

--

1 Write programs in 'C' Language to demonstrate the working of the following constructs:

- i) do...while**
- ii) while....do**
- iii) if...else**
- iv) switch**
- v) for**

i) AIM: To demonstrate the working of do.. while construct

Objective

To understand the working of do while with different range of values and test cases

```
#include<stdio.h>
void main(){
inti,n=5,j=0;clrscr();
printf("enter a no");
scanf("%d",&i);
do{
    if(i%2==0){
        printf("%d",i);
        printf("is a even no.");
        i++;
        j++;
    }
    else {
        printf("%d",i);
        printf("is a odd no.\n");
        i++;
        j++;}
} while(i>0&&j<n);
getch();
}
```

--

Output:-

Input	Actual output
2	2 is even number
	3 is odd number
	4 is even number
	5 is odd number
	6 is even number

Test cases:**Test case no: 1****Test case name:** Positive values within range

Input=2	Expected output	Actual output	Remarks
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	Success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no: 2**Test case name:** Negative values within a range

Input = -2	Expected output	Actual output	Remarks
	-2 is even number	-2 is an even	
	-3 is odd number	number	Fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3**Test case name:** Out of range values testing

Input	Expected output	Actual output	Remarks
1234567891222222222222	123456789122222222213	234567891222222215	fail

ii) Aim: To demonstrate the working of while construct**Objective**

To understand the working of while with different range of values and test cases

```
#include<stdio.h>
#include<conio.h>
void main () {
    int i, n=5, j=1;
    clrscr();
    printf("enter a no");
    scanf("%d",&i);
    while(i>0&& j<n)
    {
        if(i%2==0)
        {
            printf("%d",i);
            printf("is a even  number");
            i++;
            j++;
        }
        else
        {
            printf("%d",i);
            printf("is a odd number");
            i++;
            j++;
        }
    }
    getch();
}
```

Output:-**Input**

2

Actual output

2 is even number
 3 is odd number
 4 is even number
 5 is odd number
 6 is even number

Test cases:

Test case no: 1

Test case name: Positive values within range

Input=2	Expected output	Actual output	Remarks
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	Success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test caseno: 2

Test case name: Negative values within a range

Input=-2	Expected output	Actual output	Remarks
	-2 is even number	-2 is an even	
	-3 is odd number	Number	Fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
1234567891222222222222	123456789122222222213	234567891222222215	fail

iii) Aim: To demonstrate the working of if else construct

Objective

To understand the working of if else with different range of values and test cases

```
#include<stdio.h>
#include<conio.h>
void main ()
{
    int i ;
    clrscr();
    printf("enter a number");
    scanf("%d",&i);
    if(i%2==0)
    {
        printf("%d",i);
        printf("is a even number");
    }
    else
    {
        printf("%d",i);
        printf("is a odd number");
    }
    getch();
}
```

Output:-

Input	Actual output
2	2 is even number
	3 is odd number
	4 is even number
	5 is odd number
	6 is even number

Test cases:

Test case no:1

Test case name: Positive values within range

Input=2	Expected output	Actual output	Remarks
	3 is odd number	3 is odd number	
	4 is even number	4 is even number	
	2 is even number	2 is even number	Sucess
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case name: Negative values within a range

Input=-2	Expected output	Actual output	Remarks
	-2 is even number	-2 is an even number	
	-3 is odd number		Fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no:3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
1234567891222222222222	123456789122222222213	234567891222222215	fail

iv) To demonstrate the working of switch constructs**Objective**

To understand the working of switch with different range of values and test cases

```
void main(){
int a,b,c;
clrscr();
printf("1.Add/n2.Sub/n3.Mul/n4.Div/n Enter Your choice");
scanf("%d",&i);
printf("Enter a,b values");
scanf("%d%d",&a,&b);
switch(i) {
    case 1:
        c=a+b;
        printf("The sum of a&b is: %d ", c);
        break;
    case 2:
        c=a-b;
        printf("The Diff of a&b is :%d ",c);
        break;
    case 3:
        c=a*b;
        printf("The Mul of a&b is: %d" ,c);
        break;
    case 4:  c=a/b;
        printf("the Div of a&b is:%d ",c);
        break;
    default: printf("Entered choice is wrong ");
        break;
}
getch();}
```

Output:-**Input**

Enter your choice: 1

Enter a, b values:3,2

Enter your choice: 2

Enter a, b values: 3, 2

Enter your choice: 3

Enter a, b values: 3, 2

Enter your choice: 4

Enter a, b values: 3, 2

--

Test cases:**Test case no:1****Test case name:** Positive values within range**Input**

Enter your choice: 1	Enter your choice: 3
Enter a, b values: 3, 2	Enter a, b values: 3,2
Enter your choice: 2	Enter your choice:4
enter a, b values: 3, 2	Enter a, b values:3,2

Expected Output	Actual output	Remarks
The sum of a&b is: 5	5	Success
The Mul of a&b is:6	6	
The diff of a&b is: 1	1	
The Div of a&b is: 1	1	

Test case no: 2**Test case name:** Out of range values testing

Input	Expected output	Actual output	Remarks
Option: 1			
a=22222222222222			
b=22222222222222	4444444444444444	-2	fail

Test case no: 3**Test case name:** Divide by zero

Input	Expected output	Actual output	Remarks
Option: 4			
a=10 & b=0	Error		fail

v) Aim: To demonstrate working of for construct**Objective**

To understand the working of for with different range of values and test cases

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    printf("enter a no");
    scanf("%d",&i);
    for(i=1;i<=5;i++)
    {
        if(i%2==0)
        {
            printf("%d",i);
            printf("is a even no") ;
            i++;
        }
        else
        {
            printf("%d",i);
            printf("is a odd no");
            i++;
        }
    }
    getch();
}
```

Output:-

Enter a no: 5

```
0 is a even no
1 is a odd no
2 is a even no
3 is a odd no
4 is a even no
5 is a odd no
```

Test cases:**Test case no: 1****Test case name:** Positive values within range

Input=2	Expected output	Actual output	Remarks
	0 is even number	0 is even number	Success
	1 is odd number	1 is odd number	
	2 is even number	2 is even number	

Test case no: 2**Test case name:** Negative values within a range

Input=-2	Expected output	Actual output	Remarks
	0 is even number	0 is an even number	Fail
	-1 is odd number	-1 is even no	
	-2 is even number	-2 is odd no	

Test case no: 3**Test case name:** Out of range values testing

Input	Expected output	Actual output	Remarks
1234567891222222222222	123456789122222222213	234567891222222215	fail

2. Aim: A program written in c language for matrix multiplication fails “Introspect the causes for its failure and write down the possible reasons for its failure”.

Objective:

Understand the failures of matrix multiplication

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3][3], b[3][3] ,c[3][3] ,i,j,k, m,n,p ,q;
clrscr();
printf(Enter first matrix no of rows & cols");
scanf("%d%d",&m,&n);
printf(Enter second matrix no of rows & cols");
scanf("%d%d",&p,&q);
printf("\n enter the matrix elements");
for(i=0;i<m; i++);
{
for(j=0;j<n; j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\n A matrix is\n");
for(i=0;i<m;i++){
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
scanf("%d\t",&b[i][j]);
}
}
printf("\n B matrix is\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
```

```

for(i=0;i<m; i++)
{
for(j=0;j<q; j++)
{
c[i][j]=0;

for(k=0;k<n;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
}
}
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
getch();
}

```

Output:

```

Enter first matrix no of rows & cols 3 3
Enter second matrix no of rows & cols 3 3
Enter matrix elements
A matrix is :  111
               111
               111
B matrix is :  111
               111
               111
Actual Output: 333
               333
               333

```

Test cases:**Test case no:1****Test case name:** Equal no. of rows & cols

Input	Expected output	Actual output	Remarks
Matrix1 rows&cols = 3 3	3 3 3	3 3 3	Success
Matrix2 rows&cols=3 3	3 3 3	3 3 3	
Matrix1: 111	3 3 3	3 3 3	

111
 111
 Matrix2:111
 111
 111

Test case no: 2

Test case name: Cols of 1stmatrix not equal to rows of 2ndmatrix

Input	Expected output	Actual output	Remarks
Matrix1 rows & cols=22 Matrix2 rows & cols=32	Operation Can't be Performed		fail

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
Matrix1 rows & cols=2 2 Matrix2 rows & cols=2 2 1234567891 2222222222 2234567891 2222222221 22222221533 213242424		56456475457	fail

3. Aim: Take any system (e.g.ATM system) and study its system specifications and report the various bugs.

Program:

Features to be tested:

1. Validity of the card.
2. Withdraw Transaction flow of ATM.
3. Authentication of the user's.
4. Dispense the cash from the account.
5. Verify the balance enquiry.
6. Change of PIN number.

Bugs Identified:

Bug-Id	Bug Name
ATM_001	Invalid Card
ATM_002	Invalid PIN
ATM_003	Invalid Account type
ATM_004	Insufficient Balance
ATM_005	Transaction Limit
ATM_006	Day limit
ATM_007	Invalid money denominations
ATM_008	Receipt not printed
ATM_009	PIN change mismatch

Bug Report:

Bug Id: ATM_001

Bug Description: Invalid card

Steps to reproduce: 1.Keep valid card in the ATM.

Expected Result: Welcome Screen

Actual Result: Invalid card

Status: Pass/Fail

Bug Id: ATM_002

Bug Description: Invalid PIN entered

Steps to reproduce:

1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Menu screen should be displayed.

Expected Result: Menu screen displayed

Actual Result: Invalid PIN screen is displayed

Status: Pass/Fail

--

Bug Id: ATM_003

Bug Description: Invalid Account type selected.

Steps to reproduce:

1. Enter a valid user PIN number.
2. Select the withdraw option on the main menu.
3. Choose the correct type of account (either savings or current account).

Expected Result: Enter the Amount screen displayed

Actual Result: Invalid Account type screen is displayed.

Status: Pass/Fail

Bug Id: ATM_004

Bug Description: Insufficient Balance

Steps to reproduce:

1. Menu screen should be displayed.
2. Select the withdraw option.
3. Select the correct type of account.
4. Enter the sufficient amount to withdraw from the account.
5. Dispense the cash screen & amount to be deducted from account

Expected Result: Collect the amount screen displayed

Actual Result: Insufficient balance in the account

Status: Pass/Fail

Bug Id: ATM_005

Bug Description: Withdraw Limit per transaction.

Steps to reproduce:

1. Menu screen should be displayed.
2. Select the withdraw option.
3. Select the correct type of account.
4. Enter sufficient amount to withdraw from the account Transaction within the limit.
5. Dispense the cash screen & amount to be deducted from account.

Expected Result: Cash is dispensed and collect the receipt

Actual Result: Transaction limit exceeded screen is displayed

Status: Pass/Fail

Bug Id: ATM_006

Bug Description: Withdraw limit per day

Steps to reproduce:

1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Enter the amount to withdraw from the account.
4. Amount enter is over the day limit (>40000)
5. Amount enter is over the day limit and display screen is displayed.

Expected Result: Cash is dispensed and collect the receipt.

Actual Result: Day limit exceeded screen is displayed.

