U S E R ’ S G U I D E

SMARTS/MSW

Software Maintenance and Regression Testing System

Ver 2.6

SOFTWARE RESEARCH, INC.

This document property of:

Name:_____

Company:_____

Address:_____

_____

Phone_____

SOFTWARE RESEARCH, INC.

625 Third Street

San Francisco, CA 94107-1997

Tel: (415) 957-1441

Toll Free: (800) 942-SOFT

Fax: (415) 957-0730

E-mail: support@soft.com

http://www.soft.com

(Last Update December 18, 1998)

documentation / user-manuals / regression / msw-regression / smarts-msw2.6.Feb97 / smarts_msw2.6.B

Table of Contents

SMARTS/MSW User's Guide

List of Figures

Preface

Congratulations!

By choosing the TestWorks integrated suite of testing tools, you have taken the first step in bringing your application to the highest possible level of quality.

Software testing and quality assurance, while becoming more important in today's competitive marketplace, can dominate your resources and delay your product release. By automating the testing process, you can assure the quality of your product without needlessly depleting your resources.

Software Research, Inc. believes strongly in automated software testing. It is our goal to bring your product as close to flawlessness as possible. Our leading-edge testing techniques and coverage assurance methods are designed to give you the greatest insight into your source code.

TestWorks is the most complete solution available, with full-featured regression testing, coverage analyzers, and metric tools.

Audience

This manual is intended for software testers who are using *SMARTS/MSW*. You should be familiar with the Microsoft Windows System and your workstation.

Typefaces Used

The typographical conventions used in this manual:

boldface Introduces or emphasizes a term that refers to **STW**'s window, its sub-menus and its options.

italics Indicates the names of files, directories, pathnames, variables, and attributes. Italics are also used for manual, book, and chapter titles.

"Double Quotation Marks"

Indicates chapter titles and sections. Words with special meanings may also be set apart with double quotation marks the first time they are used.

`courier` Indicates system output such as error messages, system hints, file output, and *CAPBAK/MSW*'s keysave file language.

`Boldface Courier`

Indicates any command or data input that you are directed to type. For example, prompts and invocation commands are in this text. (**`stw`**, for instance, invokes *TestWorks*.)

CHAPTER 1

Introduction to SMARTS/MSW

In this chapter you will learn the basic functions of *SMARTS/MSW*, how it can help you, and its role in Quality Assurance.

1.1 Automated Testing — An Overview

In the past, application and operating environments were relatively simple. Manual testing or a few written test scripts stored in batch files were usually sufficient to fully exercise the product. Today's applications, however, are much more complex, as are the environments in which they run.

The stages of software production involve multiple versions. Over a single production cycle, software may have to be tested several times. Performing these tests manually usually involves a large investment of time and money.

If an application is automated, each test can be performed automatically, accurately, and often unsupervised each time developers create a new version of the software. Although it does take some time to develop the test operation, this time is more than compensated for during the middle-to-later stages of testing. Considering the resources involved, automating test operation can drastically reduce the overall time needed to test a software product.

1.1.1 Test Planning and Script Writing

The effectiveness and reliability of any tool, manual or automated, depends greatly on the manner in which it is employed. As applications become more complex, planning assumes a more important role in automated testing.

Ad hoc testing was once an accepted and adequate method for uncovering most program errors. A few testers manually verified the product's functionality and then reported any errors to the programmer(s). When a number of bugs had been fixed, the software was shipped.

Today, this kind of testing spells disaster. Many of today's applications contain dozens of user-selectable functions, each of which can have several major and minor options.

Just as software must be developed with an eye towards both reliability and revisions, a testing procedure must mimic this capacity in its ability to verify discrepancies and maintain relevancy throughout the various incarnations of the application under test.

Therefore, the analytical process used to develop an application should also be employed to develop a testing procedure. Before attempting to write test scripts for any type of application, a test plan should be created which addresses the following basic elements:

- Scope of the application to be tested.
- Extent of testing that will be performed.
- Tools that will be required during testing as well as the tasks that each tool will execute.
- Automated methods that will be used.
- Verification methods that will be used.
- Criteria that will determine the program's quality and fitness for distribution.
- Time needed to complete testing.
- Description of the test suites.
- Test data that will be used.

Having created a comprehensive test plan, the parameters and specific goals of the test scripts to be written can now be more concisely defined. Although developing incisive test scripts can often be the most time-consuming phase of the testing process, the effort will be more than compensated by a thorough and accountable testing procedure.

While no application yet exists which can automatically produce scripts based on a testing plan (just as none can automatically produce code based on software specifications), certain tools can facilitate and expedite the testing process. They do this by automating the procedure, providing an effective means of determining discrepancies and monitoring the effects of regression.

1.2 The STW/Regression Solution

Software Research, Inc. offers a solution, *STW/Regression*™, that can automate testing following the test-planning and ATS script-writing process. *STW/Regression* is designed to overcome the tedious and error-prone process of manual testing.

Test outcomes are recorded and compared automatically with baselines. Any discrepancies are recorded and stored for further analysis. Extraneous or irrelevant discrepancies can be discarded in the comparison process. Test execution, reports and statistics are available for viewing. *STW/Regression* improves the overall quality of testing by providing technically sophisticated support for full automation of regression testing, test capture / replay, and results comparison.

STW/Regression includes the following products:

- *CAPBAK/MSW*™ is a capture-and-playback tool that creates automated tests. It incorporates captured keystrokes and mouse movements into test scripts, and can save screen or screen fragment bitmap information as files.
- *CBDIFF*™ compares screen fragments and disregards irrelevant discrepancies with its masking capabilities.
- *SMARTS/MSW*™ automates the work of controlling, executing, re-executing, and analyzing the results of complex sets of tests.

SMARTS/MSW is the focus of this manual. For complete information on use of the other *STW/Regression* products, please consult the proper manuals.

An *STW/Regression* flow chart is indicated below. Boxes with darkened backgrounds represent the main components of *STW/Regression*.

SMARTS/MSW is central to the *STW/Regression* process, controlling test executions and results. *SMARTS/MSW* can run a variety of tests, including playing back hundreds of test scripts created from *CAPBAK*. *SMARTS/MSW* can also determine if *CAPBAK/MSW*'s tests passed or failed by making a simple call to *CBDIFF* to compare saved bitmap images from *CAPBAK/MSW*'s tests.

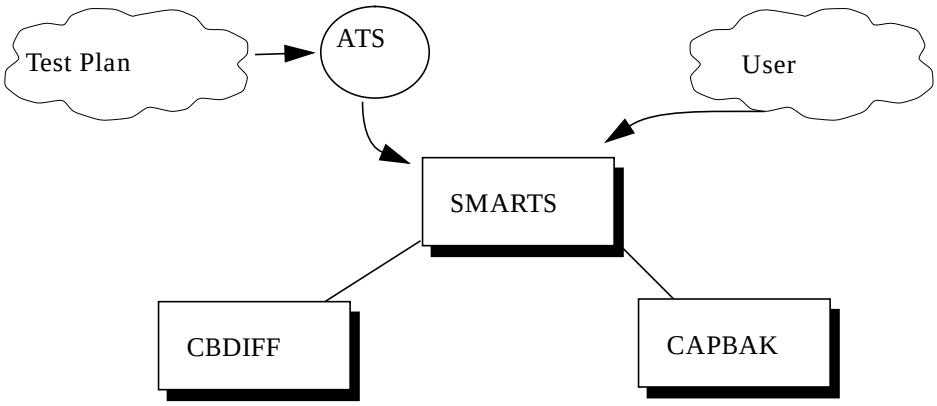


FIGURE 1 **STW/Regression Dependency Chart**

1.3 SMARTS/MSW's Role

SMARTS/MSW automates the testing process by reading a user-designed test description file, referred to as an Automated Test Script (ATS). The ATS is written in *SMARTS/MSW* code, which is a subset of the C programming language.

From the ATS, *SMARTS/MSW* is able to create a "test tree" of the groups and tests, which is similar in structure to an outline. The test tree provides a means of interactively controlling and monitoring the testing process.

SMARTS/MSW's programming capability allows the use of `if, else` and `while` control structures within its test script. Test execution can therefore be tailored to the system environment, allowing the testing process to be repeated with greater reliability than manual testing.

When generated, *SMARTS/MSW* executes the ATS, compares the test output against the expected results (test baseline), and accumulates a detailed record of the test results into a log file. Based on the log file, *SMARTS/MSW* also generates reports indicating the status and execution time of any test or group of tests, the percentage of PASS/FAIL results and test regressions.

Please refer to the Chapter 4 on page 63, "Creating an ATS" for further information on the ATS language.

By organizing all tests that apply to a given application, *SMARTS/MSW* can improve the quality of that software throughout its life cycle. Through developing and re-running a library of test suites, efforts can be focused on constructing new tests and evaluating test results — and detecting defects — rather than on the largely mechanical task of running tests and checking outputs.

The following data flow diagram depicts *SMARTS/MSW*'s processing components and procedures.

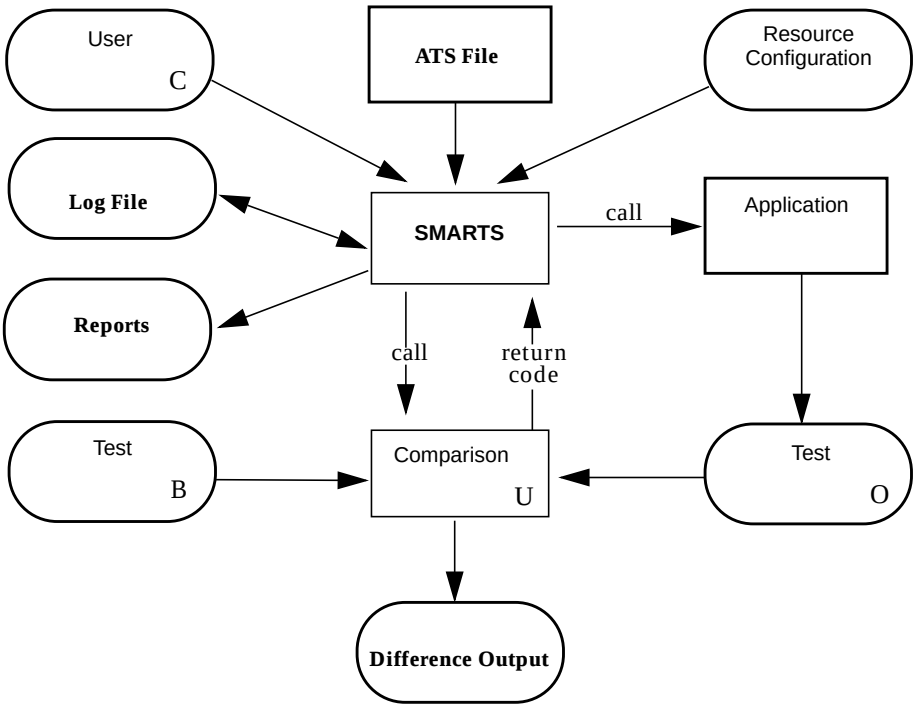


FIGURE 2 SMARTS/MSW System Chart

1.4 How SMARTS Is Used

To automate regression testing with *SMARTS/MSW*, perform the following steps:

1. Set up test baseline results.
2. Create an ATS.
3. Execute test actions specified in the ATS.
4. Evaluate test outputs (PASS / FAIL).
5. Generate test reports.

1.4.1 Establishing Test Baselines

The initial procedure to automating the testing process is to create a series of baseline files containing correct, expected (baseline) program outputs. Later, test outputs will be compared with these baseline results.

To establish a baseline, execute a test and save the correct output in a reproducible form, e.g., a text file or an image file. **SR**'s *CAPBAK/MSW* also allows the results of user sessions to be used as test baselines. These user sessions can be recorded, with the results used as test baselines, and then replayed, with the results used as test outputs.

1.4.2 Creating an Automated Test Script.

Once the baselines are established, the ATS can be created using any ASCII editor — for example, **Notepad**. The test control file must be written using *SMARTS/MSW*'s C language code.

SMARTS/MSW executes these test tree elements in sequence according to the "tree-like" group structure. See Chapter 4 on page 63, "Creating an ATS" for more detailed information.

1.4.3 Executing the ATS

Once the ATS has been constructed, testing can proceed under *SMARTS/MSW* control, with little further manual administration.

Test execution consists of two basic steps:

1. Test case selection.
2. Test case activation.

1.4.3.1 Test Case Selection. *SMARTS/MSW* executes only the group, sub-group or individual test case selected. This is an important feature which can be exploited to minimize overhead when re-testing a modified software product. The purpose of selection is to execute only the test groups or cases of interest, without invoking the remaining tests.

1.4.3.2 Test Case Activation

Once you have selected the tests you wish to run, simply click on the **Run Tests** window's **Run** button. *SMARTS* will run the selected test(s), and the output of the results will be displayed in the **Test Output** text box.

1.4.4 Generating Test Reports

Based on the log file just described, *SMARTS/MSW* produces a variety of test reports providing a means of determining which tests to examine for possible program errors.

The system produces six types of reports:

- **All** report.
- **Latest** report.
- **Regression** report.
- **Summary** report.
- **Time** Report.
- **Failed** Report.

The **All** report lists the test name(s), activation date and outcome (PASS/FAIL) of all log file test entries (as opposed to the **Latest** report, which lists only the most current tests) for a given node.

The **Latest** report gives the results of the most recent test run of the test(s) selected.

The **Regression** report indicates only those tests whose outcome has changed, thereby identifying bugs which have been fixed or introduced since the last time the tests were activated.The report lists test name, outcome, and activation date.

The **Summary** report provides a brief overview of testing status, indicating the number and percentage of tests that have passed, tests that have failed, and the total number of tests executed.

The **Time** report contains total execution time for a given test or tests.

The **Failed** report lists all the tests which have not passed. This report is cumulative.

CHAPTER 2

Installation

This chapter shows you how to install *SMARTS/MSW*.

2.1 System Requirements

Your computer system must have the following hardware configuration to install and run **SMARTS/MSW**.

- Windows 95, NT or 3.1*x*
- 486 microprocessor or better
- 6.5 MB free disk space
- 8+ MB RAM recommended

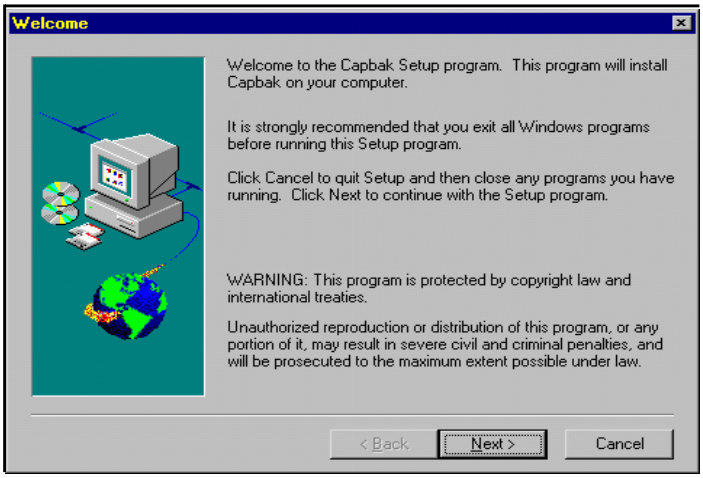
2.2 Installation Procedure

These are instructions for installing **SMARTS/MSW**.

Administrator privileges are required to properly install CAPBAK in Windows NT.

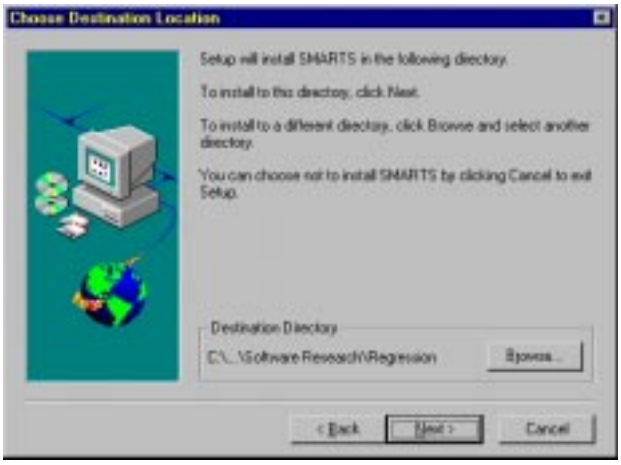
1. Insert the diskette labeled **Disk 1** in your diskette drive (these instructions assume A:).
2. **Activate setup.exe.**

- **In Windows 95 or NT 4.0:** Display the contents of the A: drive, using either the **My Computer** icon (on the desktop) or **Windows Explorer** (on the **Start** menu, **Programs** submenu). Double-click **setup.exe**.
- **In Windows NT 3.x or Windows 3.1*x*:** From **Program Manager**, choose **File|Run**, click the **Browse** button, activate the A: drive, and double-click **setup.exe**. Or, from **File Manager**, display the contents of the A: drive and double-click **setup.exe**.

