Note

The following is an excerpt from the book *Introduction to Application Development* published by Digital Equipment Corporation.

6.2 Testing the Application with VAX DEC/Test Manager

Each time you modify part of an application, it is possible that the change will alter the way the application runs. These unexpected changes are impossible to predict and often difficult to uncover.

The best method for detecting such changes is to catch them as soon as they occur so that you can identify which modification caused the change. To detect changes quickly, you need to continuously test the application and compare its actions against expected or past results. This type of testing is called regression testing.

Testing database applications that perform extensive terminal I/O is a very tedious and time-consuming process. Often someone must sit at a terminal and enter a standard set of data to exercise the application tasks. However, you can automate the entire process by using the VAX DEC/Test Manager.

The following sections provide an introduction to the DEC/Test Manager. See the *VAX DEC/Test Manager User's Guide* for a complete description of the DEC/Test Manager.

6.2.1 Creating DEC/Test Manager Test Libraries

DEC/Test Manager is a tool for creating, organizing, and performing tests of software applications. There are three main components to a DEC/Test Manager test system:

- The DEC/Test Manager library stores information about tests.
- The DEC/Test Manager test description describes an individual test and how it is run.
- The DEC/Test Manager collection identifies a collection of tests to be run as a group.

The main repository for information about DEC/Test Manager tests is the DEC/Test Manager library. It is much like a CMS library and is created in the same manner. First you create a VMS directory to use as the library, and then you create the necessary control files by using the DTM CREATE LIBRARY command.

Note

The directory you use as the DEC/Test Manager library is reserved for DEC/Test Manager. You should never attempt to create or access files in the directory except through the use of DEC/Test Manager commands.

As with CMS libraries, it is a good idea to create a new DEC/Test Manager library for each application. Following the same procedure described for CMS in Chapter 2, you can create a top-level directory to act as a root directory and create subdirectories for each DEC/Test Manager library. For example:

```
$ CREATE/DIR DUAO: [DTMLIBS]
$ CREATE/DIR DUAO: [DTMLIBS.AVERTZ]
$ CREATE/DIR DUAO: [DTMLIBS.PERSONNEL]
$ DTM CREATE LIBRARY DUAO: [DTMLIBS.AVERTZ]
_Remark: DEC/Test Manager library for the car rental application.
$ DTM CREATE LIBRARY DUAO: [DTMLIBS.PERSONNEL]
_Remark: DEC/Test Manager library for the personnel application.
```

Every time DEC/Test Manager runs a collection of tests, it creates files that store the results of each test and a comparison of the test results with the set of expected results. These files are stored in subdirectories within the DEC/Test Manager library.

6.2.2 Creating a DEC/Test Manager Test Description

Before DEC/Test Manager can run a collection of tests, it needs to know how each test should be run. You describe a test to DEC/Test Manager in a test description, using the DTM CREATE TEST_DESCRIPTION command. There are two primary files associated with a DEC/Test Manager test description:

- The template or session file

  The template file describes the test. For simple tests, the template file can be a DCL command procedure that runs the application.

- The benchmark file

  The benchmark file describes the expected output for the test.

You can create the template and benchmark files with a text editor and identify them with the /TEMPLATE and /BENCHMARK qualifiers to the DTM CREATE TEST_DESCRIPTION command. However, for interactive database applications it is much easier to let DEC/Test Manager create these files for you by using the /RECORD qualifier.

The /RECORD qualifier tells DEC/Test Manager to record the current terminal session and use it to create the template and benchmark files. In other words, you can simply run the application entering the data you wish to test, and DEC/Test Manager captures the entire terminal session as a test description.

For example, to create a test description for testing the personnel application's raise/promotion update task, you use the CREATE TEST_DESCRIPTION command with the /RECORD qualifier and specify a name for the test description:

```
$ DTM CREATE TEST_DESCRIPTION/RECORD PERS_UPDATE_RAISEPRO_TEST
_Remark: This test exercises the update raise/promotion task.
```

DEC/Test Manager then issues messages telling you it is creating a template and benchmark file, and tells you how to end the test session:

```
%DTM-I-BEGIN, your interactive test session is now beginning ... Type CTRL/P twice to
terminate the session.
```

At this point you can run the application, entering the commands and data you want the test to use. For example:

```
$ ACMS/DEBUG PERS_APPL:PERS_TASK_GROUP      ! Start the ACMS debugger.
ACMSDBG> START/ALL                          ! Start the server process.
ACMSDBG> SELECT PERS_UPDATE_RAISEPRO_TASK   ! Choose a task to perform.
```

When you are done, stop the server and exit the ACMS debugger. When you return to the DCL prompt, press CTRL/P twice to end the test session and wait for DEC/Test Manager to create the test description and the session and benchmark files.

