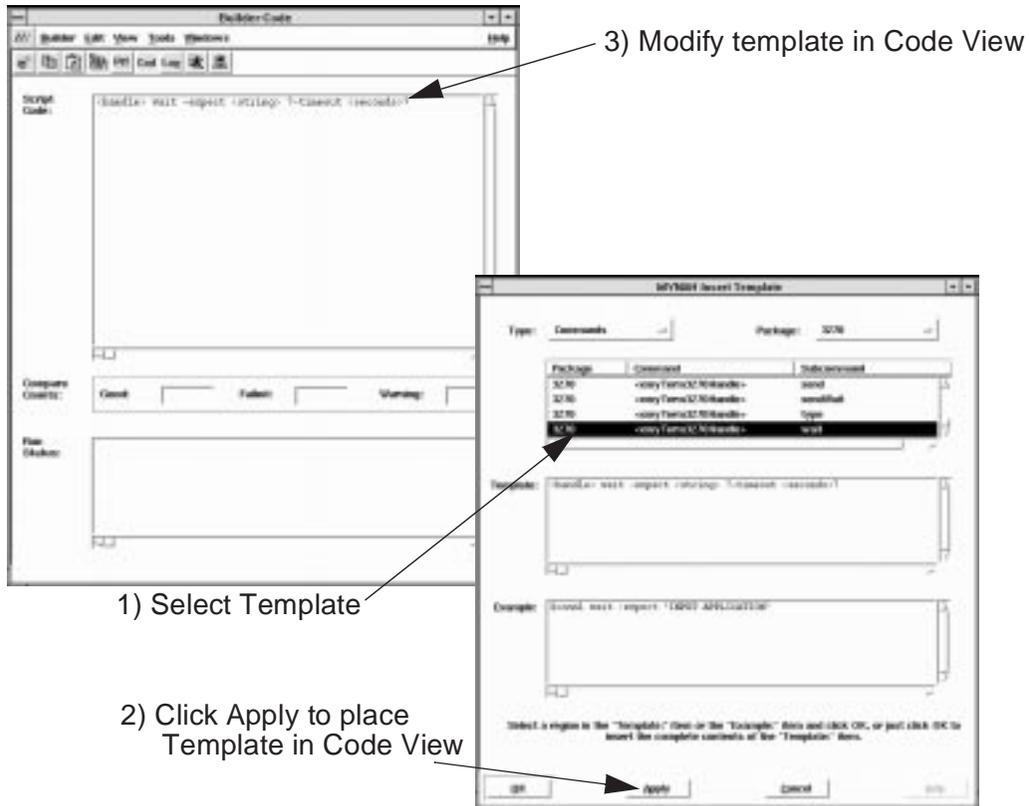
Figure 12-12. Inserting a Template into the Code View

6. To use an **Example**:

A. Drag the pointer across the portion of the **Example** to highlight it and click **Apply**.

B. Modify the **Example** in the Code View to meet your needs.

— Figure 12-13 illustrates this process.
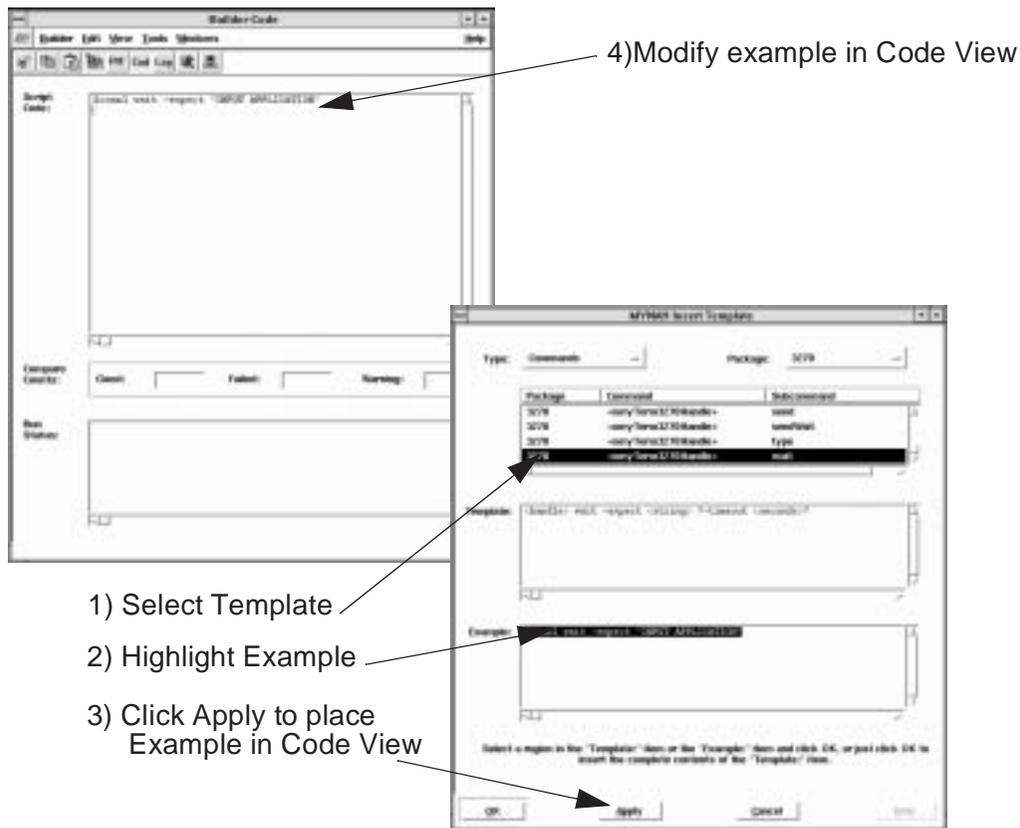
Figure 12-13. Inserting an Example into the Code View

12.7.3   Inserting Code into the Code VIew

Another way to develop scripts is to take an existing script and modify it to perform a new function. This works best if the new script is similar in function to an existing script. Also, you may want to work on a script that you already started. If you already have scripts that you want to modify or you want to continue working on a script you started, you simply insert them into the code view.

You do this with the **Insert Code** dialog which you can access by clicking on the **Edit** menu and then on the **Insert Code** menu option. There are two selections for this option:

From File   Insert code from a file in the UNIX file system.

From MYNAH Script7 Insert code from an existing Script object.

12.7.3.1   Inserting Code From a File

To insert code from a file, select **From File** and the system will display the dialog shown in Figure 12-14.

Select HOME directory

Click here to go up
one directory level

Click here to go down
one directory level
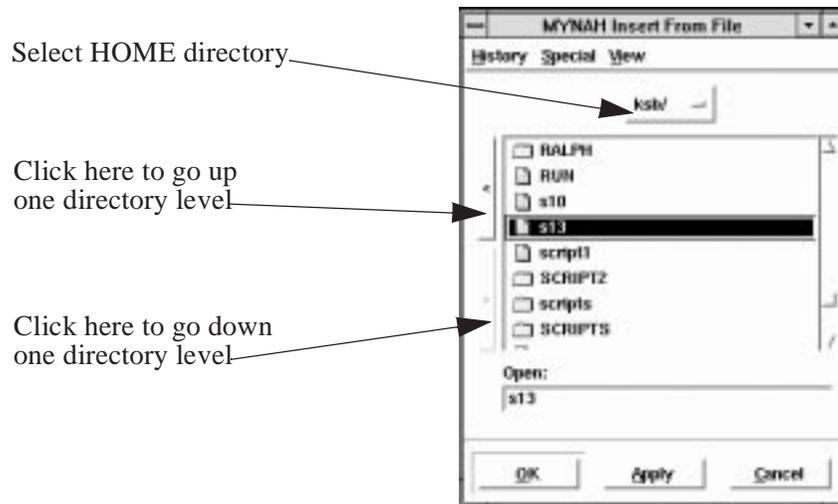
Figure 12-14. Insert Code From File Dialog

The system displays a list of directories. You can go up or down a directory level by clicking on the arrow bars on the left-hand side of the dialog. The option list at the top of the dialog allows you to select a HOME or starting directory.

Three menus appear on this dialog:

History	Shows a history of the directories you have selected in the order they were selected.
Special	The sort criteria are by: **Home**, **Mark**, and **Unmark**. This menu selection witches to your **Home** directory automatically. **Mark** and **Unmark** marks and unmarks a file. Marking a directory causes it to appear in the lower half of the **History** list so that it is easier to access.
View	The sort criteria are by: **Name**, **Date Modified**, **Date Created**. This sort allows you to change the order in which the system displays items in the scrolling list.

12.7.3.1.1   Example of Inserting Code

We selected a file in the following way:

1. Position the pointer in the Code view where you want the code inserted, and click.

2. Select the script code file you want. (We selected *s13*.)

3. Click **OK**.

OR

1. Position the pointer in the **Open** data area and type in the file name, e.g., `s13`.

2. Click **OK**.

— The system will insert the contents of the files you selected into the Code View. See Figure 12-15.
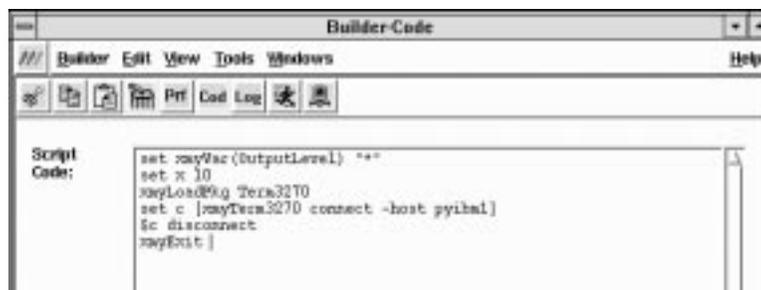
Figure 12-15. Code View with Inserted File s13

12.7.3.2   Inserting Code From a MYNAH Script

To insert code from a Script object in the MYNAH database, select **From MYNAH Script.** The system will display the dialog shown in Figure 12-16.
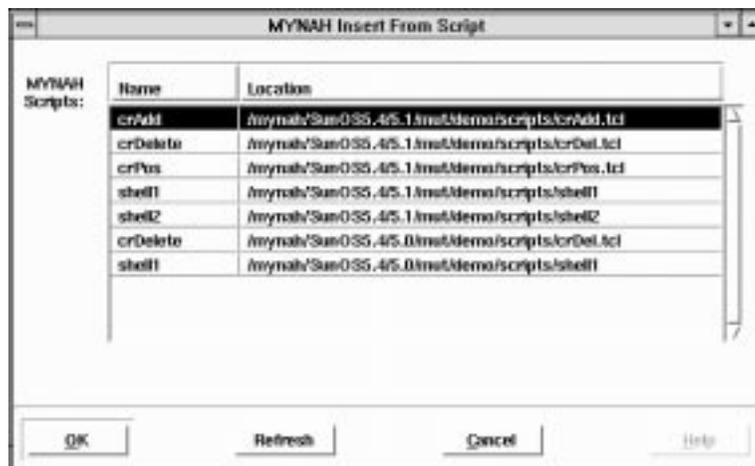
Figure 12-16. Insert Code From Script Dialog

The system displays a scrolling list of scripts. The list is based on the **Database Browser** query for Script object. Note that the only Script objects displayed here are for our Demo application. This is because our default query selected these Script objects.

The system lists the **Name** of the script and the **Location** of the script file in the UNIX system.

To insert a script:

1. Select the script you want. (We selected *crADD.tcl.*).

2. Position the pointer in the Code view where you want the code inserted, and click.

3. Click **OK**.

    — The system will insert the code at the position you indicated. We show the script code we selected in the Code View in Figure 12-17. Since there was nothing in the Code view when we inserted the code from the Script object, the system placed the code beginning at the first line in the Code View.

Figure 12-17. Code View with Inserted Script Code

12.7.4   Inserting Breakpoints into Code

Breakpoints are useful tools for analyzing script execution. A Breakpoint pauses code execution at a specified place which allows you to see just what occurred at that point in the script.

To insert a Breakpoint:

1. Position the cursor in the line above where you want to place the breakpoint. The system will place the Breakpoint on the line *below* the cursor position.

2. Click on the **Edit** menu and then on the **Insert Breakpoint** menu option.

   — The system will place an xmyBreakpoint statement in the code. Figure 12-18 illustrates inserting a breakpoint.
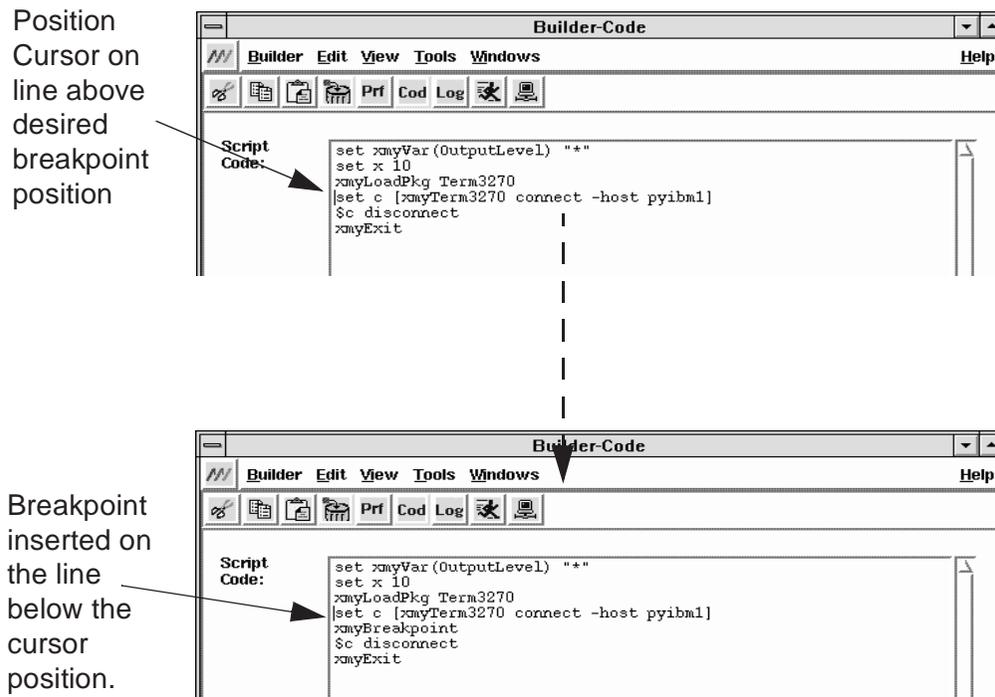
Figure 12-18. Inserting Breakpoints

12.8   Running Code

Once you have entered code, you can run it directly from the Script Builder. This is useful to see how the code you entered behaves and if you need to modify it.

You run code with the Run Code dialog. The Run Code dialog is similar to the Preferences view. You specified the default run settings on the Preferences view. You can run code using the default settings by simply clicking on **OK**.

In this sub-section we will go into some detail about the ways you can run code.

> **NOTE —** If you close the Script Builder while code is running, the system will display the message: **"Closing the Builder will cancel the currently running script. Continue?"** If you click **OK**, the Script Builder will be closed and the script or code will be cancelled.

To access the Run Code dialog, perform one of the following:

- Click on the **Builder** menu and then the **Run Code** menu selection

- Click on **Runner** icon in the Tool bar.

The MYNAH System displays a Run Code dialog similar to the one shown in Figure 12-19.
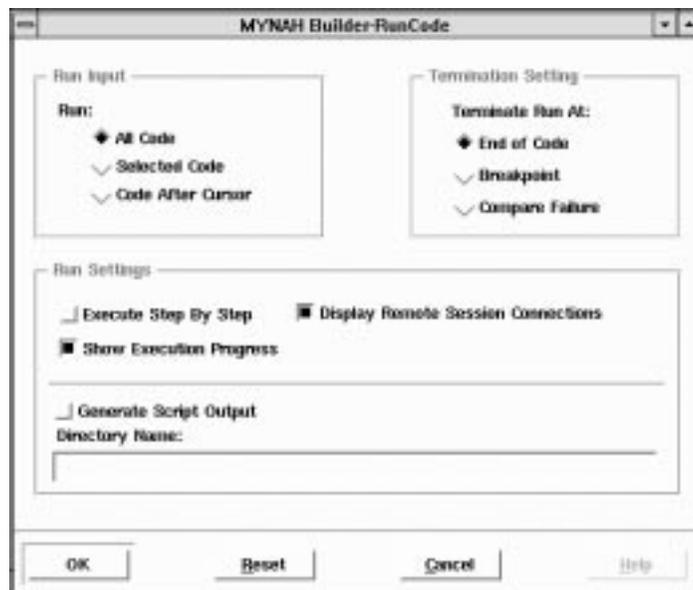
Figure 12-19. Run Code Dialog

You can change your run setting here. We described the settings on this dialog and how to change them in *Section 12.4: Using the Script Builder Preferences View.* Refer to this section for information about the settings.

When you run a script, status messages generated by the MYNAH System will appear in the **Run Status** area of the Code view. Any user defined return values or exit messages will appear here also, i.e. string arguments in **xmyExit** commands.

In the following sub-sections we will give you several examples of running code. First, we will describe the Run Progress dialog which appears whenever you run code.

12.8.1   Using the Run Progress Dialog

When you run code, the MYNAH System will display the Run Progress dialog which allows you to control execution progress. Figure 12-20 below shows the Run Progress Dialog.

Figure 12-20. Run Progress Dialog

You will see the Run Progress dialog if it is selected on the Preferences view or the Run Code dialog. It is enabled as a default, but you can deselect it. You may want to disable it if you are running many lines of code. Code will run more efficiently if you don't leave the Run Progress dialog enabled during execution. You can display the Run Progress dialog again if you want to by clicking on the **View** menu and the **Show Progress** menu selection.

As the code executes, the **Progress** slider indicates that the code is executing.

The **Log** display shows which line of code just executed and which is currently being executed.

The system displays the current status of code execution in the **Status** display area. The status of the code can be either "stepping", "running", or "paused".

This dialog provides **Close**, **Next Step**, **Pause**, **Resume,** and **Cancel** pushbuttons that allow you to control code execution.

• To close the Run Progress dialog, click on **Close**. This simply closes the Run Progress dialog, it doesn't stop the code from executing. You will still see "Running" in the Script Builder's Status Bar. To re-display the Run Progress dialog, follow the instructions above.

• To execute the next command line while in step mode, click the **Next Step** push button.

• To pause a script, click on the **Pause** pushbutton to pause the script. This actually toggles the mode between Step and non-Step.

• To start it again, click on the **Resume** pushbutton to start the script again.

• To cancel code execution, click on the **Cancel** pushbutton.

12.8.1.1   Running Code Examples

Figure 12-21 shows the code we will run to illustrate the different ways you can run code. It is simple code that opens and closes a 3270 connection.
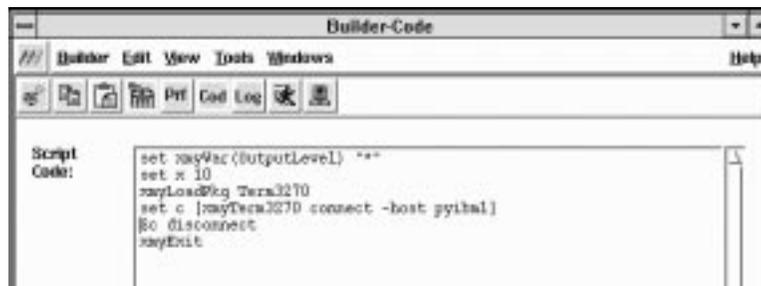
Figure 12-21. Running Code Example

12.8.1.1.1   Running All Code to Completion

In our first example we will run all the code we created to completion. To do this:

1. Click on the **OK** pushbutton if you accept the default settings.

   — The system will run the code.

2. If you need to re-set the **Run Input** selections, click on **All Code** button.

3. Click on the **Display Remote Connection** button if you want to see the connection window opened and the commands as they are executed.

4. Click on the **Generate Script Output** radio button if you want MYNAH to generate the output directory and output files.

5. If you did Step 4, position the pointer in the **Directory Name** data entry area and type in a UNIX path *only* for the MYNAH output files, e.g., `/<home>`.

6. Click on the **OK** to start execution.

   — The MYNAH System will display the **Run Progress** dialog which will show the progress of script execution.

12.8.1.1.2 Running Selected Code

We can choose to run just a part of the code we created in the Code view. We do this by highlighting the code we want in the Code view and then selecting Run Selected code. For our example we will run the first two statements.

1. In the Code view, highlight the statements you want to run by dragging the pointer across them. For our example, select the third and fourth statements, `xmyLoadPkg` and
   `set c [Term3270 connect -host pyibm] b`

2. Access the **Run Progress** dialog.

3. Click on the **Selected Code** radio button.

4. Click on **End of Code** radio button.

5. Click on the **Generate Script Output** radio button if you want an output directory for run output.

6. If you performed Step 4, position the pointer in the **Directory Name** data entry area and type in a file name and UNIX path *only* for the MYNAH output files, e.g.,
   `/<home>`.

7. Click on **OK** to start execution.

   — The MYNAH System will display the **Run Progress** dialog which will show the progress of script execution. A 3270 connection will appear at the end of execution. The connection will remain until you close it.

12.8.1.1.3 Running Code with Breakpoints

If we insert a breakpoint in our code, we could run it to completion with pauses at each breakpoint. To do this:

1. Click on the **All Code** radio button.

2. Click on **End of Code** radio button.

3. Click on **OK** to start execution.

— The MYNAH System will display the **Run Progress** dialog which will show the progress of script execution. The code will run until it encounters the breakpoint and then it will pause.

4. To start executing the code again, click on the **Resume** button on the Run Progress dialog.

— The code will continue executing until it finishes or encounters another breakpoint.

12.8.1.1.4   Running Code Until Compare Failure

If you have compares in your code, you can set the Run dialog so that when the code encounters the first failed compare, it will stop execution. To do this:

1. Click on the **All Code** radio button.

2. Click on **Compare Fail** radio Button.

3. Click on **OK** to start execution.

— The code will execute until it encounters a compare fail or until it completes.

12.8.1.1.5   Pause at a Failed Compare

If you would like the Script Builder to 'pause' at failed compares then you must use the MaxFailsHandler feature and you must have a MaxFailsHandler procedure that contains an **xmyBreakpoint** statement e.g.,

- **set xmyVar (MaxFailsHandler) maxFails**

- **set xmyVar (MaxFails) 0**

- **proc maxFails {}{
  xmyBreakpoint
  }**

You can 'configure' the Embedded Script Engine to always behave this way by setting the MaxFailsHandler in the xmyConfig file.

12.8.1.1.6   Running Code Step-By-Step

For this example we will run all the code we created, step-by-step, and stop execution after all the code has executed. To do this:

1. Click on the **All Code** radio button.

2. Click on **End of Code** radio Button.

3. Click on the **Execute Step by Step** radio button.

4. Click on **OK** to start execution.

    — The MYNAH System will display the **Run Progress** dialog which will show the progress of script execution. See Figure 12-17.

    — Since we specified **Execute Step by Step**, the system will pause after it executes the first statement in the code.

5. Click the **Next Step** push button on the **Run Progress** dialog to execute the second statement in the code.

    — The system will pause after the next step executes.

6. Repeat Step 5 until all the statements complete.

12.9   Saving Code

Once you have code in the Code view, you save it by clicking on the **Builder** menu and then on the **Save** menu selection. You can choose to **Save to File** or **Save To MYNAH Script**.

12.9.1   Saving Code to a UNIX File

If you choose to **Save to File**, the system displays the MYNAH Save To File dialog. See Figure 12-22.
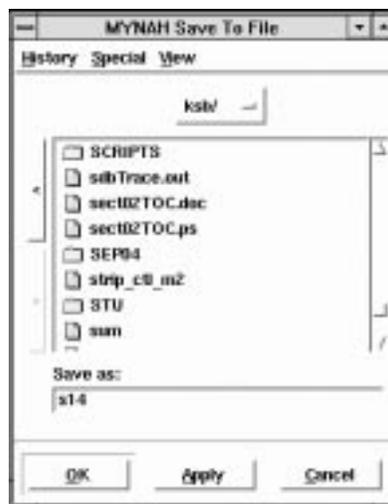
Figure 12-22. Save to FIle Dialog

This dialog is exactly like the **Insert From File** dialog. Select the file you want to save the code to and click **OK**. If you need a more detailed explanation on how this dialog works, refer to Section 12.7.3.

12.9.2   Saving Code to a MYNAH Script

If you selected **MYNAH Save to Script**, the system will display the MYNAH **Save To Script** dialog. See Figure 12-23. If you are going to save code to a MYNAH script, a Script object must exist in the MYNAH database.
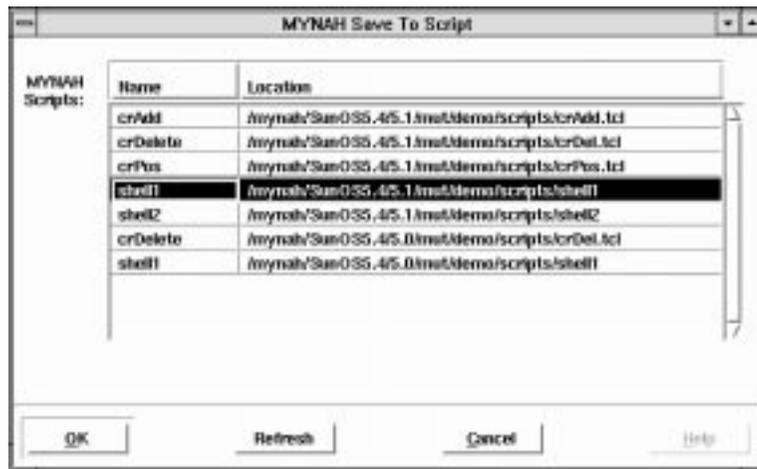
Figure 12-23. Save to MYNAH Script Dialog

This dialog operates like the **Insert MYNAH Script** dialog. If you need a more detailed explanation on how this dialog works, refer to Section 12.7.3.

To save the code to a script, select the script you want and click **OK.**

12.9.3 Saving Code When You Exit the Script Builder

If you exit the Script Builder without saving code you entered or recorded, the system will prompt you to save the code with the dialog shown in Figure 12-24. You can save to a file or a script or exit without saving the code in the Code view.
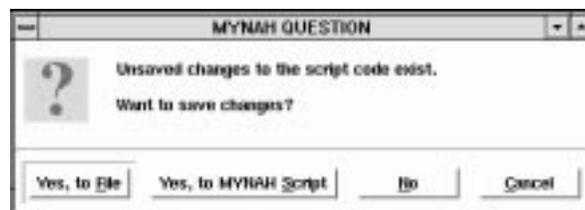
Figure 12-24. Save Code Dialog

12.10 Establishing Connections to a SUT

You can establish connections to a SUT through the Script Builder. The Script Builder offers you two types of connections: 3270 and Asynchronous. These connections allow you to record keystrokes and events occurring on the remote SUT. You can incorporate these keystrokes and events into Tcl statements.