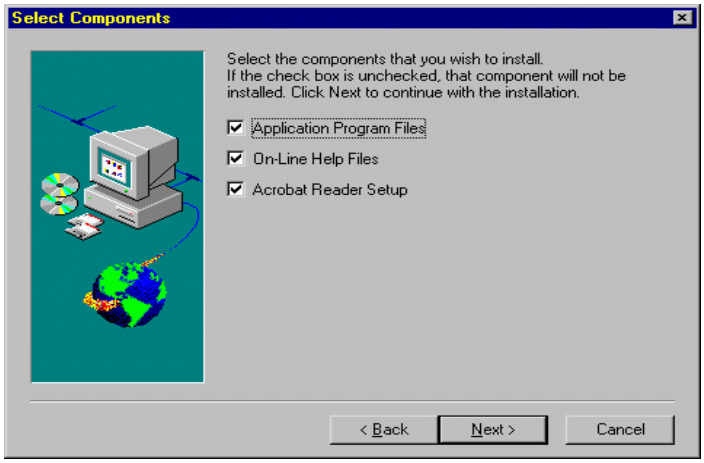


3. **setup.exe** presents you with a series of dialog boxes, beginning with the **Welcome** box shown above. Each box is a step in the installation process, and when you are satisfied with the options offered in a box you should click **Next** to go on to the next step.

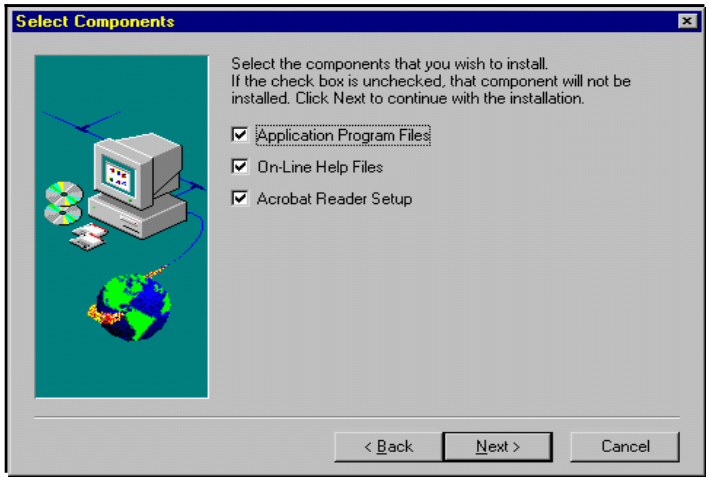
4. If you click **Next** in the **Welcome** box, a second box asks you where you would like to store the executables and the supporting files for **SMARTS/MSW**.



- Click on **Next** if you want to use the **Path** indicated and to continue the installation.
- Edit the default path to your own path, then click **Next** to continue the installation.
- Click **Cancel** to end the installation.
- If you choose **Next**, a dialog box to pops up and asks you what kind of installation you prefer. We highly recommend **Custom** installation, which allows you to install the **Acrobat Reader** software that will allow you to read the on-line help that accompanies **SMARTS/MSW**. The **Acrobat Reader** software will occupy approximately 4 MB of your computer's memory.



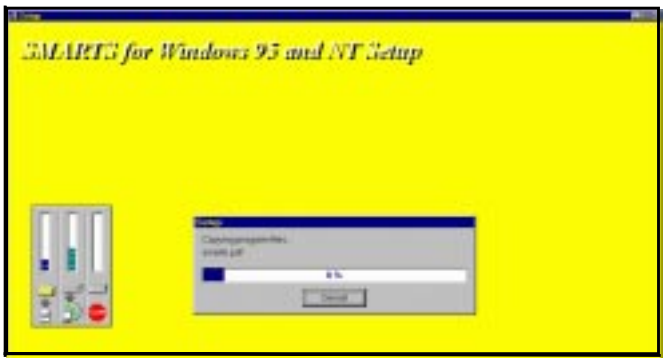
- Click **Next** if the Setup Type is the one you prefer.
- Click a different Setup Type, then click **Next** to continue the installation.
- Click **Back** to review or change previous dialog box queries.
- Click **Cancel** to end installation.



5. In Windows NT and Windows 3.1*x*, but not in Windows 95 or Windows NT 4.0, after you choose **Next**, a dialog box pops up to ask you

to choose the program group name where you would like the program icons to appear.

6. During copying, a bar gauge names the files being copied.



7. The installation process creates a *C:\Program Files\Software Research\Regression* directory (or the path you indicated). **SMARTS/MSW** will automatically store your files to this directory unless you selected otherwise.

8. The installation script also creates a program group where **SMARTS/MSW** and its utilities are installed:

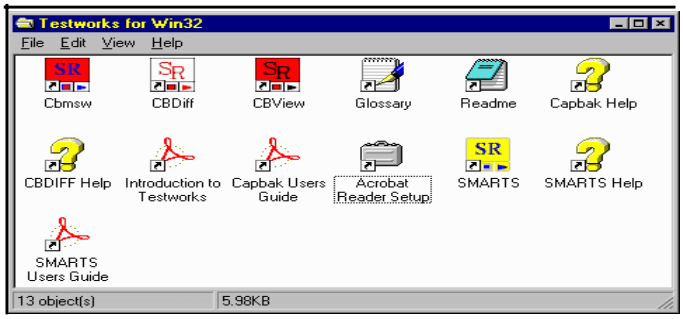


FIGURE 3 Program Group for SMARTS/MSW

9. When the installation is complete, you should include the *STW* pathname in your system environment variable.
10. To uninstall, use the following:

- In Windows NT and Windows 3.1*x*, double click the **UninstallShield** icon in the **SMARTS/MSW** program group.
- In Windows 95 and Windows NT 4.0, double click the **Add/Remove Programs** icon in the Control Panel, highlight **Regression for Win 32 (SMARTS)** and click the **Remove** button.

2.3 File list

The following files are written to your computer during the installation. The locations for these files are given for installation to a directory called *C:\Program Files\Software Research\Regression*

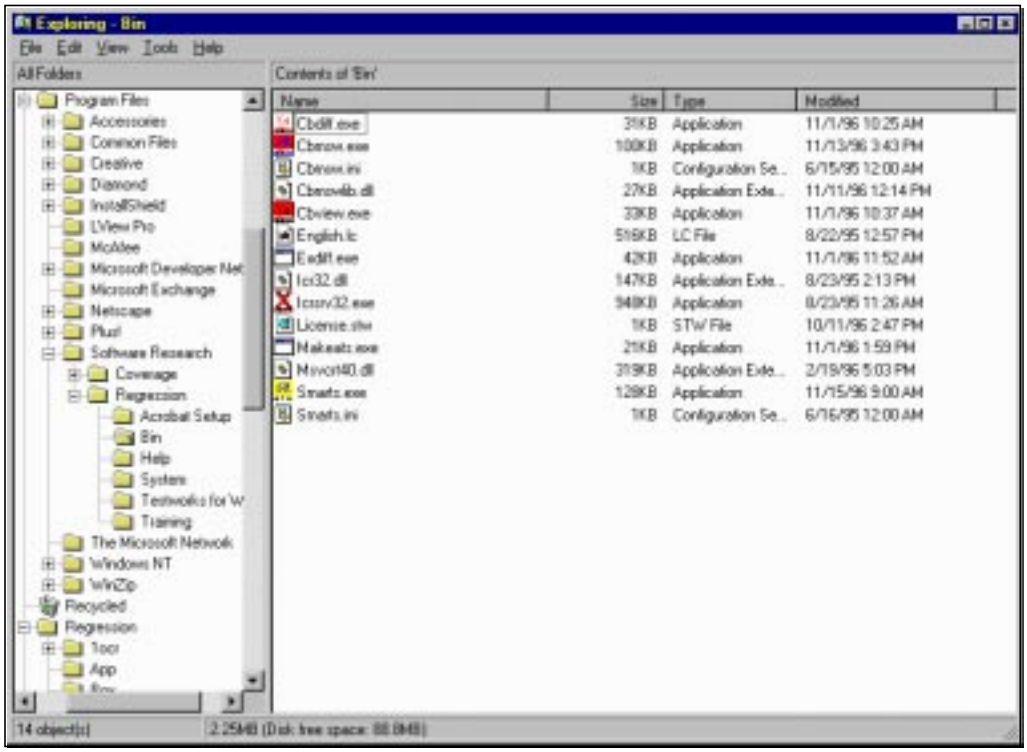


FIGURE 4 Files for SMARTS in Windows 95

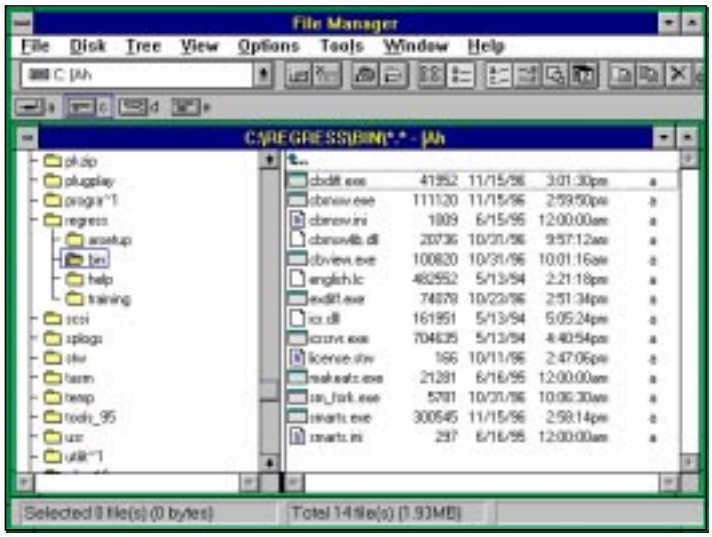


FIGURE 5 Files for SMARTS in Windows 3.1x and NT 4.0

CHAPTER 3

Quick Start

This chapter presents a step-by-step run-through of a basic test session, how *SMARTS/MSW* can help you, and its role in quality assurance

3.1 Getting Acquainted with SMARTS/MSW

3.1.1 Instructions

It is recommended that you complete the instructions in this chapter *before* continuing on to other chapters.

The *SMARTS/MSW* application program provides a directory containing demonstration files and pre-written programs. The tutorial test session performed in this chapter is invoked from this directory.

On completion of this chapter, you should be familiar with the following activities involved in executing a *SMARTS/MSW* test session: invoking the *SMARTS/MSW* application, loading a *SMARTS/MSW* Automated Test Script (ATS) and running a suite of tests, analyzing the test outcome via *SMARTS/MSW* reports, examining any test regression, purging any log files and exiting the *SMARTS/MSW* application.

For an overview of the *SMARTS/MSW* Graphical User Interface (GUI), please refer to Chapter 4 on page 45, "Understanding the Graphical User Interface (GUI)".

3.1.2 Analyzing the Test Setup

SMARTS/MSW automates the testing process by reading a user-designed test description file, referred to as an ATS. From the ATS, *SMARTS/MSW* creates a “test tree”. When the ATS is run, an outline of the test tree is displayed in the **Test Tree** display area of the **Run Tests** window. This outline provides a means of interactively controlling and monitoring the testing process.

3.2 Environment Variables

Before you invoke *SMARTS/MSW*, make sure you are already in Windows. */stw/bin*, or whatever directory you installed the SR executables in, must be in your DOS *$PATH* (see *Installation Procedures* for further details).

FIGURE 6 Typical Program Manager screen in Windows 95

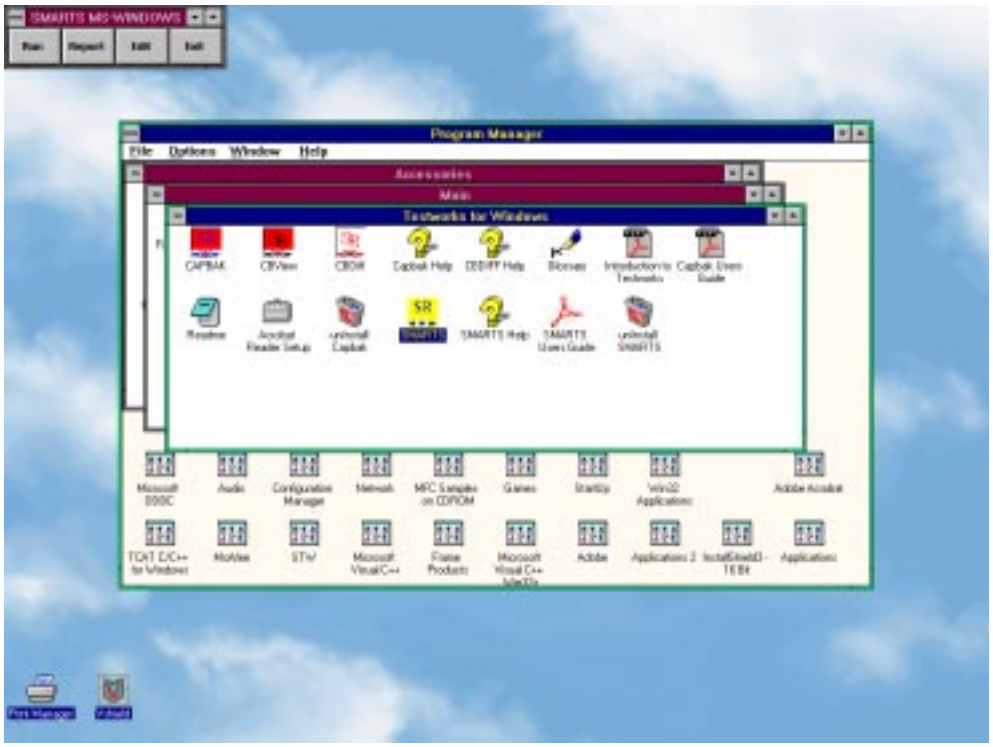


FIGURE 7 Program Manager in Windows 3.x and Windows NT 4.0

3.3 Invoking the TestWorks Window

A **TestWorks** icon should have been added to the **Program Manager** window during installation (Figure 6).

3.3.1 Step 1: Starting SMARTS/MSW

1. Double-click on the **TestWorks** icon to initiate the **TestWorks Group** window.

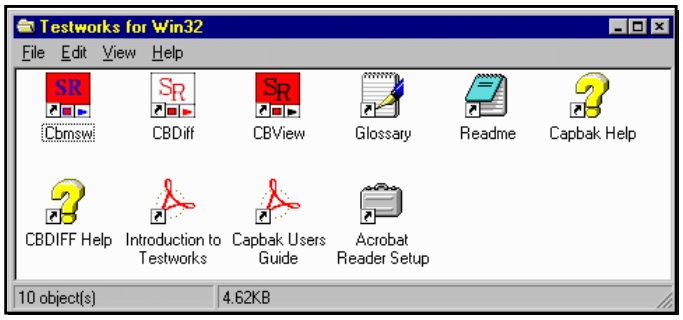


FIGURE 8 TestWorks Group Window

Each icon initiates a different *STW/Regression* utility.

- You can double-click on the *SMARTS* icon to bring up the test management utility.
- You can double-click on the *CAPBAK/MS-Windows* icon to bring up the capture / playback utility.
- You can double-click on the *CBVIEW* icon to invoke the image-displaying utility.
- You can double-click on the *CBDIFF* icon to initiate the image comparison utility.
- You can double-click on the *CB_ASCII* icon to initiate the character recognition utility.
- You can double-click on the *Glossary* icon to review testing terminology.

2. To invoke *SMARTS/MSW*, double-click on the *SMARTS* icon in the STW window.

The **Main** window pops up (Figure 9).

FIGURE 9 SMARTS/MSW Main Window

The SMARTS/MSW main window has four options.

- The **Run** button will open a window which allows you to run different tests or test groups.
- The **Report** button will show you the results of test runs in a variety of formats.
- The **Edit** button will bring up an ASCII editor (default is **Notepad**) which will allow you to edit ATS files.
- The **Exit** button will allow you to exit the *SMARTS/MSW* application.

3.3.2 Step 2: Invoking the Run Test Window

Tests are executed from the **Run Tests** window. To begin this demonstration session, you must first load an ATS file. To do so, invoke the **Run Tests** window by doing the following:

1. In the **Main** window, click on the **Run** button.

   The **Run Tests** window pops up (Figure 10).

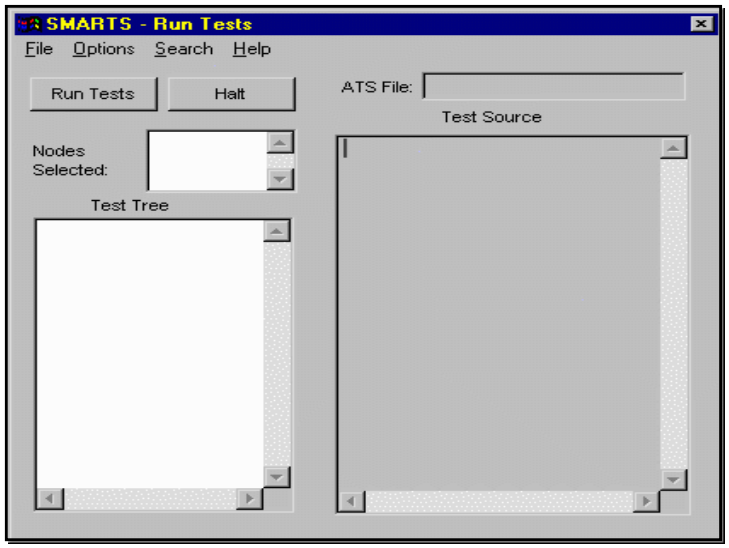


FIGURE 10 Run Tests Window

From this window, you will be able to select the ATS file, the log file, and the output file for your tests using the **File** pull-down menu.