However, before recording a DEC/Test Manager test session, you should make sure that you will be able to reproduce the test results that are being recorded. Two items you should consider are:

- The process environment
- The database environment

The following sections discuss these items.

6.2.2.1 Setting Up the Process Environment – When using DEC/Test Manager to test database applications, you should make sure that the interactive process in which you are creating the test description can be reproduced when the test is run. Several important points you should consider are:

- DEC/Test Manager does not reproduce any delays between individual inputs from the terminal. Therefore, it cannot accurately test any applications that rely on asynchronous input.

  For example, DEC/Test Manager does not reproduce the type-ahead feature that is available with many terminals. However, some programs and utilities take advantage of this feature to optimize their actions. To ensure that the test will run identically under all conditions, you should disable type-ahead before creating the test description. Use the SET TERMINAL/NOTYPE_AHEAD command to turn off the type-ahead feature.

- Once a test description is created, you can run the test in either interactive or batch mode. Make sure you create the test description in a process environment that matches the environment for batch mode processing. Some features that can result in differences between running tests in interactive and batch mode are:

  - The use of the SET PROMPT command. Use the SET PROMPT command without a string parameter to set the DCL prompt to the default.

  - Logical name definitions. Be sure that any logical name definitions you use to run the application are defined correctly in batch mode processes. (You can include the logical name definitions in your LOGIN.COM file.)

- Finally, you can run ACMS applications under the DEC/Test Manager either by invoking the ACMS Task Debugger or by logging into the ACMS terminal subsystem. However, to create or run a DEC/Test Manager test that invokes the ACMS terminal subsystem, you must be using a SYSTEM account.

6.2.2.2 Setting Up a Test Database – In addition to the template and benchmark files DEC/Test Manager uses to run the test, you can also specify two optional files that affect how a test runs:

- The prologue file

  The prologue file is a DCL command file that DEC/Test Manager executes just before running the test. You can use the prologue file to define any necessary logical names and to create the proper environment for the test.

- The epilogue file

  The epilogue file is a DCL command file that DEC/Test Manager executes immediately after running the test. You can use the epilogue file to alter the output of the test or to remove any intermediate files that the test might create.

Prologue files are particularly useful for database applications because you can use them to create a test database or test records. You can then use the epilogue file to purge the database of any test records once the test is complete. There are two reasons for using a test database:

- It protects you from corrupting important information that might be in the database.

- It ensures that any changes DEC/Test Manager detects in the test results are due to the application, not due to changes in the database.

For example, if you are testing the raise/promotion update task for the personnel application, you might run a test that updates an employee's salary from its current level (say, $10,000) to $12,000. The first time you run the test, when you select the employee record, the salary field says $10,000. However, the second time you run the test, the salary field contains the revised sum $12,000, not the previous salary of $10,000. The two test results will be different and DEC/Test Manager will indicate that the second test was unsuccessful.

To avoid such differences, you need to create a "clean" test database for the duration of the test. You can use the DEC/Test Manager prologue file to create new EMPLOYEES, SALARY_HISTORY, and JOB_HISTORY records using the RDO utility. For instance, the command file in Example 6-3 creates three test records in the personnel application.

```
$ RDO
INVOKE DATABASE FILE PERS_APPL:PERSONNEL
START_TRANSACTION READ_WRITE RESERVING
      EMPLOYEES, JOB_HISTORY, SALARY_HISTORY
      FOR SHARED WRITE
!
! Create a test EMPLOYEES record.
!
STORE E IN EMPLOYEES USING
      E.EMPLOYEE_ID = '00001';
      E.LAST_NAME = 'TESTING';
      E.FIRST_NAME = 'DTM';
      E.ADDRESS_DATA_1 = 'This record is reserved';
      E.ADDRESS_DATA_2 = 'For DTM testing. ';
      E.CITY = 'NH';
      E.POSTAL_CODE = '00000';
      E. SEX = 'M';
      E.BIRTHDAY = '01-JAN-1900';
      E.STATUS_CODE = '1'
END STORE
!
! Create a test JOB_HISTORY record for the employee.
!
STORE J IN JOB_HISTORY USING
      J.EMPLOYEE_ID = '00001';
      J.JOB_CODE = 'DTM';
      J.JOB_START = '0l-JAN-1900';
      J.DEPARTMENT_CODE = 'DTM';
      J.SUPERVISOR_ID = '00001'
END STORE
!
! Create a test SALARY_HISTORY record.
!
STORE S IN SALARY_HISTORY USING
      S.EMPLOYEE_ID = '00001';
      S.SALARY_AMOUNT = '10000';
      S.SALARY_START = '0l-JAN-1900'
END_STORE
COMMIT
EXIT
```

Example 6-3: DEC/Test Manager Prologue File for the Personnel Application

The prologue file in Example 6-3 ensures that the EMPLOYEES record associated with badge number "00001" is consistent for each test run.