You access a connection from any view in the Script Builder by either of the two following methods:

- Clicking on the **Builder** menu and then on the **Open Connection** menu selection
- Clicking on the **Terminal** icon in the tool bar.

The MYNAH System will display the dialog in Figure 12-25.

Figure 12-25. Open Connection Dialog

You select the type of connection you want from this dialog. You can also enable recording of events on the remote SUT and assign an alias to the connection. Remember, there are default settings for connections, but you can re-set these with the Preferences view.

When you access a connection, the system will display the Open Connection dialog set to the **3270**, the default setting, or to **Async** if you set this with the Preferences view.

The settings on this dialog are:

Type	Specifies the type of connection. The choices are **3270** or **Async** (asynchronous).
Record connection statement	Causes the system to include the **connect** statement in the Code View.
Record load package statement	Causes the system to include the **load package** statement in the Code view.

Record emulation	Causes the system to turn on recording as soon as the connection is displayed.
Connection Variable Name	Allows you to define a default variable name for the connection. The name must be a unique name. For example, the system supplies a name, `conn<n>`. The `conn` number starts at 1 and is incremented by 1 for each new connection.

12.10.1 Making a 3270 Connection Example

As an example, let's assume that we want to set up a 3270 connection that records the connection statement, the load package statement, and our interactions with the SUT. We want to code these statements so that you can use them in other scripts. We want to use this connection in other scripts, so we will use the default variable name (alias) assigned by the system (conn1). We could assign our own variable name as long as it was a unique name. To do this:

1. Select **3270** from the **Type** option list.

2. By default, **Record connection statement** is selected and we will accept this.

3. Click on the **Record load package statement** toggle button.

4. By default, **Record emulation** is selected, and we will accept this.

5. Click **OK**.

    — The system will display a 3270 Connection window. Refer to the next section for more information about using this window.

12.10.2 Making an Asynchronous Connection

If we wanted an asynchronous connection that used the same basic parameters as the example above, we would follow the steps above except that we would select **Async** as the **Type** and we would give it a different variable name, e.g., conn3.

12.10.3 Opening a Connection With Code

If code you are running opens a connection using a **connect** statement, i.e., a script you run has an **xmyTermAsync -connect** or **xmyTerm3270 -connect** command, **Record emulation** will be set to OFF. If you turn record emulation ON, the system will display a dialog in which you must enter a connection**Variable Name** or accept the default name supplied by the system.

12.10.4 Connection Window Layout, Menu and Icons

The 3270 and Async Connection windows are different enough from other MYNAH windows that we will describe their unique features here. You can refer to *Appendix A: MYNAH Menus and Menu Option* for a complete listing of the menus and menu selections available for the connection windows.

Table 12-1 describes the new menu items and icons available with the 3270 Connection windows.

Table 12-1. 3270 Menu Selections and Icon Functions

Menu Selection	Menu Its On	Icon	What It Does
Reset Ignore List	Edit	IgR	Resets the list of ignore statements.
Insert Ignore	Edit	Ig	Inserts a statement that specifies a screen region a compare will ignore.
Insert Compare	Edit	=?	Inserts a statements that specifies a screen region for a compare.
Enter Expression	Edit	Ee	Displays a dialog with which you can insert a Tcl expression into code.
Save Value From Screen	Edit	→V	Displays a dialog with which you can save a value from the connection screen by assigning it to a variable name.
Record Emulation On/Off	Connection	(set to OFF) OFF	Toggles record on and off.
Default Compare	Connection	none	Enables the use of Default Procedure

Table 12-1. 3270 Menu Selections and Icon Functions

Menu Selection	Menu Its On	Icon	What It Does
Location Processing	Edit	none	Enables the type of location processing you want: **Label** (available in future releases), **Tag Name, Row/Column**
14, 18	Fonts	none	Changes default font for connection window.

Table 12-1 describes the new menus and icons available with the Async Connection windows.

Table 12-2. Async Menu Selections and Icon Functions

Menu Selection	Menu its on	Icon	What It Does
Reset Ignore List	Edit	IgR	Resets the list of ignore statements.
Insert Ignore	Edit	Ig	Inserts a statement that specifies a screen region a compare will ignore.
Insert Compare	Edit	=?	Inserts a statements that specifies a screen region for a compare.
Enter Expression	Edit	Ee	Displays a dialog with which you can insert a Tcl expression into code.
Save Value From Screen	Edit	◆V	Displays a dialog with which you can save a value from the connections screen under a variable name
Record Emulation On/Off	Connection	(set to OFF) OFF	Toggles record on and off.

There is a status display to the right of the Tool Bar. This tells you the mode you are in, e.g., whether you are recording emulation or not.

The client area, which appears below the Menu and Tool Bar, serves as the 3270 or VT 100 terminal display. All messages from the remote SUT and key strokes you enter will appear in this area.

The connection windows displays the cursor location at the bottom, right-hand corner of the connection window in the format Row and **C**olumn, e.g., **R:1 C:1**

12.10.5   Using a 3270 Connection

The 3270 Terminal Emulator window provides all the functions of an IBM 3270 MOD 2, 3, 4, or 5 terminal. It will appear similar to the one shown in Figure 12-26.
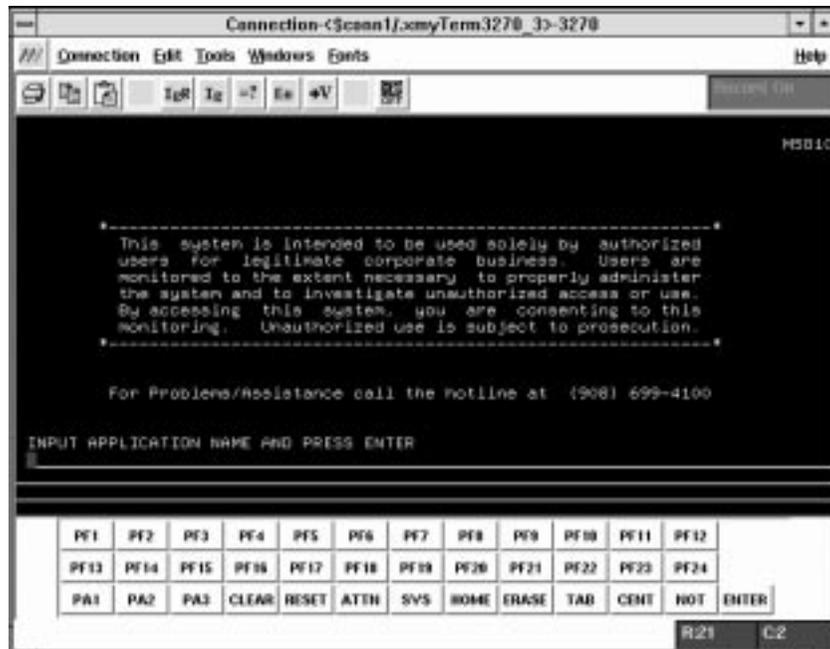
Figure 12-26. 3270 Terminal Emulator Window.

The appearance of the 3270 Connection window is very similar to the Async connection window. The difference is that below the client area are three rows of function keys:

- The first row contains PF 1 through 12

- The second contains PF 13 through 24

- The last row contains PA 1 through 3, CLEAR, RESET, ATTN, SYS, HOME, ERASE, TAB, CENT, NOT, and ENTER.

For a definition of the key functions, refer to IBM documentation for 3270 Terminals. For a description of how these keys are mapped to your workstation, change to the Preferences view and use the Function Key area to find the mapping of a 3270 key to your keyboard. We explained how to do this in Section 12.4.

12.10.6   Setting 3270 Features

The 3270 connection offers you a number of features that help you develop scripts. The following sub-sections describe these features.

12.10.6.1   Defining Default Compares

Default Compares are compare statements that MYNAH automatically codes each time a new screen response is received. You must define the region for the compare on the Preferences view. See Section *12.4.6.6* if you need to do this. You can enable or disable a default compare for the current session by:

- Clicking on the **Connection** menu and then on the **Default compare** toggle button to turn the selection ON or OFF.

12.10.6.2   Enabling and Disabling Default Compare

If you didn't enabl**e** Default Compare with the Preferences view, you can enable it here by:

- Clicking on the **Connection** menu and then on the **Default Compare** toggle button to turn it ON or OFF. Remember that you must define the **Default Compare** in the Preferences view.

12.10.6.3   Using Default Procedures

As we mentioned in Section *12.4.6.5*, a Default Procedure can be stored in a procedure library or script and is executed every time there is a new screen response from the SUT.

A default procedure can perform two actions:

- Perform an action in the connection window in response to a new SUT screen appearance. (e.g., every time a screen is received, move the cursor to row 24, column 5.)

- Write a Tcl command in the Code View (e.g., every time a screen is received, write the Tcl command in the Code View to call the procedure "checkScreenTitle.") This command will be executed only when the code is run.

12.10.6.3.1 How to Select a Default Procedure

To enable a Default Procedure you must make two entries:

1. Specify the procedure name on the Preferences view.

2. Enable the option. You may enable the option by using either of the following methods:

   • Select the option on the Preferences view. In this case, the procedure will be executed in all connections until you deselect the option in individual connection windows, as explained below.

   • Select the option on the Connection menu on individual connection windows. In this case, the procedure will be executed only in the connection window in which you selected Default Procedure.

12.10.6.3.2 How to Select a Default Procedure for an Individual Window

If you want to deselect the option for an individual connection window, you should click on the "Connection" Menu and if the Default Procedure option is highlighted, click on it to deselect the option. Make sure that you deselect this option before you type in the connection window.

For example, suppose you want to open three 3270 Connection windows (and you name them $conn1, $conn2, and $conn3) but you want the Default Procedure executed in $conn1 and $conn2 only. After entering the procedure name on the Preferences view, select Default Procedure execution so that the option will apply to all connection windows that you open. Open the connections for $conn1, $conn2, and $conn3. In the connection window for $conn3, click "Connection" menu and then click Default Procedure to deselect this option. After you deselect this option, begin typing in the $conn3 Connection window.

> **NOTE —** As with any procedure, the procedure library or script must be identified to the Script Builder. If the procedure library or script is not in the Procedure Repository, you must code and execute a source command. For more information about the source command, refer to the *MYNAH System Scripting Guide*. For more information about the Procedure Repository, see your system administrator.

12.10.6.3.3 How to Code a Default Procedure

A Default Procedure must exist in a script or Procedure Library. The procedure requires two arguments:

1. "connection handle" which is the connection's handle (e.g., xmyTerm3270_2).

2. "variable name" which is the name of the connection's variable (e.g., $conn1).

For example, in the following default procedure, the two arguments are named "handlename" and "connvar".

```
proc defproc (handlename connvar} {
   Tcl command here
```

To automatically generate a Tcl command in the Code View, you must use the "writeStatement" command. The syntax of this command is:

```
$argument1 writeStatement "Tcl command"
```

Where:

$argument1   Specifies the name of the first argument (e.g., "handlename" from the preceding example.

Tcl command   Specifies the Tcl command that you want to execute (e.g., **call procedure *checkTitle*).

> **NOTE —** The Default Procedure must not execute a send or sendwait command. These commands would cause the procedure to be called recursively.

For example, suppose you want the following actions to occur every time a new SUT screen appears.

1. Move the cursor to row 24, column 5.

2. Code a command to call the procedure "checkScreenTitle".

You code a default procedure with the following commands:

```
proc myDefProc {handlename connvar}
#Purpose: This is a default procedure that performs the following
#actions: moves the cursor to 24,5 and codes a command to call the
#procedure "checkTitle."
        $conn1 moveCursor -position{24 5}
        $conn1 writeStatement "xmyPrint -text \"call the proc checkTitle\""
        $conn1 writeStatement "checkTitle"
}
```

While you interact with the SUT, the cursor will be positioned on row 24, column 5 after every new SUT screen appearance. In addition, the following Tcl commands will appear in the Code View after every key press.

> xmyPrint -text "call the proc checkTitle"
> checkTitle

Remember that these two commands will be executed only when you run code.

12.10.7 Selecting a Location Processing Type

After the system opens a 3270 Connection you will have another chance to select the types of Location Processing. The system defaults to **row/label**. See Section 12.4 for more details on the choices.

12.10.8 Positioning the Cursor in a 3270 Field

The current cursor position is indicated by a solid rectangle. To move the cursor to the field you want, you can use either of the following methods:

Using the Mouse Move the MYNAH cursor (i.e., the arrow cursor) to the field you want and double-click on the left mouse key.

Using the Tab Key Press the tab key to move the cursor to the field you want.

12.10.9 Using an Asynchronous Connection

The Asynchronous Terminal Emulator provides you with all the functions of an asynchronous terminal. It will appear similar to the one shown in Figure 12-27.
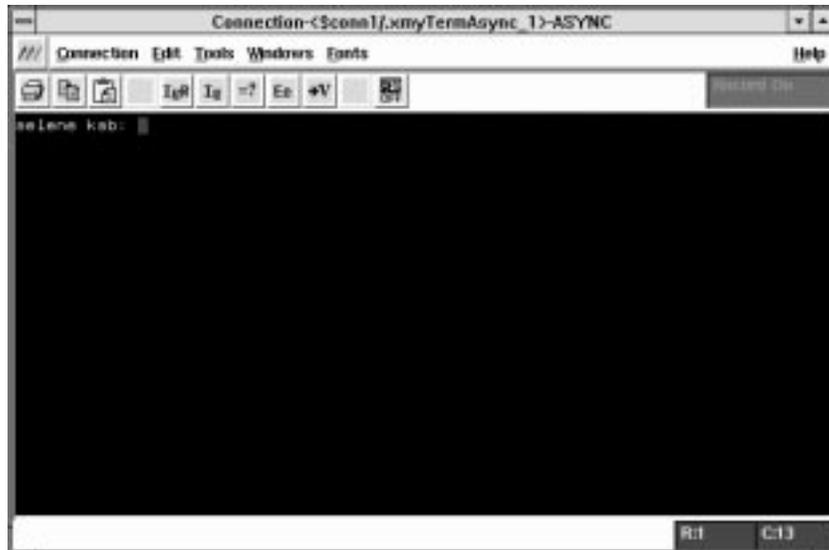
Figure 12-27. VT100 Asynchronous Terminal Emulator Window.

12.10.9.1 Defining Wait Conditions

You can define wait conditions for an asynchronous connection using the Tcl Wait Expression. The wait condition defines events on the remote system that the MYNAH System will wait for before executing the next command in a script.

There are three types of wait expressions you can define:

Default wait	A system-determined, unique, character string derived from the end of the last response from the SUT.
User Defined Prompt	A unique character string you enter. The system will wait for this character string before starting to process commands.
User Defined Expression	A Tcl expression you enter. See the *MYNAH System Scripting Guide* for information about this command.

You define a default wait condition using the Wait Expression dialog shown in Figure 12-28. To access this dialog:

• Click on the **Connection** menu and then on the **Tcl wait expression** menu selection.

Figure 12-28. Wait Expression Dialog.

There are two data display areas: one for **User Defined Prompt**, and one for **User Defined Expression**. These are also drop-down lists. Prompts and expressions you have Previously defined will appear in these lists. You can select prompts and expression you have already defined from these lists and re-use them.

To define a wait expression:

- Click on the **Default wait** radio button to accept the default wait.

- Click on the **User Defined Prompt** radio button and type in a prompt, e.g., `selene ralphm`. (This is the prompt on our system. It defines the machine name and user login ID. You can see it on the Asynchronous Connection window shown in Figure 12-27).

OR

1. Click on the **User Defined Expression** radio button and then type in the Tcl expression that defines the wait.

2. Click **OK.**

12.11 Recording Events on a Connection

We will illustrate capturing events on a remote SUT using the Async connection. This ensures that all users can try what we are describing here.

We have opened a connection from the code view and specified that we wanted to **Record connections statement**, **Record load package statement**, and **Record emulation** so that everything we do on the remote asynchronous system will be recorded in the code view.

12.11.1 Recording Keystrokes

For our example, we simply typed in the **ls** command and pressed **Return**. When we pressed **Return** again, the system records the keystrokes we made previously.

Figure 12-29 shows what occurred on the remote system. As you can see, the MYNAH System recorded the keystrokes we entered on the Connection window in the Code View.
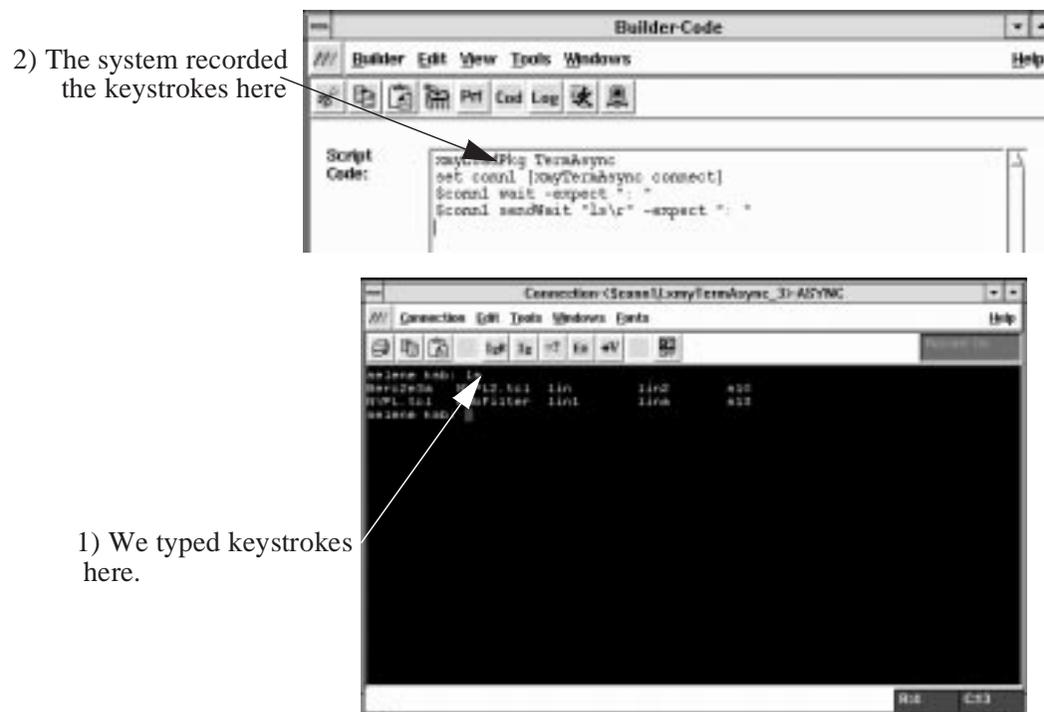
Figure 12-29. Capturing Events on the Remote Asynchronous System

12.11.2 Inserting Compares

In the example above, only the keystrokes we entered appeared in the Code view. Events that occurred on the connection window, such as the output listing for the `ls` command, were not recorded.

The Script Builder gives you an easy way to record these events and place them in the Code view. Not only will the events be recorded, but the system will insert compares into the Code view with data from the specified screen region.

You can define a screen region in a simple drag-the-pointer operation, and the contents of this screen region will be inserted into the Code view with a Compare Statement.

To do this

1. Type a command into the Connection window, e.g.,

    ls

    and press **Return**.

2. Specify the screen region you want to compare by dragging the pointer to "box" the region.

    — As you drag the pointer, the system will expand a box around the region you select. We "boxed" the listing of directories and files in our home directory which came as a result of the `ls` command.

3. Click on the **Edit** menu and then on the **Insert Compare** menu selection; or click on the **Compare** icon (=?).

    If *row/column* location processing was selected:

    — The system will record a **compare region** command for the contents of the screen region you defined. Figure 12-30 illustrates this.

    If *tagname* location processing was selected:

    — The system will record individual **compare tag** commands for all screen fields that exist in the region you selected.

    If you select an area that consists of one tag:

    • If the tag is not in the screen's *tagname* table, a "**compare region**" command is recorded.

    If you select an area that consists of more than one tag:

    • If a tag in that area does not exist in the screen's *tagname* table, no **compare** command will be recorded for that missing tag.

3) System inserts
Compare Statement
with recorded
region.

1) Box compare
region

2) Click Compare
Icon
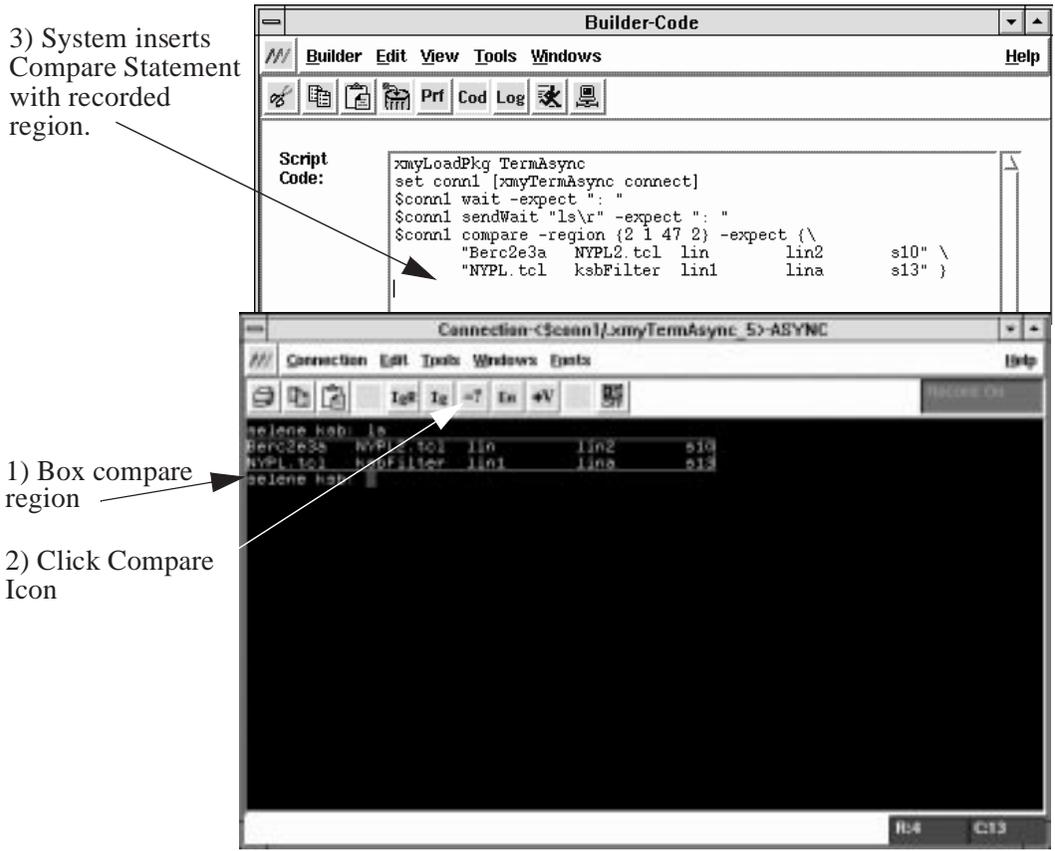
Figure 12-30. Capturing Compare Regions and Inserting Them in the Code View.

12.11.3 Ignoring Screen Regions with Compares

You may want to record part of a screen region and ignore other parts. You can do this by specifying the regions you want to ignore before you do a compare. Let's look at the example we used in the last section, except that this time we will ignore part of the display that results from the `ls` command.

> **NOTE —** The region that you specify to ignore will apply on every screen you access until you re-set the *Ignore List*. For more information about re-setting the list, see Section 12.11.3.1.

To do this

1. Type a command into the Connection window, e.g.

   ```
   ls
   ```

   and press **Return**.

2. Specify the screen region you want to ignore by dragging the pointer to "box" the region.

   — As you drag the pointer, the system will expand a box around the region you select. We "boxed" the first item in the listing - `BASELINED Untitled lin2 pwd s13`.

3. Click on the **Edit** menu and then on the **Insert Ignore** menu selection; or click on the **Ig** icon.

   — A message will appear in the status area noting which regions will be ignored.

4. Next, specify the screen region you want to compare by dragging the pointer to "box" the region.

   — As you drag the pointer, the system will expand a box around the region you select. We "boxed" the listing of directories and files in our home directory which came as a result of the `ls` command.

5. Click on the **Edit** menu and then on the **Insert Compare** menu selection; or click on the **Compare** icon (**=?**).

   — If *row/column* location processing was selected:

   The system will record a **compare region** command for the contents of the screen region you defined. The **Insert Ignore** region will be marked as an *ignore* attribute in the command. Figure 12-31 illustrates this process.

   — If *tagname* location processing was selected:

   The system will record individual **compare tag** commands for all screen fields that exist in the region you selected.

However, for those fields that exist within the *ignore* region, a **compare tag** command will not be recorded. A comment will be recorded alerting you to all the tags that were ignored.

For more information about recording **compare tag** commands, see Section 12.11.2.

5) System inserts recorded region with ignore region marked.

