Draft: Work in Progress

The General Mission Analysis Tool (GMAT) System Test Plan

Darrel J. Conway
Thinking Systems, Inc.

Steven P. Hughes
Goddard Space Flight Center

July 12, 2007

Draft: Work in Progress

Contents

List of Figures

Part I

Overview

Draft: Work in Progress

Chapter 1

Introduction

1.1 Overview

The General Mission Analysis Tool (GMAT) is a spacecraft mission analysis tool tailored to support missions involving groups of spacecraft interacting throughout a modeled time period. The potential complexity of this problem makes GMAT an intricate software system. This complexity necessitates a rigorous testing environment to ensure that the system meets its objectives.

GMAT is designed using an object-oriented architecture[GDT] and coded using extensive object-oriented structures written in C++. The object based approach employed in GMAT's design and implementation makes the system robust and relatively easy to use for experienced analysts. The extent of the object model implemented to make GMAT a complete and robust system dictates a comprehensive testing philosophy, described in the GMAT Master Test Plan[MTP]. This document describes one component of the overall testing strategy, the system testing.

System testing is a black box form of testing, designed to exercise the GMAT system from the user's perspective. The system tests are designed to exercise all of the user accessible objects in GMAT.

1.2 Purpose of this Document

This document serves as the System Test Approach for the GMAT Project. Preparation for system testing consists of three major stages:

- The Test Approach sets the scope of system testing, the overall strategy to be adopted, the activities to be completed, the general resources required and the methods and processes to be used to test the release. It also details the activities, dependencies and effort required to conduct the System Test.
- Test Planning details the activities, dependencies and effort required to conduct the System Test.
- Test Cases documents the tests to be applied, the data to be processed, the automated testing coverage and the expected results.

This document covers the first two of these items, and established the framework used for the GMAT test case development. The test cases themselves exist as separate components, and are managed outside of and concurrently with this System Test Plan.

1.3 Overview of the GMAT Development and Testing Process

The GMAT development process identifies several review points for the system. GMAT development is conducted as a cooperative effort between an analysis team, typically composed of flight dynamics specialists,

and a development team consisting of talented software developers. New requirements for the system are defined and written by the analysis team. Mathematical and design specifications are derived from these requirements and compiled into a format that can be used to code the new functionality. Requirements, Specifications, and Designs are reviewed by the development team prior to implementation. This review is typically conducted in an informal, iterative manner until the specifications are understood by all involved parties. The specifications and design documentation are then used to write the software.

During the development process, new features of a component under development may be detected that need further specification. When that happens, the new features are discussed and collected together. This may result in an immediate update to the design documents, or it may result in collection of the new feature implementation for inclusion in a final update performed when the component is ready for integration. In either case, the design documentation is updated to reflect the implemented functionality prior to formal acceptance of the related components.

During development, the software undergoes internal testing in the development team at both a unit and an integration level. Unit testing is intended to exercise all of the executable paths through the code, validating that the internal working of the code behaves correctly. Integration testing takes unit tested components and builds those components, either one at a time or collectively, into the system. From time to time, the development team will interact with the analysis team during integration testing to confirm that the observed behavior of the new code conforms to the expectations of the users. Unit testing and integration testing are performed in the course of the development of the software; neither will necessarily provide test results in a formal manner, though informal communications of the component and integrated test results are strongly encouraged.

When the GMAT development team completes integration of new functionality into the system, that new functionality is ready for system test. GMAT system testing follows a more formal test procedure than unit or integration testing. New components are exercised both from the GMAT scripting language and from the GMAT Graphical User Interface (GUI). The test cases exercised are documented using the procedures described later in this document. Test cases are managed using a traceability matrix that lists all of the elements of GMAT visible at the user level, and matches those elements to test cases that are executed in system testing. This master traceability matrix is used to generate a spreadsheet of test cases each time GMAT enters a system test cycle. All tests are tracked using this spreadsheet; formal system test is complete when every test case has been exercised and the results of the tests have been tabulated and accepted after review.

1.4 System Test Objectives

At a high level, System Test intends to prove that

- The functionality, delivered by the development team, is as specified by the Mathematical and Design Specifications[1].
- The software is stable and of high quality.
- The software models spacecraft missions faithfully.
- The software interfaces correctly with other systems, specifically MATLAB.
- The software user interfaces are stable, complete, and understandable by novice and experienced users.

These objectives are addressed through the development of a suite of test cases exercised on builds of the GMAT system. Each major release of GMAT is tested using this suite, and the results of the tests are collected and reviewd by all interested parties prior to release. This document describes the procedures followed for system testing.

[1] System test does not provide a formal mechanism for mapping the system requirements to the implemented functionality; that is the responsibility of Acceptance testing. The system test validates that the implemented functionality is correct.

1.5 Formal System Testing

While system tests can be performed as soon as new features are available, there is not a requirement that they must be performed at that time. However, system tests shall be performed prior to each major release of GMAT to the aerospace community. Part of the GMAT release process includes a review of the system test matrices and results to ensure that the system has maintained its integrity for the release. The review performed at each major release:

- Checks the System Test matrices to ensure full system coverage for User Elements, Parameters, Commands, and GUI Widgets.
- Ensures that the system tests have been run for all test cases.
- Ensures that the data produced from GMAT is consistent with known "truth" data.
- Ensures that system tests that failed have documented the cause or causes of the failure
- Ensures that any failures that must be addressed for the release have (1) been addressed and (2) that the resulting correction has been validated to meet the expected results.
- Ensures that all scripting elements of GMAT have been exercised, and function correctly.
- Ensures that all GUI elements of GMAT have been exercised, and function correctly.
- Ensures that the system is stable. Stability in this context means that GMAT
  - Does not crash
  - Produces identical results on rerun
  - Produces comparable results on all supported platforms
  - Allocates and releases memory consistently, without long term memory artifacts (aka "memory leaks")
  - Produces identical results when configured from the GUI, from a script file, and when saved to file and reloaded, both into the running instance and into a new image.
- Ensures that GMAT performs efficiently, both when executing mission sequences, and when saving and loading missions.

System test review is performed by members of the analysis and development teams. Detailed testing of the system numerics and scripting is performed by the domain experts on the analysis team. GUI testing is performed by the development team.

While the formal test responsibilities are as described in the previous paragraph, both teams are encouraged to exercise the features being tested by the other team to help identify any additional issues that exist. For example, the analysis team is encouraged to create all test cases using the GMAT GUI, and to report any difficulties encountered when following this approach. Similarly, the development team is encouraged to test the GUI in such a way as to produce functional models, to run those models, and to report any resulting anomalous behavior. This cross checking of functionality ensures that the system has been exercised as much as possible, given the resources available for development of GMAT.

1.6 Items Not Addressed in System Tests

The system tests described in this document are used to validate the stability and accessibility of GMAT components to users attempting to use the system to solve flight dynamics problems. These tests do not address several key system elements. Those elements are covered by other components of the GMAT test suite.

Specifically, the tests defined in this document do not address these items:

- Internal data representations and data flow in the GMAT code. These elements are tested in the GMAT unit and integration test processes.
- Numerical fidelity of the models. The detailed numerical testing of the components are part of the GMAT acceptance tests.
- Data range validation. The data range tests are performed as part of the integration tests.
- Requirements Validation. The mapping of GMAT capabilities to the system requirements is made and validated in the GMAT acceptance tests.

1.7 Document Layout

The remainder of this document describes the procedures followed to prepare for, conduct, and document the GMAT system tests. Chapter 2 describes the procedured followed when preparing for the system tests. Chapters 3 and 4 document the procedures followed when running the test cases. Chapter 5 describes the data collection and review procedures followed for the system. The Appendices at the end of the document provide additional information that may be useful during system test.

Part II

System Test Procedures

Draft: Work in Progress

Chapter 2

System Test Preparation

The GMAT system tests are designed to perform a "black box" examination of GMAT as an assembled system. The system tests exercise all of the elements of the system from both the scripting and graphical user interface perspectives. Traceability matrices are maintained to ensure that the entire system is exercised upon completion of the system tests. This chapter describes these matrices, and provides instructions about how to maintain and extend them.

2.1 Test Process

System testing is performed in three stages: test preparation, system testing (consisting of Script based Testing and GUI Testing), and test result reporting. The test preparation phase, described in this chapter, is used to update the system test cases with tests covering new capabilities of GMAT, and to add or update existing test cases based on lessons learned from previous testing. Procedures followed when executing the script based are described in Chapter 3. GUI testing procedures are given in Chapter 4. Both of those chapters include descriptions of the data collection for individual tests. Chapter 5 describes the process of accumulating the test results so that the status of the system can be evaluated.

2.2 Test Preparation

GMAT system testing is managed from a set of OpenOffice[OOo] spreadsheets. The test case structure and mapping between system functionality and corresponding tests is tracked using the "SystemTestMatrix.ods" spreadsheet[1]. This spreadsheet contains pages identifying detailed GMAT functionality and defined system test cases, and maps each element of functionality to one or more test cases.

The spreadsheet includes a summary page, shown in Figure 2.1, which computes coverage for the elements tabulated on the detail pages. If the tables in the spreadsheet are up to date, then the summary page is an indicator of the readiness of the system tests. Hence the first task that testers perform when preparing for system testing is to update the test matrices. Once the test matrices have been updated, the test cases are updated to cover any new functionality in the system. Test preparation is finished when a complete set of test cases has been developed, covering all of the elements in the updated test matrix tables.

To summarize, when a new piece of functionality is added to GMAT that users can access, the test team, working with the developers and users, updates the test matrices by performing three steps:

1. Identify and add all new elements of the system to the test matrices.

[1] All of the GMAT test tracking components are configuration controlled. Interested parties can obtain the current versions of these testing artifacts by contacting one of the GMAT team leads.