From the **File** pull-down (Figure 11) select **Load ATS**. The **Select Test Script** window (Figure 12) will appear.

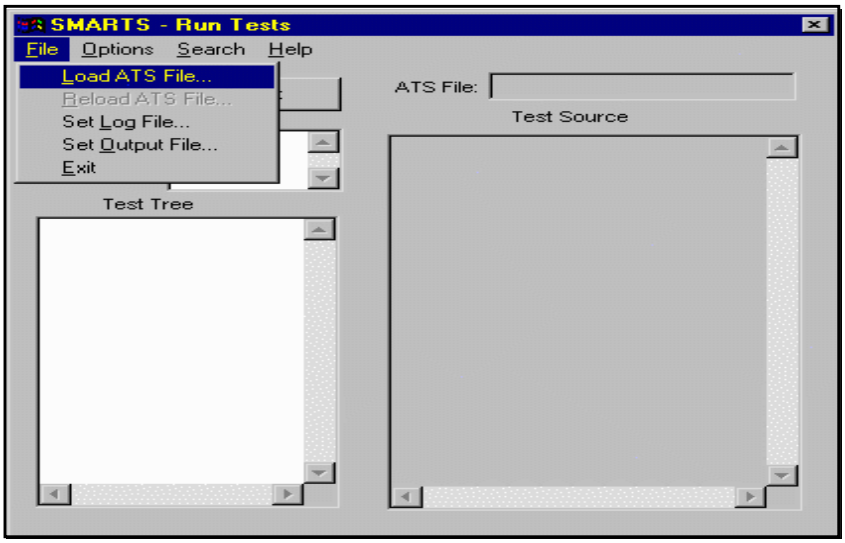


FIGURE 11 File Pull-Down Menu

3.3.3 Step 3: Selecting an ATS File

Before running any tests, you must specify the ATS file. After selecting **Load ATS** from the **Run Tests** window's **File** pulldown menu, The **Select Test Script File** window appears.

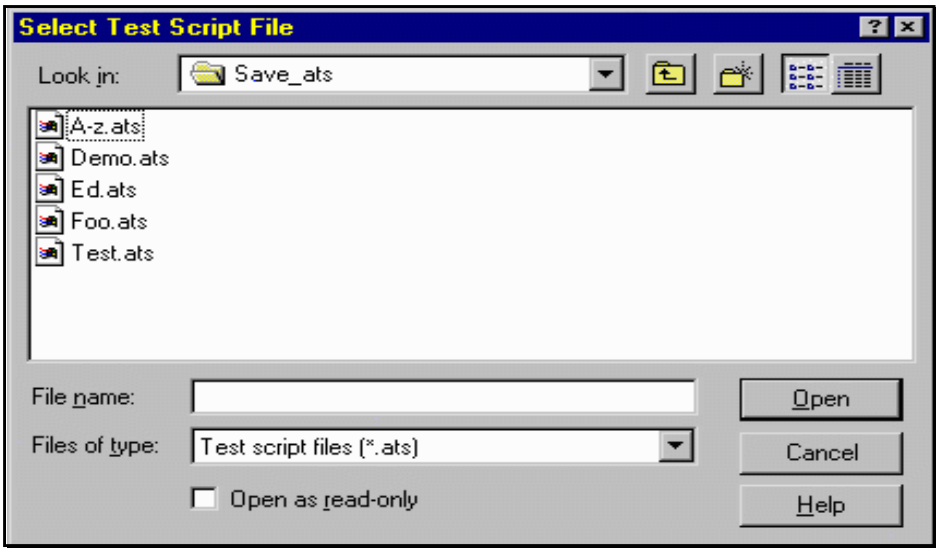


FIGURE 12 Select Test Script File Window

To select an ATS file, you can type in the file name in the **File Name** box, or you can double-click on a file name in the box below, or highlight a file name from the box and click **OK**.

For this tutorial, use the supplied *demo.ats*.

After selecting your ATS, the **Run Tests** window (Figure 10) will come up on the screen once again, with the name of the file you selected displayed in the **ATS File** text box.

3.3.4 Step 4: Setting the Log File

After establishing the ATS file, you can name a log file to which the results of the tests will be written. Do this from the **File** pulldown menu (Figure 11) by selecting **Set Log File**. The **Select Log File Name** box (Figure 14) will appear.

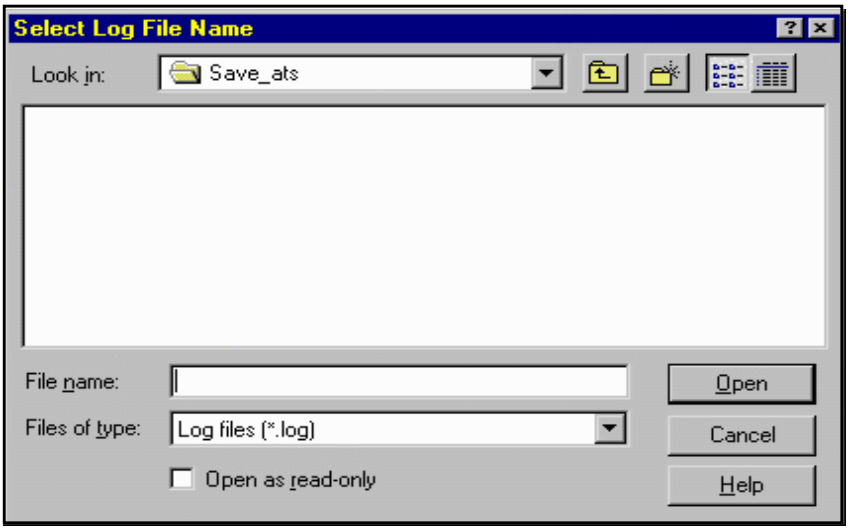


FIGURE 13 Select Log File Name Window

To select a log file, you can type in the file name in the **File Name** box, or you can double-click on a file name in the box below, or highlight a file name from the box and click **OK**.

For this tutorial, use the default *log.log*.:

NOTE: If you do not choose a log file name, all the test information will be written to the default logfile, *log.log*. The log file will be saved in your current working directory if you do not type another pathname in the **File Name** text box.

3.3.5 Step 5: Setting an Output File

After setting the log file name, you can set a name for the output file. This file is the one displayed in the **Test Source/Test Output** section of the **Run Tests** window (see Figure 78) while the actual ATS is being executed; it contains all the information about the test (s) execution, including what kind of test it was, beginning and end time of test, and whether it passed or failed.

You set the output file name from the **Run Tests** window's **File** pull-down menu (Figure 11) by selecting **Set Output File**. The **Select Output File Name** window (Figure 13) will appear.

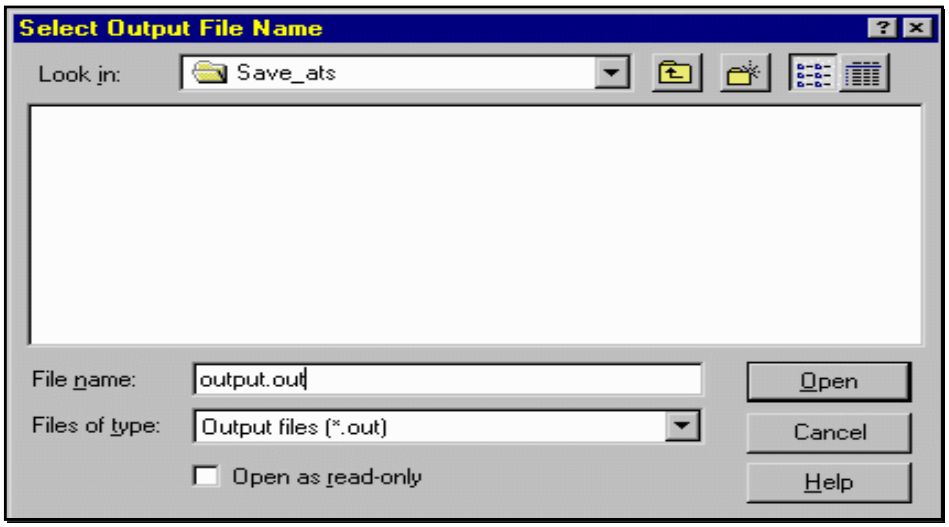


FIGURE 14 Select Output File Name Window

To select an output file, you can type in the file name in the **File Name** box, or you can double-click on a file name in the box below, or highlight a file name from the box and click **OK**.

For this tutorial, use the default *output.out*.

NOTE: If you do not choose an output file name, all the test information will be written to the default output file, *output.out*.This file will be in your current working directory if you do not type another pathname in the **File Name** text box.

3.3.6 Step 6: Running Your ATS

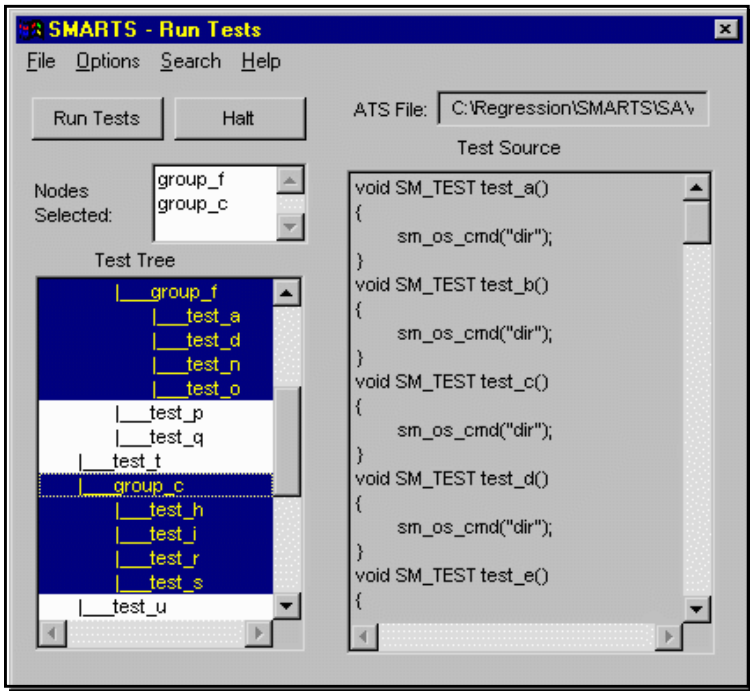


FIGURE 15 Run Tests Window after selection of ATS

After you select an ATS, a log file name, and an output file name, the **Run Tests** window will appear again, with the ATS appearing in the **Test Tree** window (the ATS name will appear in the **ATS File** box in the upper-right corner of the window). To select which parts of the ATS to run, click on the sections desired. In Figure 15 , *demo4, demo, demo2, demo, demo3*, and *demo2* (highlighted) will be run (notice that in this example, *demo* and *demo 2* belong to more than one test group and are each run twice).

If you click on the **root** name, the entire ATS will be highlighted. You can also click on individual file names to select and de-select groups or individual tests. For instance, you can select a group, yet de-select tests within that group. The names of the tests and groups selected will appear in the **Nodes Selected** text box.

Once the test (s) you want to run are highlighted, click on the **Run Tests** button. Execution will begin.

NOTE: After testing, to toggle back and forth between viewing the Test Source and Test Output, double-click on the words "Test Source". The box will again display the test output. Toggling between source and output is a default setting in the s*marts.ini* file, and can be changed there or in the **Run Tests** window's **Options** pull-down menu.

A test case which passes is indicated by the statement:

(*test name)* PASSED [*date time*]

A failed test case is indicated by the statement:

(*test name)* FAILED [*date time*]

As the *test name* is listed in the display area of the **Run Tests** window, the location of a failed test is easily determined.

2. When the test session has completed executing, the end-of-test-execution message appears in the **Test Output** box:

   `Run Ended [date time]`

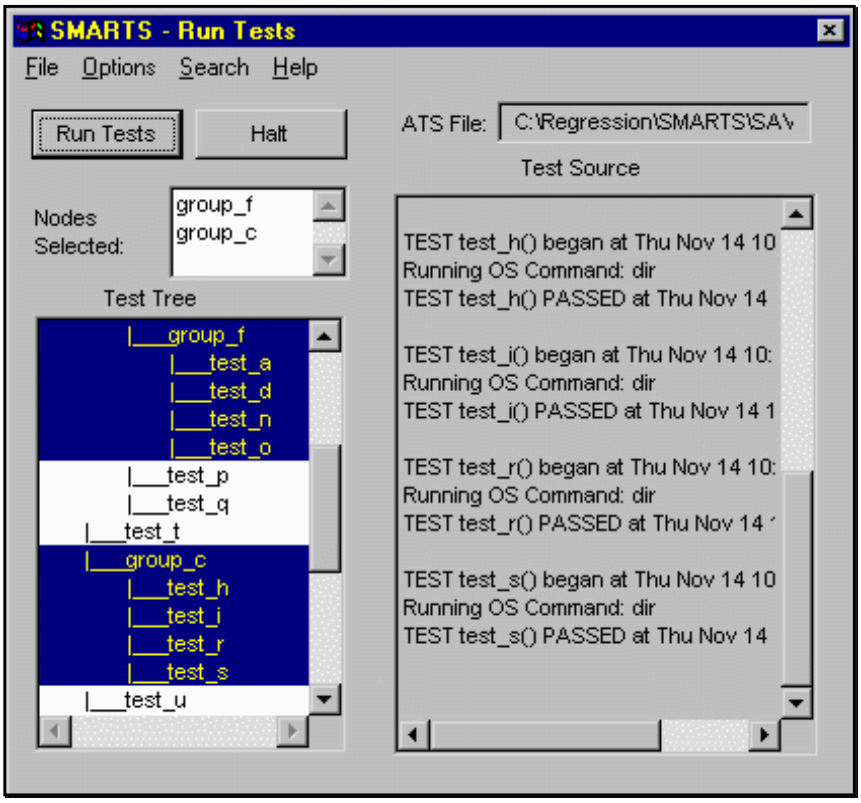


FIGURE 16 Sample Output File After ATS is run

3.3.7 Step 7: Editing the ATS

After a test is executed for the first time, users may have ideas on improving the ATS file. This step shows you *SMARTS*' editing utility that lets you amend the actual ATS.

The default editor for *SMARTS/MSW* is **Notepad** (Figure 17) the standard editor supplied with Windows. You can specify a different ASCII editor at the initialization stage of *SMARTS/MSW* by editing the *smarts.ini* file.

NOTE: Because modifications will affect the ATS behavior, **do not** actually edit the sample ATS.

In the **Main** window, click on the **Edit** button.

The **Notepad** window appears, displaying *demo.ats*. Depending on how your initialization file is set up, the **Notepad** window will either display a blank screen or the current ATS file. See Appendix B, "INITIALIZATION FILE PROCESSING", for more information on selecting an ASCII editor.

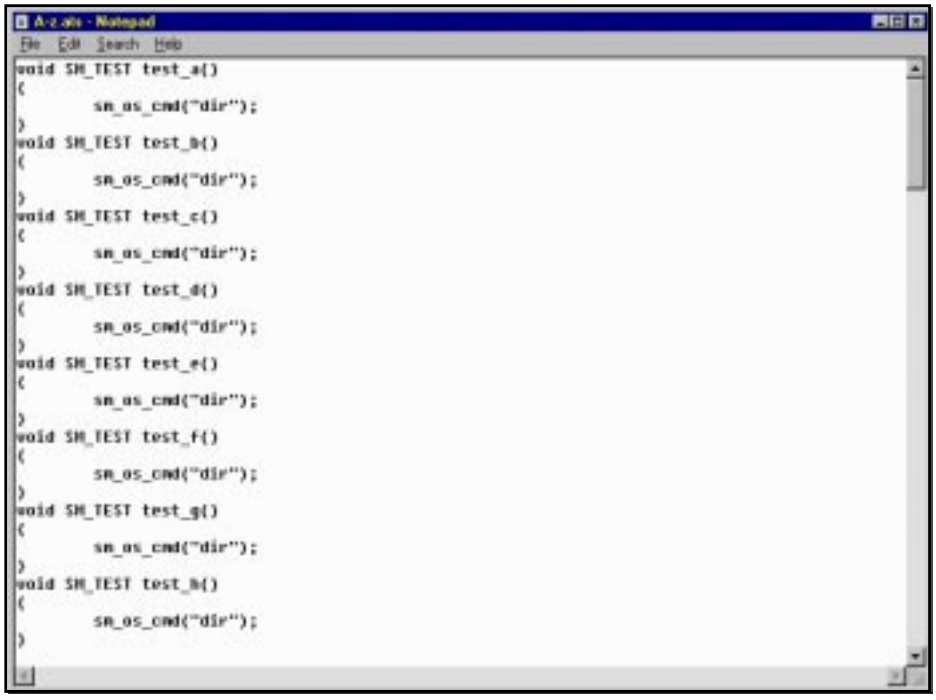


FIGURE 17 Notepad Window

3.3.7.1 Confirming Changes to the ATS

If you make changes to the ATS file in any way, and then attempt to close the **Notepad** window, the message box in Figure 18 will appear.