1) Box ignore region

2) Click Ignore Icon

3) Box compare region

4) Click Compare Icon
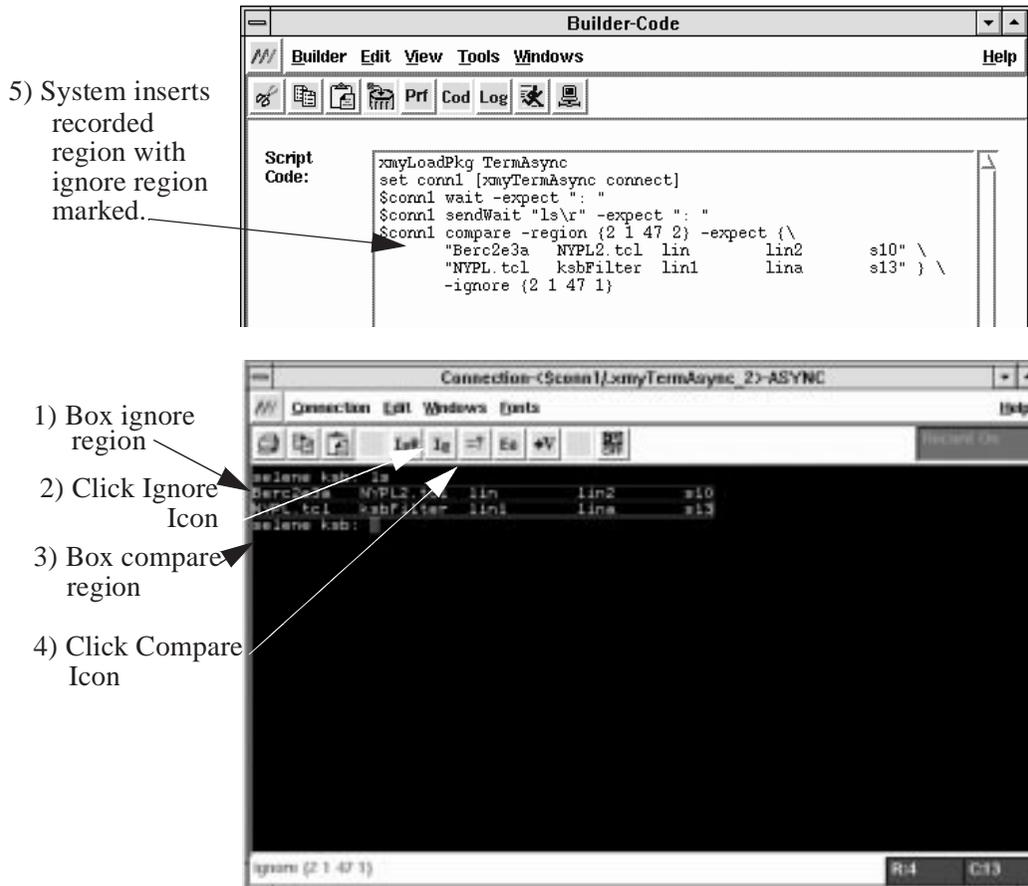
Figure 12-31. Ignoring Screen Regions in a Compare

12.11.3.1 Re-setting the Ignore List

The MYNAH System keeps a list of every ignore region specification you create. You can reset this list by performing one of the following:

- Clicking on the **Edit** menu and then on the **Reset Ignore List** menu selection.
- Clicking on the **Ignore Region** icon (**IgR**).
    - The system will display a message in the status area stating that the ignore list has been reset.

12.11.4 Saving Values From a Connection Window

You can record values that appear on a connection window and save them by assigning the value to a variable. You may use the variable elsewhere in the code.

To do this:

1. On the connection screen, box the items you want to save, e.g., `ls`.

2. Click on the **Edit** menu on the connection window and then on the **Save Value From Screen** menu option or click the **Save Value** icon (**->V**).

    — The system will display the MYNAH Requesting Information dialog shown in Figure 12-32.
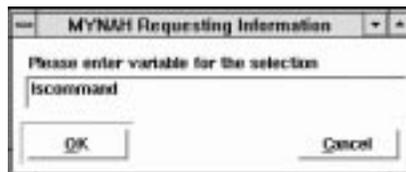
Figure 12-32. Requesting Information Dialog

3. Type in a name for the variable and click **OK** (e.g., *lscommand*).

    — The system will save the value to the specified variable and construct a Tcl statement in the Code View. Figure 12-33 illustrates this.

1) Box the items you want.

2) Click here.

3) Type in a name for the variable.

4) Value will be coded in a Tcl statement

    in the Code view.
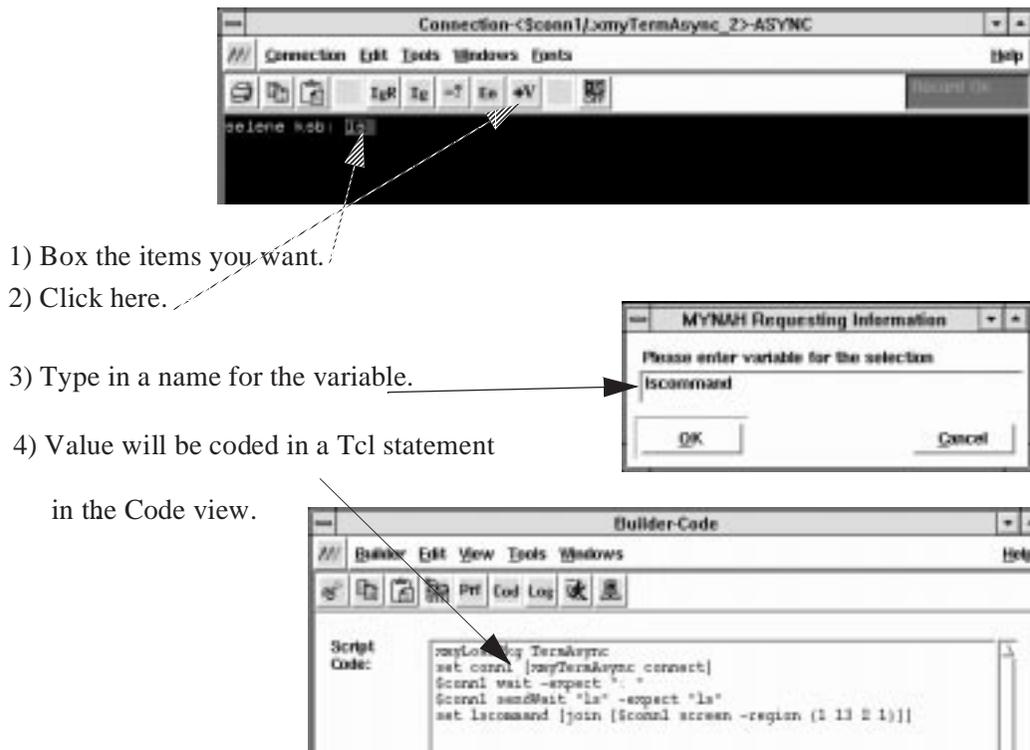
Figure 12-33. Saving a Value From a Screen Example

12.11.5 Entering Expressions

The Script Builder gives you a convenient way to send complex strings to the System Under Test through the Enter Expression confirm. When you type on a Connection screen directly, a literal string is sent to the SUT and coded in your script. If instead you wanted to send a string that was previously saved in a variable, Enter Expression would allow you to type in the **$<variable name>** and have the value sent to the SUT. Enter Expression also allows you to send the result of commands to the SUT in a asynchronous connection.

The following input is supported in the Enter Expression confirm:

- Unquoted literal strings, e.g., exit

- Variables, e.g., **$logname**

- Commands, e.g., [expr 1 + 1] .

In our example we will assign the value *ls -al* to the variable **lsCommand** and use Enter Expression to code the command and send it to the SUT in an asynchronous connection.

1. In the Code view, type

   ```
   set lsCommand "ls -al\r"
   ```

2. Select this code, and click Run. On the RunCode dialog, make sure **Run: Selected Code** is enabled. Click **OK** on the RunCode dialog. After the statement executes, make sure that the cursor in the Code view is on a line after that statement. By running this code you have assigned the value **ls -al\r** to the variable **lscommand**.

   > **NOTE —** For a 3270 connection, position the connection's cursor in the screen field in which you want the expression entered.

3. Click on the Edit menu on the connection window and then on the Enter Expression menu option or click the **Enter Expression** icon (**Ee**).

   — The system will display the Requesting Information dialog shown in Figure 12-34.

Figure 12-34. Requesting Information Dialog

4. Type

   ```
   $lsCommand
   ```

   in the Requesting Information dialog (Figure 12-35) and click **OK**.
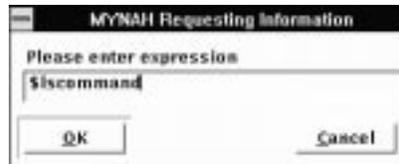
Figure 12-35. Specifying a Name for an Expression

— The system will code a statement containing **$lsCommand** (Figure 12-36) and
   send the value of **$lsCommand** to the SUT.
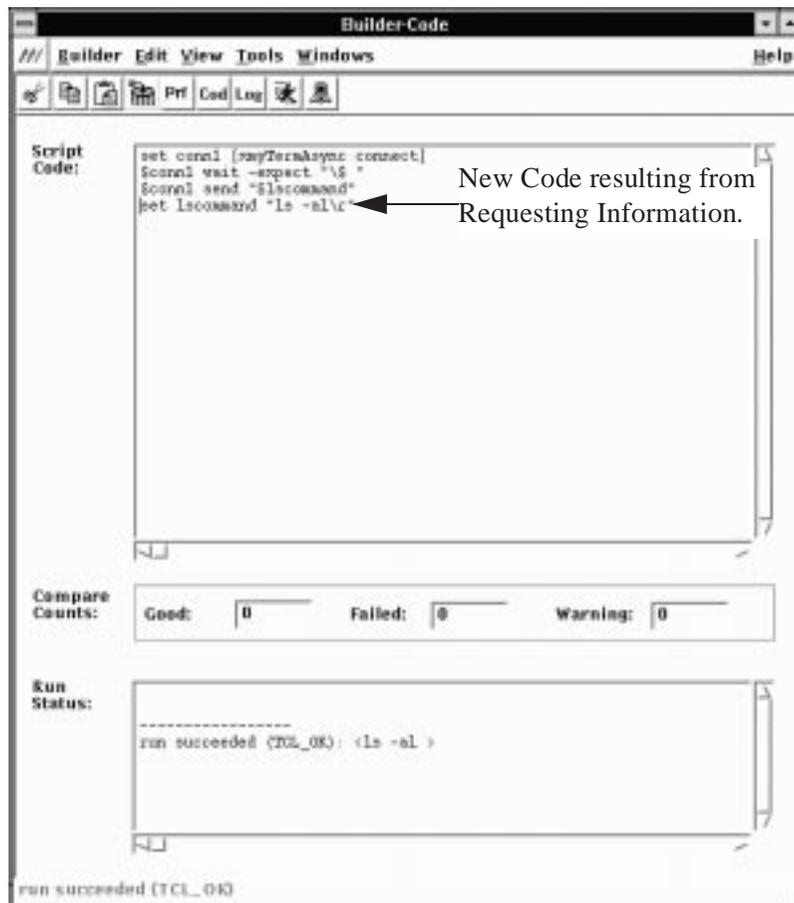
Figure 12-36. Code View with Expression

12.11.6 Receiving Extra Screens for a 3270 Connection

When you capture events on a 3270 connection, an extra (or asynchronous) screen may be received in the connection. If this occurs, the extra screen will probably be generated when the code is executed, and this condition may cause a timeout. You must correct this condition or else your code will not execute.

The way to correct this condition is to tell the connection something unique about the next screen that you care about. You tell the connection about this expected screen by specifying a character string for the **-expect** option for the **sendWait** command. Consider the following examples.

- If you are waiting for your application's **Main Menu** screen to be displayed and if that screen contains the character string **MAIN MENU,** you could use **MAIN MENU** as the expected character string. For example,

  ```
  $conn1 sendWait -key enter -expect "MAIN MENU"
  ```

- If the string *Command Line:* appears on many SUT screens, you would not use that string because it does not provide a unique string for the screen that you are expecting.

> **NOTE —** It is advisable to specify a character string that appears near the botton of the screen. This will tell the connection to wait long enough for the screen to be completely displayed.

12.11.6.1 How MYNAH Responds to Extra Screens

During a Record Session, the MYNAH System checks to see if one or more extra screens are received. If extra screens are received, the system will emit a beep sound and enter the following message in the connection's message area to alert you that one or more extra screens have been received:

```
WARNING: Extra screen received. See Code view.
```

In the Code view, a block of commented lines will be inserted. Within that block, there will be a warning that describes the condition and a template for the appropriate **sendWait** command. The following example illustrates the original recorded code and the block of comments.

```
$conn1 type position "21 2" -text "ttso tksct10"
$conn1 sendWait -key enter
#
#WARNING: Extra screen received.
#The last recorded send may need an "-expect" option
#$conn1 sendWait -key <KEY> -expect "<EXPECT>"
#
```

12.11.6.2 Modifying the Code to Fix the Extra Wait Commands

To correct this condition, you must edit the code using either of the methods described in the next subsections. These subsections refer to the example presented in .

12.11.6.2.1 Editing the Initial sendWait

You can edit the initial **sendWait** statement, e.g,

```
$conn1 sendWait -key enter
```

by performing the following steps:

1. Add the **-expect** option and appropriate string (e.g., *MAIN MENU*), e.g.,

```
$conn1 sendWait -key enter -expect "MAIN MENU"
```

2. If desired, delete the block of comment lines, e.g.,

```
#
#WARNING: Extra screen received.
#The last recorded send may need an "-expect" option
#$conn1 sendWait -key <KEY> -expect "<EXPECT>"
#
```

12.11.6.2.2 Editing the sendWait Template

You can edit the commented **sendWait** template, e.g.,

```
#$conn1 sendWait -key <KEY> -expect "<EXPECT>"
```

by performing the following steps:

1. Modify the commented template:

    • Replace the KEY placeholder with the key that should be pressed (e.g., *enter*).

    • Replace the EXPECT placeholder with the appropriate character string (e.g., *MAIN MENU*).

    • Remove the pound sign (*#)* from the template statement.

    For example, the template appears as

```
#$conn1 sendWait -key <KEY> -expect "<EXPECT>"
```

    After modifying the template, the statement appears as

```
$conn1 sendWait -key enter -expect "MAIN MENU"
```

2. Delete the original **sendWait** statement, e.g.,

```
$conn1 sendWait -key enter
```

3. If desired, delete the remaining comment lines, e.g.,

```
#
#WARNING: Extra screen received.
```

12.11.6.2.3 Example of Modified Code

Before modifying the code, the Code view contains the following statements:

```
$conn1 type position "21 2" -text "ttso tksct10"
$conn1 sendWait -key enter
#
#WARNING: Extra screen received.
#The last recorded send may need an "-expect" option
#$conn1 sendWait -key <KEY> -expect "<EXPECT>"
#
```

After modifying the code, the Code view contains the following statements:

```
$conn1 type position "21 2" -text "ttso tksct10"
$conn1 sendWait -key enter -expect "MAIN MENU"
```

12.12 Using Other MYNAH Packages with the Script Builder

The MYNAH System provides you with a number of domain packages that allow you to test and develop scripts in a number of different environments. We introduced you to domain packages in Section 1.

There are no graphical objects you can use to access these domain packages other than the 3270 and asynchronous terminal emulation packages implemented as connections in the Script Builder. However, you can enter the Tcl code statements that implement these packages through the Code view of the Script Builder and you can access these with the Insert Template dialog.

See the *MYNAH System Scripting Guide* for information about using these domain packages.

12.13 Using the Log View

The Log view is a window which displays the events related to script execution. There are no graphical objects to represent these events, so they are listed here as text line-items. The amount of information displayed in this view depends on the Output Level. The default value for these output settings is contained in the MYNAH configuration file, *xmyConfig*. Figure 12-37 shows a **Run Log** view.

Figure 12-37. Log View

12.14 Reinitializing the Script Builder

You can reinitialize the Script Builder which means you are shutting down the current Script Builder Tcl interpreter and starting a new one. You might want to do this, for example, when you need to reset or remove variables.

If you do this:

- All terminal connections are cleaned up and closed

- All Tcl variable and loaded procedures disappear

- The Run Status field, the Run Log view, and Compare Counts field are cleared.

> **NOTE —** This does not clear the Script Code area.

12.15 Reviewing Script Output

If you selected the "Generate Output" option at runtime, you can look at the Output file. The Output file will contain information from all the files that the MYNAH System generated for your run. For more information about the "Script Output" window, including the contents and views, see *Section 10.9.2.1*.

To view the output, click on the **View** menu and then click on the Show Output menu option. Note that the name in the title bar will be "builder.tcl.out," followed by the date and time.

13. Format Object

The Format object (Figure 13-1) is part of the AETG System, and is only available if you license the AETG System with the MYNAH System. For information on the Format object (and all AETG objects), see the *AETG System Users Guide*.
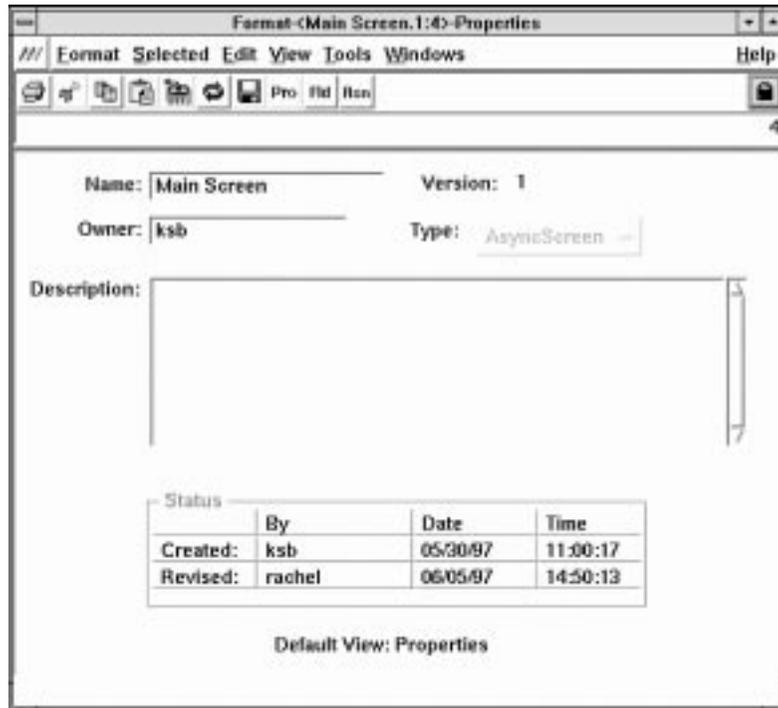
Figure 13-1. AETG Format Object Properties View

14. Examining Results and Reporting on Test Activities

A primary purpose of testing is to obtain results so that you can see if your system is performing the way it was designed to perform. The MYNAH System provides you with multiple ways to monitor test or scripting activities:

- Job Status window

- Reports

- Tests and associated Result objects.

The MYNAH System treats scripts and tests separately. A script may implement one or more tests, or a script may simply perform a useful function. In either case, the MYNAH System provides execution information for scripts through the Job Status window and through the use of Runtime objects. Runtime objects store script execution information such as start and stop times and execution status.

The MYNAH System provides Test object status information through the use of Result objects. Result objects are produced either by executing scripts that are related to Test objects or by recording a manual run of a test.

In this section we will describe:

- What a Result object is and how to use it

- How to generate reports using the MYNAH System's reporting feature

- Multiple methods for performing analysis of script execution or test result information.

14.1 How Result Objects and MYNAH Reports Help You

After planning and implementing your test plan, you will want to know how the system you are testing is performing and how your test efforts are going. These are difficult questions if you are managing a number of testers who are working full-time on your test plan. The MYNAH System provides you with two tools to help you manage this.

Result objects provide you with information about how a system performs, for a given release or test cycle, when tested. They can be associated with both automated and manual tests, and they are used in testing progress reports.

The MYNAH System helps you to manage your overall test effort with a series of reports that summarize testing efforts. You will be able to generate these reports and see at a glance how your test efforts are progressing.

14.2   Using Result Objects

One of your most important tasks will be to examine results and analyze them. The
MYNAH System supports these key activities with Result objects.

Result objects are automatically created by the MYNAH System and associated with a Test
object each time you execute a script, provided either of the following are applicable:

- There is one and only one test associated with the script

- There is one or more xmyBegin and xmyEnd statements in the script.

> **NOTE —** These statements allow you to implement more
> then one Test in a single script (see *BR 007-252-008,
> MYNAH System Scripting Guide*, for more information).

In these cases, if you executed the same test three times, the system would create three
Result objects.

Result objects contain the results of any compares done during script execution. You can
also record analysis information and enter comments with a Result object.

All executed scripts (parent or child) should exist in the database as script ojects or
MYNAH will be unable to create Result objects. Example: If you run a script which
submits a child script containing xmyBegin/xmyEnd blocks, but the child script is not in
the MYNAH database no Result object is created.

You can manually create Result objects for tests by using the **Record Manual Run** option
on the **Test** menu (see Section 14.3, "Creating Results Objects Manually"). The Result
object produced in this way is the same as a Result object that is produced by an automated
script, so all results can be included in a single report.

14.2.1   Viewing Result Objects

After you run a script, the MYNAH System will create a Result object. You can access
Result objects directly from a Runtime or Test object.

You can also access Result objects through the Database Browser. To do this, select Result
as the object type and open the Result object you want. (You may have to click on the **View**
menu and then the **Include** menu selection to obtain the specific list of Result objects you
are interested in. For example, locate all Result objects that were produced since yesterday).

> **NOTE —** You can control how many objects appear in a
> Result object using the Include dialog. See Section 5.2.2
> for information on using the Include Dialog.

Double clicking on one of the Result objects in the list will cause the system to open a
Result object Properties view similar to Figure 14-1.

Figure 14-1. Result Object Properties View.

After you become familiar with Result objects, you may want to set the default view for
Result objects to one which is more useful then the Properties view for you, Compares or
Analysis, for example. You can set the default view from the MYNAH Desktop Default
Views. Refer to *Section 3: MYNAH Basics* to do this.

14.2.2   Viewing a Result Object's Basic Properties

The Properties view provides you with the basic properties for a Result object. These include:

Test   Displays the test the result is for. This is a ruler display which means you can open the test by double clicking on it.

This item will appear on all views of the Result object.

Generated By   The person who ran the script which generated the result or the person that created the result manually.

Label   A user assigned label that a script can create. (See *BR 007-252-008, MYNAH System Scripting Guide*, for information on the xmyBegin and xmyEnd statements.)

SUT   The SUT against which the script that produced the Result object was run (or that was set for a manual run). The full path to the SUT appears here. This is a ruler display which means you can open it by double clicking on it.

Script   The Script object that produced the Result object and its ID. This is a ruler display which means you can open it by double clicking on it (this will be blank if the result was produced manually).

Run   The Runtime object for the particular script run (or manual run) that produced the Result object. Also the run status and when the run was initiated appears here. This is a ruler display which means you can open it by double clicking on it.

Default View   The Default view for the Result object set through the MYNAH Desktop Default view.

14.2.3   Using the Compares View

Scripts that call for comparisons between expected values and encountered values will produce a compares count that will appear on the Compares view of a Result object. You access this view from the **View** menu or the icon on the Toolbar. The Compares view looks like the view we show in Figure 14-2.
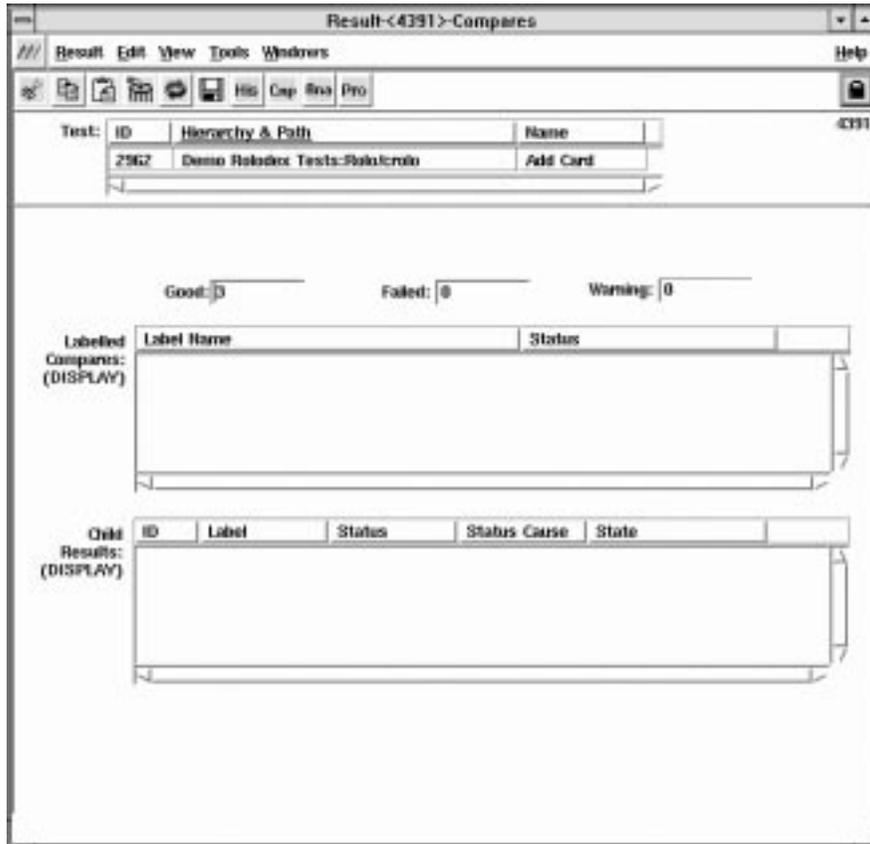
Figure 14-2. Compares View.

14.2.3.1   Examining Compare Result

The Compares view lists the result of any comparisons made during the execution of a
script. The listings are: **Good**, **Failed** and **Warning**.

Labeled Compares	Displays any labeled compares for this result (result of **xmyBegin**, **xmyEnd** and labeled **xmyCompare** statements in script).
Child Result	Displays results for child tests if this is a parent test (result of nested **xmyBegin** and **xmyEnd** statements in script).