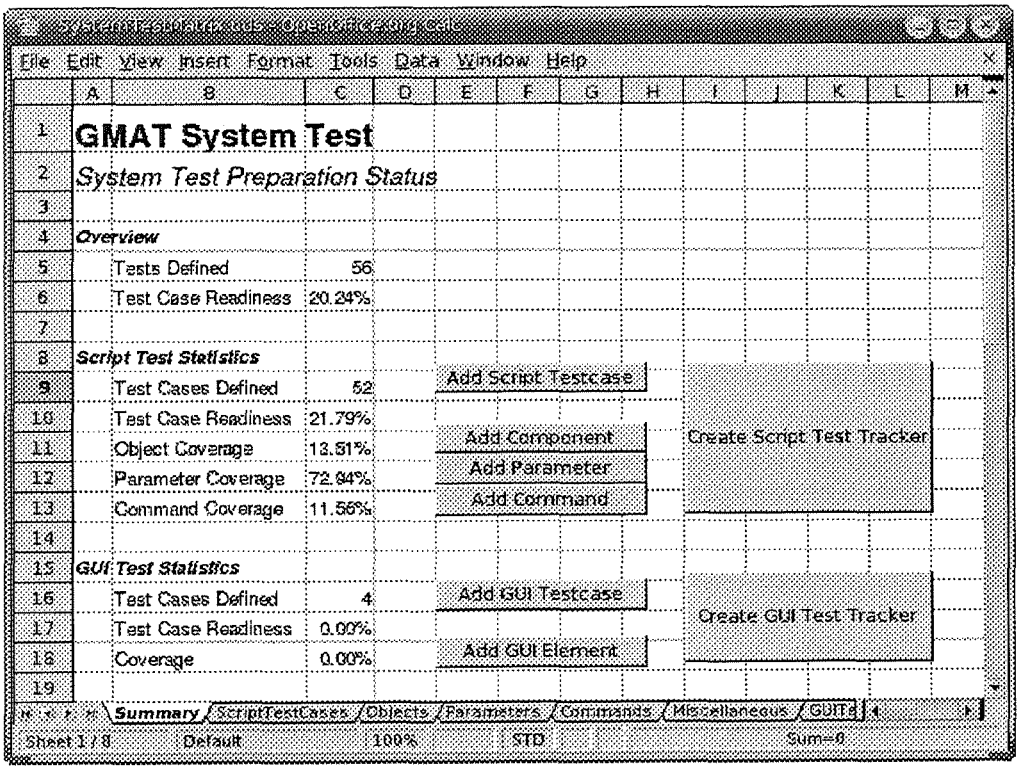


Figure 2.1: The System Test Summary Page

2. Identify test cases that cover the new elements. This may involve modifying existing test cases or creating new test cases, depending on the functionality of the new element.

3. Create or update the test cases as needed to implement the planned coverage identified in item 2.

When these steps have been performed, the coverage matrices are up to date, and the test team is ready to run the system test by executing all of the test cases in the matrices. The following paragraphs describe the procedure for executing these steps.

2.3 Updating the Element Lists in the Test Matrices

Figure 2.2 shows an example of the matrices used to identify GMAT's implemented functionality. Separate tables exist for the user accessible Components (Spacecraft, Solvers, Propagators, and so forth), Parameters that GMAT can calculate, Commands used when defining the mission sequence, Graphical User Interface elements (GuiElements), and miscellaneous other configurable elements. These tables capture a static view of every item that a user can interact with when running GMAT.

Each table lists the configurable elements in column A, and constructs, when appropriate, configurations and subconfigurations of those objects in columns B (labeled "Cases") and C ("Subcases"). Column D, "Notes", is used to indicate other considerations. Elements that are not yet scheduled for testing can be entered in the tables; when that happens, the entry in the "Notes" column should be set to the keyword "DEFERRED".

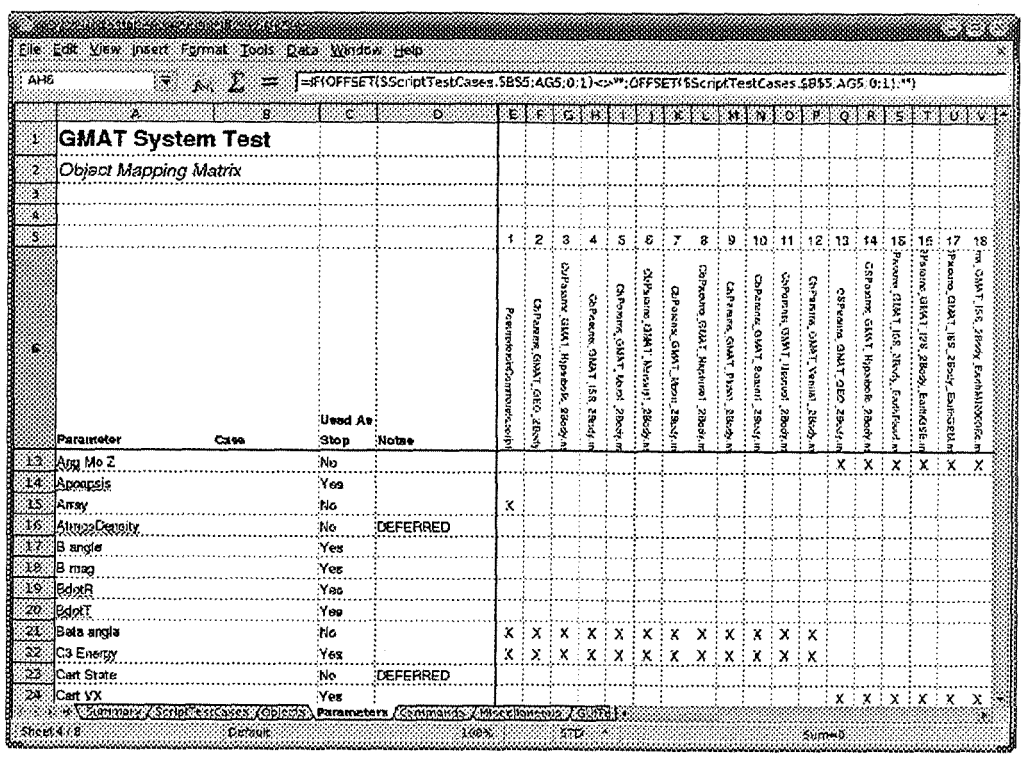


Figure 2.2: An Object Test Matrix

The first step in updating the test matrices is to ensure that the lists of accessible elements are complete, capturing any new elements and configurations added to the system since the last time the matrix was updated. Testers have two options for performing these updates: they can either edit the tables by hand, and check that all related formatting and equations are updated correctly, or they can use the macros built into the spreadsheet to add the new elements. The preferred approach is to use the macros, because that approach ensures that the calculations performed by the tables are correct.

The summary page, shown in Figure 2.2, for the spreadsheet contains four buttons used to add elements to the test matrices: "Add Resource", "Add Parameter", "Add Command", and "Add GUI Element". When a user presses one of these buttons, a dialog box opens that is used to set some basic information for the new element that is being tested. Figure 2.3 shows an example of this dialog.

When this dialog is opened, users can change the type of new element being configured using the Element Type combo box. This option is provided in case the user selected the wrong button from the summary page. The user enters the name of the new element in the ElementName field.

Many of the elements that are tested can be exercised more than one way; for example, the Impulsive Burn element can be set to run using Velocity-Normal-Binormal (VNB) delta-V vectors or a coordinate system based delta-V vector. Each of these modes should be tested independently, so a separate line should exist for each on the spreadsheet. The user reserves multiple lines on the spreadsheet by entering the number of lines required in the "Spreadsheet Lines Needed" field.

After setting the data correctly on the new element dialog, the user presses the 'OK" button. When this action is taken, the test matrix corresponding to the type of the new element is updated. New rows are inserted into the spreadsheet for the new element, and the formulas for the new rows are set. Finally,

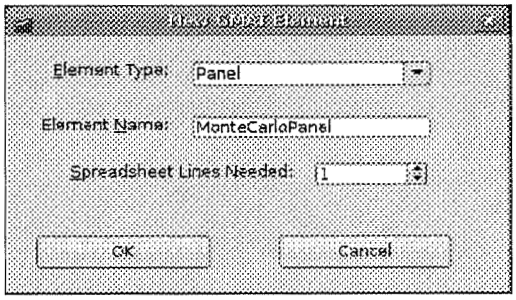
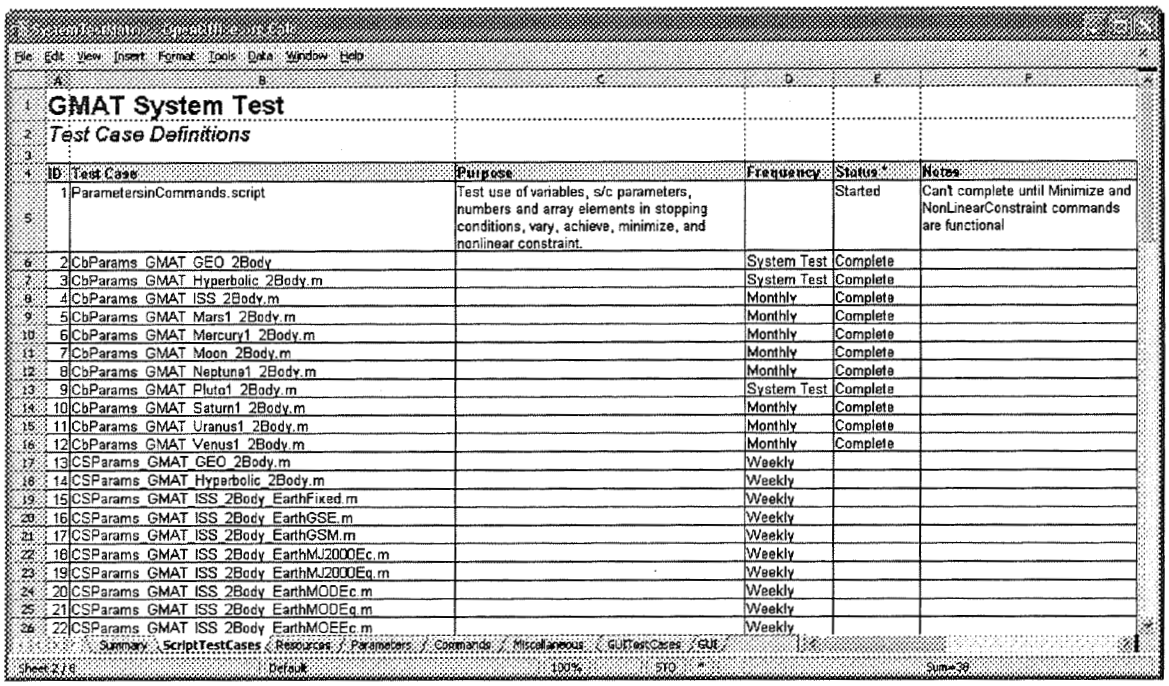


Figure 2.3: The New Element Dialog

the fields that are used to calculate the test preparation statistics are updated. If more than one row was inserted, the spreadsheet page is set to the page containing the new element, with the active cell selected to the "Cases" field for the new element, so that the user can enter the test cases required for the new element. Each test case and subcase should be entered at this time so that the element descriptions in the test matrix reflect the capabilities that need to be tested.