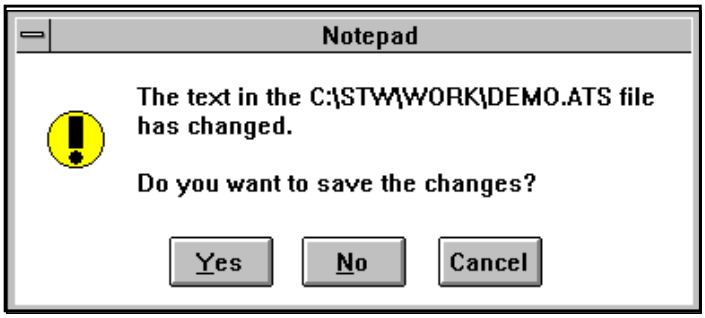


FIGURE 18 Notepad Message box

If you want to keep the changes you've made to the ATS, click **Yes**, if not, click **No**. If you want to return to the **Notepad** window without implementing changes, click on **Cancel**.

3.3.8 Step 8: Analyzing Test Status - Report on Tests Window

The various reports generated following test execution can be viewed from the **Report on Tests** window. To invoke the **Report on Tests** window, perform the following:

1. In the **Main** window, click on the **Report** button.
2. The **Report on Tests** window pops up (Figure 19).

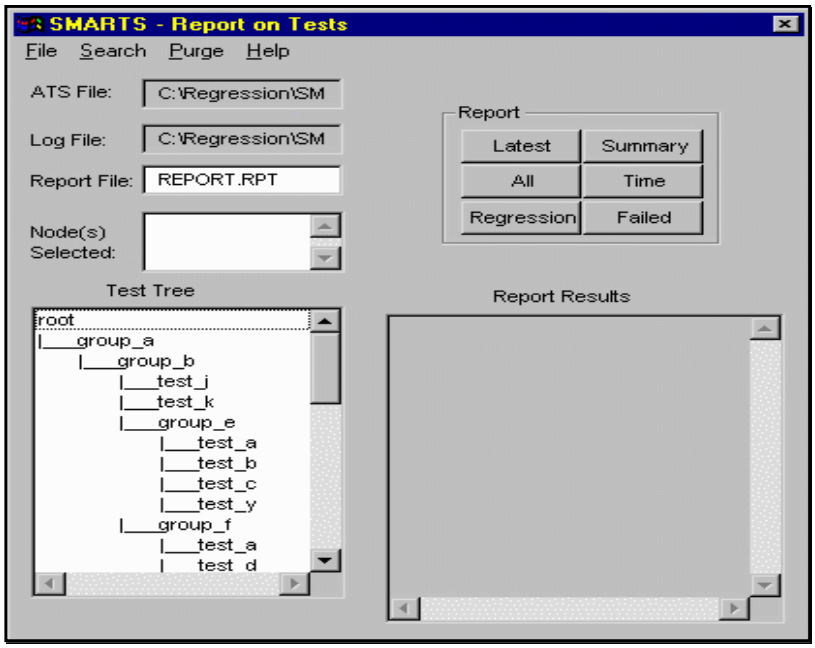


FIGURE 19 Report on Tests Window

When tests are executed in the **Run Tests** window, *SMARTS/MSW* runs a difference check on the test output against the test baseline files, and accumulates a detailed record of the test outcomes into a log file (Section 3.3.4 on page 30). Based on the log file, *SMARTS/MSW* also generates six different reports that can be viewed via the **Report on Tests** window.

- The **All** report lists the test name(s), activation date and outcome (PASS / FAIL) of all log file test entries (as opposed to the **Latest** report, which lists only the most current tests) for a given node.
- The **Latest** report gives the results of the most recent test run of the test(s) selected.
- The **Regression** report indicates only those tests whose outcome has changed, thereby identifying bugs which have been fixed or introduced since the last time the tests were activated. The report lists test name, outcome, and activation date.
- The **Summary** report provides a brief overview of testing status, indicating the number and percentage of tests that have passed, tests that have failed, and the total number of tests executed.
- The **Time** report contains total execution time for a given test or tests.
- The **Failed** report lists all the tests which have not passed. This report is cumulative.

To view a particular report:

1. From the **Report on Tests** window, click on the appropriate button. The selected report is shown in the **Report Results** display area.
2. Report text can be further viewed using the scroll bars.

NOTE: For more detailed information on viewing *SMARTS* reports, please refer to Chapter 6 on page 93, "Viewing Execution Reports".

3.3.9 Step 9: Viewing the Regression Report

In this demonstration, we will look at the **Regression** report.

1. In the **Report on Tests** window, click on the **Regression** button.
2. The **Regression** report appears in the display area of the **Report on Tests** window (Figure 20).

This report displays only those test cases whose PASS / FAIL outcomes have changed from a previous execution.

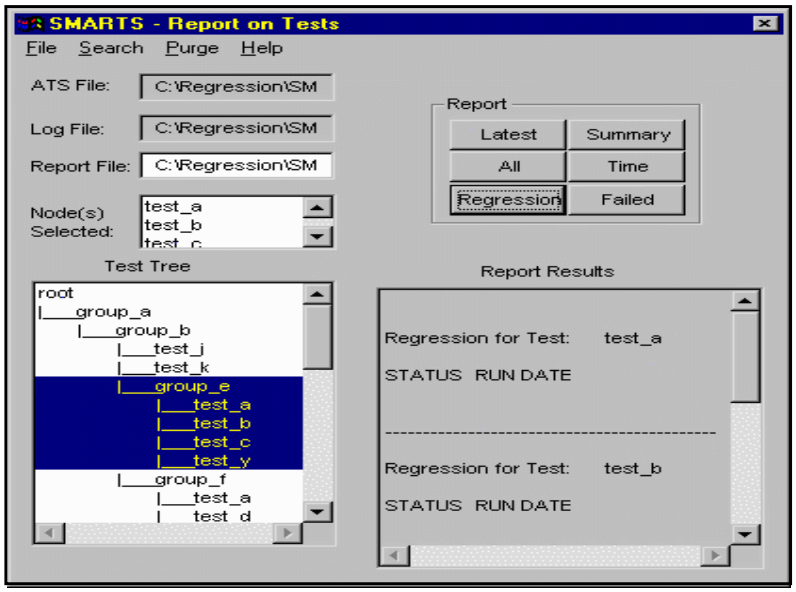


FIGURE 20 Sample Regression Report

3.3.10 Step 10: Purging the Log File

After running a number of tests, you may want to eliminate unwanted data in the logfile. You can do this by purging the logfile.

1. In the **Report on Tests** window, click on the **Purge** pull-down menu and select **Purge Log File** (the only choice). The message box in Figure 21 pops up.

FIGURE 21 Purge Log File Message Box

2. Confirm the request by clicking on the **OK** button of the message box. To cancel the purge log file operation, click on the message box's **Cancel** button.

   Only the testing data from the last run of the test cases will remain in the log file. For example, if twenty two tests have been run, following a **Purge Log File** execution, only the testing data for the twenty-second test execution would remain in the log file.

3.3.11 Step 10: Using the Supplied Demonstration Files

The supplied demo ATS file *demo.ats* is a simple example of how *SMARTS/MSW* can run tests and groups of tests. Some of these tests will pass; some will fail. Running these tests and examining the reports *SMARTS/MSW* creates from them should provide an overview of how SMARTS can organize test runs and deal with test successes and failures.

Identified by the keyword **SM_TEST**, the functions *demo()*, *demo2()*, and *demo3()* are simple individual tests. The test *demo()* first calls *sm_capbak()* to play back a simple *CAPBAK/MSW* keysave file to run a test. It then calls *sm_image_diff()* to compare an image that *CAPBAK/MSW* created during playback with an image *CAPBAK/MSW* captured during recording. The test will fail; the baseline image is not the same as the response image. The test *demo2()* is almost the same as *demo()*, except that it compares different images.

The test *demo3()* first uses *sm_os_cmd()* to call a DOS command to create an ASCII file. It then uses *sm_ascii_diff()* to compare the file created with a baseline file. Because the baseline and response files are the same, the test should pass.

Identified by the keyword **SM_GROUP**, the group *demo4()* and *demo5()* simply call individual tests. So by calling these groups, the user can call several tests with one call. *demo6()* calls the groups *demo4()* and *demo5()*, illustrating how *SMARTS* / MSW can be used to organize tests and groups of test hierarchically.

3.3.12 Step 12: Exiting the SMARTS Product

To exit the *SMARTS/MSW* application:

1. Exit any open windows by either selecting **Exit** from the **File** pull-down menu, or clicking on the System Pull-Down of the window and choosing **Close**.
2. Having closed any open *SMARTS/MSW* windows, the application itself can now be exited.
3. From the **Main** window, click on the **Exit** button to terminate the current test session.

3.4 Summary

If the preceding steps are successfully completed, you've seen and practiced the basic skills you need to use *SMARTS/MSW* productively. In this chapter, you have examined how to invoke *SMARTS/MSW*, how to run a suite of tests, how to analyze test outcome, how to look for test regression, and how to purge a log file.

For further practice it is suggested to:

- Repeat this procedure without the manual.
- Re-examine the product-supplied *demo.ats* to review the ATS "test tree structure", supplemental commands, arguments and evaluation methods.
- Refer to Chapter 4 on page 63, "Creating an ATS" for complete information on creating an ATS file, running *SMARTS*' tests, and viewing reports.

CHAPTER 4

Understanding the Graphical User Interface (GUI)

This chapter summarizes *SMARTS/MSW* windows, menus and commands. Individual commands are described in detail in the relevant chapters of this guide.

4.1 Basic MS-Windows Graphical User Interface

This section demonstrates using file selection dialog boxes, help menus, message dialog boxes, option menus, and pull-down menus. If you are familiar with the basic MS-Windows graphical user interface style, you can go on to Section 4.2 on page 52.

4.1.1 File Selection Windows

SMARTS/MSW file selection windows allow you to select or specify test file names or select saved image files.

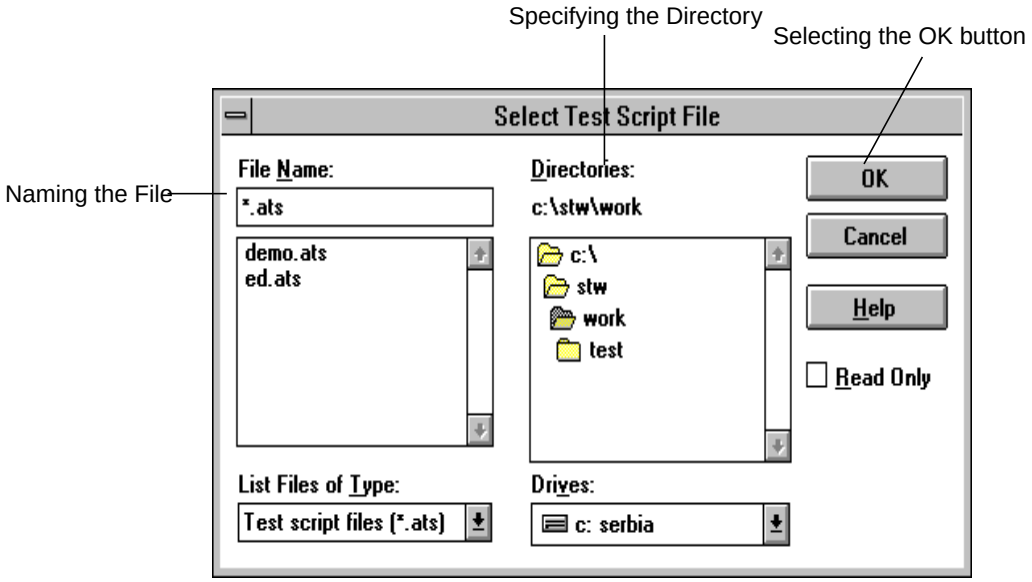


FIGURE 22 File Selection Window

File Name entry box

Selects and enters a file name.

File Name list box — Lists files in the path defined in the **List Files of Type** area.

List Files of Type — Specifies which files are listed in the **File Name** area. The current type of file and its extension are displayed.

Directories list box

Lists directories in path defined in the **Filter** entry box. Use it to locate the desired directory.

Drives — Selects your system's current drive.

Scroll bars — Move up / down and side / side in the **Directories** and **File Name** list boxes. You use them to search for the appropriate directory or file.

Use the three push buttons at the right of the dialog box to issue commands:

OK — Accepts the directory and file in the **File Name** entry box as the new file or the file to be opened and then exits the dialog box.

Help — Supplies on-line help.

Cancel — Cancels any selections made and then exits the dialog box. No file is selected as a result.

To use a file selection dialog box:

1. Click the directory name where an existing file is located or where you want a new file to be placed.
2. Select an existing file name in the **File Name** list box, or type in a new file name in the **File Name** entry box, with the usual DOS limit on file name length and a pathname of no longer than 128 characters.
3. The convention for naming ATS files is *basename.ats*, where *basename* is the file name *and ats* represents an ATS file. Output files are identified by *basename.out*, and log files by *basename.log*.

   If you are using *CAPBAK/MSW* files in your testing, captured images take the form of *basename.bxx*, *basename.sxx*, *basename.rxx*, where *b* represents a baseline image, *s* identifies an image captured for synchronization, *r* represents a response file, and *xx* represents the original sequence in which the image was captured.
4. To select a file name, do one of these three things:
   - Double-click on the file in the **File Name** list box.
   - Highlight the file in the **File Name** list box or type in the file name in the **File Name** entry box and click on **OK**.
   - Highlight or type in the file name and press the **Enter** key.

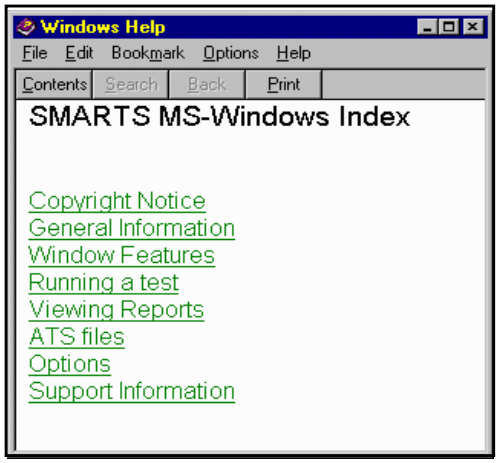
4.1.2 Help Windows

SMARTS/MSW 's help is available for each of its windows as well as the **CBDIFF** windows. Its on-line help automatically brings up the pertinent text corresponding to the topic you choose.

To use the help:

1. Click on the **Help** menu.
2. Select the **Run Help** submenu if you need help for the **Main** window or the **CBDIFF Help** submenu for the **CBDIFF** window.
3. The **Help** window pops up with the contents of the help information.
4. Simply click on the topic you want information for and the **Help** window automatically displays it.

NOTE: If this is the first time you've used on-line help, you might want to choose How To Use Help from the Help menu. You can also refer to your *Microsoft Windows User's Guide* for complete information on using Help menus.



Click on the desired topic

FIGURE 23 Help Window

4.1.3 Pull-Down Menus

Pull-down menus are located within the menu bar of *SMARTS/MSW*'s windows. They often contain several options. To use pull-down menus and their options, follow these steps.

1. Move the mouse pointer to the menu bar and over the menu containing the item.
2. Hold the left mouse button down. This displays the items on the menu.
3. While holding down the left mouse button, slide the mouse pointer to the menu item you want to select. The menu item is highlighted in reverse shadow.

NOTE: An ellipse (...) following an option (as in Figure 24 , Load ATS File...) indicates that selecting the item will bring up a pop-up window, such as a file selection window.

A dimmed (not visible) option (Figure 24 , Reload ATS File...) indicates that you may not be able to use the option with your application at the current time. For example, you may need to select another item before using this command.

---

4. To choose an item from a selected menu, click the item, or type the letter that is underlined in the item name, or use the arrow keys until you reach the item you want to select, and then press the **Enter** key

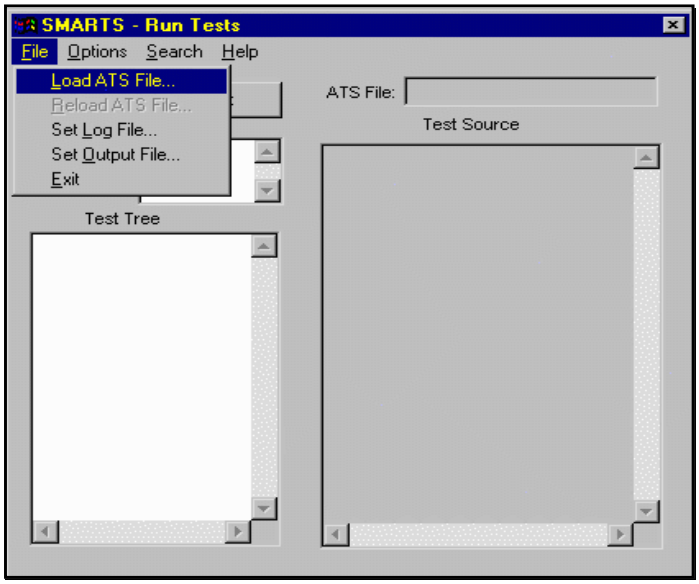


FIGURE 24 Sample Pull-Down Menu

4.2 The Main Window

The **Main** window is used to initiate test sessions.

FIGURE 25 SMARTS/MSW Main Window

This is the window which appears when you click on the *SMARTS* icon. Each of the buttons and their corresponding windows are briefly described next.