Status: Pass/Fail

--

Bug Id: ATM_007

Bug Description: Amount enter denominations

Steps to reproduce:

1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Enter the amount which should be in multiples of 100.
4. Cash Dispenser screen is displayed.

Expected Result: Collect the amount screen is displayed.

Actual Result: Amount enter not in required denominations.

Status: Pass/Fail

Bug Id: ATM_008

Bug Description: Statement not printed

Steps to reproduce:

1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Select the mini statement.
4. Current balance is displayed on the screen.
5. Collect printed receipt of the statement.

Expected Result: Collect the mini statement receipt

Actual Result: receipt not printed.

Status: Pass/Fail

Bug Id: ATM_009

Bug Description: PIN mismatch

Steps to reproduce:

1. Keep a valid card in ATM.
2. Enter the authorized PIN.
3. Select the change PIN option on the menu.
4. Enter the current PIN.
5. Enter the new PIN.
6. Retype the new PIN
7. PIN successfully changed displayed on the screen.

Expected Result: PIN change successful.

Actual Result: PIN mismatched due to wrong PIN entered

Status: Pass

4. Aim: Write the test cases for any known application (e.g. Banking application)**Objective:** Understand the guidelines of banking application

TEST CASE ID	1
TEST SCENARIO	Validate the login page enter invalid/ wrong user name and valid password.
TEST CASE	Enter invalid user name and valid password in SBI online Banking login page.
EXPECTED RESULT	System should not allow the customer to login the SBI online Banking login page and it should display the message like "please enter valid user name and password".
ACTUAL RESULT	Customer is not able to login SBI online banking account.
STATUS	Pass.
TEST DATA	USER-ID:abcdef PASSWORD:xyz123.

TEST CASE ID	2
TEST SCENARIO	validate the login page enter invalid user name and invalid password
TEST CASE	Enter invalid user name and invalid pass word in SBI online Banking login page
EXPECTED RESULT	System should not allow the customer to login the SBI online Banking login page and it should display the message like " please enter valid user name and password
ACTUAL RESULT	Customer is not able to login SBI online Banking account
STATUS	Pass.
TEST DATA	USER-ID:abcd PASSWORD:xyz12

TEST CASE ID	3
TEST SCENARIO	Validate the login page enter valid user name and invalid password
TEST CASE	Enter valid user name and invalid password in SBI online Banking login page
EXPECTED RESULT	System should allow the user to login the SBI online Banking login page
ACTUAL RESULT	Customer is not able to login SBI online Banking account
STATUS	Pass.
TEST DATA	USER-ID:abcdefg PASSWORD:xyz1234

TEST CASE ID	4
TEST SCENARIO	Validate the login page enter valid user name and valid password
TEST CASE	Enter valid user name and valid password in SBI online Banking login page
EXPECTED RESULT	System should allow the user to login the SBI online Banking login page
ACTUAL RESULT	Customer is logged into SBI online Banking login page
STATUS	Pass.
TEST DATA	USER-ID:abcd PASSWORD:xyz

TEST CASE ID	5
TEST SCENARIO	Validate the login page enter invalid user name and valid password.
TEST CASE	Enter valid user name and valid password in SBI online Banking login page.
EXPECTED RESULT	System should not allow the user to login the SBI online Banking login page.
ACTUAL RESULT	Customer does not logged into SBI online Banking login page.
STATUS	Pass.
TEST DATA	USER-ID:abcd PASSWORD:xyz

TEST CASE ID	6
TEST SCENARIO	Validate the user information or detail in the profile page
TEST CASE	a) User should able to login SBI login page User should able to click on profile link c) On clicking profile link uses should able to see all user details like 1) User /custome r name 2) User/cust omer address details b) User/cust omer phone number
EXPECTED RESULT	a) User/cus tomer should able to login SBI login page with valid b) Custome r should be able to click profile link. c) Custome r should see all the custome r informat ion once he clicking on profile hyper link
ACTUAL RESULT	Customer is not able to see phone or mobile number
STATUS	Fail
TEST DATA	USER-ID:abcdsc PASSWORD:xyzuyu

5. Create a test plan document for any application (e.g. Library Management System)

Library Management System (LMS)
Test Report
Version 1.0

Author

#Name of the Student

Revision History

Date	Version	Description	Author
29th January, 2019	1.0	First version	Test Team

1. BACKGROUND

The Library Management System is an online application for assisting a librarian in managing a book library in a University. The system would provide basic set of features to add/update clients, add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification.

2. INTRODUCTION

This test report is the result for testing in the LMS. It mainly focuses on two problems: what we will test and how we will test.

3. RESULT**3.1 GUI test**

Pass criteria: librarians could use this GUI to interface with the backend library database without any difficulties

Result: pass

3.2 Database test

Pass criteria: Results of all basic and advanced operations are normal (refer to section 4)

Result: pass

3.3 Basic function test**3.3.1 Add a student**

Pass criteria:

- ☐ Each customer/student should have following attributes: Student ID/SSN (unique), Name, Address and Phone number.

Result: pass

- ☐ The retrieved customer information by viewing customer detail should contain the four attributes.

Result: pass

3.3.2 Update/delete student

Pass criteria:

--

- ☐ The record would be selected using the student ID
Result: pass
- ☐ Updates can be made on full. Items only: Name, Address, Phone number
Result: pass
- ☐ The record can be deleted if there are no books issued by user.
Result: Partially pass
When no books issued by user, he can be deleted. But when there are books Issued by this user, he was also deleted. It is wrong.
- ☐ The updated values would be reflected if the same customer's ID/SSN is called for. Result: pass
- ☐ If customer were deleted, it would not appear in further search queries.
Result: pass

3.3.3 Add a book

Pass criteria:

- ☐ Each book shall have following attributes: Call Number, ISBN, Title, Author name.
Result: pass
- ☐ The retrieved book information should contain the four attributes.
Result: pass

3.3.4 Update/delete book

Pass criteria:

- ☐ The book item can be retrieved using the call number
Result: did not pass. Cannot retrieve using the call number
- ☐ The data items which can be updated are: ISBN, Title, Author name
Result: pass
- ☐ The book can be deleted only if no user has issued it.
Result: partially pass. When no user has issued it, pass. When there are user having issued it, Did not pass
- ☐ The updated values would be reflected if the same call number is called for Result: pass
- ☐ If book were deleted, it would not appear in further search queries.
Result: pass

3.5 Search for book

Pass criteria:

- ☐ The product shall let Librarian query books' detail information by their ISBN number or Author or Title.
Result: pass
- ☐ The search results would produce a list of books, which match the search parameters with following Details: Call number, ISBN number, Title, Author
Result: pass
- ☐ The display would also provide the number of copies which is available for issue Result: pass
- ☐ The display shall provide a means to select one or more rows to a user-list Result: pass
- ☐ A detailed view of each book should provide information about check-in/checkout status, with the Borrower's information.

Result: pass

- ☐ The search display will be restricted to 20 results per page and there would be means to navigate from sets of search results.

Result: pass

- ☐ The user can perform multiple searches before finally selecting a set of books for check in or checkout. These should be stored across searches.

Result: pass

- ☐ A book may have more than one copy. But every copy with the same ISBN number should have same detail information.

Result: pass

- ☐ The borrower's list should agree with the data in students' account Result: pass

3.3.6 Check-in book

Pass criteria:

- ☐ Librarians can check in a book using its call number Result: pass
- ☐ The check-in can be initiated from a previous search operation where user has selected a set of books.
Result: pass
- ☐ The return date would automatically reflect the current system date. Result: did not pass.
- ☐ Any late fees would be computed as difference between due date and return date at rate of 10 cents a day.
Result: did not pass
- ☐ A book, which has been checked in once, should not be checked in again Result: pass

3.3.7 Check-out book

Pass criteria:

- ☐ Librarians can check out a book using its call number Result: pass
- ☐ The checkout can be initiated from a previous search operation where user has selected a set of books.
Result: pass
- ☐ The student ID who is issuing the book would be entered Result: pass
- ☐ The issue date would automatically reflect the current system date. Result: did not pass
- ☐ The due date would automatically be stamped as 5 days from current date. Result: did not pass
- ☐ A book, which has been checked out once, should not be checked out again Result: pass
- ☐ A student who has books due should not be allowed to check out any books Result: did not pass
- ☐ The max. No of books that can be issued to a customer would be 10. The system should not allow checkout of books beyond this limit.
Result: pass

--

3.3.8 View book detail

Pass criteria:

- ☐ This view would display details about a selected book from search operation Result: pass
- ☐ The details to be displayed are: Call number, IBN, Title, Author, Issue status (In library or checked out), If book is checked out it would display, User ID & Name, Checkout date, Due date
Result: for checkout date and due date, did not pass
- ☐ Books checked in should not display user summary
Result: pass
- ☐ Books checked out should display correct user details.
Result: pass

3.3.9 View student detail

Pass criteria:

- ☐ Librarians can select a user record for detailed view
Result: pass
- ☐ The detail view should show:
 - a. User name, ID, Address & Phone number
Result: pass
 - b. The books issued by user with issue date, due date, call number, title
Result: did not pass
 - c. Late fees & Fines summary and total
Result: did not pass
- ☐ The display should match existing user profile
Result: pass
- ☐ The books checked out should have their statuses marked
Result: pass
- ☐ The book search query should show the user id correctly.
Result: pass

3.4 Network test

Pass criteria: Results of operations (ping, ftp and ODBC connectivity check) are normal

Result: did not test this item, because no enough machines and no available environment.

4 ENVIRONMENT USED

4.1 **Hardware:** Core2Duo

4.2 **Software:** Microsoft Windows XP

5 RESOURCES

- 5.1 Developers of the system are involved in testing process (debugging, unit testing, even integrity testing)
- 5.2 Users of the system are involved in testing process (integrity testing)

6. Aim: Study of Win Runner Testing Tool and its implementation

Win Runner is a program that is responsible for the automated testing of software. Win Runner is a Mercury Interactive's enterprise functional testing tool for Microsoft windows applications.

Importance of Automated Testing:

1. Reduced testing time
2. Consistent test procedures – ensure process repeatability and resource independence.
Eliminates errors of manual testing
3. Reduces QA cost–Upfront cost of automated testing is easily recovered over the life time of the product
4. Improved testing productivity–test suites can be run earlier and more often
5. Proof of adequate testing
6. For doing tedious work–test team members can focus on quality areas.

a) Aim: Win runner Testing Process and Win runner User Interface.

Objective: Student should be able to

- ☐ Describes the benefits of automated testing
- ☐ Understand the Win Runner testing process
- ☐ Work with Win Runner user interface

Theory:

The Win Runner Testing Process

Testing with Win Runner involves six main stages:

1. Create the GUI Map

The first stage is to create the GUI map so Win Runner can recognize the GUI objects in the application being tested. Use the Rapid Test Script wizard to review the user interface of your application and systematically add descriptions of every GUI object to the GUI map. Alternatively, you can add descriptions of individual objects to the GUI map by clicking objects while recording a test.

2. Create Tests

Next is creation of test scripts by recording, programming, or a combination of both. While recording tests, insert check points where we want to check the response of the application being tested. We can insert check points that check GUI objects, bitmaps, and databases. During this process, Win Runner captures data and saves it as expected results—the expected response of the application being tested.