At this point, all of the functionality in GMAT should be represented by rows in the test matrices. The next step is to plan test cases that cover elements of the system that are not already handled in the test suite.



GMAT System Test

Test Case Definitions

ID	Test Case	Purpose	Frequency	Status *	Notes
1	ParametersinCommands.script	Test use of variables, s/c parameters, numbers and array elements in stopping conditions, vary, achieve, minimize, and nonlinear constraint.		Started	Can't complete until Minimize and NonLinearConstraint commands are functional
2	CbParams_GMAT_GEO_2Body		System Test	Complete	
3	CbParams_GMAT_Hyperbolic_2Body.m		System Test	Complete	
4	CbParams_GMAT_ISS_2Body.m		Monthly	Complete	
5	CbParams_GMAT_Mars1_2Body.m		Monthly	Complete	
6	CbParams_GMAT_Mercury1_2Body.m		Monthly	Complete	
7	CbParams_GMAT_Moon_2Body.m		Monthly	Complete	
8	CbParams_GMAT_Neptune1_2Body.m		Monthly	Complete	
9	CbParams_GMAT_Pluto1_2Body.m		System Test	Complete	
10	CbParams_GMAT_Saturn1_2Body.m		Monthly	Complete	
11	CbParams_GMAT_Uranus1_2Body.m		Monthly	Complete	
12	CbParams_GMAT_Venus1_2Body.m		Monthly	Complete	
13	CSParams_GMAT_GEO_2Body.m		Weekly		
14	CSParams_GMAT_Hyperbolic_2Body.m		Weekly		
15	CSParams_GMAT_ISS_2Body_EarthFixed.m		Weekly		
16	CSParams_GMAT_ISS_2Body_EarthGSE.m		Weekly		
17	CSParams_GMAT_ISS_2Body_EarthGSM.m		Weekly		
18	CSParams_GMAT_ISS_2Body_EarthMJ2000Ec.m		Weekly		
19	CSParams_GMAT_ISS_2Body_EarthMJ2000Eq.m		Weekly		
20	CSParams_GMAT_ISS_2Body_EarthMODEc.m		Weekly		
21	CSParams_GMAT_ISS_2Body_EarthMODEq.m		Weekly		
22	CSParams_GMAT_ISS_2Body_EarthMOEEc.m		Weekly		

Figure 2.4: A Test Case List

2.4 Updating the Test Case Lists

There are two categories of test cases used in system testing GMAT, designed to exercise the system using scripting and the graphical user interface. When new components are added to GMAT, the test coverage matrix is updated to exercise those new elements using the procedure described above. This update produces holes in the system test suite, requiring either an update of the current test cases or the development of new test cases, depending on the nature of the new components.

The test case lists are broken into two groups: tests based on script files designed to exercise all components used in modeling a mission, and user interface exercises designed to test the functionality and completeness of the graphical user interface. The test tracking spreadsheet has separate pages for the GUI and script based test cases. Figure 2.4 shows the page for the script cases; the GUI test case page is similar.

When a test case is added to the test case list using the spreadsheet macros described below, that test case name is automatically picked up on the coverage tables. Once this update has been made and the new test cases have been added to the system test suite, users of the test matrix spreadsheet edit the matrices to indicate the covered functionality. In summary, the procedure for incorporating a new test case is to perform these three steps:

1. *Test case planning:* Identify and name the new test cases, and update the spreadsheet to list these cases.
2. *Test case writing:* Write the new test cases, and update any older test cases that need updating.
3. *Test Matrix Mapping:* Working from the new test cases, fill in the coverage tables for each new or changed test case to reflect the features actually covered.

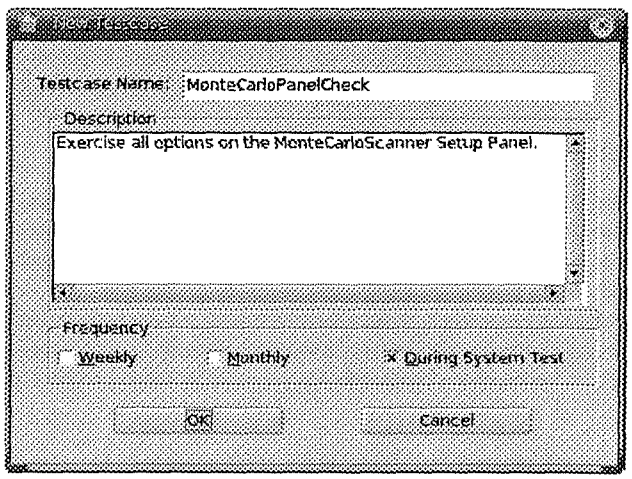


Figure 2.5: The New Test Case Dialog

The procedure for adding a test case to the test case list is similar to the procedure for adding a new element to the test matrices. Test cases are added to the system test matrices using the "Add Script Testcase" and "Add GUI Testcase" buttons on the summary page of the spreadsheet. Pressing either of these buttons opens the New Test Case dialog, shown in Figure 2.5.

When a new test case has been identified, a user will open the system test spreadsheet and press the button for the desired test case type, opening this dialog. The user then enters the name of the new test case. The user enters a summary description of the test case as well to help track the goal of the test case.

Finally, the user selects the desired frequency for execution of the test case; cases that can be automated and run frequently, or that test critical features of the system, should be set to run more frequently than those that are labor intensive or that test rarely used GMAT features.

The user accepts the new test case by selecting the "OK" button on the spreadsheet. When this action is taken, several things happen in the tables in the spreadsheet. First new test case is added to the appropriate page of the spreadsheet, along with its descriptions and execution frequency. The status of the test case is set to "Not started", indicating that the test case itself is not yet in the system test suite of test cases. The new test case is added to the column labels of the test matrices on the subsequent pages, and the formulae in in the spreadsheet are updated to track the new tests.

This step completes the test case planning phase of the preparation process. The next step is to write the test cases themselves.

2.5 Constructing the Test Cases

The steps described so far ensure that there is a plan in place to test every element of GMAT for a black box perspective. At this point, the test cases requires for the system test have been identified. Next the test team needs to write the test cases, given the new functionality of the system. The goal for each test case is to test an integrated set of system elements when executing a specified set of goals.

For the script based tests, this usually involves assembling a set of elements together and performing some computations in a mission sequence. The results of the execution of the script are compared to known good data in order to validate that the execution behaved as expected. Additionally, the script based testing checks to see that scripting errors are handled gracefully, producing error messages that are clear for typical GMAT users.

GUI based scripts have similar goals. The goals of the GUI test cases are to ensure that the GMAT user interface lets users configure all of the elements of the system, that this configuration is reflected in the internal components of the system, and that the user interface handles anomalous conditions robustly.

The following paragraphs describe the approach taken to ensure that these goals are met.

2.5.1 Updating Script Based Test Cases

Script based test cases consist of a script file and validated output files generated from the script. All script based tests should be created from the GMAT GUI, so that any related user interface issues can be identified during the process. Once a scripted test has been constructed, it should be saved with the same file name as entered in the test case table.

Each script based test should generate output in the form of a text file, using GMAT's reporting capabilities. Unless explicitly stated otherwise, the output file name should be the same as the script file name with the file extension ".report". The header comments on the script based tests should indicate the following information:

- The first line of the script should be "%% Id". This ensures that the CVS version information is stored with the script. This CVS information is the tracking identifier for each system test case.
- The primary elements being tested.
- Any ancillary items that should also be examined in the execution of the test.
- Any dependencies that need to be met to run the test successfully. For example, the FminconOptimizer requires a GMAT build that includes the MATLAB interfaces, a valid licensed MATLAB executable on the test machine, and a valid licensed copy of MATLAB's Optimization Toolbox.
- The name of the output files generated, is their name differs from the standard output file name.
- Whether the output data is expected to match data from previous runs.

- Any special steps that should be taken, either prior to the run or after it completes.