4.2.1 Run Tests Window

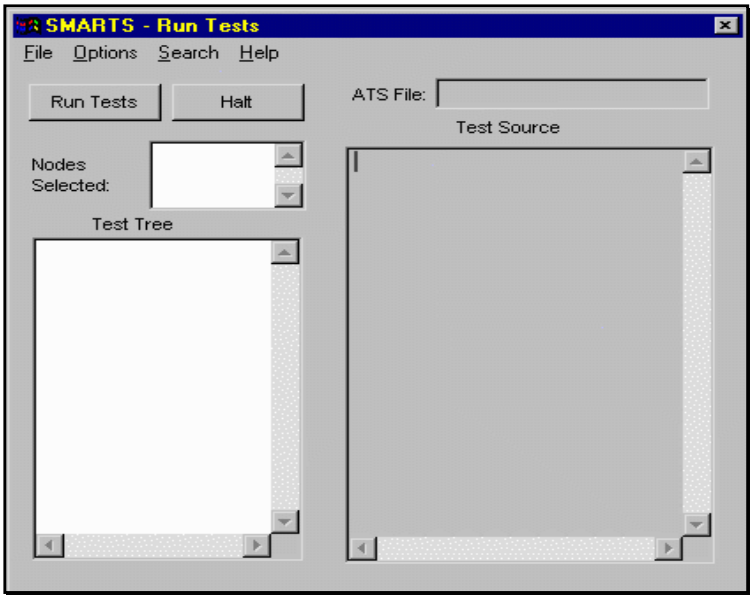


FIGURE 26 Run Tests Window

4.2.1.1 File Pull-Down Menu

- Load ATS

  This allows you to choose the ATS file which will be used in the testing.
- Set Log File

  This allows you to set the name of the file which will be accumulating your test data.
- Set Output File

  This allows you to set the name of the file where your test output will be displayed.
- Exit

  This allows you to exit the **Run Tests** window.

4.2.1.2 Options Pull-Down Menu

- Set Output Options

  This allows you to delete or retain certain files depending on whether the differencing succeeds or fails. It also gives you the option of remaking the baseline output file if differencing fails.

- Set Display Options

  This allows you to either show (in the **Run Tests** window) the test output always, the test source always, or toggle between the two (toggling is the default setting). It also allows you to display the included ATS file.

- Set Actions Options

  This allows you to determine whether or not *SMARTS* will run an ATS once or multiple times, and whether it will quit on the first test failure, or quit after a specified number of test failures. Changes made to any of *SMARTS/MSW*'s options are not saved between invocations of *SMARTS/MSW*. To make permanent changes to options, please refer to Appendix B, "CUSTOMIZING SMARTS / MSW", for instructions on editing the *smarts.ini* file.

4.2.1.3 Search Pull-Down Menu

- Find

  This will allow you to go to a particular node within the ATS file structure. You can search backward or forward, starting at the beginning or end of the ATS file.

4.2.1.4 Help Pull-Down Menu

This brings up the main *SMARTS* help window, from which you can choose your desired topic.

4.2.1.5 Boxes within the Run Tests window

- Test Tree Box

  This box displays the entire test tree for the current ATS. All of the tree may not fit in the window; you may have to use the scroll bars to view the entire tree.

- Nodes Selected

  This box displays the specific tests that you have chosen to run. These will be the tests which are highlighted in the **Test Tree** box.

- Test Source / Test Output

  This shows the actual source code for the tests being run, and the output of the tests are they are being run. As mentioned previously, the default setting for this window is toggling between the test source and test output.

4.2.2 Report on Tests Window

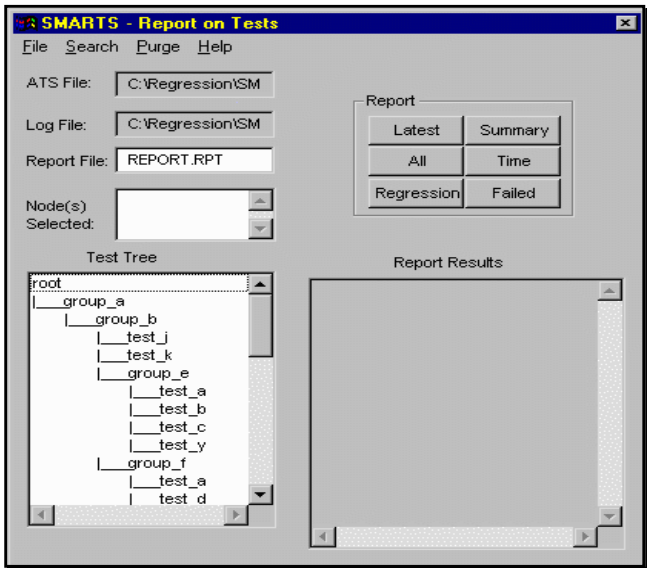


FIGURE 27 Report on Tests Window

4.2.2.1 Pull-Down Menu Options

- File

  This allows you to select an ATS file, a Log file, and a Report file.

- Search

  This allows you to find a particular node in the ATS for which you want to display reports. This is especially helpful with larger ATS files.

- Purge

  This allows you to purge the log file. This is helpful after you have run many tests because it eliminates data from all but the last test session.

- Help

  This invokes the *SMARTS/MSW* Help window, from which you can choose the desired topic for which you require assistance.

4.2.2.2 Report Buttons

- Latest

  The **Latest** report gives the results of the most recent test run of the test(s) selected.

- All

  The **All** report lists the test name(s), activation date and outcome (PASS / FAIL) of all log file test entries (as opposed to the **Latest** report, which lists only the most current tests) for a given node.

- Regression

  The **Regression** report indicates only those tests whose outcome has changed, thereby identifying bugs which have been fixed or introduced since the last time the tests were activated. The report lists test name, outcome, and activation date.

- Summary

  The **Summary** report provides a brief overview of testing status, indicating the number and percentage of tests that have passed, tests that have failed, and the total number of tests executed.

- Time

  The **Time** report contains total execution time for a given test or tests.

- Failed

  The **Failed** report lists all the tests which have not passed. This report is cumulative.

4.2.2.3 Text Boxes

- Test Tree Box

  This box displays the current ATS structure, with selected tests highlighted.

- Report Results Box

  This box displays the results of the tests selected in the Test Tree box.

4.2.3 Edit Window

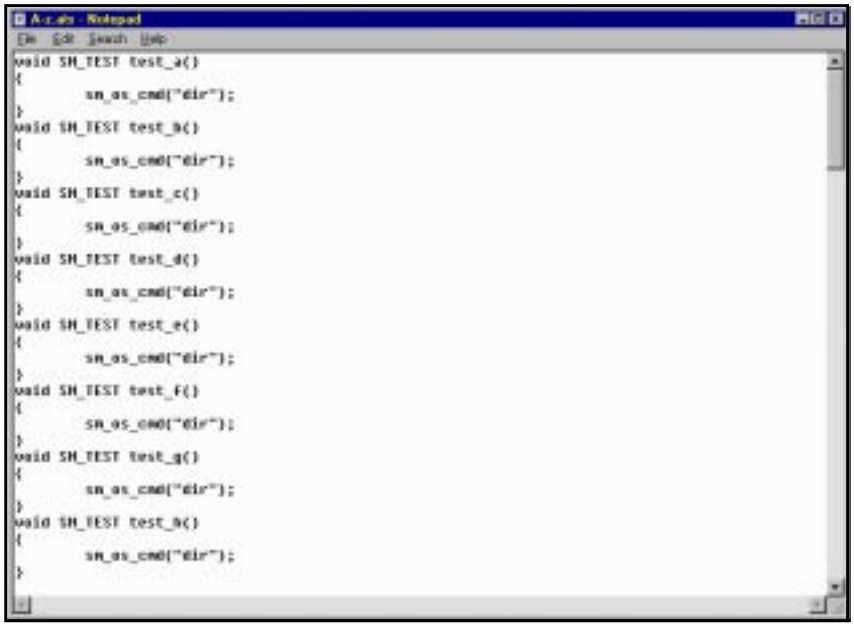


FIGURE 28 "Notepad" Editor Window

This is the window which appears when you click on the **Main** window's **Edit** button. It allows you to make changes to the ATS file. **Notepad** is the default ASCII editor; if you want to use a different ASCII editor, specify it in the *smarts.ini* file.

CHAPTER 5

Creating an ATS

This chapter explains the Automated Test Script (ATS) language, and how to establish an ATS.

5.1 Automated Test Script

After developing test scripts based on a comprehensive test plan, an ATS is created. The ATS file is a structured description file which references a test suite. Using the *SMARTS* C Interpreter Language, the relationally-organized tests can be built with test playback commands, operating system calls and PASS/FAIL evaluation methods. When executed, *SMARTS* performs the pre-stated actions, runs a difference check on the outputs against the baseline, and accumulates a detailed record of the test results.

5.2 ATS Structure

The ATS is organized into tests and groups. Individual tests are simply C functions that are identified by the keyword SM_TEST in their declarations. This test calls other functions (see Section 5.3 on page 68 for the functions available) to perform testing actions. This example sets up test "c":

```
void SM_TEST c()
 {
            sm_capbak("c.ksv");
            sm_image_diff("c.b01",
"c.r01", "");
 }
```

The test "c" calls a function to playback a *CAPBAK* keysave file, then calls another function to compare two image files (see section 21.2.1 for information on how to set up an individual test).

A group is a C function identified by the keyword SM_GROUP in its declaration. In the following example, the group "a" is set up:

```
void SM_GROUP a()
{
            c();
            d();
}
```

"a" calls the test functions "a" and "b". By calling the group "a", the user can execute tests "c" and "d". Users can run any combination of tests by calling them from various groups.

The following ATS file shows how a user can group tests together.

```
void SM_GROUP a()
{
      c();
      d();
}
void SM_GROUP b() {
      c();
      d();
      e();
      f();
 }
void SM_TEST c()
{
      sm_capbak("c.ksv");
```

```
        sm_image_diff("c.b01", "c.r01", "");
}
void SM_TEST d()
{
        sm_os_cmd("dir > d.rsp");
        sm_ascii_diff("d.bsl", "d.rsp", "");
}
void SM_TEST e()
{
        sm_capbak("e.ksv");
        sm_image_diff("e.b01", "e.r01", "");
} void SM_TEST f()
{
        sm_os_cmd("dir > f.rsp");
        sm_ascii_diff("f.bsl", "f.rsp", "");
}
```

FIGURE 29 SHOW.ATS Test Tree

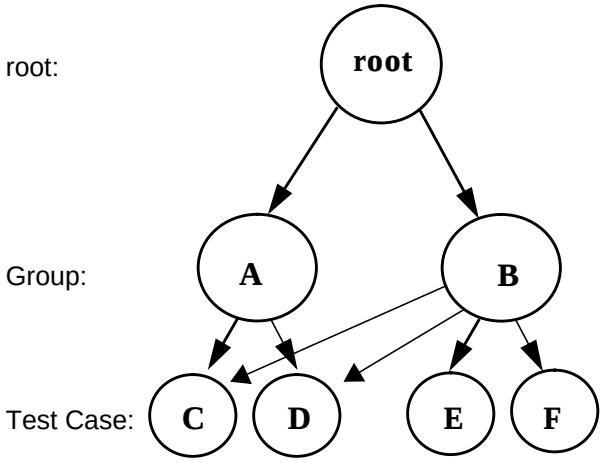


FIGURE 30 Relational Structure of the Test Tree

The tree is structured relationally, not strictly hierarchically. A test can belong to more than one group. In the example in Figure 30 , tests “c” and “d” can be run from both group a and from group b. If the user wants to run just tests “c” and “d”, group “a” should be run. But if the user wants to run tests “e” and “d” along with tests “e” and “f”, group “b” can be run.

Figure 31 shows how the *SMARTS* Run window displays this tree.

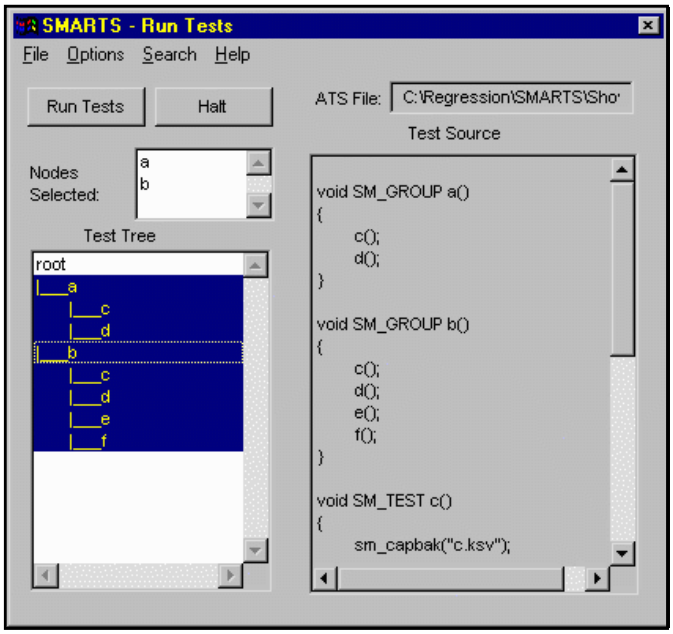


FIGURE 31 "SHOW.ATS" File Structure"

NOTE: Remember, any group can call to any other test, just as any C function can call any other C function. A test, however, should not call another test. This would interfere with *SMARTS*' internal tracking of tests' pass/fail results. Only one test should be "active" at a time.

5.3 ATS Test Function Description

A test under *SMARTS* consists of two steps:

1. Running the test itself.
2. Checking the test's results.

5.3.1 Calling CAPBAK

Under *SMARTS*, you usually run a test by calling *CAPBAK* to replay a recorded *CAPBAK* keysave file. This is done by calling the function **sm_capbak()** with the name of the keysave file. For example:

```
sm_capbak("C:\\STW\\WORK\\DEMO.ATS");
```

SMARTS calls *CAPBAK*, which replays the keysave file *C:\STW\WORK\DEMO.ATS*. (Remember, *SMARTS* uses C as its language, so a backslash " \ " in a path name must be doubled, i.e. " \\ ". Otherwise, *SMARTS* interprets the single backslash as an escape sequence.)

In a single test, you can call **sm_capbak** any number of times - either to replay the same script or to play several different scripts.

5.3.2 Image Differencing

Once you have run the test, you next check the results. *SMARTS* provides two ways to do this -- by either comparing images or ASCII files created during testing. The first comparison function, **sm_image_diff()**, checks to see if the baseline image, captured when the test was recorded, matches the image captured when the test was replayed. For example:

```
sm_image_diff("DEMO.B01","DEMO.R01","");
```

This function call compares the DIB files *DEMO.B01* and *DEMO.R01*(if a full pathname isn't given in the call, *SMARTS* assumes the files are in the current working directory.)

sm_image_diff(), like the *CBDIFF* utility it is based on, can ignore, or "mask out", unimportant differences. It uses mask files, created by *CBDIFF*, which detail which areas of the image to ignore. For example:

```
sm_image_diff("DEMO.B01","DEMO.R01","DEMO.M01");
```

This function call again compares the DIB files *DEMO.B01* and *DEMO.R01*, but this time it reads the mask file *DEMO.M01* to determine what areas of the images to ignore. (See Chapter 9, "COMPARING IMAGE FILES," of the *CAPBAK/MSW* user manual for more details on using image mask files and how to create them.)

5.3.3 ASCII Differencing

The second comparison function, **sm_ascii_diff()**, compares ASCII files. If it finds differences, it records them in a third file (this function call actually calls the SR utility **exdiff** to compare the files.) For example:

```
sm_ascii_diff("DEMO.BSL","DEMO.RSP","DEMO.DIF", "");
```

Here, sm_ascii_diff() compares the ASCII files *DEMO.BSL* and *DEMO.RSP*. If it finds any differences, it places them in the file *DEMO.DIF*. Like image comparison, ASCII comparison allows the user to mask out unimportant differences. For example:

```
sm_ascii_diff("DEMO.BSL","DEMO.RSP","DEMO.DIF",
"DEMO.RC");
```

Again, this function call actually calls the SR utility exdiff to compare the ASCII files *DEMO.BSL* and *DEMO.RSP*. If it finds any differences, it places them in the file *DEMO.DIF*. But, in this case, it reads the file *DEMO.RC* to determine which differences to ignore. (See Chapter 9, "COMPARING IMAGE FILES," in the *CAPBAK/MSW* user manual for more details on using image mask files and how to create them.)

5.3.4 Using Image and ASCII differencing

Both **sm_image_diff()** and **sm_ascii_diff()** return values based on whether they find differences in the files. They return a 0 if they find differences, a 1 if they do not.

The user can use these return values in programming the ATS file. For example, the user may want to perform one action if differences are found, another action if none are. Such a case may look like this:

```
 void SM_TEST demo()
{
      int               result;

      sm_capbak("DEMO.KSV");
      result = sm_image_diff("DEMO.B01",
"DEMO.R01", "");

      if (result == 0)
                  sm_capbak("FAIL.KSV");
      else
                  sm_capbak("SUC-
CEEED.KSV");
}
```

Here the CAPBAK script *DEMO.KSV* is run, then the image files *DEMO.B01* and *DEMO.R01* are compared. If the images are the same, then the script *SUCCEED.KSV* is run. If they are not, then *FAIL.KSV* is run.

Whenever **sm_image_diff()** or **sm_ascii_diff()** is called from within a test function, *SMARTS* automatically keeps track of whether differences are found. A test function can call these functions several times. If any of these calls find differences, *SMARTS* marks the test function as having "FAILED".