3. Debug Tests

Run tests in Debug mode to make sure they run smoothly. One can set break points, monitor variables, and control how tests are run to identify and isolated effects. Test results are saved in the debug folder, which can be discarded once debugging is finished. When Win Runner runs a test, it checks each script line for basic syntax errors, like incorrect syntax or missing elements in **If**, **While**, **Switch**, and **For** statements. We can use the **Syntax Check** options (**Tools>Syntax Check**) to check for these types of syntax errors before running your test.

4. Run Tests

Tests can be run in Verify mode to test the application. Each time Win Runner encounters a check point in the test script, it compares the current data of the application being tested to the expected data captured earlier. If any mismatches are found, Win Runner captures them as actual results.

5. View Results

Following each test run, Win Runner displays the results in a report. The report details all the major events that occurred during the run, such as check points, error messages, system messages, or user messages. If mismatches are detected at check points during the test run, we can view the expected results and the actual results from the Test Results window. In cases of bit map mismatches, one can also view a bit map that displays only the difference between the expected and actual results.

We can view results in the standard Win Runner report view or in the Unified report view. The Win Runner report view displays the test results in a Windows-style viewer. The Unified report view displays the results in an HTML-style viewer (identical to the style used for Quick Test Professional test results).

6. Report Defects

If a test run fails due to a defect in the application being tested, one can report information about the defect directly from the Test Results window.

This information is sent via e-mail to the quality assurance manager, who tracks the defect until it is fixed.

Using Win runner Window

Before you begin creating tests, you should familiarize yourself with the Win Runner main window.

☐ To start Win Runner:

Choose **Programs> Win Runner>Win Runner** on the **Start** menu.

The first time you start Win Runner, the Welcome to Win Runner window and the

—What's New in Win Runner || help open. From the Welcome window you can create a new test, open an existing test, or view an overview of Win Runner in your default browser.

CASE TOOLS & SOFTWARE TESTING LAB MANUAL

If you do not want this window to appear then exit time you start Win Runner, clear the **Show on Start up** check box. To show the **Welcome to Win Runner** window up on start up from within Win Runner, choose **Settings> General Options**, click the **Environment** tab, and select the **Show Welcome screen** check box.

The Main Win Runner Window

The main Win Runner window contains the following key elements:

Win Runner title bar

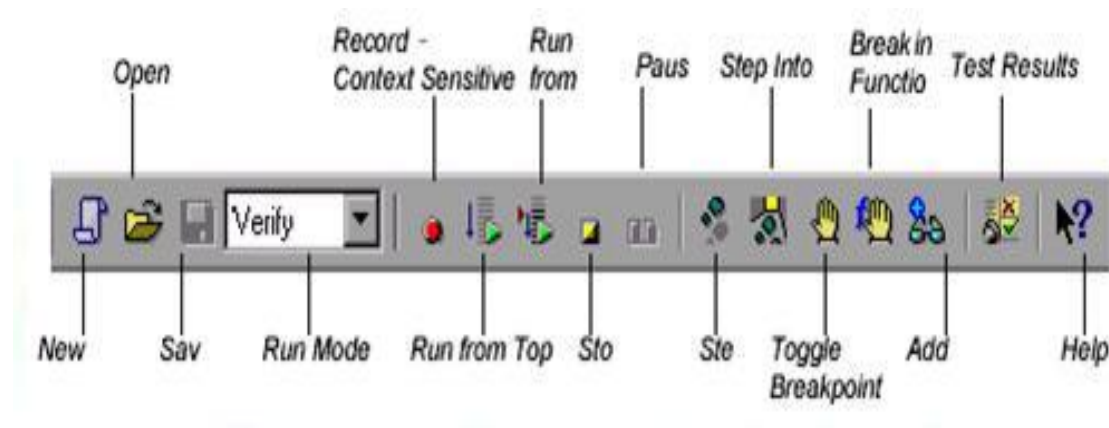
Menu bar, with drop-down menus of Win Runner commands

Standard tool bar, with buttons of commands commonly used when running a test

User tool bar, with commands commonly used while creating a test

Statusbar, with information on the current command, the line number of the insertion point and the name of the current results folder

The Standard tool bar provides easy access to frequently performed tasks, such as opening, executing, and saving tests, and viewing test results



Standard Toolbar

The User toolbar displays the tools you frequently use to create test scripts. By default, the User toolbar is hidden. To display the User toolbar, choose **Window > User Toolbar**. When you create tests, you can minimize the Win Runner window and work exclusively from the toolbar. The User toolbar is customizable. You choose to add or remove buttons using the **Settings>Customize User Toolbar** menu option. When you re-open Win Runner, the User toolbar appears as it was when you last closed it. The commands on the Standard toolbar and the User toolbar are described in detail in subsequent lessons.

Note that you can also execute many commands using soft keys. Soft keys are key board shortcuts for carrying out menu commands. You can configure the soft key combinations for

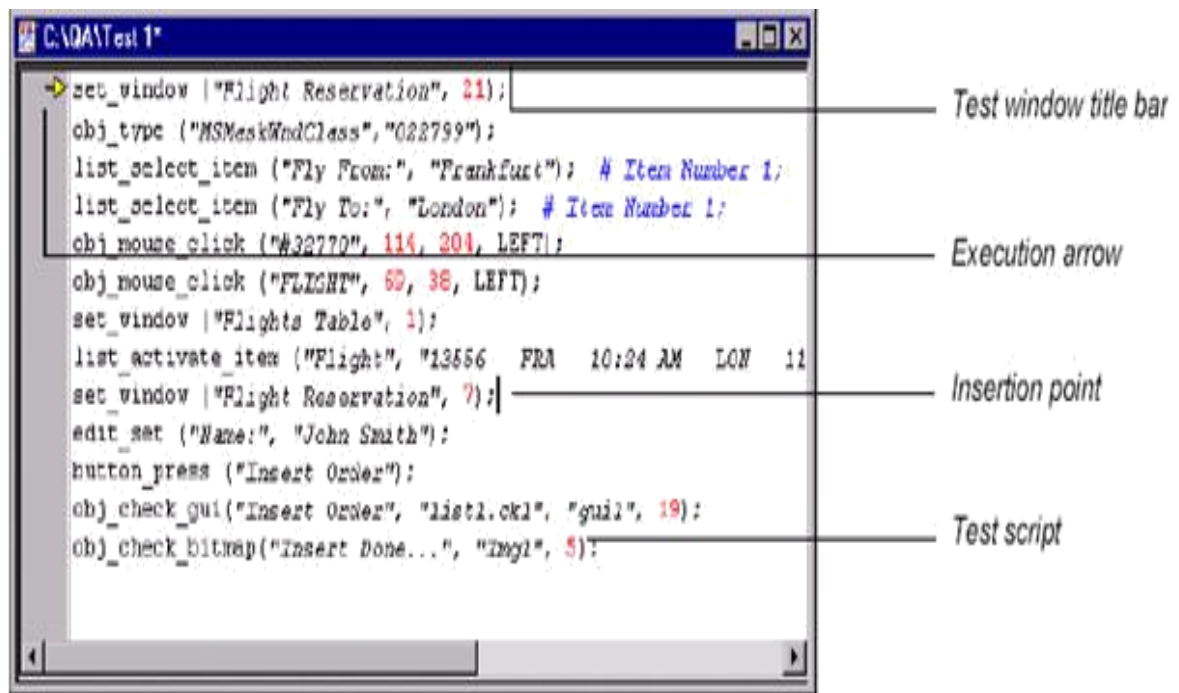
your key board using the Soft key Configuration utility in your Win Runner program group. For more information, see the—Win Runner at a Glance chapter in your Win Runner User's Guide.

Now that you are familiar with the main Win Runner window, take a few minutes to explore these window components before proceeding to the next lesson.

The Test Window

You create and run Win Runner tests in the test window. It contains the following key elements:

- Test window title bar, with the name of the open test
- Test script, with statements generated by recording and/or programming in TSL, Mercury Interactive's Test Script Language
- Execution arrow, which indicates the line of the test script being executed during a test run, or the line that will next run if you select the Run from arrow option Insertion point, which indicates where you can insert or edit text



b) Aim: How Win Runner identifies GUI (Graphical User Interface) objects in an application and describes the two modes for organizing GUI map files.

Objective: Student should be able to

- ☐ Describes the benefits of **GUI (Graphical User Interface) objects**
- ☐ Understand the Win Runner **GUI map files**
- ☐ Work with Win Runner **identifies**

Theory:

Win Runner Uses:

1. With Win Runner sophisticated automated tests can be created and run on an application.
2. A series of wizards will be provided to the user, and these wizards can create tests in an automated manner.
3. Another impressive aspect of Win Runner is the ability to record various interactions, and transform them into scripts. Win Runner is designed for testing graphical user interfaces.
4. When the user makes an interaction with the GUI, this interaction can be recorded. Recording the interactions allows determining various bugs that need to be fixed.
5. When the test is completed, Win Runner will provide with detailed information regarding the results. It will show the errors that were found, and it will also give important information about them. The good news about these tests is that they can be reused many times.
6. WinRunner will test the computer program in a way that is very similar to normal user interactions. This is important, because it ensures a high level of accuracy and realism. Even if an engineer is not physically present, the Recover manager will troubleshoot any problems that may occur, and this will allow the tests to be completed without errors.
7. The Recover Manager is a powerful tool that can assist users with various scenarios. This is important, especially when important data needs to be recovered.

The goal of Win Runner is to make sure business processes are properly carried out. Win Runner uses TSL, or Test Script Language.

Win Runner Testing Modes

Context Sensitive

Context Sensitive mode records your actions on the application being tested in terms of the GUI objects you select (such as windows, lists, and buttons), while ignoring the physical location of the object on the screen. Every time you perform an operation on the application being tested, a TSL statement describing the object selected and the action performed is generated in the test script. As you record, Win Runner writes a unique description of each selected object to a GUI map.

--

The GUI map consists of files maintained separately from your test scripts. If the user interfaces of your application changes, you have to update only the GUI map, instead of hundreds of tests. This allows you to easily reuse your Context Sensitive test scripts on future versions of your application.

To run a test, you simply play back the test script. Win Runner emulates a user by moving the mouse pointer over your application, selecting objects, and entering key board input. Win Runner reads the object descriptions in the GUI map and then searches in the application being tested for objects matching these descriptions. It can locate objects in a window even if their placement has changed.

Analog

Analog mode records mouse clicks, keyboard input, and the exact x - and y-coordinates traveled by the mouse. When the test is run, Win Runner retraces the mouse tracks. Use Analog mode when exact mouse coordinates are important to your test, such as when testing a drawing application.

c) **Aim: How to record a test script and explains the basics of Test Script Language (TSL).**

Objective: Student should be able to

- ☐ Describes Context Sensitive and Analog record modes
- ☐ Record a test script
- ☐ Read the test script
- ☐ Run the recorded test and analyze the results

Theory:

Choosing a Record Mode

By recording, you can quickly create automated test scripts. You work with your application as usual, clicking objects with the mouse and entering keyboard input.