A sample script test case is provided here:

```
%% $Id: BasicProp.m,v 1.5 2006/10/11 16:37:00 dconway Exp $
%% GMAT System Test Script File
%
% This test case is designed to test the following elements:
%
% 1.  Spacecraft state specification in Earth MJ2000 Cartesian, Keplerian, and
%     Modified Keplerian Coordinates.
% 2.  Force models appropriate to LEO, HEO and GEO orbits.
% 3.  Basic orbit Propagation.
%
% The only output file is BasicPropHEOReport.txt, which contains various output
% parameters for the HEO spacecraft.  The data in this report should be the same
% from run to run.
%
% There are no external dependencies.
%
% This file has been edited to reduce size, so that it can be used as an example
% in the System Test Plan.

Create Spacecraft LEO;
GMAT LEO.DateFormat = TAIModJulian;
GMAT LEO.Epoch = 21545;
GMAT LEO.CoordinateSystem = EarthMJ2000Eq;
GMAT LEO.StateType = Cartesian;
GMAT LEO.X = 7100;
GMAT LEO.Y = 0;
GMAT LEO.Z = 1300;
GMAT LEO.VX = 0;
GMAT LEO.VY = 7.35;
GMAT LEO.VZ = 1;

Create Spacecraft HEO;
GMAT HEO.DateFormat = TAIGregorian;
GMAT HEO.Epoch = 12 Sep 2006 21:28:00.000;
GMAT HEO.CoordinateSystem = EarthMJ2000Eq;
GMAT HEO.StateType = Keplerian;
GMAT HEO.SMA = 43200;
GMAT HEO.ECC = 0.8;
GMAT HEO.INC = 78;
GMAT HEO.RAAN = 15;
GMAT HEO.AOP = 35;
GMAT HEO.TA = 120;

Create Spacecraft GEO;
GMAT GEO.DateFormat = UTCGregorian;
GMAT GEO.Epoch = 25 Dec 2010 00:00:00.000;
GMAT GEO.CoordinateSystem = EarthMJ2000Eq;
GMAT GEO.StateType = ModifiedKeplerian;
```

```
GMAT GEO.RadPer = 42164.5;
GMAT GEO.RadApo = 42165.5;
GMAT GEO.INC = 0.5;
GMAT GEO.RAAN = 90;
GMAT GEO.AOP = 90;
GMAT GEO.TA = 90;

Create ForceModel LeoProp_ForceModel;
GMAT LeoProp_ForceModel.CentralBody = Earth;
GMAT LeoProp_ForceModel.PrimaryBodies = {Earth};
GMAT LeoProp_ForceModel.Drag = Exponential;
GMAT LeoProp_ForceModel.Gravity.Earth.Degree = 20;
GMAT LeoProp_ForceModel.Gravity.Earth.Order = 20;
GMAT LeoProp_ForceModel.Gravity.Earth.PotentialFile = c:/GmatDataFiles/gravity/earth/JGM2.grv;
GMAT LeoProp_ForceModel.Drag.AtmosphereBody = Earth;

Create Propagator LeoProp;
GMAT LeoProp.FM = LeoProp_ForceModel;
GMAT LeoProp.Type = RungeKutta89;

Create ForceModel HeoProp_ForceModel;
GMAT HeoProp_ForceModel.CentralBody = Earth;
GMAT HeoProp_ForceModel.PrimaryBodies = {Earth};
GMAT HeoProp_ForceModel.Drag = MSISE90;
GMAT HeoProp_ForceModel.SRP = On;
GMAT HeoProp_ForceModel.Gravity.Earth.Degree = 4;
GMAT HeoProp_ForceModel.Gravity.Earth.Order = 4;
GMAT HeoProp_ForceModel.Gravity.Earth.PotentialFile = c:/GmatDataFiles/gravity/earth/JGM3.grv;
GMAT HeoProp_ForceModel.Drag.InputSource = Constant;

Create Propagator HeoProp;
GMAT HeoProp.FM = HeoProp_ForceModel;
GMAT HeoProp.Type = RungeKutta89;

Create ForceModel GeoProp_ForceModel;
GMAT GeoProp_ForceModel.CentralBody = Earth;
GMAT GeoProp_ForceModel.PrimaryBodies = {Earth};
GMAT GeoProp_ForceModel.PointMasses = {Sun, Luna, Jupiter, Venus};
GMAT GeoProp_ForceModel.SRP = On;
GMAT GeoProp_ForceModel.Gravity.Earth.Degree = 4;
GMAT GeoProp_ForceModel.Gravity.Earth.Order = 4;

Create Propagator GeoProp;
GMAT GeoProp.FM = GeoProp_ForceModel;
GMAT GeoProp.Type = PrinceDormand78;
Create ReportFile HeoReport;
GMAT HeoReport.Filename = BasicPropHEOReport.txt;
GMAT HeoReport.Precision = 16;
GMAT HeoReport.Add = {LEO.A1Gregorian, LEO.A1ModJulian, LEO.ElapsedSecs, ...
   LEO.ElapsedDays, LEO.Earth.SMA, LEO.Earth.ECC, LEO.EarthMJ2000Eq.INC, ...
   LEO.EarthMJ2000Eq.RAAN, LEO.EarthMJ2000Eq.AOP, LEO.Earth.TA};
```

```
100
101 %%-------------------------------------------------------------
102 %%----------  Mission Sequence
103 %%-------------------------------------------------------------
104 Propagate LeoProp(LEO, {LEO.ElapsedSecs = 8640.0});
105 Propagate HeoProp(HEO, {HEO.ElapsedSecs = 432000.0});
106 Propagate GeoProp(GEO, {GEO.ElapsedDays = 30.0});
```

If a script test case fails any of the system test criteria specified in Chapter 3, the tester creates a test report summarizing the nature of the failure. A sample completed report is shown here:

```
1  $Id: MatlabApsidesCheck.txt,v 1.3 2006/11/23 00:27:43 dconway Exp $
2
3
4  Tester: ___D. Conway_________________      Date: _11/21/06_____________
5
6
7  Platform:   _X_ Windows, Version: XP, Service Pack 2____
8
9             ___ Macintosh, OS X Version: _____________
10
11            ___ Linux, Distribution: ___________________
12
13
14 Description:
15
16 This test validates the MATLAB interface, including passing of arrays into
17 MATLAB and receipt of data back from MATLAB.
18
19
20 Script Test Results:
21
22 Loads Correctly:  [XX] Pass   [ ] Fail  Bug# ______
23
24 Runs Correctly:   [XX] Pass   [ ] Fail  Bug# ______
25                  [ ] Unable to evaluate
26
27 3D Visualization: [ ] Pass   [ ] Fail  Bug# ______
28                  [XX] Not Applicable
29                  [ ] Unable to evaluate
30
31 Plots:           [ ] Pass   [ ] Fail  Bug# ______
32                  [XX] Not Applicable
33                  [ ] Unable to evaluate
34
35 Output:          [XX] Pass   [ ] Fail  Bug# ______
36                  [ ] Not Applicable
37                  [ ] Unable to evaluate
38
39 Truth Data:      [ ] Pass   [XX] Fail  Bug# _511__
40                  [ ] Not Applicable
41                  [ ] Unable to evaluate
```

```
42
43 Rerun:            [XX] Pass   [ ] Fail  Bug# ______
44                   [ ] Not Applicable
45                   [ ] Unable to evaluate
46
47 Save and Load:    [ ] Pass   [XX] Fail  Bug# _512__
48                   [ ] Unable to evaluate
49
50 Summary:
51
52    Number of passed test elements      __4___
53
54    Total number of test elements       __6___
55
56    Test case status                          [ ] Pass  [X] Fail
57
58
59 Bugs Reported:
60
61    511, 512
62
63 Notes:
64
65 1. Truth data file shows a defect in data handling when receiving data from
66 MATLAB.  The MATLAB return only has 6 digits of precision.  A bug needs to be
67 entered into Bugzilla for this defect.
68
69 2. Save fails when there are multiple conditions on an If command.
```

2.5.2 Updating the GUI Test Cases

GUI based test cases consist of a text file describing the test. The GUI test cases may include additional files, depending on the nature of the test. For example, the script reading GUI test includes a script that needs to be read. The purpose of the GUI tests is to validate that the build is stable, and that the user interface panels provide complete coverage of the elements of the system visible to the user.

The GUI test cases forms are relatively simple. They provide, in outline form, guidelines for testing the GUI elements. Detailed instructions for the GUI tests are provided in Chapter 4.

A sample GUI test case is provided here:

```
1  $Id: ImpulsiveBurnPanel.txt,v 1.4 2006/10/13 19:22:24 dconway Exp $
2
3  Description: This test validates the functionality of the Impulsive Burn
4  configuration panel.
5
6  Procedure:
7
8  1. Open GMAT.  Create an ImpulsiveBurn resource.
9
10    [ ] Pass   [ ] Fail  Bug# ______
11
12 2. Open the panel for the new ImpulsiveBurn.
```

```
13
14    [ ] Pass   [ ] Fail  Bug# ______
15
16  3. Evaluate the aesthetic qualities of the panel.
17
18    [ ] Pass   [ ] Fail  Bug# ______
19
20  4. Evaluate the panel functionality by exercising these elements:
21
22       Axes ComboBox              [ ] Pass   [ ] Fail  Bug# ______
23
24       Vector Format ComboBox     [ ] Pass   [ ] Fail  Bug# ______
25
26       Vector Element 1 Text      [ ] Pass   [ ] Fail  Bug# ______
27
28       Vector Element 2 Text      [ ] Pass   [ ] Fail  Bug# ______
29
30       Vector Element 3 Text      [ ] Pass   [ ] Fail  Bug# ______
31
32       Origin ComboBox            [ ] Pass   [ ] Fail  Bug# ______
33
34  5. Evaluate panel save/cancel/restore functionality.
35
36       Cancel           [ ] Pass   [ ] Fail  Bug# ______
37
38       Apply            [ ] Pass   [ ] Fail  Bug# ______
39
40       Save             [ ] Pass   [ ] Fail  Bug# ______
41
42       Restore          [ ] Pass   [ ] Fail  Bug# ______
43
44       Window Icons     [ ] Pass   [ ] Fail  Bug# ______
45
46  6. Evaluate rename functionality.
47
48    [ ] Pass   [ ] Fail  Bug# ______
49
50  7. Validate that the configured object is correct on run.
51
52    [ ] Pass   [ ] Fail  Bug# ______
53
54  8. Perform additional experiments with the panel controls.
55
56  Summary:
57
58  Test case status:
59
60    [ ] Pass   [ ] Fail
61
62  Bugs Reported:
63
```

```
64  Notes:
65
66
67
68
69  Tester:  ____________________________
70
71  Date:    ____________________________
```

Failed GUI tests provide information about the nature of the failure durectly on the test case form; there is no supplementary report for GUI test failures.

