CORE FLIGHT SYSTEM
Stored Command
BUILD 2.5.0.0

FLIGHT SOFTWARE BUILD VERIFICATON TEST REPORT

Flight Software Branch – Code 582

Version 1.0

SIGNATURES

Submitted by:

X WALTER MOLESKI

Digitally signed by WALTER MOLESKI
DN: c=US, o=U.S. Government, ou=NASA, ou=People, cn=WALTER MOLESKI, 0.9.2342.19200300.100.1.1=wmoleski
Date: 2016.10.31 06:21:54 -04'00'

Walt Moleski/582
cFS Flight Software Tester

Approved by:

X SUSANNE STREGE

Digitally signed by SUSANNE STREGE
DN: c=US, o=U.S. Government, ou=NASA, ou=People, cn=SUSANNE STREGE, 0.9.2342.19200300.100.1.1=sstrege
Date: 2016.10.30 21:40:27 -04'00'

Susanne Strege/582
cFS Flight Software Product Development Lead

PLAN UPDATE HISTORY

Version	**Date**	**Description**	**Affected Pages**
1.0		Initial Release	All

TABLE OF CONTENTS

1 INTRODUCTION

1.1 DOCUMENT PURPOSE

This Test Report describes the test results from the Core Flight System (cFS) Stored Command (SC) Flight Software (FSW) Test Team build 2.5.0.0 verification testing. It is used to verify that the SC FSW has been tested in a manner that validates that it satisfies the functional and performance requirements defined within the cFS SC Requirements Document. This Test Report summarizes the FSW test history, the build verification process, the build test configuration, and the test execution and results.

1.2 APPLICABLE DOCUMENTS

Unless otherwise stated, these documents refer to the latest version.

Parent Documents (Mission and FSW)

582-2007-019	cFS Stored Commands Requirements Document, Version 1.5
582-2008-012	cFS Deployment Guide, Version 3.1

Reference Documents

All of the references below can be found on the Code 582 internal website at http://fsw.gsfc.nasa.gov/

582-2003-001	FSB FSW Test Plan Template
582-2004-001	FSB FSW Test Description Template
582-2004-002	FSB FSW Test Scenario Template
582-2004-003	FSB FSW Test Procedure Template
582-2004-004	FSB FSW Test Execution Summary Template
582-2004-005	FSB Test Product Peer Review Form
582-2000-002	FSB FSW Unit Test Standard

1.3 DOCUMENT ORGANIZATION

Section 1 of this document presents some introductory material.

Section 2 provides a flight software overview and context along with the test history and testing overview.

Section 3 describes the build verification process including procedure development and execution and test products produced.

Section 4 describes the build test configuration which includes an overview of the testbed and the requirements verification matrix.

Section 5 describes the test execution and results by subsystem.

Appendix A - provides the Requirements Traceability Matrix

Appendix B - provides the Command, Telemetry, and Events Verification Matrix

1.4 DEFINITIONS

There were 3 verifications methods used during build verification testing. They were:

- Demonstration: Show compliance with system requirement by exhibiting the required capability (e.g. by demonstrating interactive capability, display capability, print capability, etc.
- Inspection: Show compliance with a system requirement by visual verification of the software (e.g. verifying preparation for delivery, proper interfacing)
- Analysis: Perform detailed analysis of code, generated data (both intermediate data and final output data), etc., to determine compliance with system requirements.

The fields in the Requirements Verification Matrix in Section 4.3 are defined as follows:

- Requirements Tested Passed: Requirement was fully tested in a build test procedure and passed all tests.
- Requirements Tested Failed: Requirement was fully tested in a build test procedure and failed one or more aspect of the testing.
- Requirements Tested Partially: Requirement was tested partially in a build test procedure. To be fully tested, the partially tested requirement is either tested additionally in one or more other test procedures within the same build **and/or** other aspects of the requirement must be tested in a later build, due to capabilities not present in the current build
- Total Tested: Total number of requirements fully tested in a build test procedure. Includes total passed and total failed, but does **not** include requirements tested partially, **unless** (included as a separate entry) testing in multiple procedures within the same build constitutes total testing of a particular requirement. Total Requirements Tested is computed this way in order to avoid multiple counting of individual requirements that are tested partially in more than one procedure.
- Deferred: Number of requirements that were planned to be tested in current build, but were not tested due to some FSW capability or necessary system component not being present.
- Total: Total Requirements Tested + Number of Requirements Deferred

In each software test section in Section 5 there is a table of DCR's. The state definitions are as follows:

- Opened: The DCR is currently being addressed
- Assigned: The DCR was accepted and the modification is being addressed
- InTest: The DCR was corrected and is currently in test
- Validated: The DCR was corrected and tested and has been validated, needs to have a CCB to close the DCR
- Closed: The DCR is closed and have been resolved and tested to satisfaction
- Closed with Defect: The DCR is closed and the defect is most likely assigned a differed DCR number associated with another subsystem.