Win Runner records your operations and generates statements in TSL, Mercury Interactive Test Script Language. These statements appear as a script in a Win Runner test window. Before you begin recording a test, you should plan the main stages of the test and select the appropriate record mode. Two record modes are available: Context Sensitive and Analog.

Context Sensitive

Context Sensitive mode records your operations in terms of the GUI objects in your application. Win Runner identifies each object you click (such as a window, menu, list, or button), and the type of operation you perform (such as press, enable, move, or select).

For example, if you record a mouse click on the **OK** button in the Flight Reservation Login Window, Win Runner records the following TSL statement in your test script: button press ("OK"); When you run the script, Win Runner reads the command, looks for the **OK** button, and presses it.

When choosing a record mode, consider the following points

Choose Context Sensitive if...	Choose Analog if...
The application contains GUI objects.	The application contains bitmap areas (such as a drawing area).
Exact mouse movements are not required.	Exact mouse movements are required.
You plan to reuse the test in different versions of the application.	

Recording a Context Sensitive Test:-

In this exercise you will create a script that tests the process of opening an order in the Flight Reservation application. You will create the script by recording in Context Sensitive mode.

- 1 Start Win Runner.**
- 2 Open a new test.**
- 3 Start the Flight Reservation application and log in.**
- 4 Start recording in Context Sensitive mode.**
- 5 Open orders #3.**
- 6 Stop recording.**

7 Save the test.**Recording in Analog Mode:-**

In this exercise you will test the process of sending a fax. You will start recording in Context Sensitive mode, switch to Analog mode in order to add a signature to the fax, and then switch back to Context Sensitive mode

1 Open the Fax Order form and fill in a fax number.

2 Select the Send Signature with Order check box.

3 Sign the fax again in Analog mode.

4 Stop Recording.

5 Save the test.

Running the Test

You are now ready to run your recorded test script and to analyze the test results. Win Runner Provides three modes for running tests. You select a mode from the toolbar.

☐ Use Verify mode when running a test to check the behaviour of your application, and when you want to save the test results.

☐ Use Debug mode when you want to check that the test script runs smoothly without errors in syntax.

☐ Use Update mode when you want to create new expected results for a GUI Check point or bitmap checkpoint.

To run the test:

1 Check that Win Runner and the main window of the Flight Reservation application are open on your desktop.

2 Make sure that the saved test window is active in Win Runner.

3 Make sure the main window of the Flight Reservation application is active.

4 Make sure that Verify mode is selected in the toolbar

5 Choose Run from Top.

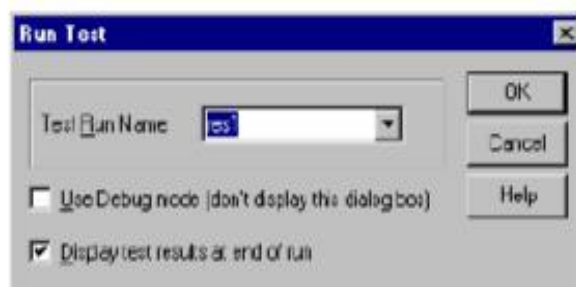
Choose Run > Run from Top or click the Run from Top button. The Run

Test dialog box opens.

6 Choose a Test Run name.

7 Run the test.

8 Review the test results.

**Win Runner – TSL Functions Introduction:**

TSL stands for “Test Scripting Language”. The test scripts are written in Test Scripting Language in win Runner. TSL is an enhanced, C-like programming language designed for testing. The advantages of TSL are:

1. It is easy to use.
2. It is similar to other programming languages. So a person who is in touch with basic concepts of programming can write test scripts easily.
3. It is a high level language.
4. The language provides various types of functions, which makes scripting easy.

Types of TSL Functions

Four basic types of Functions exist in Test Scripting Language.

- Analog functions
- Context Sensitive functions
- Standard functions
- Customization functions

All these functions can be used in Win Runner whereas they are not available in some other automation tools.

Analog Functions

Analog functions record and execute operations at specified screen coordinates. When you record in Analog mode, these functions are used to depict mouse clicks, keyboard input, and the exact coordinates traveled by the mouse. When you run a test, Analog functions retrace the mouse tracks and exactly resubmit the input you recorded. Analog functions also support different test operations such as synchronization, verification, and text manipulation.

The analog functions can again be classified bases on their operation. The various analog functions available are:

Bitmap Checkpoint Functions:

check window – Compares a bitmap of an AUT window to an expected bitmap.Input Device Functions:

Click, get, move, mtype, type are the various input device functions.

Synchronization functions:

wait window is an example of synchronization functions.

Table Functions:

Various table operations can be performed through these functions. Classic examples of table operations are clicking a table cell, double clicking a table cell, dragging a table.

Text Checkpoint Functions:

Click on text, find, get, move locator to a specified text are some of the examples of text checkpoint functions.

Context-Sensitive Functions

Context Sensitive functions depict actions on the application under test in terms of GUI objects (such as windows, lists, and buttons), ignoring the physical location of an object on the screen. In Context Sensitive mode, each time you record an operation on the application under test (AUT), a TSL statement is generated in the test script which describes the object selected and the action performed.

--

Different context-sensitive functions can be summarized as follows:

- Active Bar Functions
- ActiveX/Visual Basic Functions.
- Bitmap Checkpoint Functions
- Button Object Functions
- Calendar Functions
- Database Functions
- Data – driven test Functions
- GUI related Functions
- Java Functions
- List and Menu object Functions
- Oracle Functions
- WAP Functions
- Web Functions, etc.

Standard Functions

Standard functions include the general elements of a programming language, such as basic input and output, control-flow, mathematical, and array functions. By combining these elements with Analog and Context Sensitive functions, you can transform a simple test into an advanced testing program.

The various standard functions can be summarized as follows:

- Arithmetic Functions
- Array Functions
- Call Statements
- Compiled Module Functions
- I/O Functions
- Load Testing Functions
- Operating System Functions, etc.

Customization Functions

Customization functions allow you to enhance your testing tool so that it better supports your specific needs. For example, you can add functions to the Function Generator, or create custom GUI checkpoints.

The various customization functions are:

- Custom Record Functions
- Custom User Interface Functions
- Function Generator Functions
- GUI Checkpoint Functions

--

d) Aim: How to synchronize a test when the application responds slowly.

Objective: Student should be able

- ☐ Describes when you should synchronize a test
- ☐ Synchronize a test
- ☐ Run the test and analyze the results

When Should You Synchronize?

When you run tests, your application may not always respond to input with the same speed.

For example, it might take a few seconds:

- ☐ to retrieve information from a database
- ☐ for a window to pop up
- ☐ for a progress bar to reach 100%
- ☐ for a status message to appear

Win Runner waits a set time interval for an application to respond to input. The default wait interval is up to 10 seconds. If the application responds slowly during a test run, Win Runner's default wait time may not be sufficient, and the test run may unexpectedly fail.

If you discover a synchronization problem between the test and your application, you can either:

- ☐ Increase the default time that Win Runner waits. To do so, you change the value of the **Timeout for Checkpoints and CS Statements** option in the Run tab of the General Options dialog box (**Settings > General Options**). This method affects all your tests and slows down many other Context Sensitive operations.
- ☐ Insert a synchronization point into the test script at the exact point where the problem occurs. A synchronization point tells Win Runner to pause the test run in order to wait for a specified response in the application. This is the recommended method for synchronizing a test with your application.

In the following exercises you will:

- ✓ create a test that opens a new order in the Flight Reservation application and inserts the order into the database
- ✓ change the synchronization settings
- ✓ identify a synchronization problem
- ✓ synchronize the test
- ✓ run the synchronized test

Creating a Test

In this first exercise you will create a test that opens a new order in the Flight Reservation application and inserts the order into a database

- 1 Start Win Runner and open a new test.
- 2 Start the Flight Reservation application and log in.
- 3 Start recording in Context Sensitive mode.
- 4 Create a new order.
- 5 Fill in flight and passenger information.
- 6 Insert the order into the database.
- 7 Delete the order.

--

8 Stop recording.

9 Save the test.

Changing the Synchronization Setting

1 Open the General Options dialog box.

2 Click the Run tab.

3 Change the value to 1000 milliseconds (1 second).

4 Click OK to close the dialog box.

Identifying a Synchronization Problem

1 Make sure that the lesson4 test window is active in Win Runner.

2 Choose Run from Top.

3 Run the test.

4 Click Pause in the Win Runner message window.

Synchronizing the Test

1 Make sure that the lesson4 test window is active in Win Runner.

2 Place the cursor at the point where you want to synchronize the test.

3 Synchronize the test so that it waits for the “Insert Done” message to appear in the status bar.

4 Manually change the 1 second wait in the script to a 10 second wait.

5 Save the test

--

e) Aim: How to create a test that checks GUI objects and compare the behavior of GUI objects in different versions of the sample application.

Objective: Student should be able to

- ☐ Explain how to check the behavior of GUI objects
- ☐ Create a test that checks GUI objects
- ☐ Run the test on different versions of an application and examine the results

How Do You Check GUI Objects?

When working with an application, you can determine whether it is functioning properly according to the behavior of its GUI objects. If a GUI object does not respond to input as expected, a defect probably exists somewhere in the application's code. You check GUI objects by creating GUI checkpoints. A GUI checkpoint examines the behavior of an object's properties. For example, you can check: the content of a field whether a radio button is on or off whether a pushbutton is enabled or disabled

Adding GUI Checkpoints to a Test Script

- 1 Start Win Runner and open a new test.
- 2 Start the Flight Reservation application and log in.
- 3 Start recording in Context Sensitive mode.
- 4 Open the Open Order dialog box.
- 5 Create a GUI checkpoint for the Order No. check box.
- 6 Enter "4" as the Order No.
- 7 Create another GUI checkpoint for the Order No. check box.
- 8 Create a GUI checkpoint for the Customer Name check box.
- 9 Click OK in the Open Order dialog box to open the order.
- 10 Stop recording.
- 11 Save the test.

--

f) Aim: How to create and run a test that checks bitmaps in your application and run the test on different versions of the sample application and examine any differences, pixel by pixel.

Objective: Student should be able to

- ☐ Explains how to check bitmap images in a application
- ☐ Create a test that checks bitmaps
- ☐ Run the test in order to compare bitmaps in different versions of an application
- ☐ Analyze the results

Theory:

How Do You Check a Bitmap?

If your application contains bitmap areas, such as drawings or graphs, you can check these areas using a bitmap Check point. A bitmap checkpoint compares captured bitmap images pixel by pixel.

Adding Bitmap Checkpoints to a Test Script

- 1 Start Win Runner and open a new test.
- 2 Start the Flight Reservation application and log in.
- 3 Start recording in Context Sensitive mode.
- 4 Open orders #6.
- 5 Open the Fax Order dialog box.
- 6 Enter a 10-digit fax number in the Fax Number box.
- 7 Move the Fax Order dialog box.
- 8 Switch to Analog mode.
- 9 Sign your name in the Agent Signature box.
- 10 Switch back to Context Sensitive mode.
- 11 Insert a bitmap checkpoint that checks your signature.
- 12 Click the Clear Signature button.
- 13 Insert another bitmap checkpoint that checks the Agent Signature box.
- 14 Click the Cancel button on the Fax Order dialog box.
- 15 Stop recording.
- 16 Save the test.

g) Aim: How to Create Data-Driven Tests which supports to run a single test on several sets of data from a data table.