Once the test is complete, you need to remove the test records from the database to ensure that the next test runs properly. You can use a DEC/Test Manager epilogue file to remove all the test records and any additional records the test run might have created. The command file in Example 6-4 deletes any records associated with the test EMPLOYEES record.

```
$ RDO
INVOKE DATABASE FILE PERS_APPL:PERSONNEL
START_TRANSACTION READ_WRITE RESERVING
      EMPLOYEES. JOB_HISTORY. SALARY_HISTORY. DEGREES
      FOR EXCLUSIVE WRITE
FOR J IN JOB_HISTORY WITH J.EMPLOYEE_ID = '00001'
      ERASE J
END_FOR
FOR S IN SALARY HISTORY WITH S.EMPLOYEE ID = '00001'
      ERASE S
END_FOR
FOR D IN DEGREES WITH D.EMPLOYEE ID = '00001'
      ERASE D
END_FOR
FOR E IN EMPLOYEES WITH E.EMPLOYEE ID
      ERASE E
END_FOR
COMMIT
EXIT
```

Example 6-4: DEC/Test Manager Epilogue File for the Personnel Application

Once you create the prologue and epilogue files, you can associate them with a test description you are creating by using the /PROLOGUE and /EPILOGUE qualifiers with the CREATE TEST_DESCRIPTION command. You can also add prologue and epilogue files to existing test descriptions by using the MODIFY TEST_DESCRIPTION command.

Remember, you want the session you record to match the test runs that follow, so you should design the test in advance:

1. Plan the features you want to test.
2. Write the necessary prologue and epilogue files.
3. Set up the current process to eliminate unusual command prompts or terminal settings.
4. Invoke the prologue file.
5. Record the test session.
6. Invoke the epilogue file.

Note that you need to invoke the prologue and epilogue files only once, when you record the test session. If you use the /PROLOGUE and /EPILOGUE files with the CREATE TEST_DESCRIPTION command, these files will be invoked automatically when you run the test later.

To organize your test system, you should keep all the files associated with the tests in a central location. If you are using DEC/CMS, DEC/Test Manager lets you store the files in a CMS library. Simply specify the device and directory of a CMS library in the file specification and DEC/Test Manager automatically uses CMS to access the files. For template and benchmark files, you can also specify a CMS library as the default storage location with the DTM SET command:

```
$ DTM
DTM> SET TEMPLATE_DIRECTORY PERS_CMS "Define default template directory"
```

```
%DTM-S-NEWDEF, DUA3: [CMSLIBS.PERS] is the new default collection template directory
DTM> SET BENCHMARK_DIRECTORY PERS_CMS "Define default benchmark directory"
%DTM-S-NEWDEF, DUA3: [CMSLIBS.PERS] is the new default collection benchmark directory
```

6.2.3 Creating a Collection of Tests

Usually each test exercises a single component of the application: a single task or a single module of code. By keeping the scope of each test small, it is easier to identify the problem if a test fails.

Once you create all the tests you want to run, you can create a collection of tests with the DTM CREATE COLLECTION command. The CREATE COLLECTION command takes two parameters: the collection name and a list of tests to include in the collection. For example, the following command creates a collection with tests for each of the tasks in the personnel application:

```
$ DTM CREATE COLLECTION /PROLOGUE=SYS$LOGIN:SET_PERS.COM
_$    PERSONNEL_TESTS -
_$        PERS_ADD_TEST, -
_$        PERS_DISPLAY_TEST, -
_$        PERS_UPDATE_GENERAL_TEST,
_$        PERS_UPDATE_RAISEPRO_TEST,
_$        PERS_UPDATE_TRANSFER_TEST, -
_$        PERS_UPDATE_STATUS_TEST
_Remark: Test each of the personnel tasks
```

In addition to the prologue and epilogue files for each individual test, you can specify a prologue and epilogue for the entire collection. The preceding example uses this feature to invoke the SET_PERS.COM command file described in Chapter 2, which defines all the necessary logical names and symbols for the personnel development environment.