2 OVERVIEW

2.1 FLIGHT DATA SYSTEM CONTEXT

Figure 2-1 illustrates the cFS system context. The cFE interfaces to five external systems: an Operating System (OS), a Hardware Platform (HP), an Operational Interface (OI), Applications (APP), and other cFE-based systems.

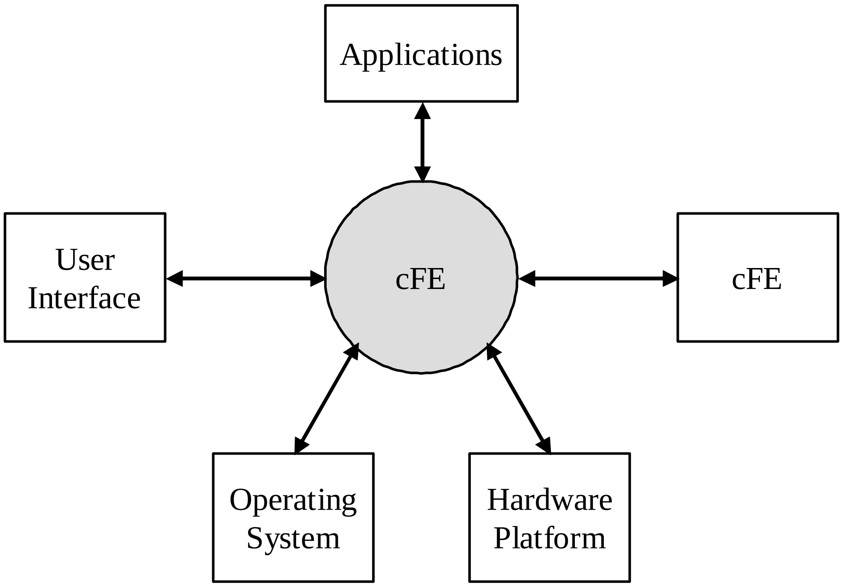

Figure 2-1 cFS System Context

The figure below shows major interfaces between the Stored Command application and other core Flight Executive (cFE) and Core Flight System (cFS) applications. Although it isn't shown explicitly, all task-to-task communications are accomplished via the cFE Software Bus (SB) application.

Inputs to the Stored Command application include: 1) Wake-up calls from the Scheduler (SCH) application which trigger processing, 2) Housekeeping requests from the Scheduler (SCH) application which trigger housekeeping data collection, 3) configuration commands from the Command Ingest (CI) application, and 4) updates to Stored Command Tables managed by the Table Services (TBL) application.

Outputs from the Stored Command application include: 1) Stored Command housekeeping messages sent to the Housekeeping (HK) application, 2) Commands sent to all applications for processing, and 3) Event messages.