2.6 Ensuring Complete System Coverage

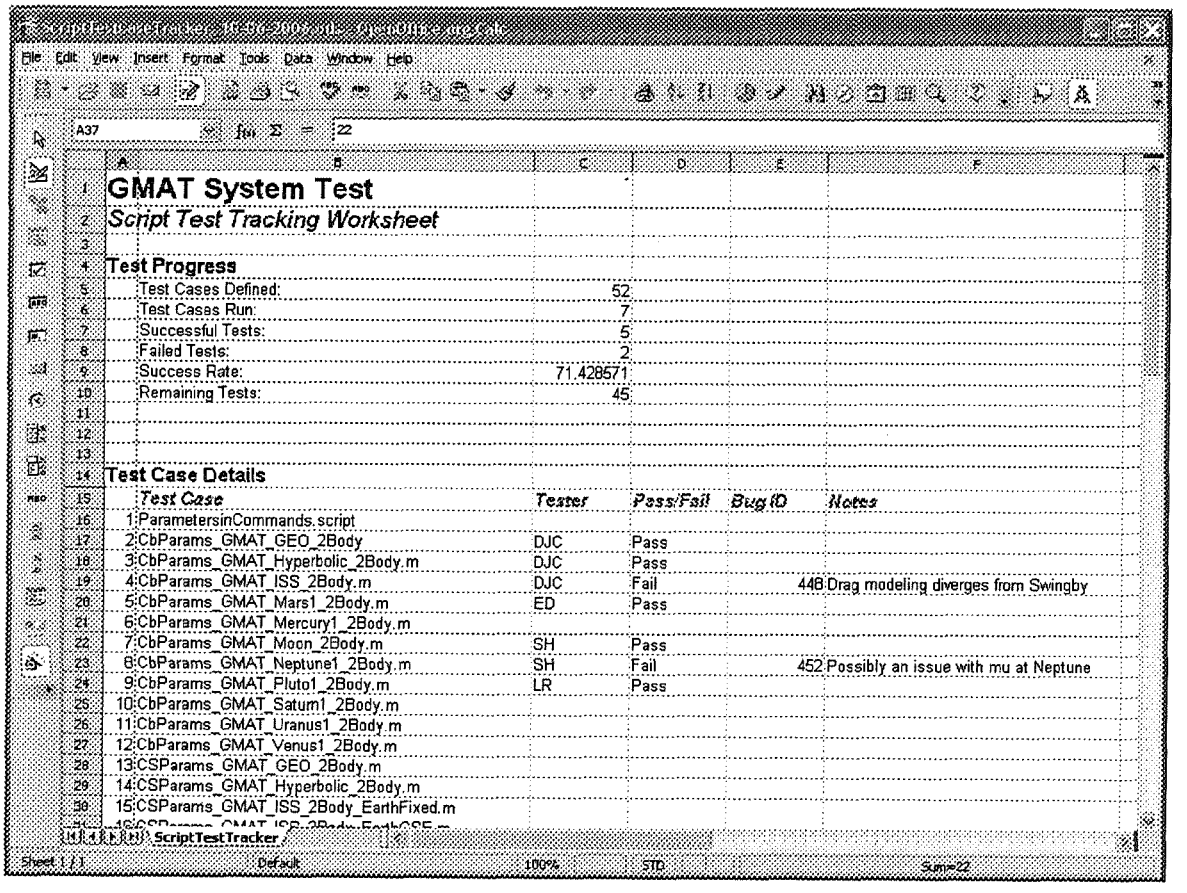
Once the test cases have been written, all that remains for test proparation is the confirmation that the test cases cover all of the new features of GMAT. This is accomplished by updating the test matrices based on the new and revised test cases. Each test case that has been added or changed since the last update is collected and used to update the matrices. For each page in the spreadsheet containing an element to test case table, the test team needs to update the matrix for these test cases. The test cases are listed across the top of the matrices. Each test case identifies the tested elements by placing an "X" marker in the row corresponding to that element. Updated test cases should be examined to ensure that elements previously tested are still tested; if an elemnet is no longer tested for a specific test case, the X for that element should be removed from the matrix.

The spreadsheet contains formulas that use these markers to determine if a given element has a corresponding test case. The far right side of the test matrices tables accumulates this data; every element that has at least one associated test case receives a coverage value of 1; uncovered elements receive a coverage value of 0. The far right side of the table also includes a column labeled "Row Count." The row count column simply counts the number of elements on the page.

The summary page examines each table in the spreadsheet and provides information about the coverage completeness of the system tests. Once the coverage statistics report that the elements of the system are covered 100%, the system tests are ready to be run. The test team then generates a new spreadsheet for each type of system test by pressing the "Create Script Test Tracker" and "Create GUI Test tracker" buttons on the summary page. These buttons generate single page spreadsheets used to track progress through the system test. An example is shown in Figure 2.6.

This spreadsheet is used to track and report system test progress. As each system test is performed, the entry in the tracking spreadsheet is updated by the test team. Examination of this spreadsheet provides a status check on the system test.

The next two chapters provide instructions about the steps performed when running the system tests.



GMAT System Test

Script Test Tracking Worksheet

Test Progress

Test Cases Defined:	52
Test Cases Run:	7
Successful Tests:	5
Failed Tests:	2
Success Rate:	71.428571
Remaining Tests:	45

Test Case Details

	Test Case	Tester	Pass/Fail	Bug ID	Notes
1	ParametersinCommands.script				
2	CbParams_GMAT_GEO_2Body	DJC	Pass		
3	CbParams_GMAT_Hyperbolic_2Body.m	DJC	Pass		
4	CbParams_GMAT_ISS_2Body.m	DJC	Fail	448	Drag modeling diverges from Swingby
5	CbParams_GMAT_Mars1_2Body.m	ED	Pass		
6	CbParams_GMAT_Mercury1_2Body.m				
7	CbParams_GMAT_Moon_2Body.m	SH	Pass		
8	CbParams_GMAT_Neptune1_2Body.m	SH	Fail	452	Possibly an issue with mu at Neptune
9	CbParams_GMAT_Pluto1_2Body.m	LR	Pass		
10	CbParams_GMAT_Saturn1_2Body.m				
11	CbParams_GMAT_Uranus1_2Body.m				
12	CbParams_GMAT_Venus1_2Body.m				
13	CSParams_GMAT_GEO_2Body.m				
14	CSParams_GMAT_Hyperbolic_2Body.m				
15	CSParams_GMAT_ISS_2Body_EarthFixed.m				

Figure 2.6: A Test Tracking Spreadsheet

Chapter 3

Executing Script Driven Tests

The tests described in this chapter are designed to exercise all accessible objects in the core GMAT engine, in as many combinations as is feasible. This object coverage is performed by running GMAT scripts designed to interact with the accessible objects from the Graphical User Interface. Each script produces output. The system testers examine this output, and, when possible, compare it with the configuration managed output produced from previous runs of the scripts. The procedure followed when running scripted tests is documented in the sections of this chapter.

3.1 Script Test Case Management

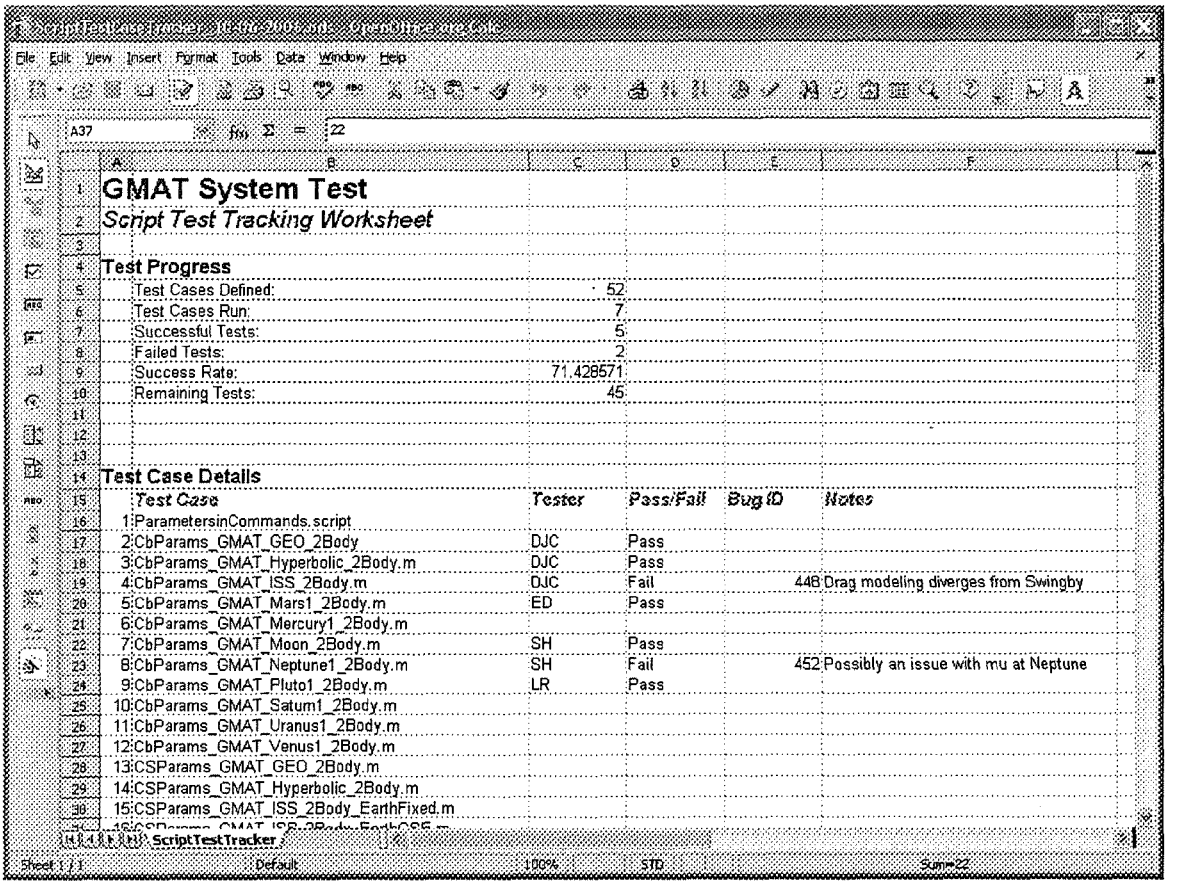
The System test cases are managed from a spreadsheet generated at the conclusion of the system test preparation process. Figure 3.1 shows an example of this test tracking spreadsheet for the script based tests[1], as it looks partway through a test cycle.

The test procedure for script based tests is relatively straightforward. Testers follow these steps when executing the system tests:

1. Obtain the latest versions of the scripts and known good results from the system test repository.
2. Identify the tests each tester needs to run.
3. Open GMAT[2].
4. Run each test following the procedure in 3.2.
5. As each test is run, record the summary results in a local copy of the test tracking spreadsheet.
6. When anomalies are found in testing, record them local test case report files.
7. At the end of each day or when testing is finished, whichever occurs first, gather the test case reports generated from the tests and place them in the folder used to gather the test results.
8. Close GMAT at the end of the test period.
9. At the end of each day or when testing is finished, whichever occurs first, save the local test tracking spreadsheet with the name <spreadsheetName>_<tester's initials> in the folder used to gather the test results.