14.2.4   Examining Analysis Information

The MYNAH System provides you with information about the results of test execution
with the Analysis view. The Analysis view (shown in Figure 14-3) provides you with basic
information about the success or failure of a test. You access this view from the **View** menu.
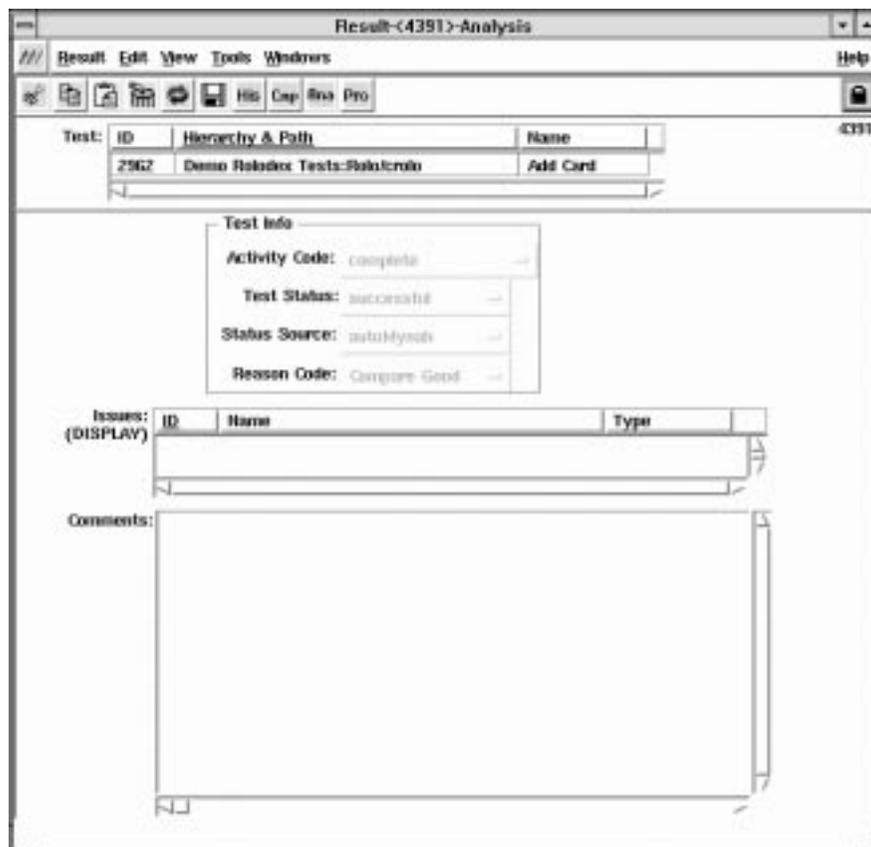
Figure 14-3. Analysis View

14.2.4.1   Viewing Analysis Information

A series of option lists display the analysis values for a test's result. If the Result object was created by an automated script then the SE (Script Engine) will set these values according to the rules described below. A person may have set these values if the Result object was created manually. In either case, a user may change the values by editing the Result object.

Activity Code   Indicates that testing is complete, or the additional activity that is required. Values depend on the **Test Status** (see next table) and the value of **Post Execution Activity Required** on the Settings view of the Test object. Values are explained in the table below.

Activity Value	Description
executionIncomplete	The test terminated abnormally.
analysisIncomplete	Set by SE because Test Status is successful, but Post Execution Activity is set to True. OR Set by SE because Test Status is unsuccessful or inconclusive. In both cases, indicates that a person needs to look further at the results.
resubmitRequired	This is never set automatically by the system. After performing analysis a person may determine that the test should be re-submitted.
complete	Set by SE because Test Status is successful, and the Post Execution Activity is set to False. OR A person has performed all necessary analysis and has determined that all work is complete. In both cases, indicates that no further work is needed.

Test Status   Records the current status of the test which produced the results. A script can set this value by using an **xmyUpdateResult** command. If the script does not use this command, the SE will set the values based on the rules described below.

Test Status	Description
inconclusive	There were one or more warning compares.
successful	There were no warning compares and no failed compares.
unsuccessful	There was a Tcl error, or one or more failed compares.

Status Source   Records the source of the **Test Status**. Values include:

Status Source	Description
autoMYNAH	The SE was the source.
autoUser	The SE was the source but the value came from an **xmyUpdateResult** command.
manualUser	A person was the source.
autoChildren	The SE was the source but the parent script had no compares, but a child script did. The SE based the Status on the compares in the child script.

Reason Code   Records the reason behind the Test Status value. The first table below contains all of the values the SE will use to explain why it set the Test Status the way it did. The second table shows additional values that a user might use when manually creating or updating a Result object. (Note that more values for Reason Code can be added by your administrator so that you can enter what makes sense for your organization).

Reason Code	Description
Ana Success	Stands for Analysis Successful. An **xmyUpdateResult** command set the Test Status to successful.
Ana Unsuccess	Stands for Analysis Unsuccessful. An **xmyUpdateResult** command set the Test Status to unsuccessful.
Ana Inconclusive	Stands for Analysis Inconclusive. An **xmyUpdateResult** command set the Test Status to inconclusive.
Script Failure	A Tcl error caused the script to abort.
Script Cancel	Script was cancelled.

Reason Code	Description
Compare Good	All compares were good.
Compare Failure	One or more compares failed.
Compare Warn	One or more compare warnings (but no failures).
Child Failed	The parent test may have been successful, but a child test had a failed compare.
Child Warn	The parent test may have been successful, but a child test had a compare warning.

The following table contains the system delivered values that are not used by the SE.

Reason Code	Description
Limbo	Meaning assigned by user organization.
Blocked	Meaning assigned by user organization. You may use this if execution is blocked by other problems in the system.
Deferred Failure	Meaning assigned by user organization. You may use this to indicate that while there is a problem, the problem will not be fixed during the current test cycle.

14.2.5   Modifying Analysis Information

If a result shows an **Activity Code** of "analysisIncomplete" or any other incomplete activity state, you may want to analyze the run and determine what needs to be done next. You may need to review the results and the Script object associated with this result to perform this analysis (See Section 14.5 for various methods for doing this).

After you complete this analysis, you can change the **Activity Code** or **Test Status** to reflect what you found. You may also want to enter comments describing what you found.

The bottom panel of the Analysis view is a free-form data entry area where you can enter comments about the results. All you have to do is:

1. Unlock the Result object.

2. Make your changes to the items in the **Test Info** area by selecting a value from one or more of the drop-down lists.

3. Position the cursor in the first free line of the **Comments** data entry area and type in your comments.

4. Click on the **Result** menu and then on the **Save** selection.

Every time you change a Result object, the system makes a record of that change. You can view the record of changes to a Result object using the **EditHistory** view.

14.2.6   Using the EditHistory View

The MYNAH System keeps a record of all the changes made to a Result object. You can see a history of these changes using the **EditHistory** view. You access this view from the **View** menu or by clicking on the **His** Toolbar icon. The system will display an **EditHistory** view similar to the one shown in Figure 14-4.
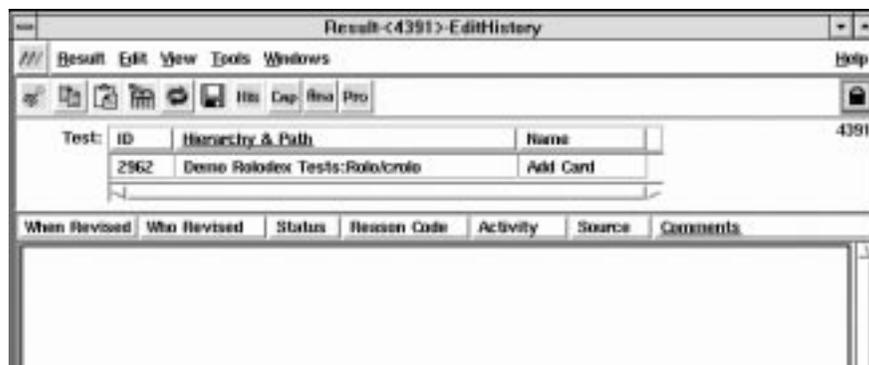
Figure 14-4. Edit History View.

This is a display only view, so you can't modify any of the information in this view. For each item in the list, the system displays: **When Revised**, **Who Revised**, **Status**, **Reason Code, Activity, Source** and **Comments.**

14.3   Creating Results Objects Manually

If the test you ran was a manual test, turning off and on your terminal for example, how would you record the results? The MYNAH System provides a way for you to do this for Test objects with the **Record Manual Run** menu option on the **Test** menu. The **Record Manual Run** menu option displays a dialog that allows you to enter results and create a Runtime object for the manual test.

When you access the **Record Manual Run** dialog from the **Test** menu of a Test object, the system displays the dialog similar to the one shown in Figure 14-5 except the dialog in the figure has been filled in already.

Figure 14-5. Record Manual Run Dialog

MYNAH System Users Guide   BR 007-252-006
Examining Results and Reporting on Test Activities   **Issue 3, October 1997**
Release 5.2

The Manual Run window lists information about the manual test run. This information includes both run information and result information:

Run Info Box

Params. Used	Names any parameters used for the test.
Run Status	Displays the current status of the test. Values include: **completed** and **aborted.**
Run Summary	Free-form data entry area where you can enter comments about the test run.
SUT	Identifies the SUT the results are for. The system pre-populates this field with the SUT in your Mynah Desktop Preferences view. You may not change it here. If the pre-populated SUT is not what you want you must Cancel, change the SUT in the Mynah Desktop Preferences view, then do **Record Manual Run** again.

Results Box

Activity	Displays the activities that still need to be accomplished. Values include: **complete**, **executionIncomplete**, **analysisIncomplete**, and **resubmitRequired**. Refer to Section 14.2.4.1 for a complete explanation of these values.
Status	Identifies the status of the Result. Values include **successful**, **unsuccessful**, and **inconclusive**. Refer to Section 14.2.4.1 for a complete explanation of these values.
Reason Code	Displays the reason for the status. Refer to Section 14.2.4.1 for an explanation of the values.
Comments	Free-form data entry area where you can enter any comments.

14.3.1 Example Manual Result Object

We will illustrate entering manual results using our example of turning a work station on and off.

1. Open the Test object that documents the test.

2. Click on the **Test** menu and then on the **Record Manual Run** menu option**.**

   — The system displays the Manual Run dialog.

3. Position the pointer in the Params. Used data entry area and type in any parameters used with the Test. (we noted the we used `none`.)

4. Click on the **Run Status** option list and select a status for the test run. (We chose `completed`.)

5. Position the pointer in the **Run Summary** data entry area and type in any comments you have. (we entered comments)

6. Select an activity code from the **Activity** option list. (We chose **analysisIncomplete**).

7. Select a status code from the **Status** option list (We chose `inconclusive`).

8. Select a reason code from the **Reason** option list. (We chose `AnaInconclusive`).

9. Position the pointer in the Comments data entry area and type in any comments you have. (We typed in comments about the analysis).

— Our dialog looked like Figure 14-5 when we were through.

We can come back later, after completing our analysis, and update this result. We can open it from the test to which it is related, from the Test object Result view.

10. Click **OK**.

— After we clicked **OK**, the system created a Runtime object and a Result object. You can see the new Result object from the Test object Result view after you do a refresh.

14.4   Using The MYNAH System's Report Feature

The MYNAH System provides you with a number of reports that will help you keep track of your testing efforts. The reports available with the MYNAH System are:

- Testing Progress - indicates how far testing has progressed for a SUT.

- Requirements Traceability - indicates whether or not a particular requirement or set of requirements has been met for a particular SUT.

- Test Specifications - a formatted print of a set of Test objects.

- Issues - a formatted print of a set of Issue objects.

You access the report feature from the Desktop Mynah menu by selecting **Reports**. When you do this, the system displays the **Report List** view. See Figure 14-6.
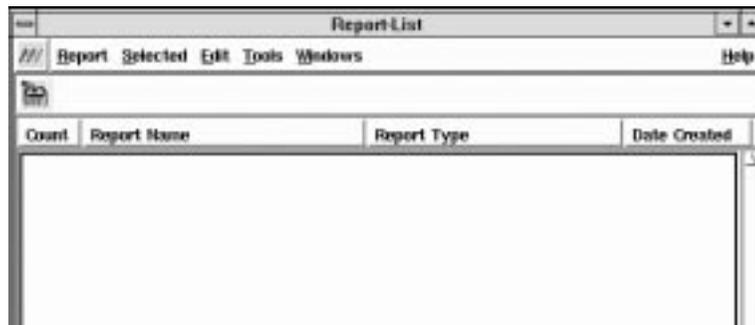
Figure 14-6. Report List View

The **Report List** view shows all the reports you have created and saved. The display in the Figure is empty because we have not yet created any reports. For each report, the system displays:

Count   A number assigned by the system when the report was created.

Report Name   Name entered by a user when the report was created.

Report Type   The type of report: Issues, Test Specifications, Requirements Traceability, Testing Progress

Date Created   Date the report was created.

14.4.1   Creating Reports

To create one of these reports:

1. Click on the **Selected** menu and then on the **New** option.

   — The system will display the New Report dialog shown in Figure 14-7.

Figure 14-7. New Report Dialog

2. Select the type of report you want to generate from the **Report Type** option list, e.g., Testing Progress.

3. Position the pointer in the **Name** data entry area and type in a name for the report.

4. Click **OK**.

   — The system will display the Report Specification window shown in Figure 14-8.

Figure 14-8. Report Specification Window.

14.4.1.1   Report Specifications

The Report Specification window allows you to specify how much detail will be in the report and the output device to which it will be sent. Information on the Report Specification window includes:

Name	The name you give the report.
Type	Displays the report type (cannot be changed here).
Output Format	Specifies the level of detail in a report. Values include: **Detailed** or **Summary**.
Destination	Specifies where the report will go. You can specify one or all of these. Values include: **Screen**, **Printer**, **File**. If you specify **Printer**, you can enter a printer or accept the preference printer. If you specify a file, you must name the file.
Weight Factor	This applies for Test Progress Report only. The default is **None**. If you choose **Importance** or **Effort**, the value from the **Ranking Info** box on the Test object Settings view will be used as a multiplying factor for the tests included in the report.
Selection Criteria	An area where you can enter queries using the Include dialog. The system uses the criteria entered to select objects for the report.
SUT Associated with Results	The SUT for which the report will be produced. This is a ruler area so you can copy information from it or paste information into it. It is pre-populated with the SUT from your Mynah Desktop Preferences view.

14.4.1.2   Entering Report Specifications and Running Reports

We will use an example to illustrate how to enter report specifications. For our example, we will request that a summary Testing Progress Report be sent to a window. The Test Specification window already displays the **Name** and **Type** of the report and we will accept the pre-populated SUT. We will need to specify the **Output Format, Destination** and **Query**. To do this:

1. Click on the **Summary** radio button in the **Output Control Options** area.

2. Click on the **Window** radio button in the **Default Destination** area.

3. Specify the Query. If we ran the report without specifying a query, it would include all of the tests in the database. We use the query to select a subset of tests to report on. You do this by entering queries into the **Selection Criteria-Query** area.

For our example, let's generate a Testing Progress Report that reports on the crolo tests. To do this on the Report Specification window:

Click on the **Edit** push button.

— The system will display the Include dialog shown in Figure 14-9. This should be familiar to you. It is the same dialog used to Include objects in the Database Browser display. See Section 5.2.2.
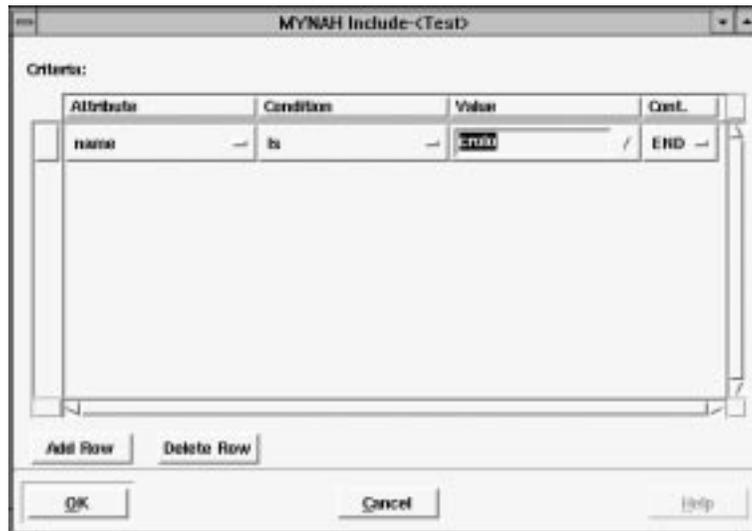
Figure 14-9. Report Query

4. Click on the **Add Row** push button.

5. Select an **Attribute**, **Condition**, **Value** and **Cont**. We selected **Name**, **is**, typed in `crolo` and accepted **END**. This will include the crolo test and all of it's descendent tests.

6. Click **OK**.

— When we were finished entering our criteria, the Specification dialog looked like Figure 14-10. Note that the Query field now has a value.

Figure 14-10. Report Specifications Window with Query

Now we are ready to run the report. This report will include resulting information relative to the populated SUT for the **crolo** tests.

7. Click on the **Report** menu and then on the **Run** menu option or click on the **Toolbar** icon.

— The system generates the report and displays a window like the one in Figure 14-11.
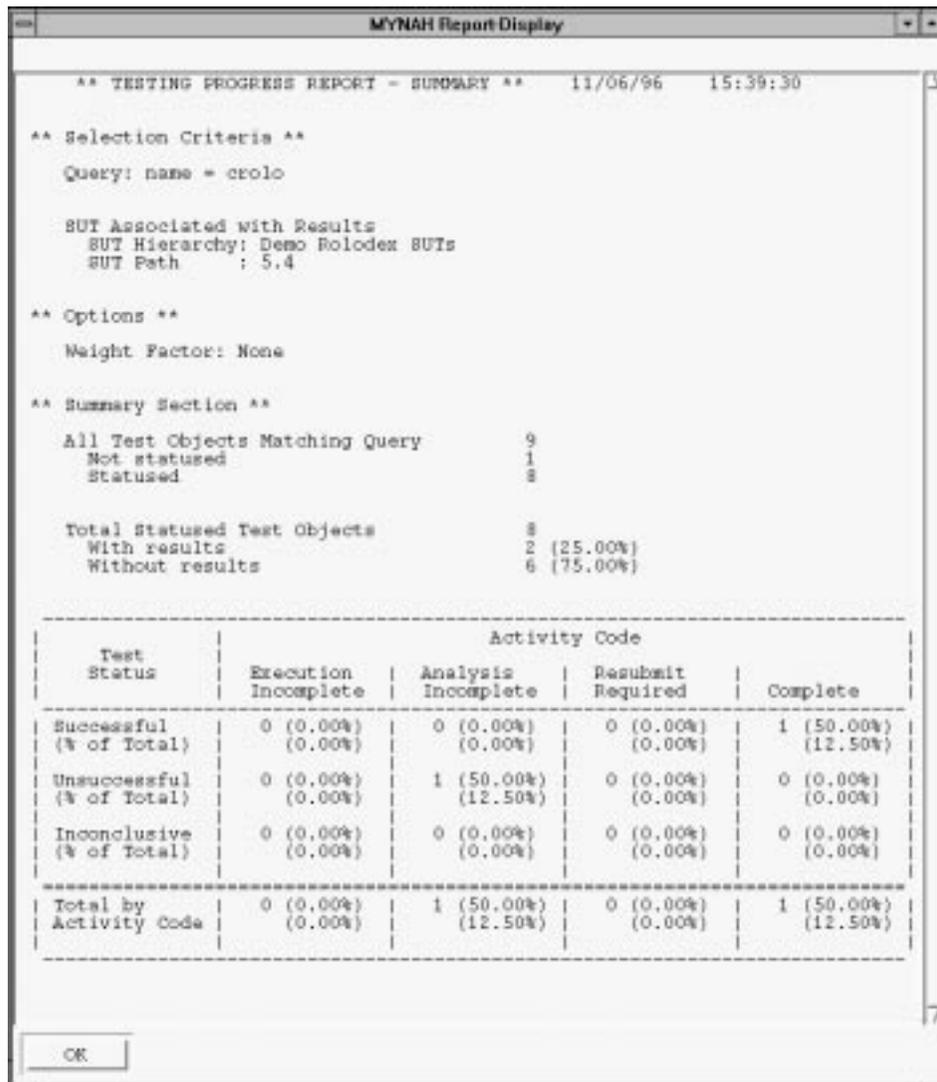
Figure 14-11. Testing Progress Report Window Output

The report output that is produced is explained in detail in Section 14.4.4.

You can generate any of the report types in the same way we generated the Testing Progress Report. The primary difference is the type of report you select from the **Report Type** option list on the New Report dialog.

14.4.2   Saving a Report

If you would like to save the report so that it can be run at any time:

- Click on the **Report** menu **Save** option.

  — The system will add your new report to the list in the Report List view as shown in Figure 14-12.
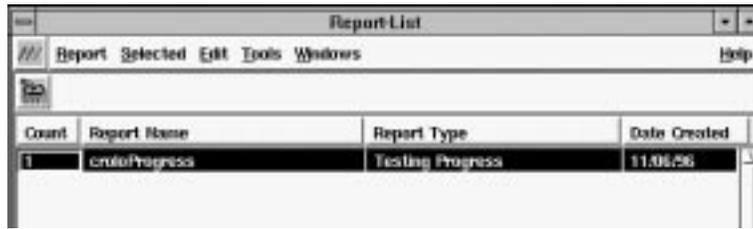
Figure 14-12. Report List View Populated

14.4.3   Deleting a Report

To delete an existing report:

1. In the Report List view, select the report row item that you want to delete.

2. Choose **Selected** menu **Delete** option (or click on the trash can).

   — The system will delete the report from the list

14.4.4   Report Input and Output

This section provides an explanation of the input parameters, processing and output content for each of the available reports.

14.4.4.1   Testing Progress

The Testing Progress report provides result information for a specific SUT for a set of tests. For example, you may wish to know how you are progressing with the first test cycle of Release 1.0. A specific SUT must always be provided as input to this report. This SUT is used to determine which Result objects will be used to determine the status of the included tests.

If you do not provide any selection query information then the report will include all the tests that are in the MYNAH database. However, you can use the selection query to specify a specific subset of tests that you are interested in. The hierarchy descendents of the tests that you choose (e.g., the children) will be automatically included.

When you run this report the system:

1. Determines the list of tests that match your selection criteria.

2. Determines, for each test in the list, if the test is Statused or not.

3. Determines, for each test in the list that is Statused, if there are any results for the specified SUT:

   — If there are no results, the test will be reported as "Result (missing)".
   — If there are one or more results, the most recent result will be used in the output.

The selection query for the report in Figure 14-13 was: **Test Name = crolo**. This was all that was needed to obtain a report on all of the children of the **crolo** Test object.

The output in Figure 14-13 provide the summary information, also we provide the meaning:

Total 10	10 tests matched the query
Nonstatused 4	4 of those tests have the Statused attribute on the Test Properties view not set, so these tests will not be counted further.
Statused 6	6 of those tests have the Statused attribute on the Test Properties view set.
Result (missing) 3	3 statused tests did not have any results for the specified SUT.
Result (found) 3	3 statused tests did have results for the specified SUT.

Summary status information is then given in a table for the results that were found. The top percentage line for each category provides a percentage of the results found. The bottom percentage line provides the percentage of the total number of statused tests.

Detailed information about the individual tests is then given (if Detail was requested). Finally, a Legend is printed that explains the codes used in the detailed information.

```
 ** Testing Progress Report - Detail **    06/08/96    19:35:49

** Selection Criteria **
    Query: name = crolo
    SUT Associated with Results
      SUT Hierarchy: Demo Rolodex SUTs
      SUT Path     : 5.3/First

** Options **
    Weight Status By Measure: NONE

** Summary Section **
    All Test Objects matching query      7
      Not statused                       1
      Statused                           6

    Total Statused Test Objects          6
      With results                       3 (50.00%)
      Without results                    3 (50.00%)

 ------------------------------------------------------------------------
 |              |                  Activity Code                        |
 |    Test      |                                                       |
 |   Status     | Execution    | Analysis     | Resubmit    |           |
 |              | Incomplete   | Incomplete   | Required    | Complete  |
 ------------------------------------------------------------------------
 | Successful   | 0 (00.00%)   | 0 (00.00%)   | 0 (00.00%)  | 1 (33.33%)|
 | (% of Total) |   (00.00%)   |   (00.00%)   |   (00.00%)  |   (16.66%)|
 |              |              |              |             |           |
 | Unuccessful  | 1 (33.33%)   | 0 (00.00%)   | 0 (00.00%)  | 0 (00.00%)|
 | (% of Total) |   (16.66%)   |   (00.00%)   |   (00.00%)  |   (00.00%)|
 |              |              |              |             |           |
 | Inconclusive | 0 (00.00%)   | 1 (33.33%    | 0 (00.00%)  | 0 (00.00%)|
 | (% of Total) |   (00.00%)   |   (16.66%)   |   (00.00%)  |   (00.00%)|
 |              |              |              |             |           |
 ========================================================================
 | Total by     | 1 (33.33%)   | 1 (33.33%)   | 0 (0.00%)   | 1 (33.33%)|
 | Activity Cd  |   (16.66%)   |   (16.66%)   |   (0.00%)   |   16.66%) |
 |              |              |              |             |           |
 ------------------------------------------------------------------------

** Detailed Section **

 Test Hierarchy = Demo Rolodex Tests
 Test Path =  Rolo/crolo
 Name            Unit Who      When              Src Act Rsn Stat
 ----------------------------------------------------------------------
 Add Card         1    hsb     06/08/96 09:46:38  aM   c   CG  S
 Delete Card      1    hsb     06/08/96 09:46:29  aM   c   CG  S
 Filters          1
 Large Database   1
 Positioning      1    hsb     06/08/96 09:46:24  aM   c   CG  S
 options          1

** Legend **

 Status Source (Src)
     aM : autoMynah
```

Figure 14-13. Testing Progress Report Output

14.4.4.2   Requirements Traceability

The Requirements Traceability report provides result information for a specific SUT for a set of requirements. For example, you may wish to know how many of a set of requirements have been met during the first test cycle of Release 1.0. A specific SUT must always be provided as input to this report. This SUT is used to determine which Result objects will be used to determine the status of the tests that are associated with the included requirements.

The selection query for the report in Figure 14-14 was: requirements that have Type = **crolo**. The query matched eight requirements.

When the report is run the system:

1. Determines the set of requirements that match the selection query.

2. Determines, for each requirement, the test(s) that are associated with that requirement. If there are no tests associated with a requirement "No Test objects found" will be reported. This indicates that the requirement is not yet covered by a test.

3. Determines, the Status of those tests (by looking for results for the specified SUT). If there are no results found "Not Run" will be reported.

4. Uses all of this information to determine the overall status of each requirement. If there is only one test for the requirement, the status of the requirement will be equal to the status of the test. If there is more than one test for a requirement, the system will determine the status of the requirement by examining the status of each of the tests. If any of the tests are 'not run' then the status of the requirement will be 'not run'. If all are run, but one or more of the tests are 'unsuccessful' then the requirement is 'unsuccessful'.