When the test function ends, *SMARTS* reports the failed test both to the output file and to the log file. In the example above, if **sm_image_diff()** finds differences, *SMARTS* marks the test **demo()** as having "FAILED".

5.3.5 Operating System Commands

One ATS function call, **sm_os_cmd()**, can both help run tests and report results. When **sm_os_cmd(<DOS_command>)** is called, it runs the system command <**DOS_COMMAND>** that the user specifies. *SMARTS* brings up a DOS window and runs the command in that window. When the command completes, the window closes and SMARTS continues. The user can use this command when running a test to set up files before a test and to clean up after it. When checking diffs, the command can be used to copy or move files.

5.4 ATS Description Language

SR uses a “C” interpreter language for use with the entire TestWorks product set. It works with *SMARTS* test scripts, as well as *CAPBAK*/X and *CAPBAK/MSW* keysave files. It supports “C” scalar data types, most “C” expressions, and some control-flow constructs. This interpreter will allow you to execute source code without going through the process of compiling and linking, and is helpful in the initial stages of the development process.

5.4.1 Input File Syntax

The input file format is a subset of the “C” language. The following sections describe the supported subset. Data Types.

5.4.2 Data Types

This interpreter supports the following scalar types:

- char
- short
- int
- long
- float
- double
- void

Arrays of the scalar type are also supported.

NOTE: The following data types are NOT supported at this time:

- **typedef**s
- structures
- unions
- **enum**s

5.4.3 Expressions

The ANSI Standard details the precedence and conversion rules. *SMARTS* follows the ANSI Standard.

5.4.4 Constants

Fixed and floating-point constants are allowed as specified by the ANSI Standard. Double-quoted strings and single-quoted characters are allowed. Long and unsigned constants are NOT supported.

5.4.5 Variables

Variables of up to 31 characters are supported with standard "C" naming conventions.

5.4.5.1 Supported (type int)

For the type **int**, the following “C” expression operators are supported.

- sizeof
- =
- +
- - (unary)
- - (binary)
- /
- %
- |
- **&** (binary)
- ^
- <
- >
- <=
- >
- = =
- **!**=
- **!**
- ++
- __
- >>
- <<
- ~
- function call
- array reference

5.4.5.2 Not Supported

The following operators are NOT supported at this time.

- ?
- casts
- ->
- &&
- | |
- ..

5.4.6 Statements

5.4.6.1 Statement Constructs Supported

- expressions
- for
- while
- if
- break
- return
- compound statements

5.4.6.2 Statement Constructs Not Supported

The following statement constructs are NOT supported at this time.

- switch
- continue
- goto
- do...while
- statement labels

5.4.7 Error Messages

The C language interpreter supports the following diagnostic error messages. Italicized words represent parameters that are replaced by variable names or character strings.

1. Expected symbol *token*.
2. Missing “]” in array declaration
3. Error in arg list. Wanted a symbol, not a *string*.
4. Bad argument syntax.
5. Can’t have nested functions.
6. Expected "{"
7. Missing "(" after function name.
8. Missing ")" in function call.
9. Missing "]"
10. Noninteger operand to "!"
11. Noninteger operand to ’‘~"
12. Bad operand to ’++’
13. Bad operand to ’_’
14. Unmatched parentheses.
15. Unexpected token in expression: *string*
16. Bad subscript to expression.
17. Missing "]" in array subscript.
18. Illegal LHS to assign up.
19. Unexpected token in expression: *’string’*
20. End of file before end of comment.
21. No main function.
22. Missing semicolon.
23. Missing ’(’ after if
24. Missing ’)’ after if
25. Missing ’(’ after while
26. Missing ’)’ after while.
27. Missing ’(’ after for.
28. Missing ’)’ after for
29. Internal error in cint, premature token list end
30. Missing ’}’
31. Bad function name *string*

5.4.8 SMARTS Functions

The following runtime library functions are available.

sm_capbak*(<keysave filename>)*

This function call calls up *CAPBAK MS-Windows* to play back the keysave file *<keysave filename>*. *CAPBAK* works the same as from the GUI, but no user intervention is needed. *SMARTS* can thus control the playback of tests that have been recorded with *CAPBAK MS-Windows*. See the documentation on *CAPBAK MS-Windows* for more information on using that utility.

sm_os_cmd*(<DOS command>)*

This function call runs the DOS system command *<DOS command>*. *SMARTS* brings up a DOS window and runs the command in that window. When the command completes, the window closes and *SMARTS* continues. The user can thus run any legal DOS command from *SMARTS* as part of a test.

sm_image_diff*(<baseline image filename>, <response image filename>, <mask filename>)*

This function call compares the images stored in the DIB files *<baseline image filename>* and *<response image filename>*. If these files have differences that are not important, the user can ignore those differences by using the mask file *<mask filename>* that was created by *CBDIFF*. See the documentation on *CBDIFF* for more information on image file comparison and the use of mask files.

sm_ascii_diff*(<baseline ASCII filename>, <response ASCII filename>, <difference filename>,<mask filename>)*

This function call calls on the utility *EXDIFF* to compare the ASCII files *<baseline image filename>* and *<response image filename>*. It stores the differences, if any, in *<difference filename>*. If these files have differences that are not important, the user can mask out those differences by using the mask file *<mask filename>*. See the documentation on *EXDIFF* for more information on how it compares files and uses masking.

sm_open(<*filename*>)

Opens the file for reading. Returns an integer file handle that is used by other *SMARTS* routines; returns zero if it can't open the file. *SMARTS* can only have five files open at a time, so close files after they are no longer used.

Example:

```
int i;
i=sm_file_open ("c:\\foo.text")
```

sm_close(<*file_handle*>)

Closes the identified by the file handle.

Example:

```
int i;
i=sm_file_open ("c:\\foo.text")
```

cb_close

sm_read_string(<*char_array*>, <*delimiter*>, <*file_handle*>)

Reads a string from the opened file (<*file_handle*>) and fills (<*char_array*>) with it. It continues filling the string <*char_array*> until it reaches the character identified by <delimiter>. It then puts a NULL character at the end of the array and returns the number of characters it read. It does NOT put the delimiter in the array, it returns.

Example:

```
int i;
char text [80];
i=sm_file_open ("c:\\foo.text")
sm_read_string (text,",",i);
sm_close(i);
```

CHAPTER 6

Executing Tests

This chapter explains how to execute tests from the **Run Tests** window.

6.1 Invoking the Run Tests Window

This chapter describes test execution. Viewing execution reports is examined in the following chapter. Tests are executed from the **Run Tests** window. To invoke the **Run Tests** window, perform the following:

1. From the **Main** window, click on the **Run** button.
2. The **Run Tests** window indicated in Figure 32 pops up.

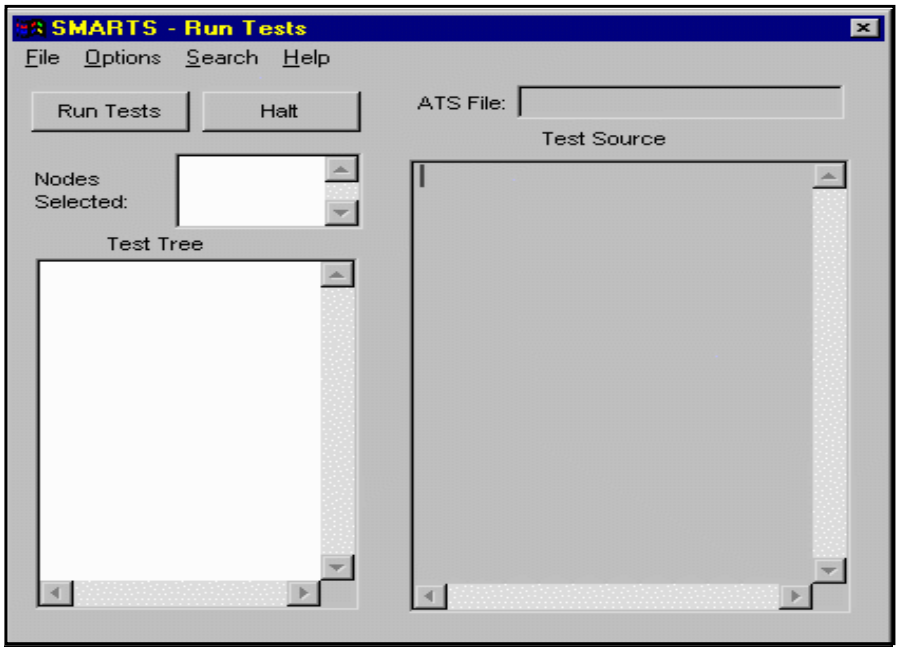


FIGURE 32 Invoking the Run Tests Window

6.2 Selecting the Test to Run

To select the tests you want to run, you must first load an ATS file. This is done by choosing **Load ATS** from the **Run Tests** window's **File** pull-down menu.

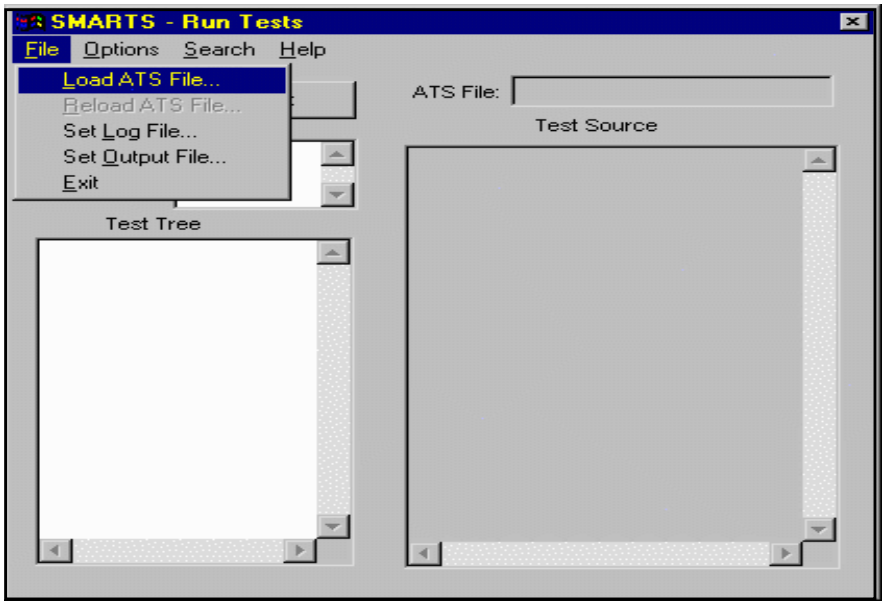


FIGURE 33 File Pull-Down Menu from Run Tests window

After loading the ATS, your **Run Tests** window should appear like Figure 96. Use the mouse to click on a test or test group in the **Test Tree** box of the **Run Tests** window. The tests that will be run are the ones which appear highlighted (with darkened backgrounds as in Figure 33) in the **Test Tree** box. They are identified in the **Nodes Selected** box of the **Run Tests** window. You can use the scroll bars at the right-hand side of the **Nodes Selected** box to move to different positions in the current test or test group.

6.2.1 Selecting a Log File

You must choose a file name for the test results to be written. You do this by choosing **Set Log File** from the **File** pull-down menu. If no log file is chosen, *SMARTS* will write the information to the default file *log.log*.

6.2.2 Selecting an Output File

You must also choose a file name to which the test output will be written. This is done by choosing Set Output File from the File pull-down menu. If no output file is chosen, the default *output.out* is used.

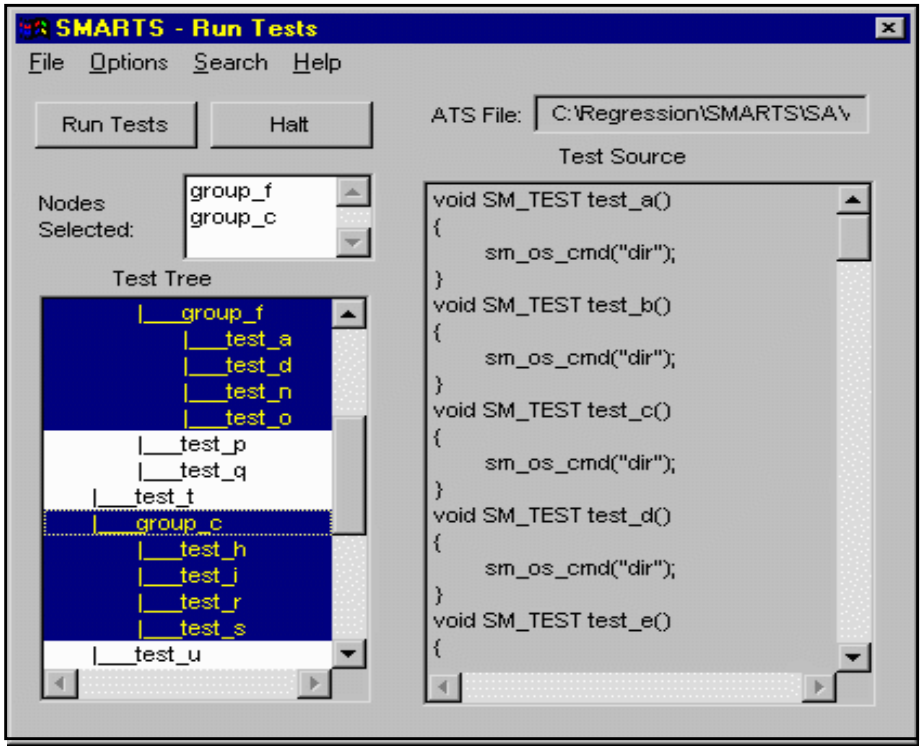


FIGURE 34 Selecting Tests to Run from the Run Tests Window

In Figure 34 , *demo4, demo, demo2, demo, demo3,* and *demo2* (highlighted) will be run. Note that *demo* and *demo2* belong to more than one group and are each executed twice. Also note that you can use the **Search** pull-down menu to expedite location of particular tests or test group in larger trees.

If you click on the **root** name, the entire ATS will be highlighted. You can also click on individual file names to select and de-select groups or individual tests. For instance, you can select a group, yet de-select tests within

that group. The names of the tests and groups selected will appear in the **Nodes Selected** text box.

Once the test (s) you want to run are highlighted, you can choose execution options, which are discussed next.

6.3 Selecting Execution Options

There are three settings which can be controlled from the **Options** pull-down menu: output options, display options, and actions outputs. These three settings control what information is kept and deleted after test executions, how the information is displayed within the **Run Tests** window, and how many times tests are run. Each will be explained in further detail in the succeeding pages.

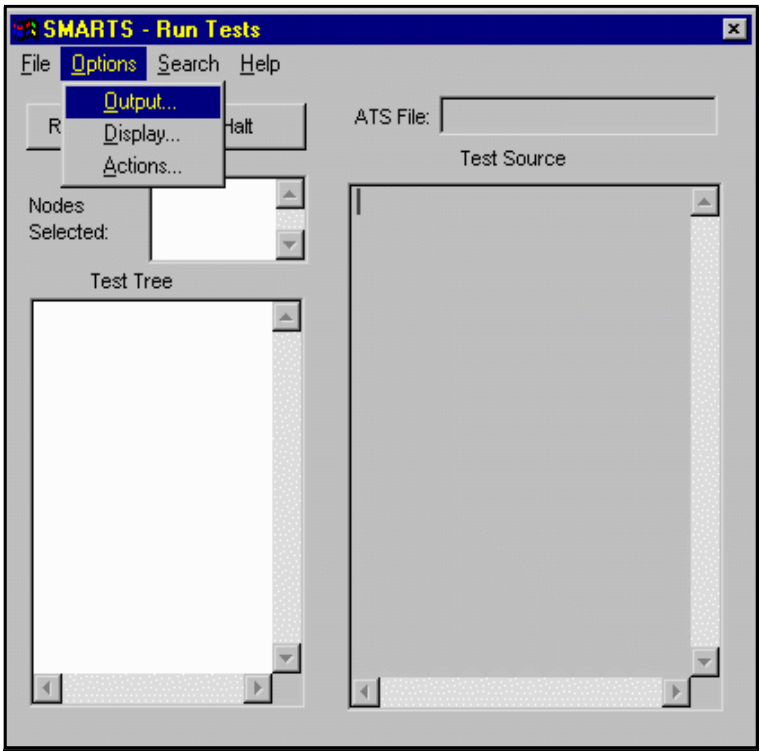


FIGURE 35 Options Pull-Down Menu

6.3.1 Output Options

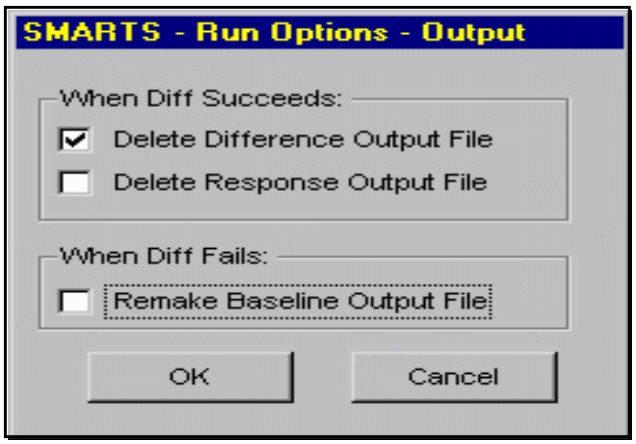


FIGURE 36 Output Options Window

This window allows you to delete the difference output file and/or the response output file when differencing succeeds. It also gives you the option of remaking the baseline output file if differencing fails. You can select any combination of these options by clicking on the appropriate boxes next to each: if an option is selected, its box will be filled with an "x".