[1]The test tracking spreadsheets, unlike the traceability matrix spreadsheet, can be saved in either OpenOffice or Excel format.

[2]GMAT should only be opened one time for any given testing period. All tests run during that test period – typically a morning or afternoon – should be run in the same instance of GMAT. This helps ensure that the system is stable over long periods of time. If the system is shut down, either by the user or through a system crash, that event should be noted.



GMAT System Test

Script Test Tracking Worksheet

Test Progress

Test Cases Defined:	52
Test Cases Run:	7
Successful Tests:	5
Failed Tests:	2
Success Rate:	71.428571
Remaining Tests:	45

Test Case Details

Test Case	Tester	Pass/Fail	Bug ID	Notes
1:ParametersinCommands.script				
2:CbParams_GMAT_GEO_2Body	DJC	Pass		
3:CbParams_GMAT_Hyperbolic_2Body.m	DJC	Pass		
4:CbParams_GMAT_ISS_2Body.m	DJC	Fail	448	Drag modeling diverges from Swingby
5:CbParams_GMAT_Mars1_2Body.m	ED	Pass		
6:CbParams_GMAT_Mercury1_2Body.m				
7:CbParams_GMAT_Moon_2Body.m	SH	Pass		
8:CbParams_GMAT_Neptune1_2Body.m	SH	Fail	452	Possibly an issue with mu at Neptune
9:CbParams_GMAT_Pluto1_2Body.m	LR	Pass		
10:CbParams_GMAT_Saturn1_2Body.m				
11:CbParams_GMAT_Uranus1_2Body.m				
12:CbParams_GMAT_Venus1_2Body.m				
13:CSParams_GMAT_GEO_2Body.m				
14:CSParams_GMAT_Hyperbolic_2Body.m				
15:CSParams_GMAT_ISS_2Body_EarthFixed.m				

Figure 3.1: The Script Test Tracking Spreadsheet

10. Upon completion of all assigned test cases, report that status to the system test lead.

3.2 Running the Scripted System Tests

By their very nature, the GUI based tests described in Chapter 4 follow a relatively unstructured execution sequence that mandates more structured test case documents to ensure complete system testing. In contrast, the script based tests follow a linear execution sequence once the scripts have been written and debugged. The rest of this chapter describes the procedure followed for the scripted tests.

3.2.1 Procedure

Each scripted test case has an associated, configuration managed script. Most script test cases also have output data files used to compare the obtained script outputs with validated GMAT output files. A tester follows this procedure to perform the associated system test:

1. Open a blank test case report file[3].

[3]The test case report file is only needed for script based tests is an anomaly is found during testing. In practice, the test

2. Open the script in GMAT.
3. Compare the resources displayed in GMAT with the resources defined in the script. Enter any anomalies in the test case report.
4. Compare the mission sequence in the script with the mission sequence displayed in GMAT. Enter any anomalies in the test case report.
5. Run the script.
6. Examine each plot and 3D view that opens. Enter any anomalies on the in the test case report.
7. Compare the output results from the run with the known good data. Enter any anomalies in the test case report.
8. Press the run button.
9. Examine each plot and 3D view that opens. Enter any anomalies on the in the test case report.
10. Compare the output results from the run with the known good data. Enter any anomalies in the test case report.
11. Open the script in the editor window, and press the "Build and Run" button.
12. Examine each plot and 3D view that opens. Enter any anomalies on the in the test case report.
13. Compare the output results from the run with the known good data. Enter any anomalies in the test case report.
14. Save the script to a new file with the name Saved_<Test case name>.
15. Load the saved script into GMAT.
16. Repeat steps 3 through 11
17. If any anomalies have been found, fill in the header and summary data on the test case report, and save it with the file name "<test case>_YYYYMMDD.report", where YYYYMMDD indicate the year, month and day the test was run.

3.2.2 A Note on Run Frequency

The script based tests can be run much more frequently than is feasible for the GUI tests. Scripts that are identified as being run more frequently than at the system test frequency follow a somewhat abbreviated procedure from that defined at the system test level. The purpose of the more frequent testing is to help catch errors in the system prior to format system testing. Teh abbreviated test procedure performed for each weekly or monthly test is presented here:

1. Open the script in GMAT.
2. Run the script.
3. Examine each plot and 3D view that opens. Report any anomalies.
4. Compare the output results from the run with the known good data. Report any anomalies.
5. If any anomalies have been found, enter a new anomaly in the bug tracking system.

These tests follow the full system test procedure when run as part of the system test suite.

case report only needs to be opened when an anomaly is found.

3.2.3 Reporting Results

At the start of the system test process, a central location was established for collection of the test results. The final step performed by the system testers is to copy their test case worksheets and local test tracking worksheet to this central location. This action is performed each day the system tests are run so that the progress of the system test execution can be evaluated. Upon completion of all system testing by a specific tester, a final update is made and the system test lead is notified that that tester has completed the assigned tests. Chapter 5 describes the consolidation of the collected test results into a system test report.

Chapter 4

Executing Tests for the Graphical User Interface

The tests described in this chapter are designed to exercise all of the controls and other elements visible from the GMAT graphical user interface (GUI). The GMAT GUI is designed to present a consistent, easy to use interface into the underlying engine so that users of the system can view, configure, and interact with the elements of the system during all phases of mission modeling. System testers work with these elements, using them both to perform the expected tasks and to attempt to perform undesired actions. The former set of actions exercises the engine to ensure that the system can be configured correctly. The latter tests are run to ensure that users cannot configure GMAT incorrectly.

4.1 GUI Test Case Management

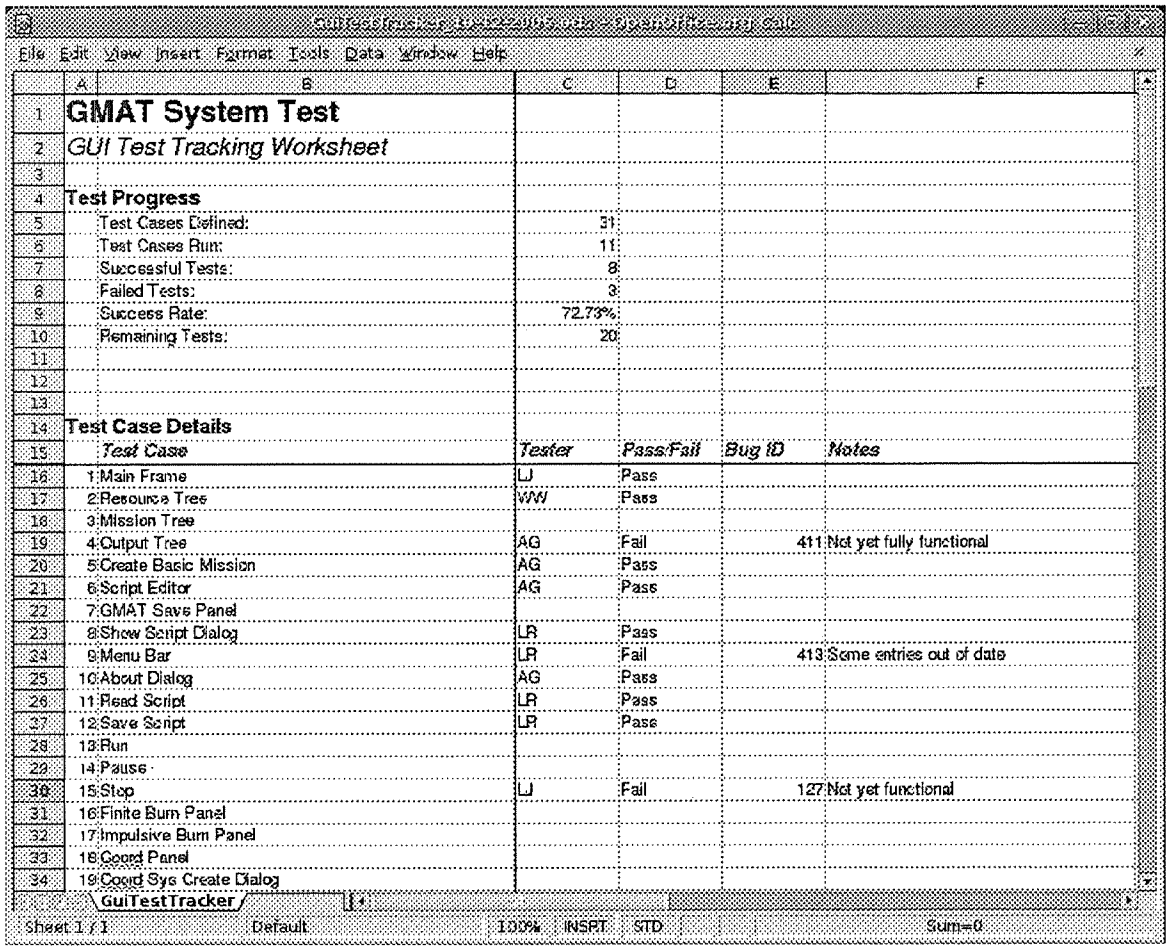
The GUI test cases are managed using a test tracking spreadsheet generated at the end of test preparation, described in Chapter 2. Figure 4.1 shows an example of this spreadsheet partway through a testing cycle.

The test procedure for GUI based tests requires extensive exercising of the components in the GUI. Testers follow these steps when executing the system tests:

1. Obtain the latest versions of the GUI test cases and a local copy of the test case tracking spreadsheet[1].
2. Identify the tests that the tester needs to run.
3. Open GMAT[2].
4. Run each test following the procedure in Section 4.2.
5. As each test is run, record the results of the test on the test case worksheet retrieved in step 1.
6. When anomalies are found in testing, record them on the test case worksheet and enter them in the bug tracking database.
7. Close GMAT at the end of the test period.
8. At the end of each day or when testing is finished, whichever occurs first, gather the completed test case worksheets and place them in the folder used to gather the test results.

[1]The test tracking spreadsheet is generated from the System Test Matrix spreadsheet using an OpenOffice macro, as described in Section 2.6.