Summary information is provided at the bottom of the report.

```
** Requirements Traceability Report - Detail **    06/09/96  10:28:15
** Selection Criteria **
   Query: type = CROLO
   SUT Associated with Results
     SUT Hierarchy: Demo Rolodex SUTs
     SUT Path     : 5.3/First

** Detailed Section **

Requirement Type: CROLO

 Name: R0-Options (ID: 20)
   Status:       No Test objects found
   ----------------------------------------------------------

 Name: R1-Add (ID: 21)
   Test name                              ID     Status
   ----------------------------------------------------------
   Add Card                               2401   successful

   Status : Successful
   ----------------------------------------------------------

 Name: R2-Delete (ID: 22)
   Test name                              ID     Status
   ----------------------------------------------------------
   Delete Card                            2402   not run

   Status : Not Run
   ----------------------------------------------------------

 Name: R3-Redraw (ID: 23)
   Status:       No Test objects found
   ----------------------------------------------------------

 Name: R4-Keys (ID: 24)
   Test name                              ID     Status
   ----------------------------------------------------------
   Positioning                            2403   unsuccessful

   Status : Unsuccessful
   ----------------------------------------------------------

 Name: R5-Move (ID: 25)
   Status:       No Test objects found
   ----------------------------------------------------------

 Name: R6-Filter (ID: 26)
   Test name                              ID     Status
   ----------------------------------------------------------
   Filters                                2404   inconclusive

   Status : Inconclusive
   ----------------------------------------------------------

 Name: R7-Clear Filter (ID: 27)
   Test name                              ID     Status
   ----------------------------------------------------------
   Filters                                2404   inconclusive

   Status : Inconclusive
   ----------------------------------------------------------

 ** Summary Section **

    Requirement Objects matching query      8
     Successful                             1
     Inconclusive                           2
     Not Run                                1
     Unsuccessful                           1
     No Test object
```

Figure 14-14. Requirements Traceability Report Output

14.4.4.3   Issue

The Issue report prints the contents of the selected issues. The selection query can be based on: Status, Type, Owner, Associated SUT id, and Created By attributes.

```
 ** Issues Report - Detailed **     06/09/96    19:28:56

** Selection Criteria **

    Status     : Open
    Type       : General
    Owner      : hsb

** Detailed Section **

  Name : disk space                            ID : 5
  (External Name : ZA-96-123456)

   Status: Open     Type: General    Owner: hsb      Created by: hsb
   Description:
       Need more disk space to hold a large rolo database.

   ----------------------------------------------------------------
  Name : reqmts coverage                       ID : 516
  (External Name : none)

   Status: Open     Type: General    Owner: hsb      Created by: hsb
   Description:
       Discovered function not covered in test plan.

   ----------------------------------------------------------------
** Summary Section **

   Total Issue objects..........................2
```

Figure 14-15. Issue Report Output

14.4.4.4   Test Specification

The Test Specification report prints either summary or detailed information about the selected tests. The summary information includes: Type, Owner, Created By, and Description attributes. The detailed information adds the Step List to the output.

The selection query determines which tests will be reported. All descendents of selected tests will also be reported.

Figure 14-16 shows sample output from this type of report.

```
** Test Specifications Report – Detailed **     06/10/96     15:21:49

** Selection Criteria **

  Query :  name = "Add Card"

 ** Detailed Section **

  Test Path: Rolo/crolo

  Name: Add Card                                 ID: 2401

   Type: Test Case          Owner: rachel              Created by: rachel
    Description  :
      Verify the user ability to add one or more cards to the database.

   Step No: 1      Name: login                   ID: 549
     Type: initialization
     Description:
       rlogin $machineName using $password
     Expected Outcome:
       unix prompt on $machine

   Step No: 2      Name: setup database          ID: 550
     Type: initialization
     Description:
       type 'cp $XMYHOME/demo/data/rolo.db /tmp/.roloAdd'
     Expected Outcome:
       unix prompt

   Step No: 3      Name: start application       ID: 551
     Type: initialization
     Description:
       type 'crolo  -f /tmp/.roloAdd'
     Expected Outcome:
       database displayed

     . . .



  ---------------------------------------------------------------------

 ** Summary Section **

    Total Test objects matching the query = 1
```

Figure 14-16. Test Specification Report Output

14.4.5   Creating Command Line Report Files

You can run reports that you have created any time through the GUI. However, you can also run report commands from the CLUI (Command Line User Interface). The CLUI is particularly useful if you wish to have a report run on a regular basis. You can put the command in a file and then that file can be run with the UNIX cron facility.

Complete information about these commands can be found in *Section 17*. However, an easy method for creating these command line files is by using the Report specification window, **Report menu Create Command File** option. This option will prompt you for a file name, and then create the appropriate CLUI report command in that file.

It is recommended that if you wish to use the CLUI report feature, that you create the command files in this manner.

14.5   Methods and Procedures For Performing Analysis

There are numerous ways to begin the process of looking at either script execution status or test results. This section provides an outline for some of these methods.

14.5.1   Script Execution Status

For all scripts that have been run in the Background Execution Environment you can start looking with either the Job Status window, the Database Browser window or from an opened Script object.

14.5.1.1   Starting with Job Status

The Job Status window provides you with dynamically updated information about the execution status of your scripts that are running, or have run, in the BEE (Background Execution Environment). You can request this window at any time by using the **Tools** menu.

The row items in the Job Status window represent the Runtime objects that are created by the system whenever you run a script in the background. You can open any one of these by double clicking on the row item. From an opened Runtime object you click on **SHOW** button to get to the Script object or the file output that the script produced.

14.5.1.2 Starting with the Database Browser

You can use the Database Browser to select the Runtime objects that you might be interested in. For example, you might wish to determine what scripts ran yesterday. You would make this selection on the View Include dialog.

Once the Runtimes that you are interested in are presented, you can open any one of them by double clicking on the row item and on the **SHOW** button.

14.5.1.3 Starting with a Script

From an opened Script object you can see when a script has been executed and what the termination status was by switching to the **Run History** view. The row items in this view represent Runtime objects. Any Runtime object can be opened by double clicking on it.

If you need to analyze the execution of the script, from the opened Runtime object you can get to the file output that was produced, by the execution of the script, by clicking on the **SHOW** button.

14.5.2 Test Result Status

You may need to analyze inconclusive or unsuccessful test results that were produced automatically by automated scripts.

14.5.2.1 Starting with a Test

If you have a Test object already opened and would like to check on the latest result for that test, simply change view to the **Result** view. A history of all results is listed. You may open the one you are interested in by double clicking on the row item.

If this result was produced by running an automated script then you may wish to look at the detailed information that was produced. From the opened Result object, you can get to Runtime object (from the Properties view) that is associated with the result.

From the Runtime object you can get to:

- The Script that produced the result (if any.)
- The script output directory (you do this by clicking on the **SHOW** button.)

14.5.2.2   Starting with the Database Browser

You can use the Database Browser to select the Result objects that you might be interested in. For example, you might wish to determine what results were produced yesterday. You would make this selection on the View Include dialog.

Once the results that you are interested in are presented you can open any one of them by double clicking on the row item.

From an opened Result object you can get to:

- The Runtime object. From the Runtime you can get to the Script object, or the file output that the script produced. (you do this by clicking on the **SHOW** button.)

14.5.2.3   Starting with a Report

You can run a Test Progress report to determine which test results need to be examined. The tests on the report that show either of the following:

- Status of "Inconclusive"

- Activity State of "Analysis Incomplete".

need to be analyzed further.

Once you have these tests identified you can use the "Starting with a Test" steps to perform the analysis.

15. Task Automation with the MYNAH System

So far in this manual we have described the test related features of the MYNAH System, but the MYNAH System is more than just a test system. The Tcl scripting language and the MYNAH extensions to Tcl give you a powerful tool for Task-automation. By task-automation we mean routine repetitive tasks that you may have to do. For example, downloading data from a legacy system or data verification.

You still can do all your scripting through the Script Builder but there is a version of the Script Builder you can use which operates separately from the MYNAH GUI as a whole. This version is called the Standalone Script Builder. You can use the complete GUI if you like, but the complete GUI requires a database. If you wish to only create script code in UNIX files and execute those scripts through the CLUI, then you do not need a database or the complete GUI.

The Standalone Script Builder has all the features of the Script Builder you access through the GUI, but it uses fewer system resources and comes up quicker then the complete GUI.

15.1   Starting the Standalone Builder for Task-Automation

To start the Script Builder without the complete GUI, type:

```
xmyRunMynah -b
```

The **-b** option tells the system to start the Script Builder only.

Figure 15-1 shows the Standalone Script Builder code view.
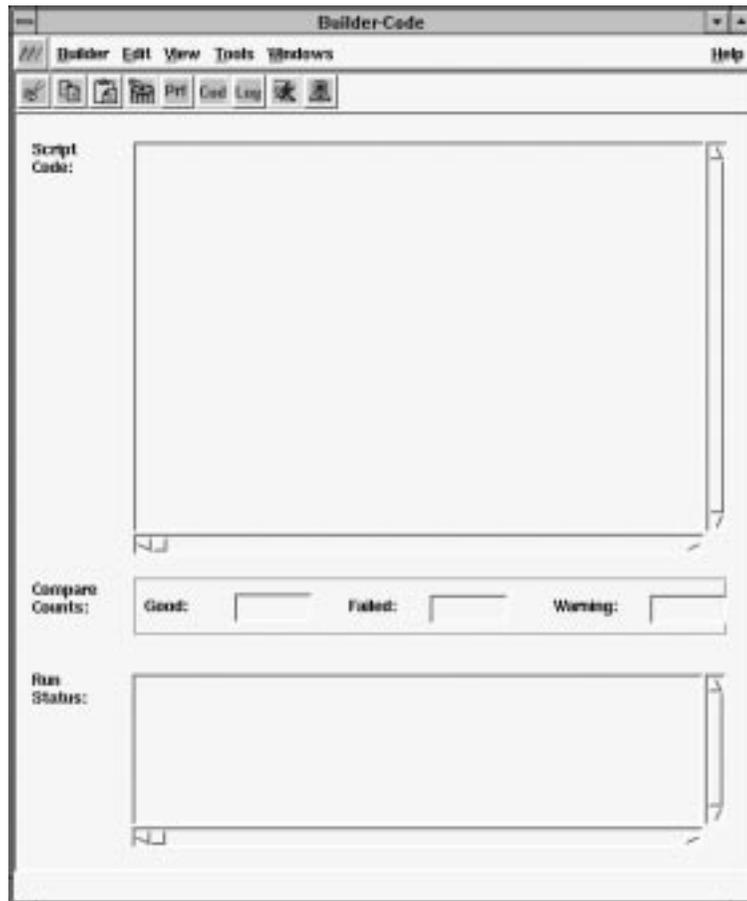
Figure 15-1. Script Builder for Task Automation

15.2   Using the Script Builder for Task Automation

As you can see the Standalone Script Builder appears the same as the Script Builder you used to develop test scripts. The major difference between this Script Builder view and the one you access through the GUI, is this one has no connection to the MYNAH database. This means that you cannot:

- Import a script using the **Insert Script** menu selection found on the **Edit** menu.

- Save to a MYNAH Script object using the **Save Code** selection on the **Builder** menu. You can however save any code you construct here to a UNIX Sfile by using the **Save Code To File** menu option.

Refer to Section 12: Developing Script Code with the Script Builder Feature for information about using this feature. Also, Refer to *BR 007-252-008, MYNAH System Scripting Guide*, for a complete guide to MYNAH Tcl and its use.

16. Understanding the Background Execution Environment (BEE)

The CLUI provides access to the Background Execution Environment (BEE). The purpose of the BEE is to allow you to run many scripts unattended in parallel.

In this section we will describe the BEE so you can better appreciate its power.

16.1 Description of the BEE

When you first use the MYNAH System, you will probably be satisfied running scripts interactively in the GUI. It is easy, convenient, and gets the job done. But it does have its limitations, especially when the number of scripts you want to run grows.

When you want to run a number of scripts simultaneously, you need the BEE. The default BEE is made up of a default SD which is configured to use a default SE group that contains 3 SEs. (The number of SDs and SEs can be changed by your system administrator based on the capacity of the platform you use at your site). The relationship between SDs and SE Groups is depicted in Figure 16-1.
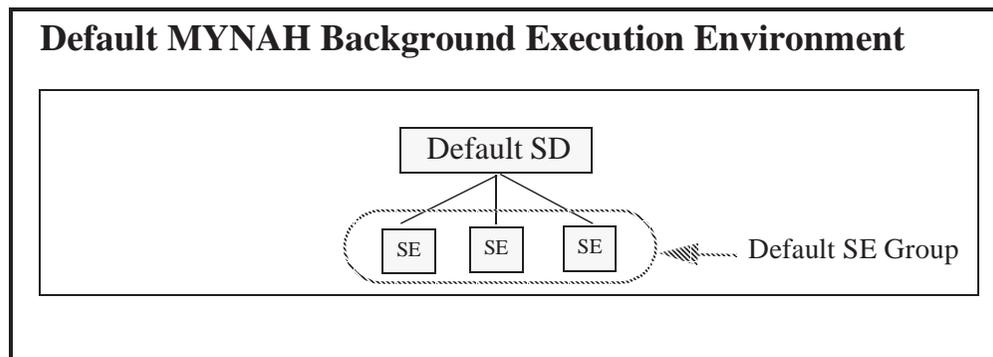
Figure 16-1. Background Execution Environment

16.1.1 How the BEE Works

For example, let's say there are 30 scripts and you submit them to the BEE. The SD will schedule the first three scripts on the 3 available SEs and will queue the remaining 27 requests. When one script finishes and an SE becomes available, the SD will 'dispatch' another script to the available SE. In this way the SD, in combination with the SEs, is providing concurrency control (controlling the number of scripts running simultaneously) on the number of executing scripts.

You might ask why not just use 30 SEs to run the 30 scripts. There are two reasons why this might not be feasible. First, the System Under Test (SUT) may not be able to handle 30

concurrent scripts. Second, the platform on which MYNAH is running may not be able to handle 30 concurrently running SEs. Your system administrator should consider these two factors when determining the ideal number of SEs in the default SE group.

You can submit scripts to the BEE from either the CLUI or the GUI.

16.1.2   How Script Engine Groups Make Testing More Efficient

Suppose that all 30 scripts have to log into the application that is being tested. One way to do this is to have the SEs remain logged into the application so that each new script starts at the application's main screen. You will be saving the time that it takes for each of the 30 scripts to log on and disconnect from the SUT.

You can do this using the Connection Only mode for the SEs that are in the default group that your system administrator can set by changing a value in the configuration file. This will provide you with an even more efficient way to execute the 30 scripts with the BEE, and you can do this with one default SE Group.

This methodology of SDs and SE Groups provides you with other efficiencies as well. Let's make our example even more sophisticated. Let's suppose that we want to expand the scope of testing to include validating data in a second application on the SUT. We will need new scripts to do this and these new scripts will log into a different application than the first 30. We might want to create a new SE group for this.

We will use the new group to maintain connections to the new application. This new group will also be run in the Connection Only mode. You or your system administrator need to configure this new group and give it a name. Lets call it SEgroupApp2. We will also need an SE Group for the first 30 scripts which we can call SEgroupApp1.
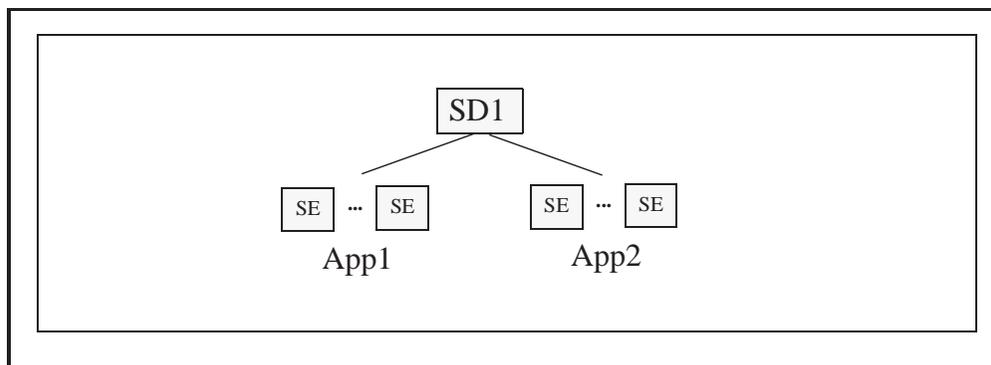
Figure 16-2. Adding SE Groups

Let us assume you create 20 scripts to test the new application. These 20 scripts will run on SEgroupApp2. You now have 50 scripts to run on the SUT and you can run all these scripts using the BEE. The SD will handle dispatching the scripts to the correct SE group and will

keep all SEs in both groups busy. You will have to specify an SE group when you submit the scripts to an SD unless you defined the scripts in the database with a default group setting.

16.1.3   Using Multiple Script Dispatchers

So far in describing the capabilities of the BEE, we have assumed that there is one SUT we want to test. But suppose there are two. The MYNAH System can be configured to have two SDs which means we will have one SD to test each of the SUTs. These SDs will provide concurrency control for all scripts that are submitted to them. So we can submit a large number of scripts to one System Under Test, and at the same time submit another set of scripts to interact with the second SUT. Figure 16-3 illustrates this.
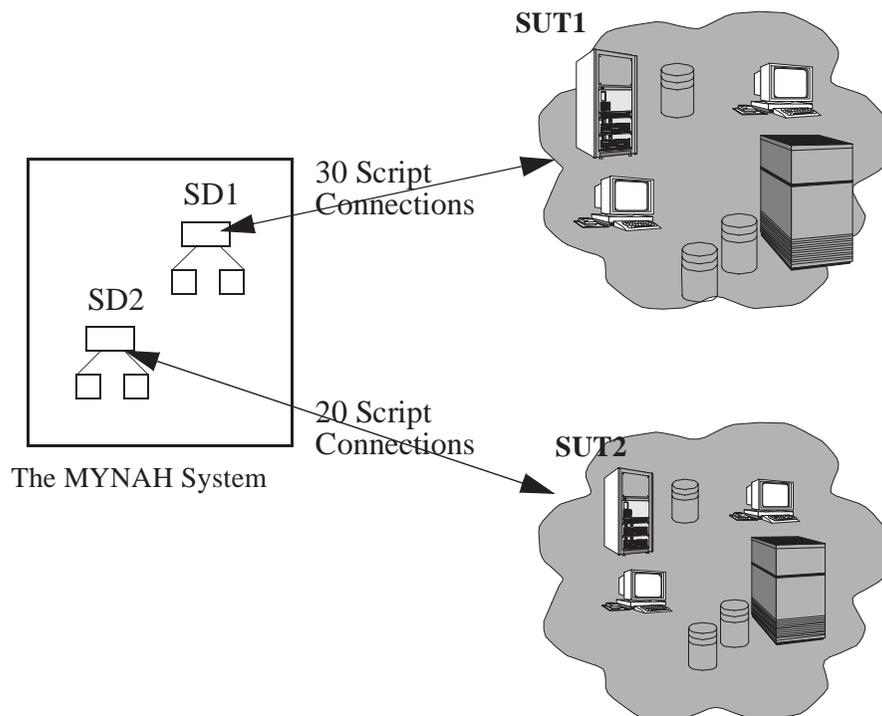
Figure 16-3. Testing Two SUTs Simultaneously with the BEE

We have tried to show you just how flexible and powerful the MYNAH Background Execution Environment can be. As you gain experience using the new MYNAH Test System, you will find that the BEE will increase your test productivity.

17. Using The Command Line User Interface (CLUI)

Many of you will want to do your testing or general scripting using the UNIX command line. The MYNAH System accommodates your needs with a Command Line User Interface — what we call the CLUI.

The CLUI consists of a set of commands which allow you to perform scripting-related tasks from the command line. You can accomplish both user and administrator tasks with the CLUI, but here we will only discuss the user capabilities of the CLUI. If you want information about using the administrator features of the CLUI, please refer to the *MYNAH System Administration Guide*.

The CLUI provides a command-line version of the MYNAH Tcl shell, as well as access to the Background Execution Environment (BEE). Refer to the previous section of this Guide for an explanation about the BEE. The CLUI also provides a method of generating reports from the database, as well as a way to create database script objects from the command line.

In this section we will:

- Introduce you briefly to the command-line MYNAH Tcl shell.

- Explain how to use the **xmyCmd** command, including how to use "sub-commands" to accomplish your work.

- Discuss **xmyCreateScriptObject**, the tool with which you can create database script objects.

- Describe how to use the report-generation facility of the CLUI.

17.1   The MYNAH Tcl Shell

The MYNAH Tcl Shell provides the user with an interactive Tcl shell with MYNAH extensions. It is invoked by the command.

```
xmytclsh
```

When you invoke **xmytclsh**, you'll get a Tcl prompt (the character ">"), and you can start typing in MYNAH-extended Tcl commands, just as if you were writing a MYNAH script to be run on a Script Engine (SE) later on. The difference is that each command is executed right away in **xmytclsh.**

> **NOTE —** When you run code using **xmytclsh**, no database updates occur and no MYNAH script output files are generated.

17.2   Using the xmyCmd Command

The **xmyCmd** command is the primary command you will use to interact with the MYNAH Test System through the CLUI. It interacts with the MYNAH System through the BEE. It takes a number of arguments that allow you to run scripts, find the status of scripts that are running, operate on running scripts, and find the status of SDs and SEs. The first argument it takes is the sub-command which tells **xmyCmd** which function you want to perform.

The basic syntax of **xmyCmd** is

```
xmyCmd <sub-command> args
```

where <sub-command> is the sub-command which will accomplish the requested operation. Sub-commands themselves take arguments and options. We will explain these below when we explain the sub-commands.

17.2.1   Getting Help with CLUI Commands

You can see a list of available CLUI sub-commands, together with a brief description of each one, by typing

```
xmyCmd -h
```

In addition, if you type in any sub-command with a **-h** option after it, the system will display a usage message that lists the options available for that sub-command and gives a brief explanation of each one. For example, to request help for the sub-command **cancel** type:

```
xmyCmd cancel -h
```

The **-H** (capital 'H') option to the **submit** sub-command gives an even more detailed explanation of the options. This option is only available for the **submit** sub-command.

17.2.2   CLUI Environment Basics

When you use the options for the sub-commands, you need to understand something about the settings in your environment. This will help you decide which options to use to send your command to the proper environment (for example, the Script Dispatcher that you need). This is particularly important if you are overriding the default settings for the Background Execution Environment (BEE). For example, you might want to submit scripts to an SD other than the default one defined in the MYNAH configuration file.

For example, suppose the default Script Dispatcher (SD) for your environment is **SD1** (as set in a central configuration file, with no other variables set).

- You determine that you need to submit a script for execution to Script Dispatcher **SD2**.

- After you submit your script to **SD2**, you determine that you need to pause that script's execution.

- To pause your script, you must specify **SD2** as the Script Dispatcher in the pause command.

The follow subsection explains how you could use environment variables or command-line options to override the default settings.

17.2.2.1   Order of Precedence

When you execute a CLUI command, the MYNAH System determines which environment settings to use to process your command. The settings are based on an order of precedence.

For example, suppose all of the following items are set for the **SD**:

- The default SD, as specified in the MYNAH configuration file, is **SD1**.

- The environment variable XMYSD is set to **SD2**.

- On the CLUI command line, you specified **SD3**, via the **-d** option.

- The database script object contains a value of SD4 in the 'ScriptDispatcher" field.

How does the MYNAH System know which SD to use for your command? The MYNAH System determines the SD based on the order of precedence for the SD. So, for the preceding example, based on the rules listed below, MYNAH will use **SD4**, the value from the CLUI command line.

The following list summarizes the order of precedence for the items that you can control in your commands.

— *Script Dispatcher:*

1. Command-line **-d** option takes precedence.

2. If that is missing, value of the **XMYSD** environment variable is used.

3. If that variable is not set, the **SD** associated with the script object in the database is used.

> **NOTE —** If the XMYSD variable is not set, then for the **submit** sub-command only, the next place CLUI looks to determine the SD to **submit** to, is the database (unless you are running in cloaked mode, i.e., using the **-c** option to the **submit** sub-command). In other words, CLUI submits the script to the SD associated with the script's *script-object* in the database. For sub-commands other than **submit** (e.g., cancel, info), CLUI skips this step -- i.e., does not consult the database.

4. If none of the above are set, the default **SD** from the MYNAH configuration file is used.

— *Script Engine:*

1. Command-line **-e** option takes precedence.

2. If that is missing, value of XMYSEGROUP environment variable is used.

3. If that variable is not set, the SE group associated with the script object in the database is used.

> **NOTE —** If the XMYSE variable is not set, then for the **submit** sub-command only, the next place CLUI looks to determine the SE to **submit** to, is the database (unless you are running in cloaked mode, i.e., using the **-c** option to the **submit** sub-command). In other words, CLUI submits the script to the SE associated with the script's *script-object* in the database. For sub-commands other than **submit** (e.g., cancel, info), CLUI skips this step -- i.e., does not consult the database.

4. If none of the above are set, the default SE group for the SD that it gets submitted to is used (as specified in the MYNAH configuration file).

> **NOTE —** If XMYSEGROUP is set so as to override the SD's default SE group, but you want the default group anyway, you can use the **-e** option to override it, by specifying **-e** "(default)". The double-quotes must appear since the shell treats parentheses specially. For example,
>
>   xmyCmd submit **-**e "(default)" myscript.tcl

— *SUT Info ID (numeric):*

1. command-line **-S** option takes precedence.

2. If that is missing, value of XMYSUT environment variable is used

3. If that is unset, a SUT Info ID of 2 is used as a default, since that is guaranteed to exist in the database.

17.2.3   xmyCmd Common Argument Definitions

The following list identifies and defines the standard or common arguments used by many of the xmyCmd sub-commands. The sub-sections that describe the sub-commands will refer you to this list when appropriate:

Arguments	Definitions
-h	The system will display a usage message that lists the options available for that sub-command and gives a brief explanation of each one.
-d *SD-name*	Lets you specify an SD to communicate with.
-t *time-limit*	Specifies the timeout interval. The CLUI will wait for this amount of seconds for a response from the SD. If you say **-t0**, the CLUI will wait indefinitely for a response. If you do not set the timeout, the CLUI will wait 30 seconds for a response.
-v	Provides verbose feedback from the SD. For example, in the case of the summit sub-command, in addition to telling you the script Id, the SD will send a message back to you when the script is actually dispatched to an SE. The message tells you which SE the script was dispatched to.
target-list	Specifies the script or scripts for which you want the status. For more information on target-list, See Section 17.2.4.

17.2.4 Specifying a Target List for xmyCmd Sub-commands

A target list specifies how you tell the MYNAH system about the script(s) on which you want to perform some action. Target list only apply to the list, info, Info, pause, resume, and cancel sub-commands. A target list may specify one or many scripts. The simplest type of target list used with xmyCmd sub-commands consists of one individual script. You specify the script you want by giving that script's execution ID. The execution ID is in the form of:

```
<login_name>.<N>
```

where:

<login_name> is the login-id of the person who submitted the script for execution.

<N> is a positive integer indicating that this was the Nth script that this person submitted to the SD since the SD came up.