Objective: Student should be able to

- ☐ Use the Data Driver Wizard to create a data-driven test
- ☐ Use regular expressions for GUI object names that vary with each iteration of a test
- ☐ Run a test with several iterations and analyze the results.

Theory:

Once you have successfully debugged and run your test, you may want to see how the same test performs with multiple sets of data. To do this, you convert your test to a data-driven test and create a corresponding data table with the sets of data you want to test. Converting your test to a data-driven test involves the following steps:

- ☐ Adding statements to your script that open and close the data table.
- ☐ Adding statements and functions to your test so that it will read from the data table and run in a loop while it applies each set of data.
- ☐ Replacing fixed values in recorded statements and checkpoint statements with parameters, known as parameterizing the test. You can convert your test to a data-driven test using the Data Driver Wizard or you can modify your script manually. When you run your data-driven test, Win Runner runs the parameterized part(s) of the test one time (called an iteration) for each set of data in the data table, and then displays the results for all of the iterations in a single Test Results window.

1 Create a new test

2 Run the Data Driver Wizard.

3 Create a data table for the test.

4 Assign a table variable name.

5 Select global parameterization options.

6 Select the data to parameterize.

7 Open the data table.

8 Add data to the table.

9 Save and close the table.

10 Save the test.

11 Locate the Fax Order window in the flight1a.gui GUI map files.

12 Modify the window label with a regular expression.

13 Close the Modify dialog box.

14 Modify the **tl_step** statements.

Locate the first **tl_step** statement in your script. Delete the words “total is correct.” and replace them with, “Correct.” tickets" tickets at \$"price" cost \$"total".” **tl_step** ("total", 0, "Correct. "tickets" tickets at \$"price" cost \$"total".");

Use the same logic to modify the next **tl_step** statement to report an incorrect result.

For example:

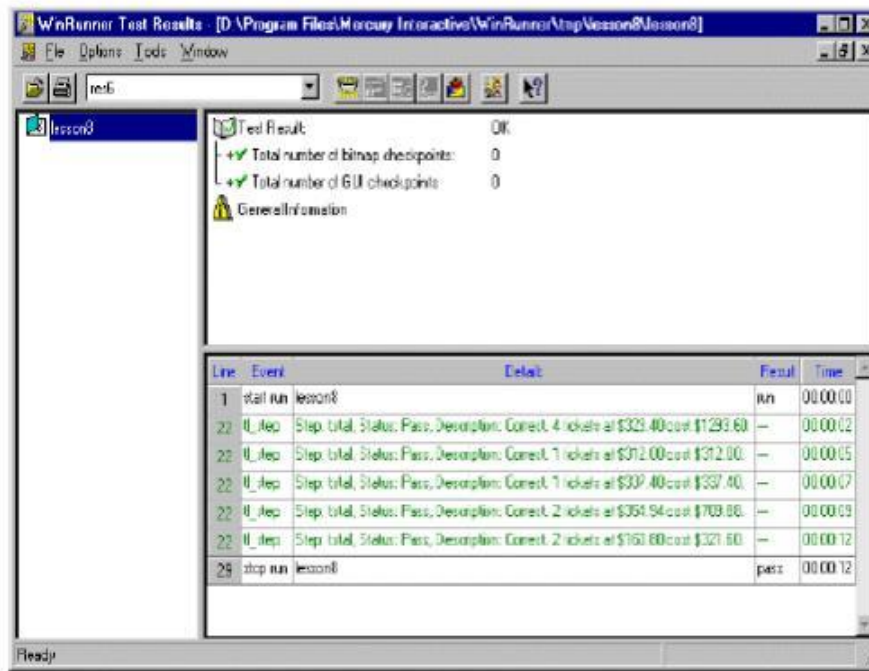
tl_step ("total", 1, "Error!"tickets" tickets at \$"price" does not equal \$"total".");

Now you will be able to see which data is used in each iteration when you view the results.

15 Save the test.

Conclusion

Regd. No.



The screenshot shows the WinRunner Test Results window for a test named 'Lesson8'. The window has a menu bar (File, Options, Tools, Window) and a toolbar. The left pane shows a tree view with 'Lesson8' selected. The right pane displays the test results, including a 'Test Result' section with 'OK' status and two passed checkpoints. Below this is a 'General Information' section. At the bottom, a table lists the test steps, their results, and execution times.

Line	Event	Detail	Result	Time
1	start run	Lesson8	run	00:00:00
22	tl_step	Step total, Status: Pass, Description: Correct, 4 tickets at \$323.40 cost \$1293.60	—	00:00:02
22	tl_step	Step total, Status: Pass, Description: Correct, 1 tickets at \$312.00 cost \$312.00	—	00:00:05
22	tl_step	Step total, Status: Pass, Description: Correct, 1 tickets at \$337.40 cost \$337.40	—	00:00:07
22	tl_step	Step total, Status: Pass, Description: Correct, 2 tickets at \$354.94 cost \$709.88	—	00:00:09
22	tl_step	Step total, Status: Pass, Description: Correct, 2 tickets at \$163.60 cost \$327.60	—	00:00:12
25	stop run	Lesson8	pass	00:00:12

--

h) Aim: How to read and check text found in GUI objects and bitmaps.

- ☐ Reading text describes how you can read text from bitmaps and non-standard GUI objects shows you how to teach Win Runner the fonts used by an application lets you create a test which reads and verifies text
- ☐ Lets you run the test and analyze the results How Do You Read Text from an Application? You can read text from any bitmap image or GUI object by adding text checkpoints to a test script. A text checkpoint reads the text from the application. You then add programming elements to the test script which verify that the text is correct. For example, you can use a text checkpoint to: verify a range of values
- ☐ calculate values Perform certain operations only if specified text is read from the screen To create a text checkpoint, you indicate the area, object, or window that contains the text you want to read. Win Runner inserts a win_get_text or obj_get_text statement into the test script and assigns the text to a variable. To verify the text you add programming elements to the script. Note that when you want to read text from a standard GUI object (such as an edit field, a list, or a menu), you should use a GUI checkpoint, which does not require programming. Use a text checkpoint only when you want to read text from a bitmap image or a non-standard GUI object.
- ☐ In the following exercises you create a test that:
 - ✓ opens a graph and reads the total number of tickets sold
 - creates a new order for the purchase of one ticket
 - opens the graph again and checks that the total number of tickets sold was dated
 - reports whether the number is correct or incorrect

Text Checkpoint Tips

- ☐ Before you create a script that reads text, determine where the text is located. If the text is part of a standard GUI object, use a GUI checkpoint or TSL functions such as edit_get_text or button_get_info. If the text is part of a non-standard GUI object, use the Create > Get Text > Object/Window command. If the text is part of a bitmap, use the Create > Get Text > Area command.
- ☐ When WinRunner reads text from the application, the text appears in the script as a comment (a comment is preceded by #). If the comment #no text was found appears in the script, WinRunner does not recognize your application font. Use the Font Expert to teach WinRunner this font.
- ☐ TSL includes additional functions that enable you to work with text such as win_find_text, obj_find_text, and compare_text.

--

i). How to create a batch test that automatically runs the tests.

Using Batch Tests

Imagine that you have revised your application and you want to run old test scripts on the revised product. Instead of running each test individually, you can use a batch test to run several tests, leave for lunch, and see the results of all your tests on your screen when you get back. A batch test looks and behaves like a regular test script, except for two main differences: It contains call statements, which open other tests. For example:

```
call "c:\\qa\\flights\\lesson9";
```

During a test run, Win Runner interprets a call statement, and then opens and runs the called test. When the called test is done, Win Runner returns to the batch test and continues the run. You choose the Run in batch mode option in the Run category of the General Options dialog box (Tools > General Options) before running the test. This option instructs Win Runner to suppress messages that would otherwise interrupt the test. For example, if Win Runner detects a bitmap mismatch, it does not prompt you to pause the test run. When you review the results of a batch test run, you can see the overall results of the batch test (pass or fail), as well as the results of each test called by the batch test.

Programming a Batch Test

In this exercise, you will create a batch test that:

► calls tests

► Runs each called test three times to check how the Flight Reservation application handles the stress of repeated execution

1 Start Win Runner, open a new test and load the GUI map. If Win Runner is not already open, choose Start > Programs > Win Runner > Win Runner. If the Welcome window is open, click the New Test button. Otherwise, choose File > New. A new test window opens. If you are working in the Global GUI Map file mode, confirm that the GUI map is loaded. To do this, choose Tools > GUI Map Editor. In the GUI Map Editor, choose View > GUI Files and confirm that flight4a.GUI is contained in the GUI File list.

2 Program call statements in the test script

In your test script, replace c:\\qa\\flights with the directory path that contains your tests.

3 Define a loop that calls each test 3 times. Add a loop around the call statements

4 Choose the Batch Run option in the General Options dialog box. Choose Tools > General Options. In the General Options dialog box, choose the Run category. Select the Run in batch mode check box and click OK to close the General Options dialog box.

5 Save the batch test.

Choose File > Save or click the Save button. Name the test batch.

--

J). Aim: How to update the GUI object descriptions which in turn supports test scripts as the application changes.

Maintaining Your Test Scripts

- ☐ Explains how the GUI map enables you to continue using your existing test scripts after the user interface changes in your application
- ☐ Shows you how to edit existing object descriptions or add new descriptions to the GUI map
- ☐ Shows you how to use the Run wizard to automatically update the GUI map

What Happens When the User Interface Changes? Consider this scenario: you have just spent several weeks creating a suite of automated tests that covers the entire functionality of your application. The application developers then build a new version with an improved user interface. They change some objects, add new objects, and remove others. How can you test this new version using your existing tests? Win Runner provides an easy solution. Instead of manually editing every test script, you can update the GUI map. The GUI map contains descriptions of the objects in your application. It is created when you use the Rapid Test Script wizard to learn the objects in your application. This information is saved in a GUI map file.

An object description in the GUI map is composed of:

- ☐ A logical name, a short intuitive name describing the object. This is the name you see in the test script. For example: button press ("Insert Order"); Insert Order is the object's logical name.
- ☐ A physical description, a list of properties that uniquely identify the object. For example: {class: push-button label: "Insert Order"} the button belongs to the push-button object class and has the label "Insert Order." When you run a test, Win Runner reads an object's logical name in the test script and refers to its physical description in the GUI map. Win Runner then uses this description to find the object in the application under test. If an object changes in an application, you must update its physical description in the GUI map so that Win Runner can continue to find it during the test run.