[2]GMAT should only be opened one time for any given testing period. All tests run during that test period -- typically a morning or afternoon -- should be run in the same instance of GMAT. This helps ensure that the system is stable over long periods of time. If the system is shut down, either by the user or through a system crash, that event should be noted.



File Edit View Insert Format Tools Data Window Help

	A	B	C	D	E	F
1		**GMAT System Test**				
2		*GUI Test Tracking Worksheet*				
3						
4		**Test Progress**				
5		Test Cases Defined:	31			
6		Test Cases Run:	11			
7		Successful Tests:	8			
8		Failed Tests:	3			
9		Success Rate:	72.73%			
10		Remaining Tests:	20			
11						
12						
13						
14		**Test Case Details**				
15		*Test Case*	*Tester*	*Pass/Fail*	*Bug ID*	*Notes*
16	1	Main Frame	LJ	Pass		
17	2	Resource Tree	WW	Pass		
18	3	Mission Tree				
19	4	Output Tree	AG	Fail	411	Not yet fully functional
20	5	Create Basic Mission	AG	Pass		
21	6	Script Editor	AG	Pass		
22	7	GMAT Saves Panel				
23	8	Show Script Dialog	LR	Pass		
24	9	Menu Bar	LR	Fail	413	Some entries out of date
25	10	About Dialog	AG	Pass		
26	11	Read Script	LR	Pass		
27	12	Save Script	LR	Pass		
28	13	Run				
29	14	Pause				
30	15	Stop	LJ	Fail	127	Not yet functional
31	16	Finite Burn Panel				
32	17	Impulsive Burn Panel				
33	18	Coord Panel				
34	19	Coord Sys Create Dialog				

GuiTestTracker

Sheet 1 / 1 Default 100% INSRT STD Sum=0

Figure 4.1: The GUI Test Tracking Spreadsheet

9. At the end of each day or when testing is finished, whichever occurs first, save the local gui test tracking spreadsheet with the name <spreadsheetName>_<tester's initials> in the folder used to gather the test results.

10. Upon completion of all assigned test cases, report that status to the system test lead.

The procedure for running a single test case is described next.

4.2 Running the GUI System Tests

By their very nature, the script based tests described in Chapter 3 follow a linear execution sequence once the scripts have been written and debugged. In contrast, interactions performed using the GMAT GUI are less structured -- users can use the controls on the GUI in a seemingly random fashion -- so the test cases for the GUI include allowances for interacting with the GUI elements by the tester in a more free form manner than the script based tests allow.

4.2.1 Sample GUI Test Case

A sample GUI test case is shown here:

```
$Id $


Tester: _______________________________      Date: _________________________


Description:

This test validates the functionality of the OpenGL panel.
(* indicates sub-panel whose functionality is tested separately)


Procedure:

1. Create and open the appropriate object panel.

   Create OpenGL Resource                    [ ] Pass  [ ] Fail  Bug# ______

   Open OpenGL Resource                      [ ] Pass  [ ] Fail  Bug# ______


2. Evaluate the aesthetic qualities of the panel.

   Panel Aesthetics                          [ ] Pass  [ ] Fail  Bug# ______


3. Evaluate the individual panel elements.

   Show Plot Check Box                       [ ] Pass  [ ] Fail  Bug# ______

   Collect Data Text Field                   [ ] Pass  [ ] Fail  Bug# ______

   Update Plot Text Field                    [ ] Pass  [ ] Fail  Bug# ______

   Number of Points to Redraw Text Field     [ ] Pass  [ ] Fail  Bug# ______

   Draw Wireframe Check Box                  [ ] Pass  [ ] Fail  Bug# ______

   Draw Targeting Check Box                  [ ] Pass  [ ] Fail  Bug# ______

   Draw Ecliptic Plane Check Box             [ ] Pass  [ ] Fail  Bug# ______

   Draw XY Plane Check Box                   [ ] Pass  [ ] Fail  Bug# ______

   Draw Axes Check Box                       [ ] Pass  [ ] Fail  Bug# ______

   Draw Grid Check Box                       [ ] Pass  [ ] Fail  Bug# ______
```

Draw Earth/Sun Lines Check Box [ ] Pass [ ] Fail Bug# _____

Spacecraft List [ ] Pass [ ] Fail Bug# _____

Selected Spacecraft List [ ] Pass [ ] Fail Bug# _____

Celestial Object List [ ] Pass [ ] Fail Bug# _____

Selected Celestial Object List [ ] Pass [ ] Fail Bug# _____

--> (Add) Selection Button [ ] Pass [ ] Fail Bug# _____

<-- (Remove) Selection Button [ ] Pass [ ] Fail Bug# _____

< = (Remove All) Selection Button [ ] Pass [ ] Fail Bug# _____

Show Object Check Box [ ] Pass [ ] Fail Bug# _____

Orbit Color Select Box [ ] Pass [ ] Fail Bug# _____

Target Color Select Box [ ] Pass [ ] Fail Bug# _____

Use Initial View Definition Check Box [ ] Pass [ ] Fail Bug# _____

Use Perspective Mode Check Box [ ] Pass [ ] Fail Bug# _____

Use Fixed FOV Angle Check Box [ ] Pass [ ] Fail Bug# _____

Field of View Text Field [ ] Pass [ ] Fail Bug# _____

Coordinate System Combo Box [ ] Pass [ ] Fail Bug# _____

View Point Reference Combo Box (see 4a) [ ] Pass [ ] Fail Bug# _____

View Point Vector Combo Box (see 4b) [ ] Pass [ ] Fail Bug# _____

View Scale Factor Text Field [ ] Pass [ ] Fail Bug# _____

View Direction Combo Box (see 4c) [ ] Pass [ ] Fail Bug# _____

Coordinate System Combo Box [ ] Pass [ ] Fail Bug# _____

Axis Combo Box [ ] Pass [ ] Fail Bug# _____

4. Evaluate panel-specific functionality.

a. Select 'Vector' for View Point Reference

Vector 1 Text Field [ ] Pass [ ] Fail Bug# _____

```
100         Vector 2 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
101
102         Vector 3 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
103
104      b. Select 'Vector' for View Point Vector
105
106         Vector 1 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
107
108         Vector 2 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
109
110         Vector 3 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
111
112      c. Select 'Vector' for View Direction
113
114         Vector 1 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
115
116         Vector 2 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
117
118         Vector 3 Text Field                    [ ] Pass  [ ] Fail  Bug# ______
119
120      Use Perspective Mode Check Box            [ ] Pass  [ ] Fail  Bug# ______
121      --- select checkbox to check following
122
123      Use Fixed FOV Angle Check Box             [ ] Pass  [ ] Fail  Bug# ______
124      --- select checkbox to check following
125
126      Field of View Text Field                  [ ] Pass  [ ] Fail  Bug# ______
127
128
129   5. Evaluate data.
130
131      Data elements appear complete             [ ] Pass  [ ] Fail  Bug# ______
132
133      Show Script                               [ ] Pass  [ ] Fail  Bug# ______
134
135
136   6. Evaluate panel control.
137
138      Tab Key Navigation                        [ ] Pass  [ ] Fail  Bug# ______
139
140      Cancel                                    [ ] Pass  [ ] Fail  Bug# ______
141
142      Apply                                     [ ] Pass  [ ] Fail  Bug# ______
143
144      OK (Save)                                 [ ] Pass  [ ] Fail  Bug# ______
145
146      Help [DEFERRED]
147
148      Restore                                   [ ] Pass  [ ] Fail  Bug# ______
149
150      Minimize                                  [ ] Pass  [ ] Fail  Bug# ______
```

```
151
152     Maximize                                [ ] Pass  [ ] Fail  Bug# ______
153
154     Close                                   [ ] Pass  [ ] Fail  Bug# ______
155
156
157  7. Evaluate rename functionality.
158
159     Rename (on resource tree)               [ ] Pass  [ ] Fail  Bug# ______
160
161
162  Summary:
163
164     Number of passed test elements      ______
165
166     Total number of test elements       ______
167
168     Test case status                        [ ] Pass  [ ] Fail
169
170
171  Bugs Reported:
172
173
174
175  Notes:
176
177
```

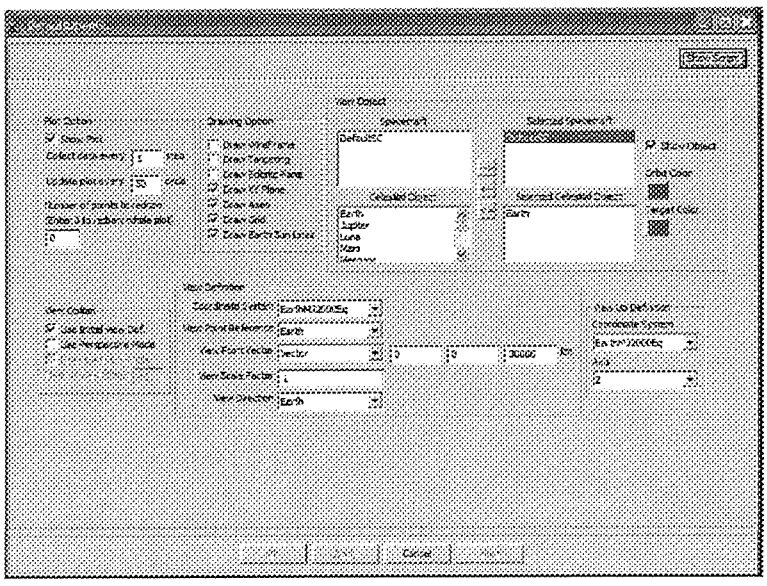


Figure 4.2: The OpenGLPlot Setup Panel

The test case worksheet shown here is the test case for the OpenGL plot setup panel. The panel, shown in Figure 4.2, is a fairly complex GUI panel, containing text fields, combo boxes, check boxes, text lists, and action buttons which open color selection dialogs. Each element is included in the test plan worksheet, along with the standard control processes that need to be exercised. Each test criterion is evaluated using this worksheet, and given a pass or fail evaluation.