So for example, the fifth script submitted for execution by user *geowash* would be *geowash*.5. If George Washington wanted to find out the status of the fifth script he submitted he would enter:

```
xmyCmd list geowash.5
```

Our example shows you the simplest target list you can use. The CLUI gives you the flexibility to enter a number of different notations for the target list. The following list shows you all the valid target lists you can enter. You may use only one target list on a command line.

all	Indicates all scripts submitted by you.
active	Indicates all scripts submitted by you that are active in an SE. This does not include scripts that are in an SE that have been paused in that SE (see the "pause" sub-command below).
paused	Indicates all the scripts submitted by you that have been paused while they were active in an SE.
queued	Indicates all queued scripts submitted by you. These are scripts that the SD has not yet dispatched to an SE.
dequeued	Indicates all de-queued scripts submitted by you. These are scripts that are currently being handled by an SE, whether or not they have been paused in that SE. (In other words it is the union of the "active" and "paused" target lists.)
<logid>.all	Indicates all scripts submitted by the user with the given login id.
<logid>.active	Indicates all scripts submitted by the user with the given login id, that are active in an SE.
<logid>.paused	Indicates all scripts submitted by the user with the given login id that have been paused while they were active in an SE.

<logid>.queued	Indicates all queued scripts submitted by the user with the given login id. These are scripts that the SD has not yet dispatched to an SE.
<logid>.dequeued	Indicates all de-queued scripts submitted by the user with the given login id. These are scripts that are currently being handled by an SE, whether or not they have been paused in that SE.
All	Indicates all scripts in the SD regardless of who submitted them.
Active	Indicates all scripts submitted to the SD that are currently active in an SE, regardless of who submitted them.
Paused	Indicates all scripts submitted to the SD that have been paused while they were active in an SE, regardless of who submitted them.
Queued	Indicates all queued scripts in the SD regardless of who submitted them. These are scripts that the SD has not yet dispatched to an SE.
Dequeued	Indicates all de-queued scripts submitted to the SD, regardless of who submitted them. These are scripts that are currently being handled by an SE, whether or not they have been paused in that SE.

One final thing to note about the logical target lists enumerated above: the scripts that compose the target list do not include child scripts, but only "root scripts", that is, scripts that were submitted to the SD via the GUI, the CLUI or an embedded SE, but not from one of the SEs that is controlled by the SD. To be sure, the **Info** sub-command will give you information about the child scripts of scripts that are in the target list, but those child scripts are not part of the target list itself. This is an important point to understand especially for the sub-commands that operate on the scripts in the target list—i.e., the sub-commands that pause, resume or cancel the scripts. These sub-commands are discussed below.

Examples

So, for example, if you wanted to see a list of all of geowash's scripts that are still on queue, you would type:

```
xmyCmd list geowash.queued
```

If you wanted to get extensive information about all of the scripts you submitted including child scripts, no matter what state they were in, you would type.

```
xmyCmd Info all
```

If you wanted to see a moderate amount of information about all scripts in the SD that anybody submitted that were currently being handled by an SE, whether or not they were paused in that SE, you would type.

```
xmyCmd info Dequeued
```

(The capital 'D' in the word "Dequeued" indicates that you want information on everyone's dequeued scripts, not just those that you submitted).

17.2.5   Executing Scripts with the CLUI (submit)

Syntax

```
xmyCmd submit ?-h? ?-H? ?-R? ?-d sd-name? \
           ?-e SE-group-name? ?-v? ?-E? ?-c? \
           ?-G s1=v1,s2=v2...? ?-g symbol-val-file? \
           ?-S sut-info-id? ?-T low/high? ?-F trace-file? \
           ?-l? ?-f list-file? script1 script2 ...
```

Description

The **submit** sub-command lets you execute scripts through the CLUI. **submit** lets you submit one or more script(s) to a Script Dispatcher (SD) for execution by a Script Engine (SE). When the SD receives the request, it will respond with an indication of the script ID as the SD knows it.

The script ID is always in the format *your_login_name.N*, where the *N* is a positive integer that indicates that this is the Nth script you've submitted to that SD since it was brought up. The script ID is important because you must use it for any further interaction with the SD concerning that script (e.g., pausing, resuming or canceling it).

In addition to the arguments listed in Section 17.2.3 (-h, -v, -d), **xmyCmd submit** accepts the following arguments:

Arguments	Definitions
-H	Displays a detailed usage message
-R	Displays the release number of this program
-e *SE-group-name*	Lets you specify the SE group that you want the script to run in. The argument to the **-e** option must be the name of an SE group controlled by the SD the script is submitted to. (Otherwise the SD will refuse to run the script.) A value of "(default)" (which must be enclosed in double-quotes) will cause the SD to run the script in its default SE group. For information about the order of precedence see Section 17.2.2.1.
-E	Indicates that you want the script's 'execution results' displayed to your screen when the script finishes running. The **-E** option takes no arguments. The SE will send execution results directly to CLUI, which will then display them to you.

-c   Tells this program to run the script in 'cloaked' mode. This
                          means that you don't want any MYNAH database transactions
                          while the script runs. There will be no record in the MYNAH
                          database that the script ran, and you don't even need to define
                          the script in the database. If your installation of MYNAH is
                          configured to run without a database, this option is not needed.

-G *s1=v1,s2=v2...*   Lets you specify one or more **symbol=value** pairs for the
                          Symbol-Table. Symbols cannot have spaces in them, but
                          values can. If the value has a space in it, you must use quotes
                          around the **symbol=value**, (for example **-G "a=1 1" or
                          -G a="1 1"**). You can specify multiple **symbol=value** pairs
                          either by using the **-G** option repeatedly on the same command
                          line, or by separating the **symbol=value** pairs with a comma,
                          as in: **-Ga=b,c=d,e=f**.

                          Note that no quotes are needed even though the value has
                          spaces in it. This is different than the **-G** option (see above).

                          Leave no spaces between commas, though; this program will
                          treat the spaces as part of the symbol or value.

                          If your symbols or values have any 'meta-characters' in them,
                          i.e., characters that the UNIX shell treats specially such as '*',
                          you should place the **symbol=value** pairs in a file and use the
                          -**g** option instead of using the **-G** option. If the value has a
                          comma in it, precede the comma with a backslash. (see the
                          description for **-g** option).

-g *symbol-val-file*   The **-g** option is similar to the **-G** option, except that it takes as an argument the name of a file. The file should contain **symbol=value** pairs on each line.

In a file given as the argument to the **-g** option, you can have one **symbol=value** pair per line (e.g., **a=123**). You can use multiples **symbol=value** pairs per line, if you separate them with commas (e.g., **a=123,b=x y z**).

Leave no spaces between commas, though; this program will treat the spaces as part of the symbol or value.

If you want a comma in the value, you must precede it with a backslash (**e.g., a=\,123**). Furthermore, because backslash is such a special character, if you want a backslash itself to be part of a symbol's value, you must use four backslashes in a row in the file (**e.g., a=\\\\123**).

Additionally, you can use the **-g** option repeatedly on one command line, so you can spread the **symbol=value** pairs across multiple files. For example, **-gFile1 -gFile2 -gFile3** specifies three files containing **symbol=value** pairs.

-S *sut-info-id*   Lets you specify the SUT-INFO object associated with this run of the script(s). This only applies if you are running with a MYNAH database and you are NOT running in 'cloaked' mode (see the **-c** option). In this case, CLUI will create a runtime information object in the database to keep track of what happened with this run of the script. This runtime-information object, in turn, is associated with a SUT-INFO object, that provides information about the SUT (system under test) against which the script is being run. For information about the order of precedence see Section 17.2.2.1.

-T *low/high*   Sets the trace volume level for message-based tracing. (See the **-F** option.) The arguments for the -**T** option are:

> **low** - provide low-volume trace.

> **high** - provide high-volume trace.

If you use the **-T** option, you must also use the **-F** option to designate the file into which the trace information will go.

-F *trace-file*	Lets you specify a file into which message-based tracing should be put. When message-based tracing is in effect, all the MYNAH components that handle a run of a script (e.g., CLUI, SD, SE, etc.) will place their tracing-type messages in the same file. This facilitates trouble-shooting script execution. If you use the **-F** option, then you must also use the **-T** option to designate the trace level.
-l	Tells the SE to look in its library path for this script. The SE will search in each path specified by the value of its library path, defined in the MYNAH config file, to try to find the script, rather than searching the current directory.
-f *list-file*	Lets you specify a 'listfile' containing names of scripts to be submitted for execution. This option may be used multiple times on the same command line. In the list file, you can only place one script name per line. The system ignores blank lines and lines beginning with the pound character (#). If any line has the pattern '-f file_name' the system takes it to be an indication of yet another list file, whos content will be 'included' at that point. You can nest included list files to any level.

Any number of scripts may be submitted in one invocation of the **submit** sub-command by enumerating them on the command line, or listing them in one or more 'list files' specified with the **-f** option.

> **NOTE —** If the **-d** option is not used to specify which SD to submit the scripts to, and the XMYSD environment variable is not set, then some scripts may be submitted to one SD, while others may be submitted to another SD. This is because (assuming you are using the database and not running in 'cloaked' mode) each database script object specifies an SD, and they may not all be the same.

Examples

To submit a single script (myscript.tcl) with database connection:

```
xmyCmd submit myscript.tcl
```

To submit a single script (myscript.tcl) with feedback and result, without database connection and with a **symbol=value** pair:

```
xmyCmd submit -v -E -c -G "a=hello world" myscript.tcl
```

17.2.6   Finding the Status of a Script (list, info, and Info)

Syntax

```
xmyCmd list ?-d sd_name? ?-t timeout? target list
xmyCmd info ?-d sd_name? ?-t timeout? target list
xmyCmd Info ?-d sd_name? ?-t timeout? target list
```

Description

You can find out the status of a script that has been submitted to an SD using the sub-commands **list, info**, and **Info**. Status refers to where scripts stand in the execution process: whether they are processing, paused (i.e., their processing has been temporarily suspended), or are in the execution queue managed by the SD. You can find out the status of your scripts or indeed any of the scripts assigned to an SD using these sub-commands. The difference between these sub-commands is the amount of information that each sub-command provides for each script in the target list. The following list summarizes the information that each sub-command provides:

- **list** provides one line of information about where each script is in the execution process (e.g., active, paused, etc.).

- **info** provides similar information to **list** but adds information about the Script Engine in which the script is running.

- **Info** (capital I) provides information similar to **info,** (lower case i) but adds information about all child scripts associated with the script.

For more information about the arguments that these commands accept, refer to Section 17.2.3. For more information about the target list, refer Section 17.2.4, "Specifying a Target List for xmyCmd Sub-commands".

Each sub-command's output is divided into sections with headings for queued, active, and paused. For example, all script IDs for queued scripts appear in the "queued section. Similarly, all script IDs for paused scripts appear in the "paused" section.

Examples

To get a summary list of all your scripts that are running in the default SD:

```
xmyCmd list all
```

To get detailed information about all your scripts, including child scripts, that are running in SD **SD2**:

```
xmyCmd Info -d SD2 all
```

17.2.7   Operating on Running Scripts (pause, resume, and cancel)

Scripts can be operated on in the sense that you can pause and resume them, or cancel them all together, while they are being executed by the MYNAH System. The sub-commands that fall under the category of operation sub-commands are pause, resume and cancel.

All the script operation sub-commands use the same target-lists as described in Section 17.2.4, "Specifying a Target List for xmyCmd Sub-commands".

To get the execution IDs for the target-list, execute the **Info** subcommand described in Section 17.2.6.

17.2.7.1   Stopping Scripts Temporarily (pause)

Syntax

```
xmyCmd pause ?-d sd_name? ?-t time_out? ?-v? target_list
```

Description

The **pause** sub-command tells the SD to temporarily halt a specified script (or scripts) but not to cancel its execution.

xmyCmd pause takes the following arguments: *-h, -d, -t, -v,* and *target_list.* See Section 17.2.4 for information about how to specify a target list. For more information about the other arguments see Section 17.2.3.

You can "pause" scripts while they are executing in an SE or when they are in the queue waiting to execute. If you pause a script while it is in the queue, the system will not dispatch the script to an SE until it is "resumed." See Section 17.2.7.2 for information on how to "resume" a script once it is paused.

> **NOTE —** If a script that you are pausing has child scripts, pausing the script will NOT cause its child scripts to be paused. To pause a child script, you must specify the child script's execution ID as the target-list.

Examples

If your user ID is *geowash*, to pause only your active scripts:

```
xmyCmd pause geowash.active
```

To **pause** all *your* scripts:

```
xmyCmd pause all
```

17.2.7.2   Restarting Scripts (resume)

Syntax

```
xmyCmd resume ?-d sd_name? ?-t time_out? ?-v? target_list
```

Description

The **resume** sub-command restarts paused scripts. The scripts will start at the point they were paused.

xmyCmd resume takes the following arguments: *-h, -d, -t, -v* and *target_list.* See Section 17.2.4 for information about how to specify a target list. For more information about the other arguments refer to Section 17.2.3.

You can "resume" scripts that were paused while they were executing in an SE or while they were in the queue waiting to execute. If they were paused on queue and you resume them, then they are once again eligible to be dispatched to an SE for execution. See Section 17.2.7.1 for information on how to "pause" a script.

> **NOTE —** If a paused script has child scripts that are paused, resuming the script will NOT cause its child scripts to be resumed. To resume a child script, you must specify the child script's execution ID as the target-list.

Examples

If your user ID is *geowash*, to **resume** only your paused scripts:

```
xmyCmd resume geowash.paused
```

To resume all *your* scripts:

```
xmyCmd resume all
```

17.2.7.3   Ending Script Execution (cancel)

Syntax

```
xmyCmd cancel ?-d sd_name? ?-t time_out? ?-v? target_list
```

Description

The **cancel** sub-command stops the execution of a script and completely removes it from the SD.

xmyCmd cancel takes the following arguments: **-h**, **-d, -t**, **-v**, and **target_list**. See Section 17.2.4 for information about how to specify a target list. For information about the other arguments refer to Section 17.2.3.

You can **cancel** scripts that are executing in an SE or while they are in the queue waiting to execute.

> **NOTE —** If the scripts you **cancel** are executing in an SE and have child scripts, their child scripts (and grandchild scripts, and so on) will all be canceled as well. This is different from the **pause** and **resume** sub-commands in which child scripts are unaffected.

Examples

If your user ID is *geowash*, to cancel only your queued scripts:

```
xmyCmd cancel geowash.queued
```

To **cancel** all *your* scripts:

```
xmyCmd cancel all
```

17.2.8   Viewing SE Group Status (sestat)

Syntax

```
xmyCmd sestat ?-d sd_name? ?-t time_out? ?-v? ?se_group_name?
```

Description

The **sestat** sub-command lets you check the status of the SE Groups controlled by an SD.

xmyCmd sestat accepts the following argument: **-h**, **-d**, **-v**, **-t, and** *SE_gorup_name*. For more information about **-h**, **-d**, **-v**, **-t** refer to the arguments listed in Section 17.2.3:

SE_group_name, lets you specify the SE-group that you want status on. If you do not specify *SE_group_ name*, you will get status information on all SE-groups controlled by the SD used in the command or if **-d** is not specified, the default SD.

17.2.9   Viewing and Changing SD Parameters

The CLUI provides you with a number of sub-commands that allow you to find the status of SDs and manipulate the parameters for your SD.

Remember these are commands you as a user can execute without administrative privileges. In addition to the sub-commands listed here, there are a number of sub-commands that provide extensive information about the SD and control over its execution, but you need administrative privileges to use them. Refer to the *MYNAH System Administration Guide* for information about the administrative features of the CLUI.

17.2.9.1   Viewing SD Parameter Values (sysstats)

Syntax

```
xmyCmd sysstats ?-d sd_name? ?-t time_out? ?-v?
```

Description

The **sysstats** sub-command lets you see parameter values for an SD. These are things such as:

- The time the SD was brought up

- The SD's overall system concurrency (i.e., the maximum number of scripts the SD will run at a time.)

- Whether or not the SD is running with the MYNAH database.

xmyCmd sysstats takes the following arguments: *-d, -t* and *-v*. For information about these arguments refer to Section 17.2.3.

Examples

This example displays the parameters for the default SD. In this example the SD named "a" is displayed because that is the default SD in the MYNAH config file. You could us the -**d** option to see the parameters of another SD.

```
selene kjd: xmyCmd sysstats
SD(a): SD Program statistics on 01/08/97 at 14:31:07:

SD(a)'s parameters are as follows:
        Name:                 a
        Start Time:           12/13/96 at 13:51:35
        Overall SD Concurrency: 30
        Default User Conc.:     5
        Default User Priority:  2
        SD(a) is running with the MYNAH database
```

17.2.9.2   Viewing Your User Parameters (mystats)

Syntax

```
xmyCmd mystats ?-d sd_name? ?-t time_out? ?-v?
```

Description

The **mystats** sub-command lets you see what your user parameters are in a specified SD. User parameters are such things as:

- Your queuing priority (this helps determine the order in which queued scripts are dispatched to an SE.)

- Your actual-concurrency value (this is the maximum number of scripts you can have running at a time—you have control over this, but it cannot be set higher than your maximum-concurrency value.)

- Your max-concurrency (this is the maximum value to which you can set your "actual-concurrency" parameter.)

- The number of tests submitted by you that are currently queued, active or paused-in-an-SE.

xmyCmd mystats takes the following arguments *-d, -t* and *-v.* For information about these arguments refer to Section 17.2.3.

Examples

This example displays the parameters for the user who initiated the command (kjd).

```
selene kjd: xmyCmd mystats
SD(a):
User          Pri-   Max  Actual  Queued   Active   Paused
              ority  Conc   Conc  Scripts  Scripts  Scripts
kjd             2     5      4       0        0        0
```

17.2.9.3   Viewing SD Users (users)

Syntax

```
xmyCmd users ?-d sd_name? ?-t time_out? ?-v?
```

Description

The **users** sub-command lets you see a list of users the SD knows about. These are users who can submit scripts to the SD. These users will be defined in the MYNAH database. If there is no MYNAH database, this will be a list of users who have used the SD.

xmyCmd users takes the following arguments: *-d, -t* and *-v*. For information about these arguments refer to Section 17.2.3.

Examples

This example displays the user known to the default SD.

```
selene kjd: xmyCmd users
SD(a): The following users are known by SD(a):  mgt02 kjd adaj root daveng
silva ghia nfun wunn mehral pml3 madmin(adm)
```

> **NOTE —** Any user with administrative privileges (such as madmin in the above example) will have "(adm)" after the name.

17.2.9.4   Viewing User Parameters of SD Users (usrstats)

Syntax

```
xmyCmd usrstats ?-d sd_name? ?-t time_out? ?-v? ?login_id?
```

Description

The **usrstats** sub-command is similar to the **mystats** sub-command (Section 17.2.9.2), but it lets you see the parameters of other users of the SD besides yourself.

In addition to the arguments listed in Section 17.2.3, **-d**, **-v**, **-t, xmyCmd usrstats** accepts the following argument:

login_id, lets you specifies the login id of the person whose parameters you want to see. If you do not specify any *login_id*, the system will display the parameters of all users the SD knows about.

Examples

This example displays the parameters for the user *waln*.

```
selene kjd xmyCmd usrstats waln
SD(a):
User            Pri-   Max  Actual  Queued   Active  Paused
                ority  Conc   Conc  Scripts  Scripts Scripts
waln              2     5      5       0        0       0    (admin)
```

17.2.9.5   Displaying SE Groups Associated with an SD (segroups)

Syntax

```
xmyCmd segroups ?-d sd_name? ?-t time_out? ?-v?
```

Description

You can display the SE Groups associated with (i.e., controlled by) an SD using the **segroups** sub-command.

xmyCmd segroups takes the following arguments: *-d, -t* and *-v*. For information about these arguments refer to Section 17.2.3.

Examples

This example displays the SE Group known to the default SD.

```
selen kjd: xmyCmd segroups
SD(a): SE groups known by SD(a) are the following:  SeGp2 SeGp1
```

17.2.9.6   Setting Your Actual Concurrency in an SD (usractconc)

Syntax

```
xmyCmd usractconc ?-d sd_name? ?-t time_out? ?-v? value
```

Description

The **usractconc** sub-command lets you change your 'actual concurrency level' (maximum number of scripts you can run at a time in the SD). This is to be distinquish from your 'maximum concurrency level' which only and administrator can set. Your maximum concurrency level is the maximum value to which you can set your actual concurrency level. This scheme allows you to set your actual concurrency level to zero for example, and queue scripts up in the SD. Then you can let them go at any time by setting your actual concurrency level to a positive number. Or, you can set your actual concurrency level to one and single-thread your scripts, as another example.

In addition to the arguments listed in Section 17.2.3 (*-d, -t, -v*) **xmyCmd usractconc** accepts the following:

value, lets you specifies the number of scripts you can have running at a time*.*

Examples

This example changes the concurrency level for the user kjd to four (4).

```
selene kjd: xmyCmd usractconc 4
SD(a): 'kjd' now has an actual-conc. of 4.
User            Pri-    Max  Actual   Queued   Active   Paused
               ority   Conc    Conc  Scripts  Scripts  Scripts
kjd                2      5       4        0        0        0
```

> **NOTE —** A concurrency is the maximum number of scripts that the SE can run at a time. The **user max** concurrency is the max number to which that user can set his or her actual concurrency.

17.2.10 Encrypting Script Keys (scramble)

Syntax

```
xmyCmd scramble ?-k key? ?output-file?
```

Description

You can use the **scramble** sub-command to encrypt the key associated with a script. This will provide you with some security since only an SE can unscramble a key once you have scrambled it.

xmyCmd scramble takes the following options:

-k   Specifies a key to be saved (8 characters max).

          If you omit the *key* argument on the command line, CLUI will prompt you to enter a key.

output-file   Specifies the name of a file the scrambled key will be written to. If you don't specify a file, CLUI will write the scrambled key to standard output.

Please refer to the *MYNAH System Scripting Guide* for a complete description of when and how to use **scramble** to deal with sensitive data.

17.2.11 Merging Output Files (mergeOutput)

Syntax

```
xmyCmd mergeOutput ?-D directory? ?-a? ?-c? ?-o? ?-e? \
                   ?-s sutfile(s)?
```

Description

The **mergeOutput** sub-command lets you merge some of the files in your script's output directory. **mergeOutput** reads the "output" file in a script output directory and merges its content with the content of the files you request though the sub-command's options. If no directory is specified, **mergeOutput** assumes the current directory. Merged output is written to stdout.

xmyCmd mergeOutput takes the following options:

-D *directory*	Specifies the script output directory (default: ".").
-a	Specifies that you want to merge all files ("output", "compares", "stdout", "stderr", and sut image files).
-c	Specifies that you want to merge in the "compares" file.
-o	Specifies that you want to merge in the "stdout" file.
-e	Specifies that you want to merge in the "stderr" file.
-s *sutfile(s)*	Specifies that you want to merge in one or more sut-image files. The -**s** option is followed my the name of a sut-image file, or a comma-separated list of them (e.g., -**s SUTimage..xmyTerm3270_1,SUTimage..xmyTerm3270_2**), or the flag **all**, meaning merge in all sut-image files. The -**s** option can be specified multiple times with a single sut-image file named each time.

> **NOTE —** "stdout" and "stderr" files are not cross-indexed with the "output" file, so their output is not really merged—they are displayed prior to the "output" file. Any other files in the script output directory (such as files produced explicitly by the user) are not accessible with this sub-command.

Examples

Figure 17-1 shows an an excerpt of a *sutimages* file that has been merged with an *output* file using the command

```
selene kjd: xmyCmd mergeOutput -s SUTimage..xmyTerm3270_1
```

```
FILE(S) 'output', requested sut images
19961120:141401:script::start:SeGp1
19961120:141402:testobj::begin:3002
19961120:141402:lang:tcl:command:set numfail 3
19961120:141402:lang:tcl:command:xmyBegin -testid 3003
19961120:141402:testobj::begin:3003
19961120:141402:lang:tcl:command:xmyLoadPkg Term3270
19961120:141404:lang:tcl:command:xmyTerm3270 connect -TN3270E true -initialWaitE
xpect "INPUT APPLICATION" -tagDir /bdfs/users/stw/tagname
19961120:141407:suttiming:Term3270::.xmyTerm3270_1:sent ::0.000000
19961120:141407:sutimage:Term3270:rcv:.xmyTerm3270_1::0:0
IMAGE HEADER - Screen (index:0)
                                                                          MSG10
         *----------------------------------------------------------*
           This  system is intended to be used solely by  authorized
           users  for  legitimate  corporate  business.  Users  are
           monitored to the extent necessary  to properly administer
           the system and to investigate unauthorized access or use.
           By accessing  this  system,  you  are  consenting to this
           monitoring.   Unauthorized use is subject to prosecution.
         *----------------------------------------------------------*
           For Problems/Assistance call the hotline at  (908) 699-4100
 INPUT APPLICATION NAME AND PRESS ENTER
IMAGE FOOTER - Screen
19961120:141407:lang:tcl:command:set conn1 [xmyTerm3270 connect -TN3270E true -i
nitialWaitExpect "INPUT APPLICATION" -tagDir /bdfs/users/stw/tagname]
19961120:141407:lang:tcl:command:$conn1 type -position "21 2" -text "st8"
19961120:141407:lang:tcl:command:$conn1 sendWait -key enter
19961120:141407:sutimage:Term3270:snd:.xmyTerm3270_1::2000:0
```

Figure 17-1. mergeOutput Example

17.2.12   Comparing Received Messages (diff)

Syntax

```
xmyCmd diff ?-q? ?-f? ?-s sedScriptFile? \
            ?-l listFile? \ \
            scriptBaseFile


xmyCmd diff ?-q? ?-f? ?-s sedScriptFile? \
            ?-l listFile? \ \
            file1 file2
```

Description

The **diff** sub-command lets you compare messages received over a connection to a SUT to see if there is any difference between them. **diff** shows you what the differences, if any, were (assuming you don't run it in quiet mode with the **-q** option), or tells you that there were no differences between the two messages. You specify the two files containing the messages you want to compare against each other by either typing both file names on the command line, or typing a base file name, in which case **diff** figures out which two files to compare, based on the base file name you typed.

xmyCmd diff takes the following options:

-q	Specifies that you want the sub-command to run in "quite" mode (i.e., don't print anything to stdout, but just exit with an exit status of 0 to indicate no differences or 1 to indicate that differences were found).
-f	Specifies that you want the sub-command to run in "no-format" mode when you are comparing FCIF messages that might not be formatted (i.e., you don't want to do any formatting work on the messages before comparing them).
-s *sedScriptFile*	Specifies the name of a sed-script-file (i.e., a file containing a **sed**(1) script) to be applied to the files that will be compared to each other. If *sedScriptFile* does not start with "." or "/", it must reside in the *$XMYHOME/data/sedscripts* directory.