Note that you cannot specify the order in which DEC/Test Manager runs the tests. As a consequence, you cannot depend on one test creating a record that another test uses. Using prologue and epilogue files ensures that the database is in a consistent state for each test.

6.2.4 Running a Collection of Tests

DEC/Test Manager provides two methods for running collections of tests: in the current process or as a batch process. To run a test in your current process, use the DTM RUN command specifying the collection you wish to run:

```
$ DTM RUN PERSONNEL_TESTS
```

DEC/Test Manager executes all the necessary prologue files, test sessions, and epilogue files, displaying each test on your terminal as it is run. After all the tests are completed, DEC/Test Manager compares the results of each test to the expected results stored in the benchmark files.

If you wish to do other work while the DEC/Test Manager tests are running, you can submit the collection as a batch job by using the DTM SUBMIT command:

```
$ DTM SUBMIT PERSONNEL_TESTS /NOPRINT /NOTIFY
```

The SUBMIT command performs the same functions as the RUN command, except that it creates a batch job to do the work, freeing your terminal for other work. You can use many of the same qualifiers

that you use with the DCL SUBMIT command with the DTM SUBMIT command. The preceding example uses the /NOPRINT and /NOTIFY qualifiers.

6.2.5 Reviewing Tests

After you run a collection of tests, you can use the DTM REVIEW command to enter the review subsystem and examine the results of the tests. When you enter the review subsystem, it first displays information about the collection run:

```
$ DTM REVIEW PERSONNEL_TESTS
Collection PERSONNEL_TESTS with 6 tests was created on 11-JUN-1986 12:08:14 by the
command:
        CREATE COLLECTION/PROLOGUE=[AVERTZ]SET_PERS.COM PERSONNEL_TESTS
PERS_ADD_TEST,PERS_DISPLAY_TEST,PERS_UPDATE_GENERAL_TEST,
PERS_UPDATE_RAISEPRO_TEST,PERS_UPDATE_TRANSFER_TEST,PERS_UPDATE_STATUS_TEST "Test each
of the personnel tasks"
        Last Review Status not previously reviewed Success count = 5
        Unsuccessful count = 1
        New test count = 0
        Updated test count = 0
        Comparisons aborted = 0
        Test not run count = 0
DTM REVIEW>
```

To examine individual tests, you can use the SHOW command with a test name or select an individual test using the FIRST, LAST, NEXT, or PREVIOUS commands to step through the collection of tests and then use the SHOW command for the current test. The SHOW command takes several qualifiers, specifying the particular item you want to view. You can examine the results of the test (/RESULT), the expected output (/BENCHMARK), or the differences between the test results and expected results (/DIFFERENCES).

You can also display summary information about tests based on their current status. For example, you can see information about all the tests that were unsuccessful by using the SHOW/UNSUCCESSFUL command.

6.2.5.1 Examining Unsuccessful Tests – Perhaps the most useful command in the review subsystem is the SHOW/DIFFERENCES command. This command displays the differences between the benchmark and the corresponding test result. By default, DEC/Test Manager compares interactive tests screen by screen – with the differences highlighted in reverse video – letting you step through the screens one at a time. To use the SHOW/DIFFERENCES command, position DEC/Test Manager at an unsuccessful test with the NEXT/UNSUCCESSFUL command. Then enter the SHOW/DIFFERENCES command and press the RETURN key to display each screen that contains a difference between the test result and the benchmark.

When DEC/Test Manager first displays the differences for an unsuccessful interactive test, it displays the test result and the expected result at the same time, by dividing the screen into two parts horizontally. You see the top half of the test result on the top half of the screen and the top half of the expected result on the bottom half of the screen. However, you can change the display to suit your particular test using the keys on the numeric keypad. For example:

- The KP3 key lets you switch between displaying the bottom half or the top half of the result and benchmark screens.

- The KP8 key lets you divide the screen vertically instead of horizontally (which is useful for forms that display data in columns).
- The KP9 key lets you display the entire result or benchmark screen.

Figure 6-2 shows one screen from the DEC/Test Manager review subsystem SHOW/DIFFERENCES display with the test result and benchmark separated vertically.

Once you find a screen with different displays for the test and the benchmark, you must decide what caused the change. Three possible causes for the change are:

- The test environment has changed.
- The test database has changed.
- The application has changed.

In Figure 6-2, the department code, job code, and supervisor fields are highlighted on the test result screen. These changes indicate that the application could not find the test record, probably because the test record was not created by the prologue file.