4.2.2 Procedure

Each GUI test case has a worksheet like the one shown above. A tester follows this procedure to perform the associated system test:

1. Open the test case worksheet.
2. Follow the procedure outlined in the test case.
   - Section 4.3 provides detailed instructions about the process that should be followed when testing each type of GUI element.
   - Each item identified in the worksheet is marked as either passing or failing the test. If the item fails, an associated bug is entered or identified in the bug tracking system and listed on the worksheet.
   - After completing the tests on the worksheet, the tester experiments with the component for an additional period (typically ten to fifteen minutes), checking to be sure that the component is stable and behaves correctly when bad data is entered, and when random actions are taken using that component.
   - Once every item on the worksheet has been evaluated and the final period of usability testing has been performed, the number of pass and fail evaluations are counted and recorded in the summary section of the test case worksheet. Any bugs identified on the worksheet are listed in this section, and any additional notes that need to be recorded are also listed here[3].
3. Summarize the results of the tests.
   - Once every item on the worksheet has been evaluated, an overall pass or fail evaluation is made and recorded in the summary section. Any bugs identified on the worksheet are listed in this section, and any additional notes that need to be recorded are also listed here.
   - Add the tester's name and the data the test was run to the worksheet.
   - Save the completed test case worksheet.
4. Update the local test tracking worksheet to indicate that the test was run and the results of the run.
5. Save the test tracking worksheet.

4.2.3 Reporting Results

At the start of the system test process, a central location was established for collection of the test results. The final step performed by the system testers is to copy their test case worksheets and local test tracking worksheet to this central location. This action is performed each day the system tests are run so that the progress of the system test execution can be evaluated. Upon completion of all system testing by a specific tester, a final update is made and the system test lead is notified that that tester has completed the assigned tests. Chapter 5 describes the consolidation of the collected test results into a system test report.

[3]These data are collected using an automation tool to build a status report for the system tests.

4.3 Procedural Rules

The steps described in the preceding sections lay out the procedures followed when testing the GUI elements of GMAT. In this section, the criteria that must be evaluated are defined for these tests.

4.3.1 Test Procedures for All Elements

Aesthetics

Description: This set of tests verifies platform-specific look and feel of a panel, as extended by the GMAT GUI Philosophy document[Dove]. Each criterion must be met to pass the aesthetics tests.

- All of the data input fields and bounding boxes can be seen at the panel size displayed when the panel is first opened, for all tabs on the panel.
- The blank space surrounding the data area is not distracting, and does not dominate the appearance of the interface. As a guideline, for platforms that allow control of the surrounding white space, that region should not consume more than 20% of the total space dedicated to the panel when it is opened.
- The data area does not appear too crowded; the surrounding blank space is appropriately sized.
- The window cannot be resized so that the data cannot be seen.

General Panel Functionality

Description: This is the list of tests associated with basic panel functionality: open, close, rename, minimize, ok, cancel, help, show script, command summary. Additionally, the behavior of open panels needs to be consistent with deletion actions taken on the resource and mission trees - if an object in the tree is deleted, any open panel associated with that object should close. All of these functions must pass.

- New objects of the type being tested can be created from the appropriate tree on the Resource or Mission panels.
- Double clicking in a new object opens the panel for that object.
- Double clicking in a object that has an open panel brings the panel for that object to the front of the displayed panels.
- New objects can be renamed.
- Default objects, when they exist, can be renamed.
- Default objects, when they exist, can be deleted.
  - The object can be renamed.
  - References to the renamed object are updated in related elements of the system.
- Renaming works after making changes to the data on the object panel.
  - The object can be renamed while the panel is open.
  - A change can be made on the panel, and then the object can be renamed before the change is applied.
  - A change can be made on the panel, the change can be applied, and then the object can be renamed.

  - For each of the above cases, references to the object's name are updated throughout the system when the object's name is changed.

- Changes made on the panel and applied using the OK button appear on the panel when it is reopened.
- Changes made on the panel and applied using the Apply button are visible in the script when viewed using the Show Script dialog.
- When you open the panel, make a minor change in the panel, and click button to close the panel (on Windows, this is the small "x" button in the upper right hand corner; on the Mac, it is the red button on the left side of the frame controls, and on Linux, varies based on the configuration of the Linux window manager), you are prompted to save data before closing. Check that:
  - The prompt does appear.
  - Selecting "Yes" updates the underlying data.
  - Selecting "No" discards the changes.
- Cancelling closes the opened panel without changing the underlying data.
  - The object does not change when you open the panel and press the Cancel button without making any changes.
  - The object does not change when you open the panel, make a minor change in the data, and press the Cancel button.
  - The object does not change when you open the panel and click the close button in the panel's frame to close the panel, but the panel does close without prompting.
- The panel is minimized when the minimize button on the panel frame is pressed.
- The panel reopens to previous size when maximize icon on the minimized panel is pressed
- The tab key navigates the open panel in agreement with style and GUI design philosophy. Navigation is orderly and sensible using the tab key.

Panel Data Element Completeness and Correctness

Description: This set of tests verifies that all data elements that should appear on the panel are present on the panel. It also tests that all elements that should appear in "Show Script" dialog appear there, and that items that should not appear in show script do not appear there.

- Verify that only data elements that occur in the Range Test Plan appear in show script and that the user does not see any other object fields.
- Verify that defaults agree with the values in the Range Test Plan.
- Press the "Show Script" button, and verify that all elements on the GUI panel also appear on the show script dialog. Verify that these elements match the description in the Range Test Plan.
- Verify that all data elements that appear in Show Script also appear on the GUI. (This step validates that all scriptable settings also appear in the GUI.)

4.3.2 Procedures for Specific Control Types

The following table provides additional guidelines that should be followed when testing each specific type of control.

Table 4.1: Tests for Data Objects on All Panels

Element Type	Tests
Check Boxes	• Set all check boxes to off (unchecked), hit show script, and verify that the functionality is indeed turned off for each radio button and check box. • Set all check buttons to on (checked), hit apply, and show script and verify that the functionality is indeed turned on for each radio button and check box.
Radio Buttons	• For each radio button on panel, select the button, and ensure that it activates and all others are deactivated. Hit Apply, and then check show script to ensure that the configuration was properly saved.
Combo Boxes	• For each combo box on the panel, ensure that all options that appear in Range Test Plan appear in the pull down menu. • For each Combo box on the panel, select each allowable option, hit apply and show script and check to see that the option was correctly saved. • Check to ensure that the combo box is not editable.
Text Fields	• For each text field enter "DNE" and ensure that if GMAT should reject this string that the string is rejected. (Currently, this is not an acceptable value for any GMAT field unless the user has created an appropriate object type and named it DNE, and is using it correctly in the GUI.) • Perform all range tests as described in Range Test Plan. • For all numeric fields, enter an allowed numeric value, hit apply and show script and check that the value was saved. • If user-defined objects can appear in the combo box, create one object for all allowable object types for the particular combo box, and ensure that it appears in the combo box. Also, hit apply and ensure that each case appears in show script.
Action Buttons	• For each button ensure that clicking on the button brings up the appropriate panel. • For the panel opened up, perform all tests defined in Section 4.3.1 and Table 4.1
Selection Lists	• First Item • Second Item

Table 4.1: (Tests for Data Objects on All Panels...continued)

Element Type	Tests
Tabbed Panels	• First Item • Second Item

4.3.3 Usability Testing

The tests described in the preceding paragraphs are meant to exercise all of the elements of the graphical user interface. One important aspect of the interface not covered by those tests is the usability of the system: the GUI may perform error free as designed, and still be difficult to use in practice. Usability testing is performed to capture information about this aspect of the GUI.

Chapter 5

Reporting and Reviewing Test Results

This chapter describes the process followed for tracking the state of the system test process and for reporting the results of the testing.

5.1 System Test Status

The status of the system tests is tracked using the Script and GUI test tracking spreadsheets described in Chapters 3 and 4. System testers update their copies of these spreadsheet daily during system testing. Once a week or upon request, the system test lead consolidates these spreadsheets, collecting the test results in master system test spreadsheets that can be reviewed by interested parties.

5.2 The System Test Report

At the conclusion of system test cycle, the reports generated during system test are consolidated into a single document. This document is prepared using the following outline:

I. Overview

- A. Executive Summary
- B. Test Results
- C. Recommendations

II. Script Test Case Results

- A. Test Result Statistics
- B. Summary of Failed Tests (if any)
- C. Test Results
  - i. ParametersinCommands Test Case Report
  - ii. CbParams_GMAT_GEO_2Body Test Case Report
  - ...

III. GUI Test Case Results

- A. Test Result Statistics
- B. Summary of Failed Tests (if any)
- C. Test Results

i. Mainframe Test Case Worksheet
ii. Resource Tree Test Case Worksheet
...

5.3 System Test Review

The final step in the system test process is to perform a review of the test results. In preparation for this review, each team member and reviewer reviews the System Test Report, highlighting any issues that raise concerns. These parties then meet and discuss the findings of the system testing. The outcome of this review is a list of action items, assigned to specific individuals or teams, and a recommendation about the status of the system for release.

A typical release recommendation will fall into one of three categories: (1) GMAT is ready for release, (2) GMAT is ready for release, contingent on specific items being addressed and approved prior to that release, or (3) GMAT is not ready for release, and needs to meet specific items and be reviewed again before release will be approved.

Following this review, a summary documenting the findings of the review is written and provided to all team members and interested parties. Once GMAT has been released as an open source project, a public version of this summary is made available with the other project artifacts.

Bibliography

[Black] Rex Black, "Managing the Testing Process," Second Edition, Wiley Publishing, 2002.

[Craig] Rick D. Craig and Stefan P. Jaskiel, "Systematic Software Testing," Artech House, 2002.

[Dove] Edwin G. Dove, "GUI Philosophy for the General Mission Analysis Tool (GMAT)."

[MTP] GMAT Test Team, "General Mission Analysis Tool (GMAT) Master Test Plan."

[GDT] GMAT Development Team, "General Mission Analysis Tool (GMAT) Architectural Specification."

[hughes] Steven P. Hughes, "General Mission Analysis Tool (GMAT) Mathematical Specification."

[hughes2] Steven P. Hughes, "General Mission Analysis Tool (GMAT) User's Guide."

[matlab] The MathWorks, Inc, "MATLAB", available from http://www.mathworks.com.

[OOo] OpenOffice.org, "OpenOffice", available from http://www.openoffice.org/.

[opttools] The MathWorks, Inc, "Optimization Toolbox", available from http://www.mathworks.com.