When you are satisfied with the options chosen, click **OK**. If you wish to exit this window without implementing your choices, click **Cancel**.

6.3.2 Display Options

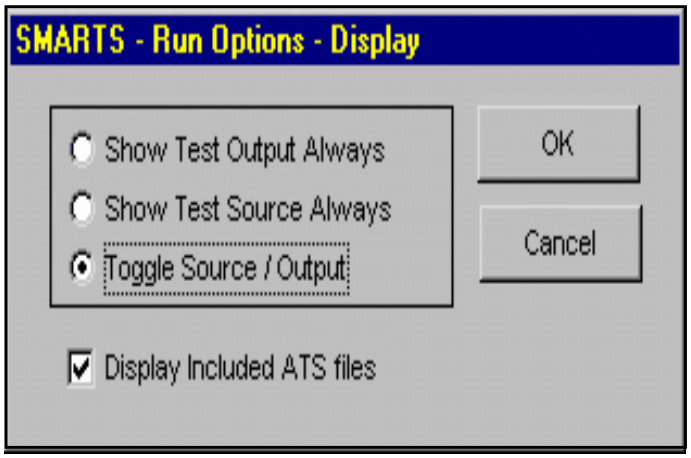


FIGURE 37 Display Options Window

This window controls what is displayed in the **Test Source/Test Output** box of the **Run Tests** window. You can have the Test Output always displayed, the Test Source always displayed, or toggle between the two (toggling is the default setting). *You can only choose one of the three options.*

You can also display the included ATS files by clicking on the box at the bottom of the window.

Once you are satisfied with your display options, click **OK**. If you want to nullify your choices, click on **Cancel**; and the window will close and the display options will return to their default settings.

6.3.3 Action options

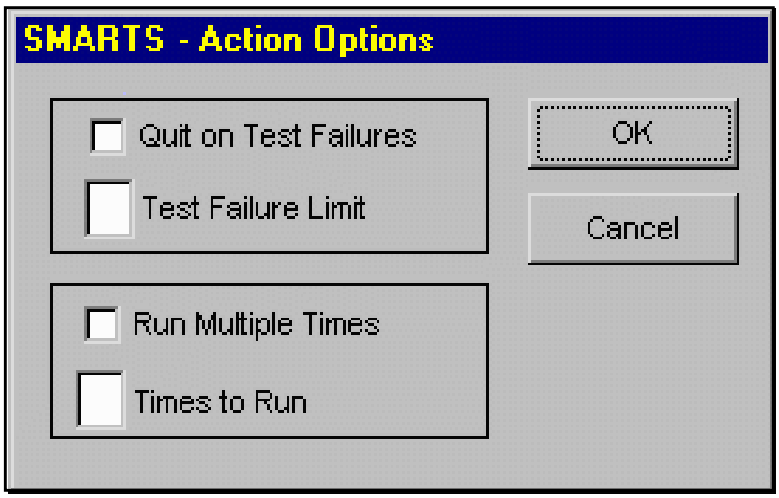


FIGURE 38 Action Options Window

This window determines when an ATS execution is completed, and how many times tests are run. You can choose to quit on a specified number of test failures (the default is 2). If you do not specify the number of test failures to quit on, the ATS will run until all tests are completed.

You may find it helpful to run a test or tests more than once. You can choose how many times to run a particular test or suite of tests using the **Run Multiple Times** option; the default is 2.

Once you are satisfied with the action options, click on **OK**. If you want to exit the window without saving your choices, click on **Cancel**.

6.4 Executing the Test Cases

After determining the type of test execution processing, execute the selected ATS test(s) by performing the following:

1. Click on the **Run Tests** activation button.
2. As each test executes, the current test outcome is compared to the baseline files specified within the test in the **Test Output** box. Clicking on the **Halt** button will stop the execution following completion of the currently executing test case (to use the **Halt** button, however, you must first quit *CAPBAK/MSW*).

The entire process, as well as test outcomes, are scrolled in the **Test Source** area of the **Run Tests** window. A test case which passes is indicated by the statement:

```
TEST PASSES (group identifier)
```

A failed test case is indicated by the statement:

```
TEST FAILS (group identifier)
```

As the *`group identifier`* is listed in the display area of the **Run Tests** window, the location of a failed test is easily determined.

3. When the test session has completed executing, the end-of-test-execution message appears in the **Test Source** box:

```
Run Ended [date]
```

NOTE: After testing, to toggle back and forth between viewing the Test Source and Test Output, double-click on the words “Test Source”. The box will again display the test output. Toggling between source and output is a default setting in the s*marts.ini* file, and can be changed there or in the **Run Tests** window’s **Options** pull-down menu.

Also note that if your ATS file makes any function calls to operating system or ASCII differencing commands, a DOS window appears on the screen and the command is run in that window. The DOS window will appear and disappear from the screen in rapid succession.

6.5 Exiting the Run Tests Window

The **Run Tests** window need not be exited following each test session. The window is exited throughout the user manual for the purpose of practice.

Exit the **Run Tests** window by selecting **Exit** from the **File** pull-down menu, or by selecting **Close** from the GUI's System Menu pull-down.

CHAPTER 7

Viewing Execution Reports

This chapter explains how to select and view execution reports.

7.1 Invoking the Report Window

When the selected test(s) are executed, *SMARTS* automatically stores all the test information in the log file established or the default *log.log* and generates various reports based on the log file data.

Reports are displayed in the **Report on Tests** window. To invoke the **Report on Tests** window, from the **Main** window click on the **Report** button. The **Report on Tests** window indicated in Figure 39 pops up (the **Test Tree** and **Report Results** sections will appear blank until an ATS file is selected and a **Report** button clicked on).

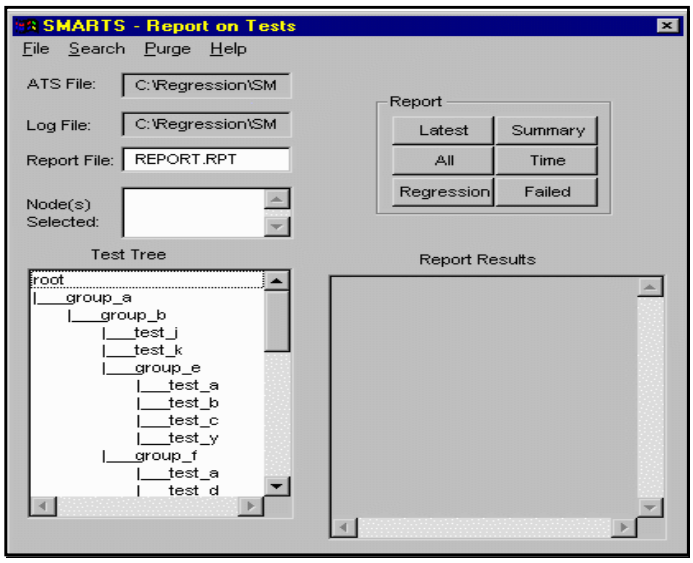


FIGURE 39 Invoking the Report on Tests Window

7.2 Selecting Reports

When selecting a report, keep in mind that the information will only reflect the most recent log file data for each test case. That is, if an entire test suite has not been executed in a while and an individual test case is selected for execution, the generated reports will indicate the recently-executed data for the individual test case. The data provided for the remaining cases in the test suite, however, will reflect information from previous test executions.

The **Report** Window offers the following reports:

- **All** report.
- **Latest** report.
- **Regression** report.
- **Summary** report.
- **Time** Report.
- **Failed** Report.

7.2.1 The Search Option

Use the **Search** pull-down menu to help locate specific tests or test groups for which you want to see reports.

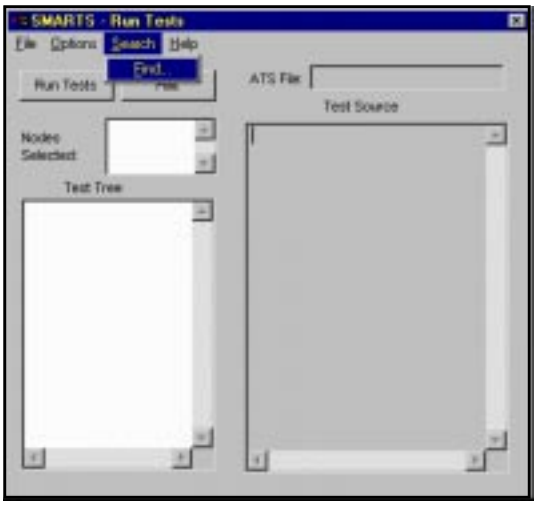


FIGURE 40 Search Pull-Down Menu

After selecting **Search**, the following dialog box will appear:

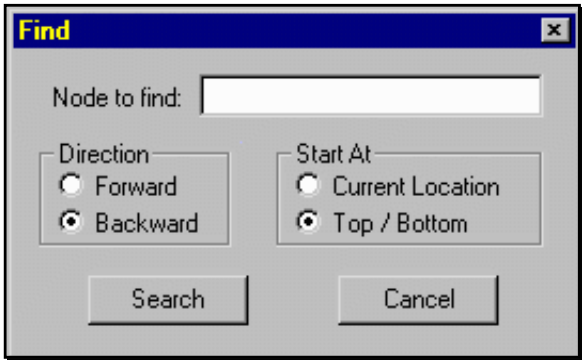


FIGURE 41 Search Dialog Box

7.2.2 Selecting the ATS file, Log file and Report file

To view one of the reports, load an ATS file from the **File** pull-down menu (Figure 42). If you want the test results written to a particular log file, then use the **Set Log File** option from the **File** pull-down menu; otherwise the information will be written to the default *log.log*. This is also true of the report file (the file which contains the outcomes of the tests and is shown in the **Report Results** box); if a particular one is not specified using the **Set Report File** option, the results will be written to the default *report.rpt*.

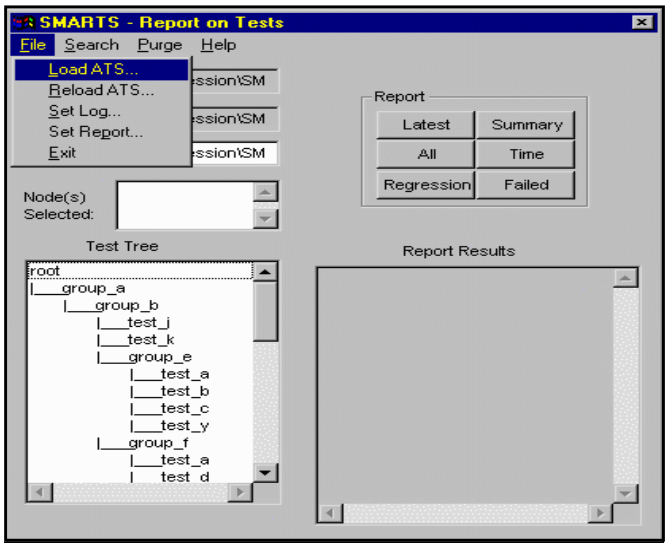


FIGURE 42 Report Window File Pull-Down Menu

After selecting an ATS, log file and report file, choose the tests you want to view reports for, using the mouse to highlight them in the **Test Tree** box. After you have selected the tests, click on the button for the desired report. If you wish to exit the *SMARTS* utility before doing any of these actions, select **Exit** from the **File** pull-down menu.

The selected report is automatically loaded into the **Report Results** box of the **Report on Tests** window. The displayed report can be traversed using the vertical and horizontal scroll bars.

Each *SMARTS* report will be discussed in further detail in the following sections, including a sample output.

NOTE: The first time you request a report of any kind, three xterms will appear and disappear from the screen in rapid succession. This will only happen the *first* time you request a report during that window session.

Also note that if you request more than one report during any one session, requests after the first report may not be immediately visible in the **Report Results** area of the window. You may have to use the scroll bars to view later reports or the entire text of exceptionally long reports.

7.2.3 Latest Report

The **Latest** report contains the name of each test, the PASS / FAIL status of each test case, the date of test activation, execution time in seconds, and includes the error value returned by each test case. Provided information can help locate test cases that failed during execution. A sample **Latest** report is shown in Figure 43 .

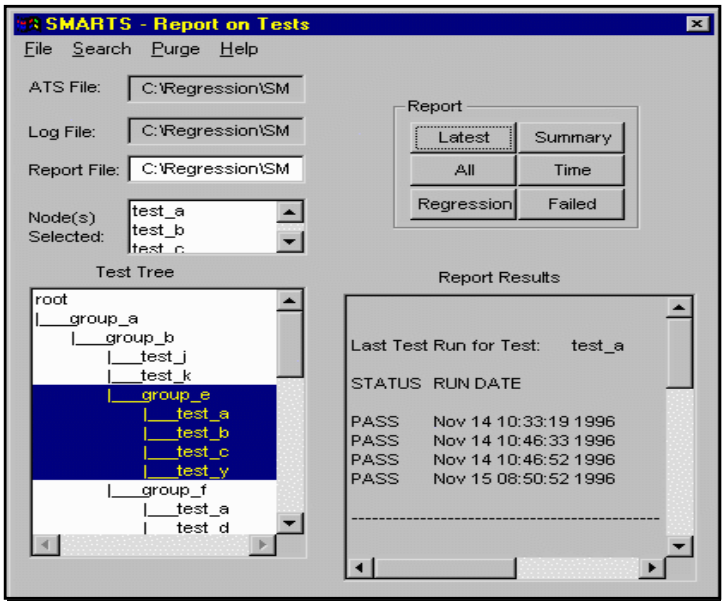


FIGURE 43 Latest Report

NOTE: Use the scroll bars at the bottom of the **Report Results** window to view **Time** and **Error** statistics for this report. They are located in the far right-hand side of the report, so use the right-pointing arrow to view them.

7.2.4 All Report

The **All** report is a summary of all test outcomes maintained in the log file, providing an overview of test regression throughout the testing process. If the log file has not been purged in a while (see Section 7.3 on page 106), then the **All** report can be quite extensive, covering all old and new test executions.

Test information is organized within the **All** Report by test name. For each test, the date(s) and PASS/FAIL status of all recorded test executions are listed. A sample **All** report is shown in Figure 44 .

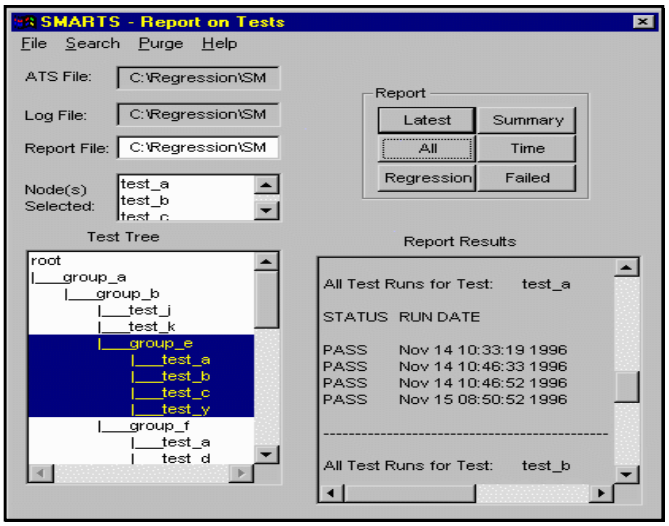


FIGURE 44 All Report

NOTE: Use the scroll bars at the bottom of the **Report Results** window to view **Time** and **Error** statistics for this report. They are located in the far right-hand side of the report, so use the right-pointing arrow to view them.

7.2.5 Regression Report

The **Regression** report indicates only the most-recently-executed test cases whose outcomes have changed since the previous execution, and thus helps to identify bugs that have been either fixed or introduced since the last time the tests were activated.

The **Regression** report contains the name of the test whose outcome has changed, the PASS/FAIL status and date of the most recent test execution, and the PASS/FAIL status and date of the previous test execution. Using this information, the source code to locate a bug can be quickly inspected. A sample **Regression** report is shown in Figure 45 .

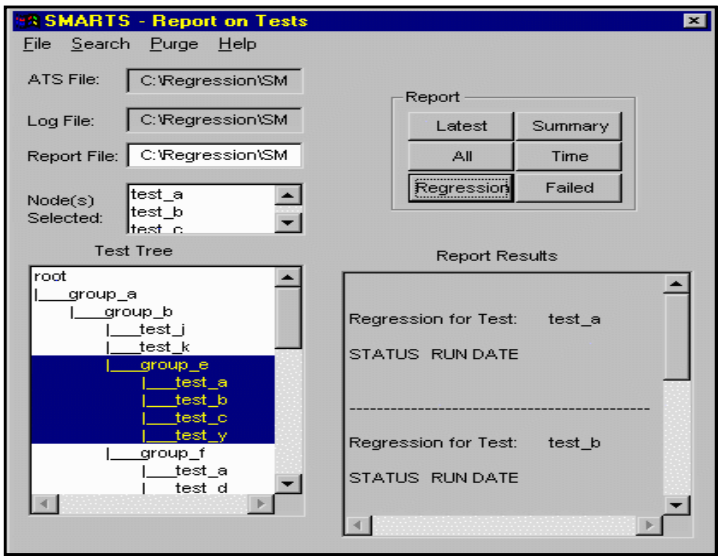


FIGURE 45 Regression Report

7.2.6 Summary Report

The **Summary** report summarizes the total number and percentage of PASS/FAIL outcomes for the current position's selected test(s).