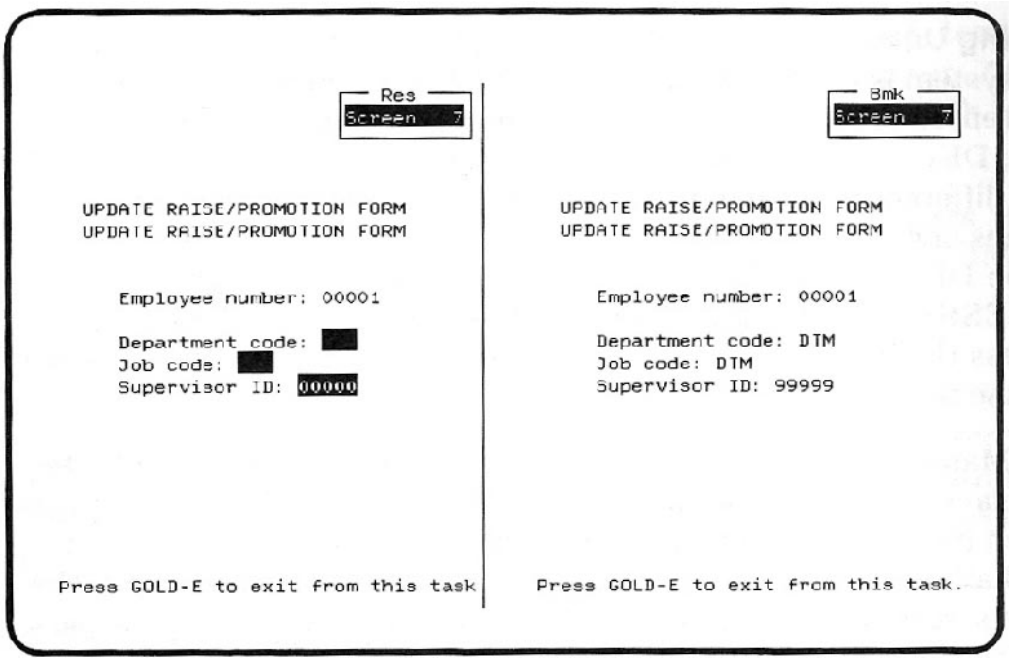

Figure 6-2: DEC/Test Manager SHOW/DIFFERENCES Display

If the change is caused by a change in either the environment or database (as in Figure 6-2), you should go back and examine the prologue and epilogue files to make sure they are setting up the test environment correctly. If the change is caused by a change to the application, you know that the application has regressed. That is, a modification to the application code has altered the way the application acts.

At this point, you should identify the specific modification that caused the regression. If you have been

using CMS and DEC/Test Manager consistently, you can use the CMS SHOW HISTORY/SINCE command to find any application modules that have changed since the collection of tests were last run. You can then use the CMS DIFFERENCES command on the two most recent generations of those elements to identify the specific modifications that might have caused the change.

6.2.5.2 Updating the Benchmark File – It is possible that the change in the test results represents some previously incorrect action that has now been fixed. For example, if you add a new field to the database – and subsequently to a TDMS form – the next test run will identify the new field as a change in the application. In this case, the change is not an error in the application, it is an enhancement. What you need to do is update the benchmark file to identify the new action as the desired and expected action. To update the benchmark file in the review subsystem you use the UPDATE command. The UPDATE command replaces the current benchmark file with the most recent results file:

```
DTM_REVIEW> UPDATE
%DTM-I-UPDATED, the benchmark for PERS_UPDATE_RAISEPRO_TEST has been updated
```

6.2.6 Making Sure You Are Testing the Entire Application

Finally, a test system is only as good as the tests it contains. A common problem is determining how complete a collection of tests is. In the car rental application, the tests must check the application to see that it correctly stores a reservation, checks out a car, and checks it back in. However, the tests should also make sure the application handles any error conditions properly.

One way to make sure that the tests are complete is to design the test system early in the development cycle, possibly as part of the application design. By designing the tests in conjunction with the software design, you can be sure that each application module has an associated set of tests.

Another way to make sure the tests are complete is to use the VAX Performance and Coverage Analyzer (PCA). By linking PCA into the program image and collecting coverage data while the test collection is run, you can identify exactly which lines of code are being executed. If you find any program code that the test collection does not execute, you know that you need to add a new test to the collection to check for that particular condition.

Chapter 7 explains how to use PCA with DEC/Test Manager to test database applications.