-l *listFile*	Specifies a file that contains a list of other files. Those other files contain sed scripts to be applied to the files that will be compared to each other. If *listFile* does not start with "." or "/", it must reside in the *$XMYHOME/data/sedscripts* directory.
scriptBaseFile	Specifies the base file name for the two files to be compared. The two files must have the extensions *.out* and *.mstr*.
	If for example, ***scriptBaseFile*** is *message*, then the two files that will be compared are *message.out* and *../message.mstr*. If *message.mstr* does not exist in the parent directory, then *message.out* will be compared to *message.mstr* in the current directory instead of the parent directory.
file1 file2	Specifies two files to be compared to each other. You can use this form instead of the **-s** *scriptBaseFile* form described above. If you do use this form, then you must specify the name of the two files to be compared, as the sub-command will not add any suffix such as ".out" or ".mstr" to either ***file1*** or ***file2***.

Files generated by the Term3270 package may contain the UNIX carriage return character, ^M (<control>-M), at the end of each line. The MYNAH System includes an **sed** script, **strip_ctl_m**, to strip out the ^M characters from these files when **xmyCmd diff** compares them. This script is located in *$XMYHOME/data/sedscripts* and can be specified as the *sedScriptFile* to the **-s** option, i.e., as **-s strip_ctl_m**.

> **NOTE —** If the files to be compared have control-M's (octal 15) in them, then you are ***strongly*** advised to use **-s strip_ctl_m** as part of the command line. If **-s strip_ctl_m** is not used, you will get unpredictable results.

For each line of the two files being compared that contains a difference, **xmyCmd diff** will display the line as it appears in each file, and then display a line with carets ('**^**' characters) to show where the differences are. If any line containing a difference is more than 75 characters long (including the newline at the end), **xmyCmd diff**'s output will break the line into multiple "chunks" of 75 characters each and will show all chunks of the line that contained the difference, including chunks that are identical.

Only the chunks that contain a difference will be followed by another line with carets showing where the difference is.

The **diff** sub-command is actually used "behind the scenes" by the SE when it executes the **xmyDiff** command in Tcl (the **xmyDiff** command is part of the MYNAH extensions to the Tcl language). For more information on **xmyDiff** and on differencing messages received over a SUT channel, see the General MYNAH Tcl Extensions chapter of the *MYNAH System Scripting Guide*.

Example

This example compares two files. **xmyCmd diff** returns the lines containing the differences. If a line occurs in both files but there are differences, **xmyCmd diff** will return both lines, indicating in which file the line was found and the differences, which are indicated by a caret (^) sign. If a line ends with *A*, it means the line was found in *file1*. If a line ends with *B*, it means the line was found in *file2*. If a line occurs in only one file, **xmyCmd diff** returns the line, indicating in which file the line was found.

First, **xmyCmd diff** finds that *file2* (*B*) has a timeout value of 60 seconds while *file1* (*A*) has a timeout value of 30 seconds. Secondly, **xmyCmd diff** finds that the line

```
$nylib sendWait "PS1=\"$ \"\r" -expect "$ "
```

occurs only in *file2* (*B*) .

```
selene ksb: xmyCmd diff NYPL.tcl NYPL2.tcl
|###################################################################
| xmyCmd(diff) OUTPUT:                       Mon Jan 20 16:52:25 EST 1997
set nylib [xmyTermAsync connect -timeout 60 -shell /bin/sh]          *B*
set nylib [xmyTermAsync connect -timeout 30 -shell /bin/sh]          *A*
                                       ^
$nylib sendWait "PS1=\"$ \"\r" -expect "$ "                          *B*
| xmyCmd(diff) return code is 1:   DIFFERENCES
|###################################################################
```

17.2.13 Creating Script Objects (xmyCreateScriptObject)

Syntax

```
xmyCreateScriptObject ?-h? ?-H? ?-f listfile? ?-d SD_name?
?-g SEgroup_name? ?-o logid? filename
```

Description

You can automatically create database Script objects to represent any existing script files from the UNIX command line using the **xmyCreateScriptObject** command. Scripts represented by Script objects can appear in the MYNAH database, which will make managing them a lot easier. *filename* is the name of the file in which there is a list of script files. The script names must appear with their complete UNIX path.

In addition to *filename*, **xmyCreateScriptObject** takes the following options:

-h   Displays a brief usage message.

-H   Displays a long usage message.

-f   Specifies a "listfile" containing the pathname(s) of the scripts for which you want to create objects.

-d   Lets you specify an SD for the Script object you create with this command. If you do not specify an SD, the system will use the default SD as specified in the MYNAH config file.

-g   Lets you specify an SE group for the Script object you create with this command. If you do not specify an SE group, the system will use the default SE group for that SD.

-o   Specifies a UNIX login ID as the owner of the script. If you do not specify an owner, the system will assume the person who is running xmyCreateScriptObject is the owner and that person's UNIX login ID will be the Owner attribute.

Example

You have two script files, *logon_test.tcl* and *stress_test.tcl*, for which you want to create Script objects. You can not use xmyCreateScriptObject directly on a script file. Therefore, you must create a file, e.g., *create_list*, that contains the full path listing each script file, one file per line, as in the following:

```
/u/kjd/CREATE/logon_test.tcl
/u/kjd/CREATE/stress_test.tcl
```

Within the directory */u/kjd/CREATE* you could the execute the following to create a Script object for each script file:

```
selene kjd: xmyCreateScriptObject -f create_list

connecting to database...
.xmySdperson..0   GET          ID=511
.xmySdscript..1   GET          ID=3780
.xmySdscript..1   RENAME
.xmySdscript..2   GET          ID=3781
.xmySdscript..2   RENAME
.xmySdscript..0   RENAME
```

17.3   Running Reports From the CLUI - xmyReport

You can create and run reports from the GUI. However, sometimes it is useful to be able to run reports from the command line. This is particularly useful if you wish to have a report run automatically, on a regular basis (through use of the Unix cron facility).

Syntax

```
xmyReport ? testProgress -H?-h? ? ? requirement -H?-h? \
     ? testSpec -H?-h? ? ? issues -H?-h? ?
```

Description

The **xmyReport** command has multiple sub-commands: **testProgress**, **requirement**, **testSpec** and **issues**. These sub-commands equate to each of the report types that are available with the system.

To get syntax help for individual sub-commands just type xmyReport followed by the sub-command, and then a -H. For example:

```
xmyReport testProgress -H
```

provides information about the testProgress report.

Three of the report sub-commands (testProgress, requirement and testSpec) provide the following two options:

```
-query_sql "sql query"
```

```
-query "query to be displayed"
```

The **query_sql** option lets you specify an ad hoc query to obtain the list of objects that you want a report on. However, writing the sql query requires detailed knowledge of the structure of the MYNAH database. If you are using the GUI, it is suggested that you use the **Report** menu, **Create Command File** option, if you wish to use queries. See Section 14.4.5 for complete information.

The **query** option provides a method to control the query information that will be used on the report output itself. Again, this option is primarily here for the automatic system generated commands that you can obtain from the GUI.

The following sub-sections provide information for each of the **xmyReport** sub-commands. Report output format is not covered here because it is covered in Section 14.4.4.

17.3.1   testProgress

Syntax

```
xmyReport testProgress ?-sut sutid? \
                  ? -list test_ids ? \
                  ? -kw {keyword_name1} {keyword_name2}? \
                  ? -ownedby owner_logid? \
                  ? -weight weight_measure? \
                  ? -report Detailed|Summary? \
                  ? -File filename ?
```

Description

The **testProgress** sub-command provides a status report for a set of Tests for a specified SUT. A *sutid* is required. You can select the list of Tests to be reported on by either providing a list of Test ids, or by specifying associated Keywords, and/or a specific Owner of the Tests.

testProgress takes the following options:

-sut	You must specify a valid SUT id. Results associated with this id will be reported on.
-list	You provide a list of test ids. If this option is used, the **-kw** and **-ownedby** options do not apply.
-kw	Lets you specify one or more keywords. The report will locate all Tests that are associated will all Keywords specified**.**
-ownedby	Lets you specify a logid. The report will locate all Tests that are owned by that person.
-weight	Lets you specify that you would like reported Tests weighted by either **priority** or **effort** (no weighting factor is the default)
-report	Lets you select either **Detailed** or **Summary** report output.
-**File**	Lets you specify a file where the report output should go. The default is for the output to come back to your screen.

Example

This example asks for a detailed report on all Tests owned by hsb.

```
selene kjd: xmyReport testProgress -sut 6 -ownedby hsb
-report Detailed
```

17.3.2   requirement

Syntax

```
xmyReport requirement ?-sut sutid? \
                ?-reqid req_ids? \
                ?-type type_name? \
                ?-report Detailed|Summary ? \
                ?-File filename?
```

Description

The **requirement** sub-command provides a status report for a set of Requirements, for a specified SUT. A sutid is required. You can select the list of Requirements to be reported on by specifying either a list of Requirements ids or by specifying a specific Requirement Type.

requirement takes the following options:

-sut	You must specify a valid SUT id. Results associated with this id will be reported on for the Tests that are associated with the included Requirements.
-reqid	You provide a list requirements ids. If this option is used, the -type option does not apply.
-type	Lets you specify a Requirement Type. The report will locate all Requirements of that Type**.**
-report	Lets you select either **Detailed** or **Summary** report output.
-**File**	Lets you specify a file where the report output should go. The default is for the output to come back to your screen.

Example

This example asks for a detailed report on all Requirements of Type CROLO.

```
selene kjd: xmyReport requirement -sut 6 -type CROLO -report
Detailed
```

17.3.3   testSpec

Syntax

```
xmyReport testSpec ?-sut sutid? \
                   ?-type type_name? \
                   ?-ownedby owner_logid? \
                   ?-who_created created_by? \
                   ?-report Detailed | Summary? \
                   ?-File filename?
```

Description

The **testSpec** sub-command provides a formatted report for a set of Tests. You can select the list of Tests to be reported on by specifying an associated SUT id, a Test Type, an Owner, or a creator. All conditions are ANDed together to form the list of Tests to be reported.

testSpec takes the following options:

-sut	You may specify a valid SUT id. Tests that are associated with this SUT will be included.
-type	Lets you specify a Test Type. The report will include all Tests of that Type**.**
-ownedby	Lets you specify a logid. The report will include all Tests that are owned by that person.
-who_created	Lets you specify a logid. The report will include all Tests that were created by that person.
-report	Lets you select either **Detailed** or **Summary** report output.
-**File**	Lets you specify a file where the report output should go. The default is for the output to come back to your screen.

Example

This example asks for a detailed report on all Tests of Type testcase that are owned by hsb.

```
selene kjd: xmyReport testSpec -type testcase -ownedby hsb \
             -report Detailed
```

17.3.4   Issues

Syntax

```
xmyReport issues ?-sut sutid? \
                  ?-status state_value? \
                  ?-type type_name? \
                  ?-ownedby owner_logid? \
                  ?-who_created created_by? \
                  ?-report Detailed|Summary? \
                  ?-File filename?
```

Description

The **issues** sub-command provides a formatted report for a set of Issues. You can select the list of Issues to be reported on by specifying an associated SUT id, an Issue Status, an Issue Type, an Owner, or a creator. All conditions are ANDed together to form the list of Issues to be reported.

issues takes the following options:

-sut	You may specify a valid SUT id. Issues that are associated with this SUT will be included.
-type	Lets you specify an Issue Type. The report will include all Issues of that Type**.**
-ownedby	Lets you specify a logid. The report will include all Issues that are owned by that person.
-who_created	Lets you specify a logid. The report will include all Issues that were created by that person.
-report	Lets you select either **Detailed** or **Summary** report output.
-**File**	Lets you specify a file where the report output should go. The default is for the output to come back to your screen.

Example

This example asks for a detailed report on all Issues with a status of open that are owned by hsb.

```
selene kjd: xmyReport issues -status open -ownedby hsb
-report Detailed
```

Appendix A: Quick Reference to Menus and Menu Options

This appendix contains a listing of all MYNAH menus and their selections. We organized menus alphabetically by object. The menus for MYNAH Tools -- Database Browser, Job Status and Job Status Script Builder -- also appear in the list in alphabetical order.

All the menus available for an object or tool appear in a table with the menu options listed. We give a brief description of the function of menu selections. We also list accelerator keys (where applicable).

All you need to do to find information about a menu is:

1. Find the object you want.

2. Look at the **Menu** name in the first column of the table.

3. Find the menu **Selection** you want.

4. For more information about how you use the menu selection, turn to the **Manual Section** listed in the last column of the table.

A.1   Database Browser

Table A-1. Database Browser Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Browser	Print	Ctrl-P	Displays the Print dialog with which you can print the contents of the List view.
	Close	Ctrl-F4	Closes the Database Browser
Selected	Open	Ctrl-O	Opens a selected (highlighted) object.
	Bulk Change Owner		Allows you to change the owner of an object or objects. Valid for Scripts or Tests.
	Bulk Associate		Allows you to associate an object or objects to the currently selected objects.
	Run	Ctrl-R	Causes the system to display the Run dialog with which you can run scripts.
Edit	Copy	Ctrl-C	Copies the currently selected object to a buffer.
	Delete	Del (key)	Deletes selected object(s) from the database and clears them from the display.
	Select All	Ctrl-/	Selects all the objects in the List view.
	Deselect All	Ctrl-\	Deselects all the objects in the List view
View	Include		Displays the Include dialog which allows you to include objects in the Database Browser display.
	Refresh		Performs the Include Query again and refreshes the display.
Tools	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.

Table A-1. Database Browser Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.2   Job Status

Table A-2. Job Status Menus and Menu Selections

Menu	Selection	Accelerator Key	Function
Status	Close	Ctrl-F4	Closes the Job Status window.
Edit	Undo	Ctrl-Z	Available in future releases
	Set Status		Displays a sub-menu that allows you to Pause, Resume or Cancel script execution.
	Select All	Ctrl-/	Selects all the objects in the List view.
	Deselect All	Ctrl-\	Deselects all the objects in the List view
View	Include		Displays the Include dialog which allows you to include specific Runtimes to display in the List view.
	Refresh		Performs the Include Query again and refreshes the display.
Tools	Database Browser		Displays the Database Browser tool with which you can find, open and run objects.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.3   Folder Object

Table A-3. Folder Object Menus and Menu Selections (Sheet 1 of 3)

Menu	Selection	Accelerator Key	Function
Folder	Save	Ctrl-S	Saves the desktop. All preferences, folders and Include queries are saved as well.
	Close	Ctrl-F4	Closes the Folder.

Table A-3. Folder Object Menus and Menu Selections (Sheet 2 of 3)

Menu	Selection	Accelerator Key	Function
Selected	New		Displays a sub-menu with which you can create a new object.
	Open	Ctrl-O	Opens a selected (highlighted) object.
	Run		Displays the Run dialog with which you can run scripts.
	Delete	Del (key)	Deletes selected object from the database and clears it from the display.
	Expand Fully		Expands a list view or hierarchy to show all levels.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected object(s) from the display and places them on the clipboard.
	Copy	Ctrl-C	Copies the currently selected object to the clipboard.
	Paste	Ctrl-V	Pastes contents of buffer below selected object. If nothing is selected, places object in the first position in the List view.
	Clear	Ctrl-L	Clears selected object(s) from the display (does not place them on the clipboard).
	Select All	Ctrl-/	Selects all the objects in the List view.
	Deselect All	Ctrl-\	Deselects all the objects in the List view
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Checks objects in the view against the database and prompts you if there is a discrepancy.

Table A-3. Folder Object Menus and Menu Selections (Sheet 3 of 3)

Menu	Selection	Accelerator Key	Function
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.4 Issue Object

Table A-4. Issue Object Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Issue	Save	Ctrl-S	Saves the issue. Information in all views is saved.
	Print	Ctrl-P	Displays the Print dialog with which you can print the Issue.
	Close	Ctrl-F4	Closes the object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Issue to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.

Table A-4. Issue Object Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.5   Keyword Object

Table A-5. Keyword Object Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Keyword	Save	Ctrl-S	Saves the Keyword. Information in all views is saved.
	Print	Ctrl-P	Displays the Print dialog with which you can print the Keyword.
	Close	Ctrl-F4	Closes the object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Keyword to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.

Table A-5. Keyword Object Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.6   MYNAH Desktop

Table A-6. MYNAH Desktop Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Mynah	Save	Ctrl-S	Saves the desktop. All preferences, folders and Include queries are saved as well.
	Reports		Displays Report dialog with which you can generate reports.
	Iconify All Windows		Reduces all open windows to icons
	Exit		Exits the MYNAH System.
Selected	New		Displays a sub-menu with which you can create a new object.
	Open	Ctrl-O	Opens a selected (highlighted) object.
	Run	Ctrl-R	Displays the Run dialog with which you can run scripts.
	Delete	Del (key)	Deletes selected object from the database and clears it from the display.
	Expand Fully		Expands Listviews and hierarchies to show all levels.
Edit	Undo	Ctrl-Z	Available in future release.
	Cut	Ctrl-X	Cuts the currently selected object to the clipboard.
	Copy	Ctrl-C	Copies the currently selected object to the clipboard.
	Paste	Ctrl-V	Pastes contents of the clipboard below selected object. If nothing is selected, places object in the first position in the List view.
	Clear	Ctrl-L	Clears selected object from the display (does not place anything on the clipboard).
	Select All	Ctrl-/	Selects all the objects in the List view.
	Deselect All	Ctrl-\	Deselects all the objects in the List view
View	Change View		Displays a list of available views so you can change the current view.

Table A-6. MYNAH Desktop Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Tools	Refresh		Checks objects in the view against the database and prompts you if there is a discrepancy.
	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.7   Person Object

Table A-7. Person Object Menus and Menu Selections

Menu	Selection	Accelerator Key	Function
Person	Save	Ctrl-S	Saves the Person. Information in all views is saved.
	Close	Ctrl-F4	Closes the object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Person to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.8   Requirement Object

Table A-8. Requirement Object Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Requirement	Save	Ctrl-S	Saves the Requirements. Information in all views is saved.
	Print	Ctrl-P	Displays the Print dialog with which you can print the Issue.
	Close	Ctrl-F4	Closes the object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Requirements to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.

Table A-8. Requirement Object Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.9   Result Object

Table A-9. Results Object Menus and Menu Selections

| Menu	Selection	Accelerator Key	Function
Result	Save	Ctrl-S	Saves the Result. Information in all views is saved.
	Close	Ctrl-F4	Closes the object.
Edit	Undo	Ctrl-Z	Available in future releases
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Result to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.10 Runtime Object

Table A-10. Runtime Object Menus and Menu Selections

Menu	Selection	Accelerator Key	Function
Runtime	Save	Ctrl-S	Saves the Runtime. Information in all views is saved.
	Close	Ctrl-F4	Closes the object.
Selected	Open	Ctrl-O	Opens a selected (highlighted) object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Runtime to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.11 Script Object

Table A-11. Script Object Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Script	Save	Ctrl-S	Saves the Script. Information in all views is saved.
	Print	Ctrl-P	Displays the Print dialog with which you can print the Issue.
	Run	Ctrl-R	Brings up the Run Script dialog so that you can submit the script to the BEE for execution.
	Close	Ctrl-F4	Closes the object.
Selected	Open	Ctrl-O	Opens a selected (highlighted) object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Script to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
	External Editor		Initiates external editor.
	Insert Template		Causes the system to display the Insert Template dialog.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.

Table A-11. Script Object Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.12   Script Builder

Table A-12. Script Builder Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Builder	Save Preferences		Save the preferences values you entered on the Preferences view.
	Save Code		Saves script code entered in the Code view.
	Run Code	Ctrl-R	Executes the code in the Code View.
	Reinitialize	Ctrl-l	Re-initializes the Script Builder to its starting state.
	Open Connection	Ctrl-O	Causes the system to display the Open Connection dialog with which you can make an asynchronous or 3270 connection.
	Close	Ctrl-F4	Closes the Script Builder.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
	External Editor		Causes the system to display the external editor specified for you environment.
	Insert Template		Displays a dialog with which you can insert a template of xmyTcl commands into the Code View.
	Insert Code		Displays dialogs with which you can insert code into the Code View either from a file or from another script.
	Insert Breakpoint		Insert breakpoints into code in the Code View.
	Select All	Ctrl-/	Selects all the code in the Code VIew.
	Deselect All	Ctrl-\	Deletes all the code in the Code VIew.

Table A-12. Script Builder Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
View	Change View		Displays a list of available views so you can change the current view.
	Show Progress		Displays the progress dialog which indicates the progression of script execution.
	Show Output		Show output files for a script.
Tools	Database Browser		Displays the Database Browser tool which allows you to find, open, and run objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.13   SUT Object

Table A-13. SUT Object Menus and Menu Selections

Menu	Selection	Accelerator Key	Function
SUT	Save	Ctrl-S	Saves the SUT. Information in all views is saved.
	Close	Ctrl-F4	Closes the object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the SUT to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.14 SUT Hierarchy Object

Table A-14. SUT Hierarchy Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Hierarchy	Save	Ctrl-S	Saves the Hierarchy.
	Close	Ctrl-F4	Closes the object.
Selected	New SUT		Creates a new SUT object and places it below the highlighted object. If nothing selected, it will be placed at the top level.
	Expand Fully		Expands the hierarchy to show all levels.
	Open	Ctrl-O	Opens a selected (highlighted) object.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected object and places it in a clipboard.
	Copy	Ctrl-C	Copies the currently selected object to a clipboard.
	Copy Object	Ctrl-B	Copies the Hierarchy to the clipboard (it can then be parted into a Folder).
	Paste	Ctrl-V	Pastes contents of clipboard below selected object. If nothing is selected, places object in the first position in the List view.
	Delete	Del (key)	Deletes selected object.
	Select All	Ctrl-/	Selects all the objects in the List view.
	Deselect All	Ctrl-\	Deselects all the objects in the List view
	Duplicate		Duplicates objects.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.

Table A-14. SUT Hierarchy Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.15 Test Object

Table A-15. Test Object Menus and Menu Selections (Sheet 1 of 2)

Menu	Selection	Accelerator Key	Function
Test	Save	Ctrl-S	Saves the Test. Information in all views is saved.
	Print	Ctrl-P	Displays the Print dialog with which you can print the Test.
	Run	Ctrl-R	Displays the Run Script dialog.
	Record Manual Run		Allows you to record results for a manual run.
	Close	Ctrl-F4	Closes the object.
Selected	New Step		Adds a new step to a StepList view.
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected item and places it on the clipboard.
	Copy	Ctrl-C	Copies the currently selected item and places it on the clipboard.
	Copy Object	Ctrl-B	Copies the Test to the clipboard (it can then be pasted into a folder or into another objects Association view).
	Paste	Ctrl-V	Pastes the contents of the clipboard into the field that has focus.
	Delete	Del (key)	Deletes the highlighted item.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains a fresh copy of the object from the database and updates all views.

Table A-15. Test Object Menus and Menu Selections (Sheet 2 of 2)

Menu	Selection	Accelerator Key	Function
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

A.16 Test Hierarchy Object

Table A-16. Test Hierarchy Menus and Menu Selections (Sheet 1 of 2)

Hierarchy	Selection	Accelerator Key	Function
Hierarchy	Save	Ctrl-S	Saves the Hierarchy.
	Close	Ctrl-F4	Closes the object.
Selected	New Test		Displays a sub-menu with which you can create a new Test.
	Duplicate		Creates a new Test object from the selected Test object.
	Expand and Fully Open	Ctrl-O	Opens a selected (highlighted) object.
	Run	Ctrl-R	Displays the Run dialog with which you can run scripts.

Table A-16. Test Hierarchy Menus and Menu Selections (Sheet 2 of 2)

Hierarchy	Selection	Accelerator Key	Function
Edit	Undo	Ctrl-Z	Available in future releases.
	Cut	Ctrl-X	Cuts the currently selected object and places it in a clipboard.
	Copy	Ctrl-C	Copies the currently selected object to a clipboard.
	Copy Object	Ctrl-B	Copies the Hierarchy to the clipboard (it can then be pasted to a Folder).
	Paste	Ctrl-V	Pastes contents of buffer below selected object. If nothing is selected, places object in the first position in the List view.
	Delete	Del (key)	Deletes selected object.
	Deselect All	Ctrl-\	Deselects all the objects in the List view
	Select All	Ctrl-/	Selects all the objects in the List view.
	Duplicate		Duplicates objects.
View	Change View		Displays a list of available views so you can change the current view.
	Refresh		Obtains fresh copy from database.
Tools	Database Browser		Displays the Database Browser with which you can find objects in the MYNAH database.
	Job Status		Displays the Job Status with which you can see the status of scripts submitted for execution.
	Script Builder		Displays the Script Builder with which you can develop scripts.
	Start GUI Test Tool		Starts the GUI Test Tool that you have specified in your preferences.
Windows	Window List		Displays a list of the windows you have currently open. Clicking on one of the listed Windows will cause it to be brought to the front of your display.

Appendix B: MYNAH Objects - Their Attributes and Associations

This appendix provides an overview of associations in the MYNAH database and also contains a complete listing of the MYNAH objects, their attributes and the other objects with which they can be associated. After Appendix B.1, which provides an overview of MYNAH associations, the sections are organized alphabetically by object name. Each object section has two tables:

- One listing attributes for the object.

- One listing associations for the object.