The **Summary** report contains the name of the tests selected, the number and percentage that passed, and the number and percentage that failed. A sample **Summary** report is shown in Figure 46 .

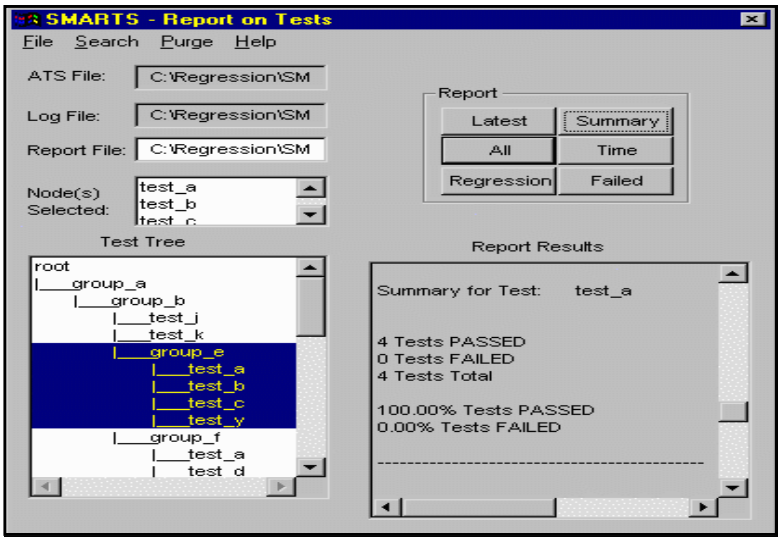


FIGURE 46 Summary Report

7.2.7 Time Report

The **Time** report contains total execution time for a given test or tests. A sample **Time** report is shown in Figure 47 .

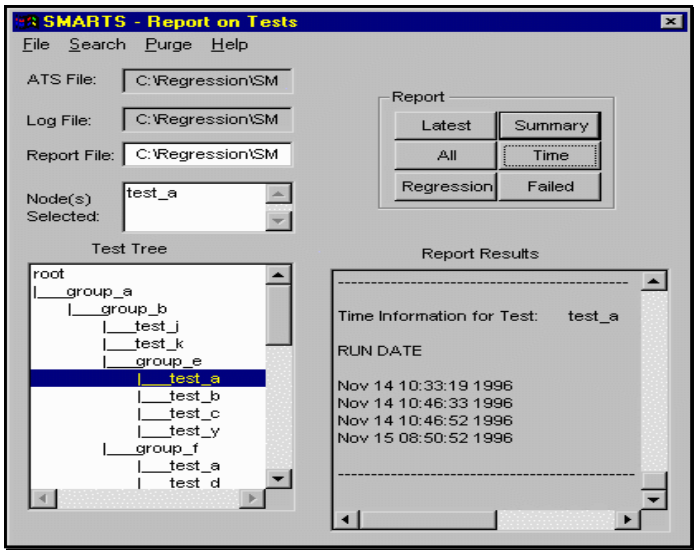


FIGURE 47 Time Report

7.2.8 Failed Report

The **Failed** report lists all the tests which have not passed. This report is cumulative. A sample **Failed** report is shown in Figure 48 .

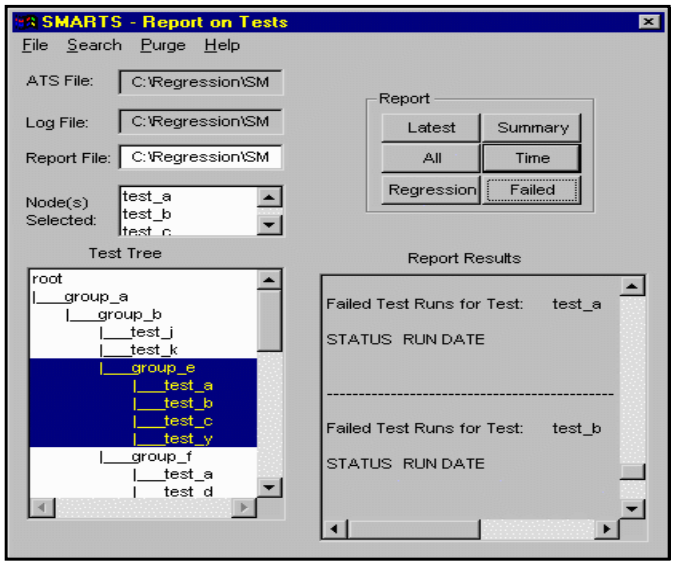


FIGURE 48 Failed Report

NOTE: Use the scroll bars at the bottom of the **Report Results** window to view **Time** and **Error** statistics for this report. They are located in the far right-hand side of the report, so use the right-pointing arrow to view them.

7.2.9 Viewing Time and Error Statistics for Latest, All and Failed Reports

The length of test execution time and number of errors for the **Latest**, **All** and **Failed** reports can be viewed by using the scroll bars at the bottom of the **Report Results** box. These figures appear in the extreme right-hand side of the report, so you must click on the right-pointing arrow until they are visible (Figure 49).

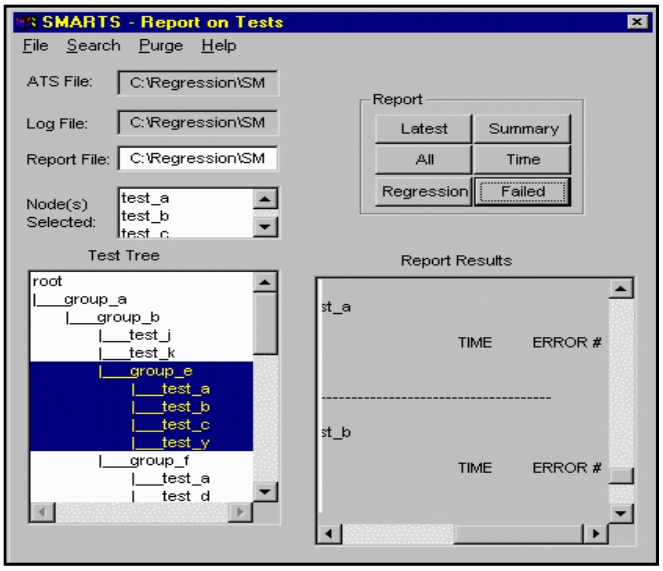


FIGURE 49 Time and Error Statistics for the Failed Report

7.3 Purging the Log File

All testing information, such as test case name, timing information and return values, is stored in the log file (the default name for this file is *log.log*)

Over a period of time, the log file can become very large from the accumulating test information and may require extensive disk space. After executing a large number of test executions, it is recommended that you purge the log file. Purging will leave only the most recent test execution information for each test case.

To purge the log file:

From the **Report** Window, click on the **Purge** pull-down menu.

FIGURE 50 Purge Log File Pull-Down Menu

There is only one choice: **Purge Log File**.

The message box below pops up:

FIGURE 51 Purge Log File Message Box

Click **OK** to purge the log file; and **Cancel** if you do not want to purge the log file. The **Purge Log File** action deletes old log file records and maintains only the most current data on each test case. If a test, for instance, were executed twenty times, only the twentieth activation would remain in the log file after purging. If the log file has been purged recently, and if you try to purge it again before any further tests have been executed, the message box in Figure 52 appears.

FIGURE 52 "Log file already purged" Message Box

Purging will not affect the **Latest** and **Summary** reports; however, the **All** report will only have the most recent data and the **Regression** report will be empty until tests are executed again.

7.4 Exiting the Report Window

The **Report** window need not be exited following each test session. The window is exited throughout the user manual for the purpose of practice.

1. Exit the **Report** window by clicking on the window's **Close** button.

Recommended Usage

This appendix explains where the setup information for SMARTS/MSW is stored and gives you instructions on how to change it.

A.1 Automated Regression Testing

SMARTS/MSW is a powerful and effective tool for use in automated software testing. It is important to appreciate, however, that even though the *SMARTS/MSW* application is a comprehensive organization tool, the "engine" of an effectively automated testing system is test planning and script writing.

In practice, using *SMARTS/MSW* is only *part* of automating the regression testing process. Other parts of the process include creating test baseline (which may involve *CAPBAK/MSW*) and employing a variety of comparison methods.

A.2 Organizing Tests

There are several common-sense guidelines to remember when organizing a regression test suite:

- Organize tests into a hierarchy to facilitate test selection.
- When possible, divide tests into smaller, function-driven test suites. Should test re-structuring be desired, modular tests will expedite the re-structuring process.
- Like tests — tests for similar features of a system — should be grouped together.
- Always try to determine the most effective way to execute comparisons between baseline and response files. If the *STW/Regression* product bundle is available on the current system, the *CBDIFF* utility can be employed.

A.3 ATS Creation

After developing comprehensive test scripts, the test plan is transformed into an ATS (Automated Test Script). The ATS file is read by *SMARTS/MSW* to automate the testing process. Creating an ATS file is the most important and time-consuming part of using *SMARTS/MSW*. Follow the suggestions below to lessen the burden.

- In many ways, the ATS file structure should emulate the recommended organization of test suites. Within the ATS file, a minimal number of function-driven test groupings should be arranged. This structure facilitates both the debugging and ATS restructuring process. For a first-time user, a large, detailed ATS file can be overwhelming.
- Indent each group and test case within the ATS to improve readability.

For complete information on organizing tests within the ATS file, please refer to Chapter 5, "CREATING AN ATS".

A.4 Executing the Test Suite and Generating Reports

Having created the ATS file, the majority of user-required procedures are completed. The remaining procedures are to execute the test suite and view generated reports.

- When a test is executed, all of the resulting information is stored in a log file. Over time this log file can become very large, accumulating extensive amounts of disk space. Therefore, use the **Report** window's **Purge Log File** option.
- To immediately determine whether a test has regressed since its previous execution, display the **Regression** report.

APPENDIX B

Customizing SMARTS/MSW

Run-time parameters for *SMARTS/MSW* may be set from the initialization file, *smarts.ini*.

B.1 Initialization File: Definition

This file consists of a series of parameters which are set one per line and listed in any order. If no parameters are specified, then the embedded default values for the *SMARTS* application are used.

B.2 Initialization File Parameters

The following parameters can be set in the initialization file:

EDITOR=”NOTEPAD.EXE”

EDIT_SELECT_FILE=YES

The editor for the ATS file. The default editor is **Notepad**. If `EDIT_SELECT_FILE` is set to the default `YES`, the current test will be displayed in **Notepad**’s window; if it is set to `NO`, a blank **Notepad** screen will appear.

RUN_MULTIPLE_TIMES=NO

TIMES_TO_RUN=2

If RUN MULTIPLE TIMES is set to the default NO, the test or group will only run once. If that parameter is set to YES, you can set TIMES_TO_RUN to any number you wish; the default is 2.

QUIT_ON_FAILURE=NO

FAILURE_LIMIT=2

The ATS will not quit if one test fails. However, if the QUIT ON FAILURE is set to YES, testing will cease after the number of failures specified by the FAILURE_ LIMIT parameter; default is 2.

DELETE_RESPONSE_FILES=NO

After test comparisons are made, maintain all response files indicated within a test case’s evaluation with baseline clause.

DELETE_DIFFERENCE_FILES=YES

Does not save the difference output when a test case fails. `YES` is the default parameter.

DISPLAY_INCLUDED_FILES=YES

Show the names of included files during start-up. `YES` is the default parameter.

DISPLAY_INCLUDED_FILES=YES

Displays the included files within the test tree hierarchy. `YES` is the default parameter.

OUTPUT_OR_SOURCE_DISPLAY=TOGGLE

Toggles between the source clause from the ATS file, and the test output, for each test case during test execution.

B.2.1 Sample Initialization File

Below is the default RC file *smarts.ini*:

```
[SMARTS]
REMAKE_BASELINE_FILES=NO
DELETE_RESPONSE_FILES=NO
DELETE_DIFFERENCE_FILES=YES
OUTPUT_OR_SOURCE_DISPLAY=TOGGLE
DISPLAY_INCLUDED_FILES=YES
EDITOR=”NOTEPAD.EXE”
EDIT_SELECT_FILE=YES
RUN_MULTIPLE_TIMES=NO
TIMES_TO_RUN=2
QUIT_ON_FAILURE=NO
FAILURE_LIMIT=2
```

MAKEATS Utility

This appendix discusses the **makeats** utility, which helps the user create their own ATS scripts.

C.1 Description of makeats

makeats is a utility that transforms a simple outline of a test tree hierarchy into an ATS file containing that structure and membership information.

You first edit a file that has a tabular structure (described below) and run that file into **makeats**. You then edit the resulting ATS file so that it incorporates the specifics of your test structure.

When managing your ATS files, it is a very good idea to have all of the "test tree structure information" in one place, i.e. all in one file. That file then can **`#include`** all of the other data -- including data found in files stored in sub-directories. The advantage of this is that you only have to edit *one* file, if for any reason you have to modify the test tree.

The test tree may be hierarchical or it may be relational. However, when you are taking advantage of the relational feature of *SMARTS* you must remember that the names of the tests are uniquely identified by their path names -- even when the structure is relational.

C.2 Invocation and Use of makeats

makeats is invoked by the command **`makeats`**. Optional run-time parameters may be specified on the command line with a dash, followed by an option code letter. If no parameters are specified, default values are assumed. Invalid parameters are ignored.

Below is the required syntax you should use for **makeats**:

`makeats [options]` *infile outfile*

These are the names of the input file and the output file. Pipeline rules are assumed. To read from standard input, just replace *infile* with the standard UNIX "-". For standard output, replace *outfile* with "-".

For example, the command:

```
makeats - -
```

will read from standard input (the keyboard) and write to standard output (the screen).

C.2.1 Command Line Options

These are the command line options:

-F	Fast Generation Switch. The input file interprets quickly into a set of `#include` structures that you can later fill in.
-help	Help Switch. Generates a full description of the correct calling parameters for **makeats**. This information is also generated whenever an incorrect calling sequence is used. No input file is needed for this option.
-M	Generate ATS file for MS-Windows.
-s *N*	Space Insertion Switch. Inserts *N* line spaces between major ATS elements.The default value is 1.
-t	Test Output Switch. Bypasses normal mode ATS production and displays the hierarchical *SMARTS* test structure for the input file that would be produced if the **-t** switch was not present.

C.2.2 ATS Sample Input File

```
test1*&
test2
test3&*
test4&*#4
test5*
test6*
test7#3&
test8#2
test9*

group 1
      test1
      test2
      test3
group 2
      test4
      test5
      test6
group 3
      test7
      test8
      test9
```

FIGURE 53 Sample ATS Input File

Each test case must be defined in the **files** area before it is called later in the script--as in C programming, tests must be defined before they can be called. It is required that this definition area of the ATS be named **files**. In the area below the definitions, the actual ATS hierarchy is specified.

A default test, with no parameters, is an image differencing (as in test 2). A test followed by an "*****" (test 5) is an ASCII file differencing. A test followed by an "**&**" specifies that a mask file will be used, and a test name followed by a "**#n**" specifies the number of images to difference,"**n**" representing the number (in test 8, two image differences are specified). You can use these parameters is various ways to achieve desired results: test 4 uses a mask file, has an ASCII differencing, and has four image differences. The ATS input file in Figure 53 is shown in output form in Figure 54 .

```
void SM_TEST test1()
{
sm_capbak("test1.ksv");
sm_ascii_diff("test1.bsl","test1.rsp","test1.diff");
}
void SM_TEST test2()
{
sm_capbak("test2.ksv");
sm_image_diff("test2.b01","test2.r01");
}
void SM_TEST test3()
{
sm_capbak("test3.ksv");
sm_ascii_diff("test3.bsl","test3.rsp","test3.diff");
}
void SM_TEST test4()
{
sm_capbak("test4.ksv");
sm_ascii_diff("test4.bsl","test4.rsp","test4.diff");
}
void SM_TEST test5()
{
sm_capbak("test5.ksv");
sm_ascii_diff("test5.bsl","test5.rsp","");
}
void SM_TEST test6()
{
sm_capbak("test6.ksv");

sm_ascii_diff("test6.bsl","test6.rsp","");
}
void SM_TEST test7()
{
               sm_capbak("test7.ksv");

sm_image_diff("test7.b01","test7.r01","test7.m01");

sm_image_diff("test7.b02","test7.r02","test7.m02");
sm_image_diff("test7.b03","test7.r03","test7.m03");
}
void SM_TEST test8()
{
sm_capbak("test8.ksv");
sm_image_diff("test8.b01","test8.r01","");
sm_image_diff("test8.b02","test8.r02","");
}
void SM_TEST test9()
{
sm_capbak("test9.ksv");
```

```
sm_ascii_diff(“test9.bsl”,”test9.rsp”,””);
}
void SM_GROUP first()
{
test1();
test2();
test3();
}
void SM_GROUP second()
{
test4();
test5();
test6();
}
void SM_GROUP third()
{
test7();
test8();
test9();
}
```

FIGURE 54 ATS Sample Output File

System Considerations

This appendix lists the present limitations of the *SMARTS/MSW* ATS structure and CINT language.

D.1 ATS Structural Limitations

- You cannot select more than than 512 items in the ATS test tree.

D.2 CINT Memory Requirements

- CINT requires 12 times the number of bytes in the ATS file.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W