7 Aim: Apply Win Runner testing tool implementation in any real time applications.

Objective:

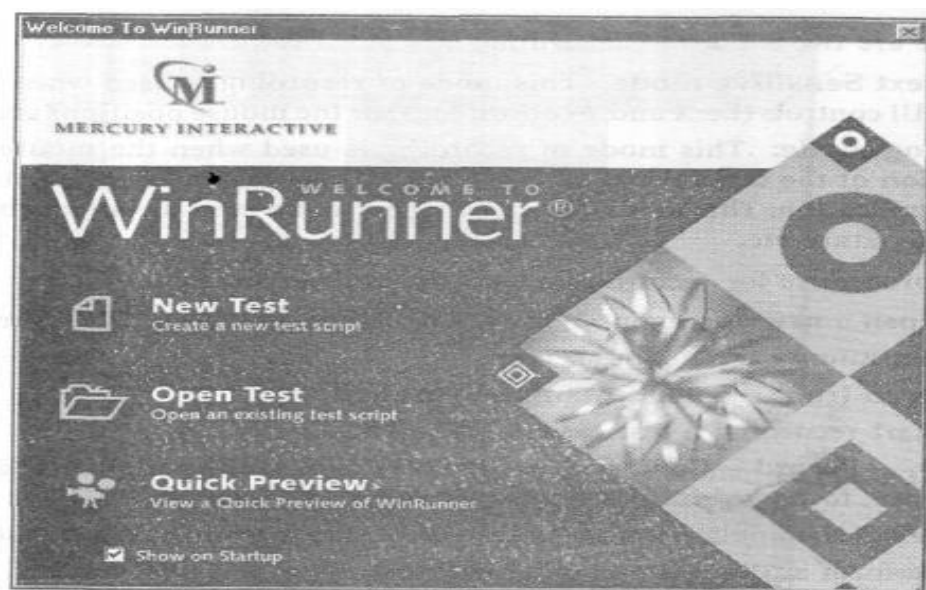
Testing an application Using Win Runner testing tool

After installing the Win Runner on your computer, invoke the Win Runner application:

- ☐ Start -> Programs -> Win Runner -> Win Runner

The opening screen of the Win Runner application is displayed, prompting you to select one of the three options:

- ☐ New Test: To create a new test script
- ☐ Open Test: To open an existing test script
- ☐ Quick Preview: To view the quick preview of Win Runner



Recording Test Cases

To test any application, first you can run the application and understand its operation. Then, you can invoke Win Runner, again run the application and record the GUI operations. During the recording mode, Win Runner will capture all your actions, which button you pressed, where you clicked the mouse etc. You need to work with the application as usual and perform all the actions to be tested. Once the recording is completed, Win Runner generates a script in TSL (Test Script Language). You can run this test script generated by Win Runner to view the results. The test results will show whether the test has passed or failed.

There are two modes of recording:

1. **Context Sensitive mode:** This mode of recording is used when the location of the GUI controls (i.e. X and Y coordinates) or the mouse positions are not necessary.
2. **Analog mode:** This mode of recording is used when the mouse positions, the location of the controls in the application, also play an important role in testing the application. This mode of recording has to be used to validate bitmaps, testing the signature etc.

The procedure for recording a test case is as follows:

Step 1: Open a new document: File -> New (or) Select "New Test" from the Win Runner's Welcome screen.

Step 2: Open (run) the application to be tested.

Step 3: Start recording a test case.

Create ->Record - Context Sensitive (or) click on the toolbar's "Record" button once, to record in Context Sensitive mode.

Step 4: Select the application to be tested by clicking on the application's title bar.

Step 5: Perform all the actions to be recorded.

Step 6: Once all required actions are recorded, stop the recording. Create -> Stop (or) Click on the toolbar's "Stop" button to stop the recording Win Runner generates the script for the recorded actions.

There are two modes for generating the test cases: "Global GUI map file mode" and "GUI map file per test mode". By default, it is in "Global GUI map file mode".

- ☐ In Global GUI map file mode, you have to explicitly save the information learnt by Win Runner. Win Runner saves it in a file with extension "gui".

When you have to run a test, you need to load the corresponding GUI map file; otherwise it will not be able to recognize the objects in the test case and displays an error message.

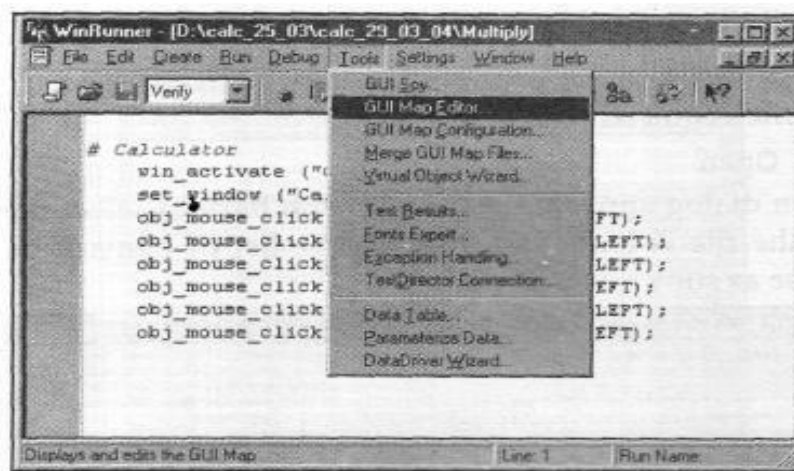
- ☐ In GUI map file per test mode, Win Runner automatically saves the information it has learnt from the application.

It is always preferred to work in Global GUI map file mode.

The procedure for saving the GUI map file in Global GUI map file mode is as follows:

Step 1: Record a test case by following the preceding procedure.

Step 2: Open the GUI Map Editor window as shown in Fig. Tools -> GUI Map Editor



Step 3: On selecting the GUI Map Editor. The screen as shown in figure is displayed



Step 4: Save the GUI Map file.

File -> Save As

A File dialog appears and you need to enter the filename.

Step 5: Close the GUI Map Editor window.

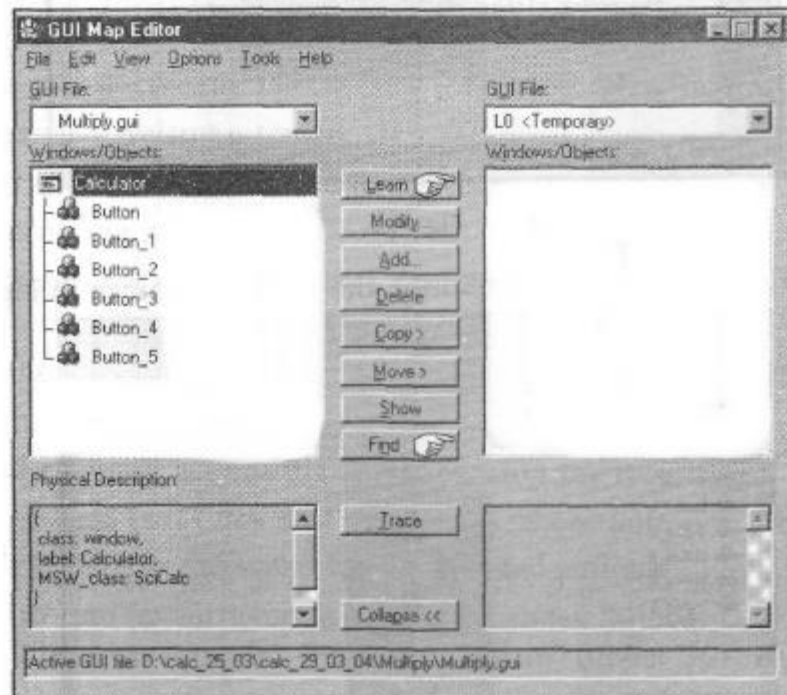
The procedure for loading the GUI map file is as follows:

Step 1: Open the GUI Map Editor.

Tools -> GUI Map Editor

Step 2: Close all the opened GUI Map files

File -> Close all.



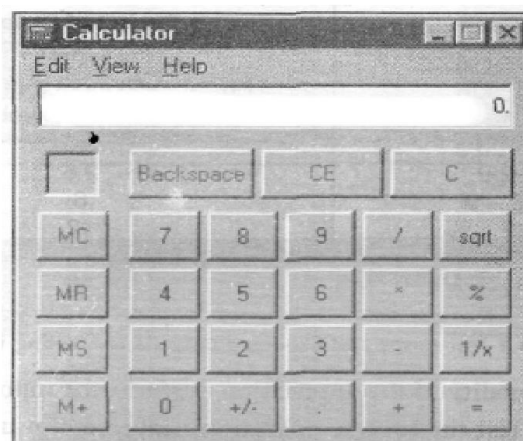
The procedure for running a test case is as follows:

Step 1: Open the test script to be executed.

Step 2: Run the test

Run -> Run from top (or) press F5 to run the test.

Win Runner executes the generated script and displays the results in the Test Results window. We will now illustrate using Win Runner to test the "Standard Calculator" application available on your Windows system. You can invoke the calculator application from the desktop Start -> Programs -> Accessories -> Calculator. The GUI of the "Calculator" application is shown in Fig.



The symbols on the buttons of Calculator application represent the following functions:

+ : To perform addition - : To perform subtraction *: To perform multiplication

/ : To perform division . : Decimal point sqrt : To find square root of a number % : To find percent

1/x : To find inverse of a number MC : To clear the memory MR : To recall from memory MS : To save in the memory M+ : To add to the memory C :To clear the current calculation CE :To clear the displayed number +/- : To give sign to a number (positive or negative) Backspace: To remove left most digit.

To test the complete functionality of the application, we need to generate test cases in such a way that all the buttons are made use of. We need to generate some test cases which will give correct output and also some test cases which will give error messages. Table gives such test cases and the expected output for each test case.

Test Cases and the Expected Output for Testing the Calculator

Test Case	Expected Output
4 1/x	0.25
- 6 sqrt	Err: "Invalid input for function"
4 C	Clears the Display
1.2 * 3	3.6
5 / 2.0	2.5
7 + 8 - 9	6
600 * 2 %	12
2, MS, C, MR	2
MC, 2, M+, 3, M+, C, MR	5

To test the functionality of the application performs the following steps:

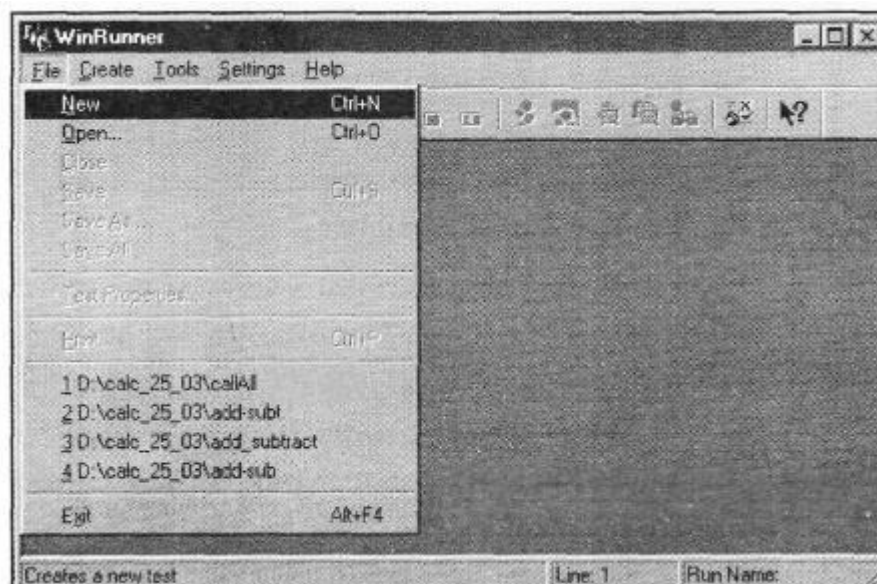
Test Case #1: To test the Inverse operation (inverse of 4 using 1/x button)

Step 1: Open Win Runner application.