B.1   Overview of Associations

Figure B-1 shows all the associations that a user can make between objects. These are the associations that are visible on the **Associations** view of individual objects. None of these associations are required by the MYNAH system. The user decides when a particular association is appropriate.
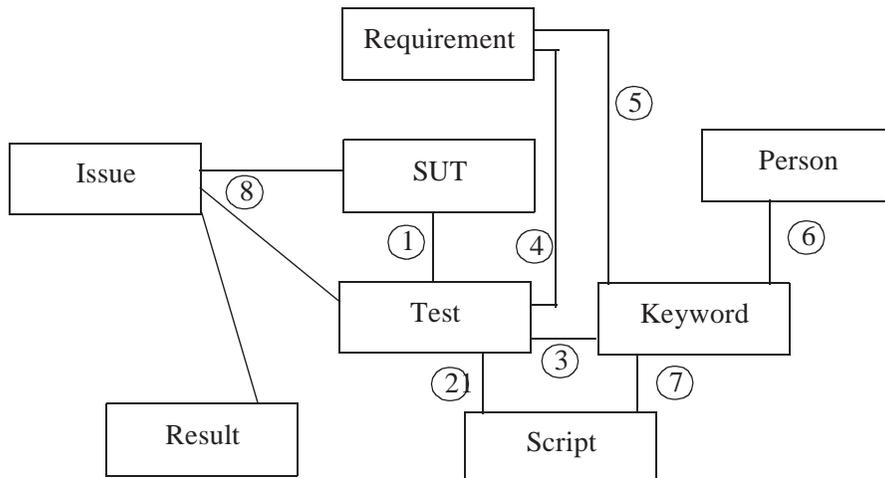
Figure B-1. User-Defined Associations

1. A user associates a Test with the SUT(s) for which the Test is valid.

2. A user associates a Test with the Script that implements the Test.

3. A user associates a Test with the Keyword(s) that characterize the Test.

4. A user associates a Requirement with the Test(s) that validate the Requirement.

5. A user associates a Requirement with the Keyword(s) that characterize the Requirement.

6. A user associates a Person with the Keyword(s) that characterize the Person.

7. A user associates a Script with the Keyword(s) that characterize the Script.

8. A user associates an Issue with the SUT, Test and/or Result that is impacted by the Issue, or that discovered the Issue.

Figure B-2 shows the associations that MYNAH automatically generates when a Script is run in the background (BEE).
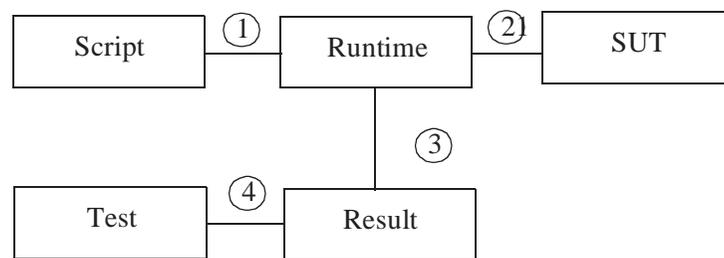
Figure B-2. Automatic Associations When Script is Run

1. When a Script is run, a Runtime is automatically created and associated with the Script. This association is visible on the Script object's **Run History** view.

2. The Runtime is automatically associated with the SUT that is populated on the Run dialog.

3. If the Script is associated with one and only one Test, or, if the Script contains an **xmyBegin** statement, then a Result is automatically created and associated with both the Runtime and the Test. The association between the Result and the Test is visible on the Test object's **Results** view.

If the Script contains more then one **xmyBegin** statement, a Result is automatically created for each referenced Test, and each Result is automatically associated with the Runtime.

Figure B-3 shows the associations that are automatically generated when a **Record Manual Run** for a Test is performed by the user.
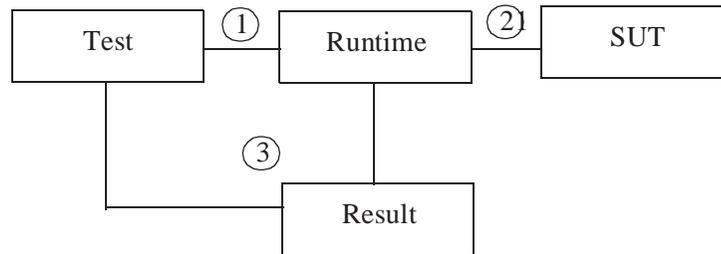
Figure B-3. Automatic Associations with Record Manual Run

1. When a user performs a **Record Manual Run** a Runtime object is created to record the date, time and who information.

2. The Runtime is automatically associated with the SUT that is populated on the Record Manual Run dialog.

3. A Result is automatically created to record the result information and the Result is automatically associated with both the Runtime and the Test. The association between the Test and the Result is visible on the **Results** view of the Test.

In addition to the automatic associations discussed above, the system also automatically creates associations with Person objects. This is done for the Owner, Created By, and Revised By attributes of other objects.

Also, for hierarchies, the system automatically creates associations between parent and child objects.

The following table lists all of the automatic system-generated associations.

Table B-1. System Associations

Object	Possible Automatic Association
Person	Created By, Revised By and Owner fields for other objects
Result	Runtime, Test
Runtime	Result, Script, SUT, Test
Test Hierarchy	Test
Test	Parent and Child Test, Step List
Step List (Test)	Step
SUT Hierarchy	SUT
SUT	Parent and Child SUT

MYNAH System Users Guide   **BR 007-252-006**
MYNAH Objects - Their Attributes and Associations   **Issue 3, October 1997**
Release 5.2

B.2 Attributes Common to All Objects

There are a few attributes that are common to all objects. They appear in the Status area of an object's Properties view. Table B.2 lists these.

Table B-2. Attributes Common to All Objects

Attribute	Description
Created By	Name of the person who created the object.
Revised By	Name of the person who updated the object.

B.3 Folder

Folders provide you with a way to organize other objects. You can place Issue, Keyword, Requirement, and Script objects into a Folder. You can also place Test or SUT hierarchies in a Folder. (Note: Folders are the only objects that **do not** appear in the database.)

Table B-3. Folder Object Attributes and Associations

Attribute	Valid Values	Description	Found on View
Name	variable	The name you give to a folder.	Properties

B.4 Issue

Issue objects help you to document any test-related issues that occur in the course of testing.

Table B-4. Issue Object Attributes

Attribute	Valid Values	Description	Found on View
Name	variable	The name you give to an issue.	Properties
External ID	path to external file	Identifies a file external to the MYNAH System where the issue is stored.	Properties
Type	General, Environment, Software	Identifies the area that the issue affects. Your System Administrator can change these values.	Properties

Table B-4. Issue Object Attributes

Attribute	Valid Values	Description	Found on View
Current Status	Open, Closed	Whether an issue still needs to be resolved (**open**) or has been resolved (**closed**).	Properties
Owner	N/A	Identifies the person who owns the issue.	Properties
Description	N/A	Free-form data entry area where you can enter comments about the issue.	Properties

Table B-5. Issue Object Associations

Associated with:	Where the Association is made
SUT	Issue
Test	Issue
Results	Issue
Requirement	Issue

B.5   Keywords

Keyword objects provide you with a way to reference a number of different objects needed for a specific task.

Table B-6. Keyword Object Attributes

Attribute	Valid Values	Description	Found on View
Name	variable	The name you give to a keyword.	Properties
Type	General, Wire Center, Configuration, Feature, Tests, cycle, tester.	Identifies what the keyword represents. Your System Administrator can change these values.	Properties
Owner	N/A	Identifies the person who owns the keyword.	Properties
Description	N/A	Free-form data entry area where you can enter a description of the keyword.	Properties

Table B-7. Keyword Object Associations

Associated with:	Where the Association is made
Test	Test
Requirement	Requirement
Script	Script
Person	Person

B.6   Person Object Attributes and Associations

Person Objects identify MYNAH users. Your System Administrator creates and deletes these objects, but you can change information in your own person object. You access person objects from the Database Browser.

Table B-8. Person Object Attributes

Attribute	Valid Values	Description	Found on View
UNIX ID	The User's UNIX login ID	Identifies the user to the system and other users.	Properties
Last Name	User's last name	Identifies the user to the system and other users.	Properties
First Name	User's first name	Identifies the user to the system and other users.	Properties
MI	User's middle initial	Identifies the user to the system and other users.	Properties
Email	User's e-mail address	For communication among users.	Properties
Phone	User's phone number	For communication among users.	Properties
Authority	Administrator, User, Inactive	Defines user's authority within the MYNAH System.	Properties

Table B-9. Person Object Associations

Associated with:	Where the Association is made
Keywords	Person

B.7   Requirements Object Attributes and Associations

Requirements objects store information about test or scripting requirements in the database.

Table B-10. Requirements Object Attributes

Attribute	Valid Values	Description	Found on View
Name	variable	The name you give to the Requirement	Properties
Type	General	Identifies the functional area the requirement covers. Your System Administrator can add or change, but not delete to these values.	Properties
Description	variable	Free-form user entered description.	Properties

Table B-11. Requirements Object Associations

Associated with:	Where the Association is made
Documents (available in future release)	Documents
Tests	Tests
Keywords	Requirements

B.8   Results Object Attributes and Associations

Results object attributes store information about the results of the execution of a script, or the results of a manually executed test. The system creates them automatically when you execute a test script, or you may create them for test that you cannot automate.

Table B-12. Results Object Attributes (Sheet 1 of 2)

Attribute	Valid Values	Description	Found on View
Test	Test name and ID.	System assigned ID number, test hierarchy and path, and name for the test the results are for.	Properties
Generated By	User name	Name of the user who ran the script that generated the result, or created the result.	Properties
Label	variable	User assigned label for the result.	Properties
SUT	SUT name and ID	System assigned ID number, SUT hierarchy and path, and name for the SUT the results are for.	Properties
Script	Script name and ID.	System assigned ID number, script name, location of the script file in the UNIX system.	Properties
Run	Script run that generated the result.	System assigned ID number, status, when run was initiated.	Properties
Activity State	complete, resubmit required, execution incomplete, analysis required	Select or view the current status of activities needed to complete testing. Your system administrator can change the values for this attribute.	Analysis
Test Status	inconclusive, successful, unsuccessful.	Select or view the current status of test the results are for. Your system administrator can change the values for this attribute.	Analysis
Reason Code	variable	Reasons for the analysis judgment. Your system administrator can change the values for this attribute.	Analysis

Table B-12. Results Object Attributes (Sheet 2 of 2)

Attribute	Valid Values	Description	Found on View
Issue	variable	Listing of issues associated with the test the results are for.	Analysis
Comments	variable	User entered free-form comments.	Analysis
Successful	number	Number of successful compares	Compares
Warning	number	Number of warnings about compares	Compares
Failed	number	Number of failed compares	Compares
Labelled Compare	Label Name, Status	Identifies labelled compare and reports their current status.	Compares
Child Results	ID, Label, Status, Status Cause, State, Script	Identifies and shows the status of results for any child scripts associated with the script the results object is for.	Compares

Table B-13. Result Object Associations

Associated with:	Where the Association is made
Issues	Issue

B.9   Runtime Object Attributes and Associations

The system creates Runtime objects every time you execute a script. You access Runtime objects from the Database Browser or from the Run History view of a Script object.

Runtime objects contain results from a run. Refer to Results objects for a description of Results attributes contained in a Runtime object. See AppendixB.8.

Table B-14. Runtime Object Attributes

Attribute	Valid Values	Description	Found on View
Run ID	variable	System assigned run identifier.	Properties
Script	variable	Identifies the script that produced the runtime object.	Properties
Created By	user	Identifies the user who ran the script.	Properties
Began At	time/date	When the script started execution.	Properties
Ended At	time/date	When the script ended execution.	Properties
SUT	variable	The SUT against which the script was run.	Properties
SUT Hierarchy	variable	The hierarchy where the target SUT resides.	Properties
SUT Path	variable	The path to where the target SUT resides.	Properties
Script Dispatcher	variable	SD which scheduled the script for execution.	Properties
Script Engine Group	variable	The SE group which executed the script.	Properties
Parameters	variable	Displays user defined parameters used during the script run.	Properties

No user defined associations for Runtime objects.

B.10   Script Object Attributes and Associations

Script objects store information about scripts used to execute tests.

Table B-15. Scripts Object Attributes

Attribute	Valid Values	Description	Found on View
Owner	user	User who owns the script. It may be the user who created it, but not necessarily.	Properties
Logical Name	variable	User assigned name for the script object.	Properties
File Name	variable	Name of the UNIX file that contains the script code.	Properties
Description	variable	User-entered, free-form, description of the script.	Properties
Script Language	MynahTcl, 4Test, Tcl, Replay Tcl	Language the script is written in.	Properties
Expected Duration	hours, minutes, seconds	The expected duration of the script run.	Run Settings
Script Dispatcher	SD name	SD the script is assigned to.	Run Settings
Script Engine	SE name	SE the script is assigned to.	Run Settings
RunTime	Run Time objects	Identifies Runtime objects created by the script's execution. Includes Run ID, Status, Executed At and Ended At information.	Run History

Table B-16. Script Object Associations

Associated with:	Where the Association is made
Tests	Tests
Keywords	Script

B.11 SUT Object Attributes and Associations

SUT object attributes store information that define a System-Under-Test.

Table B-17. SUT Object Attributes

Attribute	Valid Values	Description	Found on View
Owner	user name	User who owns the SUT object. Not necessarily the user who created it.	Properties
Name	variable	User defined name for the SUT.	Properties
Path	path	Location of the SUT in a SUT Hierarchy.	Properties
Type	Release, Platform, Cycle	The type of SUT based on testing cycle. The values for this attribute can be set by your System Administrator.	Properties
Current Status	Active, Idle	Current testing status. SUT is actively being tested (active) or awaiting testing (idle.)	Properties
Parent Type	Release, Platform, Cycle	If the SUT is a child of another SUT, the type of the parent SUT based on testing cycle.	Properties
Planned Begin On	month, day, year	Planned start date for testing a SUT.	Schedule
Planned End On	month, day, year	Planned end date for testing a SUT.	Schedule
Actual Begin On	month, day, year	Actual start date for testing a SUT.	Schedule
Actual End On	month, day, year	Actual end date for testing a SUT.	Schedule
Comment	variable	Free-form data entry area for user comments on a SUT schedule.	Schedule

Table B-18. SUT Object Associations

Associated with:	Where the Association is made
Issue	Issues
Documents (available in future release)	Documents
Tests	Tests

B.12   SUT Hierarchy Object Attributes and Associations

SUT Hierarchies help you organize Systems-Under-Test. They are the only place in the GUI where you can create and store SUT objects.

Table B-19. SUT Hierarchy Attributes

Attribute	Valid Values	Description	Found on View
Name	variable	User defined name for the SUT Hierarchy.	Properties

SUT Hierarchies have no user defined associations.

B.13   Test Object Attributes and Associations

Test objects document tests.

Table B-20. Test Object Attributes (Sheet 1 of 2)

Attribute	Values	Description	Found on View
Owner	user name	User who owns the Test object. Not necessarily the user who created it.	Properties
Name	variable	User-defined name for the Test	Properties
Version	number	Version number of the test (multiple versions will be supported in future releases).	Properties
Path	path	Location of the Test in a Test Hierarchy.	Properties

Table B-20. Test Object Attributes (Sheet 2 of 2)

Attribute	Values	Description	Found on View
Type	Area, Feature, Test, Test Case	They type of Test based on testing cycle. The values for this attribute can be set by your System Administrator.	Properties
Description	variable	Free-form data entry area where users enter a description of the Test object.	Properties
Parent Type	Area, Feature, Test, Test Case	If the Test is a child of another Test, the type of the parent Test.	Properties
Comment	variable	Free-form data entry area where users enter comments about a Test object.	Properties
Priority	High, Medium, Low	The relative priority of this test when compared to other tests.	Settings
Test Importance	number	Relative importance of this test to the test plan when compared to other tests.	Settings
Test Effort	number	Amount of effort the test will take.	Settings
Script	script name	Script that implements the test. Includes script ID, Name, Owner, Language, and Filename.	Settings
Script State	not started, complete, partial.	Describes how developed the script that implements the test is.	Settings
Dependencies	variable	Free-form data entry area where users enter a description of dependencies concerning the test.	Settings
Results	Result object listing	Displays Result object listing for results for the test. Includes ID, Date/Time, Result, SUT hierarchy and path, SUT name.	Results

Table B-21. Test Object Associations

Associated with:	Where the Association is made
Issue	Issue
SUT	Test
Keyword	Test
Requirements	Test (Documentation View)
Script	Test (Settings View) Job Status

B.14   Test Hierarchy Object Attributes and Associations

Test Hierarchies help you organize Test objects. They are the only place in the GUI where you can create and store Test objects.

Table B-22. Test Hierarchy Attributes

Attribute	Valid Values	Description	Found on View
Name	variable	User defined name for the Test Hierarchy.	Properties

Test Hierarchies have no user defined associations.

Glossary

A

AID Key — Any 3270 special program key which causes the current screen to be sent to the 3270 SUT, and causes the SUT to transmit screen data back to the client.

Aggregate — A term used to identify a portion of a Flexible Computer Interface Format (FCIF) message. An aggregate contains zero or more tag-value pairs and is contained in an FCIF section.

App-to-App — Package that allows a user to send, receive, and analyze messages to and from a SUT over an application-to-application interface or a binary synchronous printer interface.

Array — A collection of associated variable elements.

Asynchronous Terminal Interface — An interface to an operating system or application that sends and receives data in arbitrarily-sized blocks at arbitrary times.

Attributes — **1.** The values defining the characteristics of a class or a class's connection, e.g., blinking, highlighted. **2.** The sub-commands used to specify or return attribute values. **3.** A category of methods and attributes that are used to find information about the SUT's configuration characteristics.

B

Background Execution Environment — The combination of the Script Dispatchers, the Script Engine Groups, and the associated SEs.

Background Script Engine — An SE process that communicates over an channel to a controlling process.

BD — See Boot Daemon

BEE — See Background Execution Environment

Binary Synchronous Communication — An IBM communications protocol that provides access to a 3270 data stream.

Boot Daemon — A platform process required by an SD that manages the SEs running on its machine.

BSC — See Binary Synchronous Communication

BSE — See Background Script Engine

C

Character Position — A manner in which screen positions are referred to. The screen can be viewed as one long string, where the indices of that string map to a position on the screen. For instance, the first value of the string has a character position of 1, which would have a corresponding row/column value of {1 1}. The maximum character position, or the last position on the screen, varies from model type to model type, as different model types have different screen sizes. Model 2's max character

position is (24 x 80) 1920, while the bigger Model 5 has a max character position of (27 x 132) 3564.

Child Script — A Tcl script that is submitted for execution by a parent script

Class — A specific area or category of functionality.

Class Command — A command that gives you control over a class or category of functions.

Clear Tag-value Database — An ASCII file containing two columns separated by spaces or tabs. The first column contains the tag, and the second column contains the values.

CLUI — See Command Line User Interface

Concatenate — To put two items together, end to end. For example, if you concatenated the strings "uvw" and "xyz", you get "uvwxyz". If you concatenate two files, the new file contains the contents of both files, presented sequentially.

Concurrency Group — The set of all scripts that run on one Script Dispatcher

Config file — The MYNAH Configuration File (named *xmyConfig*) that resides in the directory *$XMYHOME/config*.

D

des — A UNIX command to encrypt or decrypt data using the Data Encryption Standard.

Domain — A type of interface provided to the System Under Test. Examples of domains are the asysnchronous terminal interface of an application, the application-to-application interface of an application, and the synchronous printer interface of an application.

Domain Connection — Any specific input/output interface to a SUT.

E

EAB — See Extended Attribute Bytes, used in 3270 to provide more information about a field, such as color attributes.

EHLLAPI — See Emulator High Level Language Application Programmatic Interface.

Elements — Components of a list or array.

Embedded Script Engines — Script Engines that graphically display the screens associated with Term3270 or TermAsync Packages. Embedded Script Engines (ESEs) offer script execution functionality through class methods. ESEs are not separate processes.

Emulator High Level Language Application Programmatic Interface — The IBM specification API for interacting with a 3270 host providing the essential functionality underneath the MYNAH 5.0 3270 Terminal domain.

Encrypted Database — A clear tag-value database that has been encrypted using **des**

ESE — See Embedded Script Engine

Exception — Any event that can abort a script.

Extended Attribute Bytes — Used by the Term3270 Package to provide more information about a field, such as color attributes.

Extended Tcl — See TclX.

Extensions — Commands and procedures that expand Tcl's capabilities.

F

FCIF — Flexible Computer Interface Format. FCIF is a text format developed at Bellcore for communicating messages between processes.

FMM — See Flexible Message Manager (TraxWay-provided wrapper to Telexel IPC)

Flexible Message Manager — A TraxWay-provided wrapper to Telexel IPC. An inter-process communication module used by the MYNAH System that uses the Telexel directory daemon underneath to do the actual IPC processing.

Focusing — Selecting a MYNAH GUI element and making it ready for you to enter information.

G

Global Array — An array of elements that are available to all domains.

GUI — Graphical User Interface.

H

Handle — A reference to an instance.

hllc() — The native EHLLAPI call. EHLLAPI makes use of one function, the hllc() function, which always takes four parameters. These four parameters determine what EHLLAPI function to execute, the input parameters to that function, and, afterwards, return the output of that function's execution, should there be any. The four parameters are commonly referred to as (and passed to the hllc() function in this order): Function Number, Data String, Data String Length or Buffer Size, and Presentation Space. This design refers to the specific EHLLAPI function simply as hllc(*function_number*). For instance, the EHLLAPI function Connect Presentation Space corresponds to hllc(1).

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.

Instances — Connections made to SUTs using a class command.

IPC — Inter-process Communication Telexel IPC processes

J

Job Status Container — MYNAH GUI tool used to monitor the scripts that have submitted to the BEE.

L

List — An ordered collection of elements.

Log File — A file containing a record of activity for a software product.

M

Mask — A way to identify data that will be ignored during a comparison.

Methods — Sub-commands used to perform particular actions on instances you create in Tcl.

MYNAH System — An advanced software environment that can be used in all phases of software testing to exercise and analyze mainframe, minicomputer, and workstation applications. The MYNAH System can also be used for task automation and rapid application development.

O

OA — See Operability Agent

OM — See Operability Manager

Operability Agent — A MYNAH process that manages all MYNAH required processes on a host, communicating the start, stop and status requests to individual processes and then communicating the reply back to the OM.

Operability Management — The MYNAH mechanism, consisting of the OA and OM, that lets the MYNAH Adminstrator start, stop, and get status of all of the MYNAH processes from any host.

Operability Manager — A set of commands used interact with an OA to manageall MYNAH processes. You can start or stop a process or you can determine if a process is running statis

P

Parent Script — A Tcl script that submits other scripts for execution.

-position *position* — One of the ways of specifying screen location to a 3270 Tcl command. The position is a list of two integer values, row and column in that order. Example: -position {1 1}.

Presentation space — The 3270 screen that the EHLLAPI function call will affect and/or perform its action upon.

PRINTCOM — A program that interfaces to applications on a host computer over a binary synchronous communication line, receiving and capturing the printer output sent by the applications.

Process — An executable program that is active (running).

Prt3270 — MYNAH Package that lets a user simulate PRINTCOM processes.

R

Regression Testing — The testing of a previously verified application after changes have been made to the application.

Requesting Process — A process that sends a script-execution request to the SD (this does not include an SE sending a child-script-execution request to the SD).

Resource, X-Window System — A default value that can be changed by a user. Sets of resources are commonly stored in the *~/.Xdefaults* file (i.e., the file called ".Xdefaults" in your home directory).

Root Script — A script submitted to an SD via the GUI, CLUI, CSE, or an embedded SE, but not from one of the SEs that is controlled by the SD.

Runtime — A state in which a script is being executed.

Runtime Analysis — data Analysis that occurs during the execution of a test, and provides verification that the application being tested performed as expected. An example of runtime analysis is a comparison statement in a test script.

S

Screen Definition File — A file containing a tag name table. One file exists for each screen in a user's application. The file may contain other information in addition to the tag name table. The tag name table is delimited in this file by "begin" and "end" statements.

Screen IDs File — A file containing the names of screens and other information to uniquely identify one screen from another.

Script — A file that contains one or more instructions to be performed by a domain.

Script Dispatcher — MYNAH process that provides scheduling and concurrency control for background execution of user scripts.

Script Engine — An extended Tcl interpreter that runs user scripts.

Script Engine Group — A logical set of BSEs controlled by an SD that all run on one host and run in the same mode. When a script is submitted to the BEE, it is submitted to run in a particular SE Group. It will run on one of the SEs in that group, but it doesn't matter which SE in the group it runs on.

Script Builder — A MYNAH GUI object used to create script code by capturing interactions with a system, importing templates and procedures, and existing script code. The Script Builder can be run independently of the rest of the MYNAH GUI.

Script Object — A MYNAH GUI object used to create and track script code.

SD — See Script Dispatcher.

SE — See Script Engine.

SNA — See Systems Network Architecture.

Standalone Script Engine — A MYNAH process that accepts Tcl commands from **stdin** and produces results on **stdout**. The standalone SE does not interface with the MYNAH System database, but does, however, open a channel.

String — In Tcl: A set of characters that represents the current value of a variable. In some cases, strings show what will appear on the screen or in a printout.

SUT — System Under Test.

Symbol Table — User-supplied data associated with a script. Symbol tables contain symbol-value pairs; they can be read and modified by the script during execution.

Synchronous Terminal Interface — An interface to an operating system or application that sends and receives data in blocks of predefined size at regular intervals.

Systems Network Architecture — An IBM communications protocol that provides access to a 3270 data stream.

System Under Test — The system you wish to test or which contains the application you wish to automate.

T

TagDir — A directory containing Tag Name files.

Tag Table — A formatted table in a screen definition file, containing screen information for a single screen. For each user-identified screen field this table has a name for the field (called a tag name), the field's row and column location, and the number of characters in the field.

Tag Name File — A file containing tag-value pairs, used for locating items on a synchronous screen.

Tag-value Pair — A pair of items, the first being a variable, the second being the value of that variable. Tag-value pairs reside in a symbol table.

Tags — User defined lables used by the Term3270 Package for locating items on a synchronous screen.

Tcl — Tool Command Language. An interpretive programming language, implemented as a library of C procedures, developed by John Ousterhout. Tcl is the basis for the MYNAH scripting language.

TclX — Extended Tcl, flavor of Tcl used by the MYNAH System under license from NeoSoft.

Term3270 — Package that performs 3270 synchronous terminal emulation, allowing users to build scripts that simulate an interactive work session with a SUT.

TermAsync — Package that performs asynchronous terminal emulation, allowing users to build scripts that simulate an interactive work session with a SUT.

Terminal Emulation — The use of software to emulate a type of hardware terminal (e.g., vt100, 3278).

TOPCOM — A software product that provides an interface to allow an application to establish and accept Transaction Oriented Protocol (TOP) sessions with a foreign partner, to send and receive messages to and from the partner, and to terminate the sessions. TOPCOM uses X.25 or TCP/IP transport services to transport the application data messages and TOP protocol messages between the two partners.

TOP — Extension Package that lets a user simulate TOPCOM processes.

V

Variable — A user defined quantity that can assume a value.

Index

T