Step 2: Open Calculator application.

Step 3: Create a new document as shown in Figure.

File -> New or Click Q (New) on tool bar or press Ctrl+N



Step 4: Start recording

Create -> Record-Context Sensitive (or) press F2 (or) Click # on the toolbar

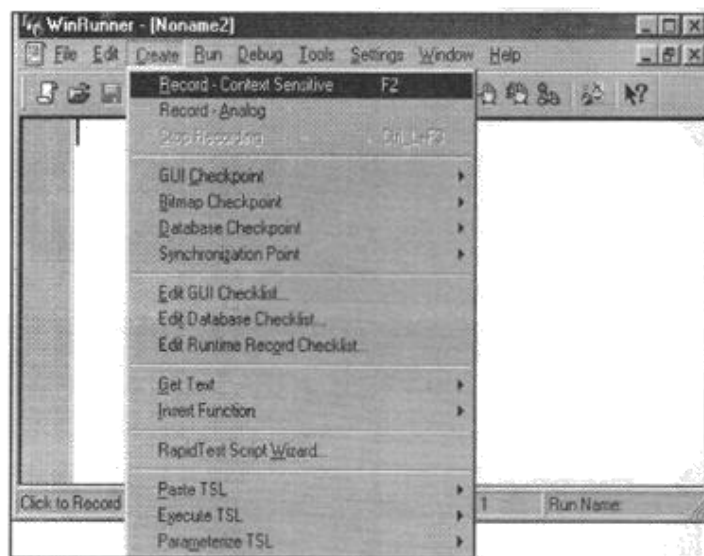
Click the (Record-Context Sensitive) button on the toolbar of Win Runner as shown in Figure or Select "Record - Context sensitive" option from the "Create" menu as shown in Figure

Step 5: Select the Calculator application and start recording the actions. a Click "4" on the Calculator

- ☐ Click the "1/x" button on the Calculator to find the inverse of 4.
- ☐ The result, 0.25 will be displayed on the Calculator.

Step 6: Stop the Recording process. Create -> Stop Recording (or) Click (Stop) on toolbar

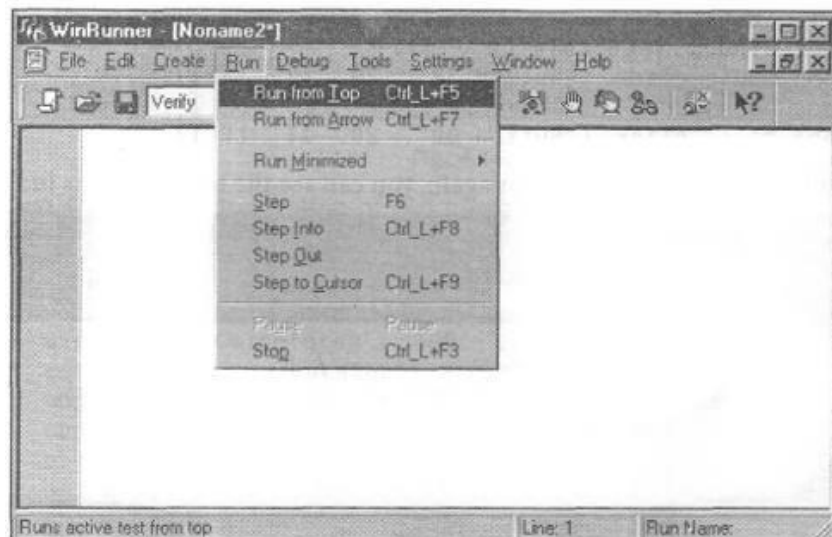
Click (Stop Recording) button on the toolbar of Win Runner as shown in Figure or Select the "Stop Recording" option from the "Create" menu as shown in Figure.



Step 8: Save the file as "inverse" in the selected folder. File -> Save

In the "Save" dialog box that appears, save the test script with name "inverse".

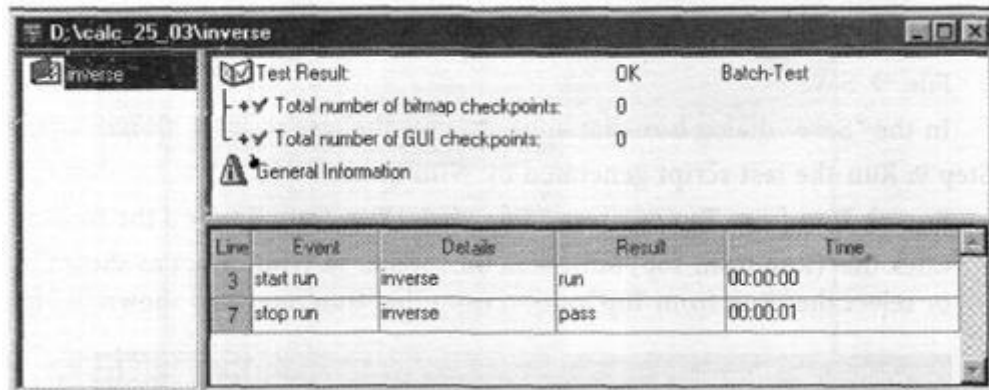
Step 9: Run the test script generated by Win Runner. Run -> Run from Top or press F5



or Click (Run from Top) on the toolbar

Click the (Run from Top) button on the toolbar of Win Runner as shown in Figure or select the "Run from Top" option from the "Run" menu as shown in Figure.

Step 10: After executing the TSL statements, Win Runner generates test results as shown in Figure. The Results column indicates whether the test has "Passed" or "Failed". The test results also give useful information such as the name of the test case, the line numbers in the test script and the time taken for executing the test case.

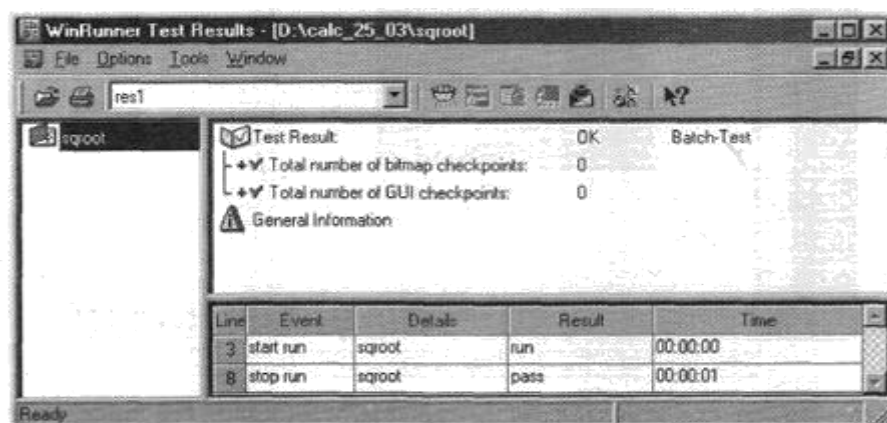


You can use the same procedure explained above for recording the test case. The following test script will be generated:

```
#Calculator winactivate("Calculator"); set_window("Calculator",1);
```

```
obj_mouse_click("Button_38",20,12,LEFT); objmousedrag("Button_35",10,15,11,14,LEFT);  
obj_mouse_click("Button_60",20,11,LEFT);
```

When you run the test script again, you can see the test results, as in Figure.



Calling the Test Cases using "call" Function

The "call" function can be reused to execute a series of test cases without any user interaction. The syntax of call function is:

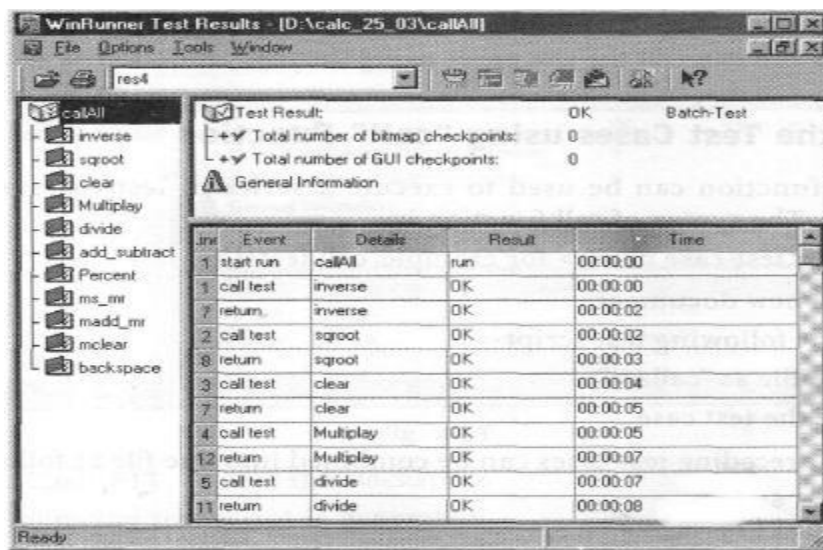
call<test-case name> for example, call test1();

- ☐ Create a new document
- ☐ Write the following test script
- ☐ Save the file as "callAll"
- ☐ Execute the test case

All the preceding test cases can be combined into one file as follows:

```
call inverseO; call sqrootO; call clearO; call MultiplayO; call divideO; call add_subtract() call
PercentO; call msjnrO; call maddmrO; call mclearO; call backspaceO;
```

When you execute this test script, all the earlier test cases are executed in one shot. The test results screen will be as shown in Figure. As you can see from the table, the "Details" column gives the various test cases executed. The "Result" column shows whether the test has passed or failed. The "Time" column gives the time taken to execute the test case.



The screenshot shows the WinRunner Test Results window for a file named 'callAll'. The window displays a list of test cases on the left and a table of results on the right. The table has columns for 'Line', 'Event', 'Details', 'Result', and 'Time'.

Line	Event	Details	Result	Time
1	start run	callAll	run	00:00:00
1	call test	inverse	OK	00:00:00
7	return	inverse	OK	00:00:02
2	call test	sqroot	OK	00:00:02
8	return	sqroot	OK	00:00:03
3	call test	clear	OK	00:00:04
7	return	clear	OK	00:00:05
4	call test	Multiplay	OK	00:00:05
12	return	Multiplay	OK	00:00:07
5	call test	divide	OK	00:00:07
11	return	divide	OK	00:00:08

When you have to retest the application using the same test cases, you can run the script in unattended mode. You can save the script in a file and run the script at specified time.

This feature of Win Runner is extremely useful for regression testing. When you are developing the software, you need to run the same set of test cases many times. So, you can run the application once, generate the test script and then keep doing the regression testing. Obviously, the productivity of the test engineers will be very high when this tool is used.

8 Aim: Study of any web testing tool (e.g. Selenium)

Selenium is a robust set of tools that supports rapid development of test automation for web-based applications. Selenium provides a rich set of testing functions specifically geared to the needs of testing of a web application. These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior.

One of Selenium's key features is the support for executing one's tests on multiple browser platforms.

Selenium Components

Selenium is composed of three major tools. Each one has a specific role in aiding the development of web application test automation.

Selenium-RC provides an API (Application Programming Interface) and library for each of its supported languages: HTML, Java, C#, Perl, PHP, Python, and Ruby. This ability to use Selenium-RC with a high-level programming language to develop test cases also allows the automated testing to be integrated with a project's automated build environment.

Selenium-Grid

Selenium –Grid allows the Selenium –RC solution to scale for large test suites or test suites that must be run in multiple environments. With Selenium-Grid, multiple instances of Selenium-RC are running on various operating system and browser configurations; Each of these when launching register with a hub. When tests are sent to the hub they are then redirected to an available Selenium-RC, which will launch the browser and run the test. This allows for running tests in parallel, with the entire test suite theoretically taking only as long to run as the longest individual test.

*Tests developed on Firefox via Selenium-IDE can be executed on any other supported browser via a simple Selenium-RC command line.

**Selenium-RC server can start any executable, but depending on browser security settings there may be technical limitations that would limit certain features.

Flexibility and Extensibility

Selenium is highly flexible. There are multiple ways in which one can add functionality to Selenium's frame work to customize test automation for one's specific testing needs. This is, perhaps, Selenium's strongest characteristic when compared with proprietary test automation tools and other open source solutions. Selenium-RC support for multiple programming and scripting languages allows the test writer to build any logic they need into their automated testing and to use a preferred programming or scripting language of one's choice.

Selenium- IDE allows for the addition of user –defined — user –extensions for creating additional commands customized to the user's needs. Also, it is possible to re-configure how the Selenium- IDE generates its Selenium-RC code. This allows users to customize the generated code to fit in with their own test frame works. Finally, Selenium is an OpenSource project where code can be modified and enhancements can be submitted for contribution.

Test Suites

A test suite is a collection of tests. Often one will run all the tests in a test suite as one continuous batch - job.

When using Selenium-IDE, test suites also can be defined using a simple HTML file. The syntax again is simple. An HTML table defines a list of tests where each row defines the file system path to each test. An example tells it all.

```
<html>
<head>
<title>Test Suite Function Tests–Priority1 </title>
</head> <body>
<table>
<tr> <td> <b>Suite Of Tests</b> </td> </tr>
<tr> <td> <a href=“./Login.html”>Login</a> </td> </tr>
<tr> <td> <a href=“./Search Values.html”>Test Searching for
Values </a></td></tr><tr><td><a href=“./Save Values.html”>Test
Save </a> </td> </tr>
</table></body>
</html>
```

A file similar to this would allow running the tests all at once, one after another, from the Selenium-IDE.

Test suites can also be maintained when using Selenium-RC. This is done via programming and can be done a number of ways. Commonly Jun it is used to maintain a test suite if one is using Selenium-RC with Java. Additionally, if C# is the chosen language, Nun it could be employed. If using an interpreted language like Python with Selenium-RC than some simple programming would be involved in setting up a test suite. Since the whole reason for using Sel-RC is to make use of programming logic for your testing this usually is not a problem. Few typical Selenium commands.

Open –opens a page using a URL.

Click /click And Wait–performs a click operation, and optionally waits for a new page to load.

Verify Title/assert Title–verifies an expected page title.

Verify Text Present–verifies expected text is somewhere on the page.

Verify Element Present–verifies an expected UI element, as defined by its HTML tag, is present on the page.

Verify Text–verifies expected text and it’s corresponding HTML tag are present on the page.

Verify Table–verifies a table’s expected contents.

Wait For Page To Load–pauses execution until an expected new page loads called automatically when click And Wait is used.

Wait For Element Present–pauses execution until an expected UI element, as defined by its HTML tag, is present on the page.

9 Aim: Study of Any Bug Tracking Tool (Bugzilla)

Bugzilla is a—Bug Tracking System that can efficiently keep track of outstanding bugs in a product. Multiple users can access his database and query, add and manage these bugs. Bugzilla essentially comes to the rescue of a group of people working together on a product as it enables them to view current bugs and make contributions to resolve issues. Its basic repository nature works out better than the mailing list concept and an organized database is always easier to work with.

Advantage of Using Bugzilla:

1. Bugzilla is very adaptable to various situations. Known uses currently include IT support queues, Systems Administration deployment management, chip design and development problem tracking (both pre-and-post fabrication), and software and hardware bug tracking for luminaries such as Red hat, NASA, Linux-Mandrake ,and VA Systems. Combined with systems such as CVS, Bugzilla provides a powerful, easy-to-use solution to configuration management and replication problems.
2. Bugzilla can dramatically increase the productivity and accountability of individual employees by providing a documented workflow and positive feedback for good performance .Ultimately ,Bugzilla puts the power in user's hands to improve value to business while providing a usable frame work for natural attention to detail and knowledge store to flourish.

The bugzilla utility basically allows to do the following:

- Add a bug into the database
- Review existing bug reports
- Manage the content

Bugzilla is organized in the form of bug reports that give all the information needed about a particular bug. A bug report would consist of the following fields.

- Product→ Component
- Assigned to
- Status (New, Assigned, Fixed etc)
- Summary
- Bug priority
- Bug severity (blocker, trivial etc)
- Bug reporter

Using Bugzilla:

- Bugzilla usage involves the following activities
- Setting Parameters and Default Preferences

Creating a New User

- Impersonating a User
- Adding Products
- Adding Product Components
- Modifying Default Field Values
- Creating a New Bug
- Viewing Bug Reports

Setting Parameters and Default Preferences:

When we start using Bugzilla, we'll need to set a small number of parameters and preferences. At a minimum, we should change the following items, to suit our particular need:

- Set the maintainer
- Set the mail_delivery_method
- Set bug change policies
- Set the display order of bug reports

To set parameters and default preferences:

1. Click Parameters at the bottom of the page.
2. Under Required Settings, add an email address in the maintainer field.
3. Click Save Changes.
4. In the left side Index list, click Email.
5. Select from the list of mail transports to match the transport we're using. If evaluating a click 2 try application, select Test. If using SMTP, set any of the other SMTP options for your environment. Click Save Changes.
6. In the left side Index list, click Bug Change Policies.
7. Select On for comment on create, which will force anyone who enters a new bug to enter a comment, to describe the bug. Click Save Changes.
8. Click Default Preferences at the bottom of the page.
9. Select the display order from the drop-down list next to the When viewing a bug, show comments in this order field. Click Submit Changes.

Creating a New User

Before entering bugs, make sure we add some new users. We can enter users very easily, with a minimum of information. Bugzilla uses the email address as the user ID, because users are frequently notified when a bug is entered, either because they entered the bug, because the bug is assigned to them, or because they've chosen to track bugs in a certain project.

To create a new user:

1. Click **Users**.
2. Click **add** a new user.
3. Enter the **Login name**, in the form of an email address.
4. Enter the **Real name**, a password, and then click **Add**.

5. Select the **Group access options**. We'll probably want to enable the following options in the row titled User is a member of these groups: Can confirm edit bugs edit components
6. Click **Update** when done with setting options. **Impersonating a User**

Impersonating a user is possible, though rare, that we may need to file or manage a bug in an area that is the responsibility of another user when that user is not available. Perhaps the user is on vacation, or is temporarily assigned to another project. We can impersonate the user to create or manage bugs that belong to that user.

Adding Products

We'll add a product in Bugzilla for every product we are developing. To start with, when we first login to Bugzilla, we'll find a test product called **Test Product**. We should delete this and create a new product.

To add a product:

1. At the bottom of the page, click **Products**.
2. In the **Test Product** listing, click **Delete**.
3. Click **Yes, Delete**.
4. Now click **Add a product**.
5. Enter a product name, such as— Widget Design Kit.
6. Enter a description
7. Click **Add**. A message appears that you'll need to add at least one component.

Adding Product Components

Products are comprised of components. Software products, in particular, are typically made up of many functional components, which in turn are made up of program elements, like classes and functions. It's not unusual in a software development team environment for different individuals to be responsible for the bugs that are reported against a given component. Even if there are other programmers working on that component, it's not uncommon for one person, either a project lead or manager, to be the gate keeper for bugs. Often, they will review the bugs as they are reported, in order to redirect them to the appropriate developer or even another team, to review the priority and severity supplied by the reporter, and sometimes to reject bugs as duplicates or enhancement requests, for example.

To add a component:

1. Click the link **add at least one component** in the message that appears after creating a new product.
2. Enter the **Component** name.
3. Enter a **Description**.
4. Enter a **default assignee**. Use one of the users we've created. Remember to enter the assignee in the form of an email address.
5. Click **Add**.
6. To add more components, click the name of product in the message that reads edit other components of product <**product name**>.

--

Modifying Default Field Values

Once we begin to enter new bugs, we'll see a number of drop-down lists containing default values. Some of these may work just fine for our product. Others may not. We can modify the values of these fields, adding new values and deleting old ones. Let's take a look at the OS category.

To modify default field values:

1. At the bottom of the page, in the **Edit** section, click **Field Values**.
2. Click the link, in this case **OS**, for the field we want to edit. The OS field contains a list of operating system names. We are going to add browsers to this list. In reality, we might create a custom field instead, but for the sake of this example, just add them to the OS list.
3. Click **Add a value**. In the **Value** field, enter—IE7. Click **Add**.
4. Click **Add a value** again.
5. In the **Value** field, enter—Fire fox3.
6. Click **Add**.
7. Where it reads **Add other values for the op_sys field**, click **op_sys**.
8. This redisplay the table. We should now see the two new entries at the top of the table. These values will also appear in the OS drop-down list when we create a new bug.

Creating a New Bug

Creating bugs is a big part of what Bugzilla does best.

To create a new bug:

1. In the top menu, click **New**.
2. If we've defined more than one component, choose the component from the component list.
3. Select a **Severity** and a **Priority**. **Severity** is self-explanatory, but **Priority** is generally assumed to be the lower the number, the higher the priority. So, a **P1** is the highest priority bug, a shows topper.
4. Click the **OS** drop-down list to see the options, including the new browser name we entered.
5. Select one of the options.
6. Enter a summary and a description. We can add any other information of choice, but it is not required by the system, although we may determine that our bug reporting policy requires certain information.
7. Click **Commit**. Bugzilla adds our bug report to the database and displays the detail page for that bug.

Viewing Bug Reports

Eventually, we'll end up with thousands of bugs listed in the system. There are several ways to view the bugs. The easiest is to click the My Bugs link at the bottom of the page. Because we've only got one bug reported, we'll use the standard Search function.

To find a bug:

1. Click **Reports**.

--

2. Click the **Search** link on the page, not the one in the top menu. This opens a page titled—Find a Specific Bug.¶
3. Select the **Status**.
4. Select the **Product**.
5. Enter a word that might be in the title of the bug.
6. Click **Search**. If any bugs meet the criteria that we have entered, Bugzilla displays them in a list summary.
7. Click the **ID** number link to view the full bug report.

Modifying Bug Reports

Suppose we want to change the status of the bug. We've review edit and have determined that it belongs to one of the users we have created earlier

To modify a bug report:

1. Scroll down the full bug description and enter a comment in the **Additional Comments** field.
2. Select—Reassign bug toll and replace the default user ID with one of the other user IDs you created. It must be in the format of an email address.