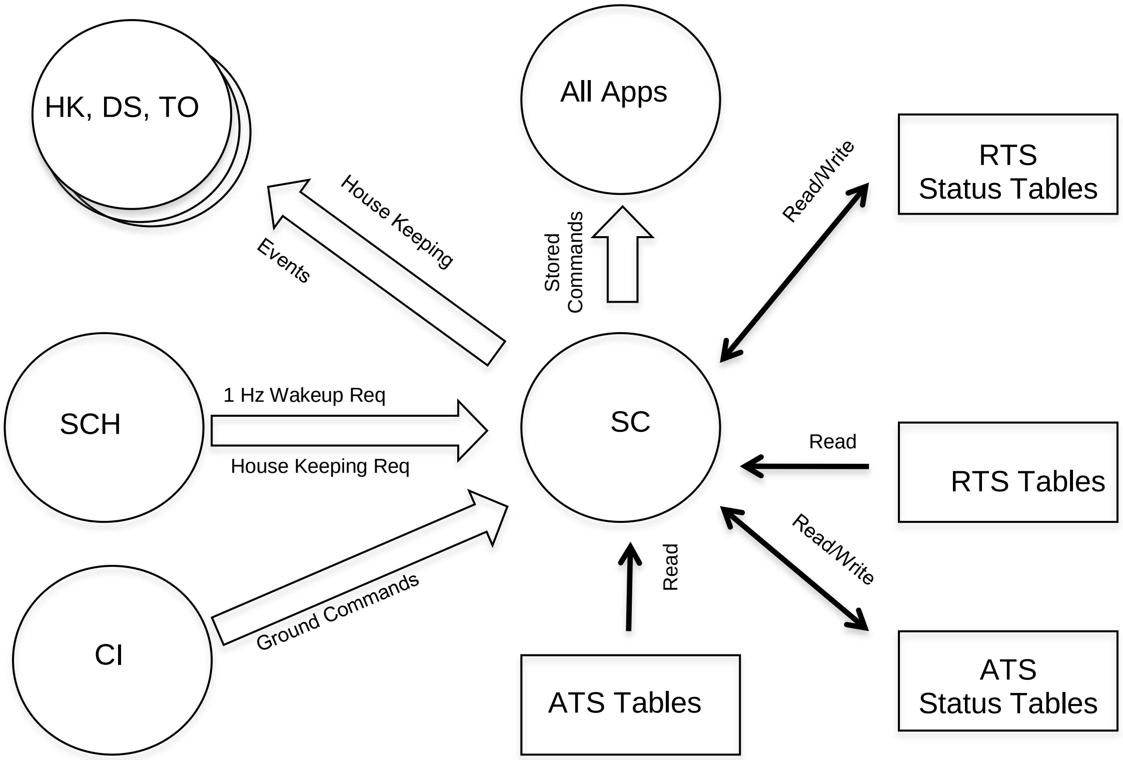

Figure 2-1 cFS SC Context

2.2 TEST HISTORY

SC 1.0.0.0 – Build Verification Testing completed 3/23/2009 by Walt Moleski
SC 2.0.0.0 – Build Verification Testing completed 8/31/2009 by Walt Moleski
SC 2.1.1.0 – Build Verification Testing completed 2/17/2011 by Walt Moleski
SC 2.2.0.0 - Build Verification Testing completed 9/8/2011 by Walt Moleski
SC 2.2.1.0 - Build Verification Testing completed 10/6/2011 by Walt Moleski
SC 2.3.0.0 - Build Verification Testing completed 1/11/2012 by Walt Moleski
SC 2.4.0.0 - Build Verification Testing completed 1/20/2015 by Walt Moleski
SC 2.5.0.0 - Build Verification Testing completed 10/28/2016 by Walt Moleski

2.3 TESTING OVERVIEW

The SC application was tested during Build Verification testing using the following:

- 1 test application: tst_sc
- 5 main test procedures: sc_atsfunc.prc, sc_gencmds.prc, sc_resetnocds.prc, sc_rtsfunc.prc, sc_stress.prc
- 11 test procedures that are called by the main procedures: sc_start_apps.prc, sc_700cmdats.prc, sc_appendfull.prc, sc_appoffend.prc, sc_atsoddbyte.prc, sc_atsoffend.prc, sc_maxcmdats.prc, sc_loadrts1.prc, sc_loadrts2.prc, sc_rtsoddbyte.prc and sc_rtsoffend.prc
- App tests require the Advanced Spacecraft Integration and System Test (ASIST) Ground Station

The TST_SC test application is used to send schedule requests for the output of SC's housekeeping data to the SC application. This was useful when performing build verification testing since it provided great control over the sequence of steps. In addition, having the test application eliminated the need to modify the SCH_LAB application and rebuild. When deployed for a mission, the Scheduler Application would provide this request. In addition, the test application has 5 ground commands defined to help with the SC testing. These commands are described below:

- TST_SC_NOOP
  - This command that issues an event and increments the command processed counter.
- TST_SC_ResetCtrs
  - This command resets the command processed and command error counters to zero (0).
- TST_SC_SetCounters
  - This command sets several Stored Command (SC) counters so that the SC_ResetCtrs command can be tested and verified.
- TST_SC_GetCRC
  - This command generates a CRC value for the supplied data and displays this value in the TST_SC housekeeping page for use by the test procedures.
- TST_SC_GetTime
  - This command retrieves the current requested time based upon the supplied type and displays this value in the TST_SC housekeeping page.

The SC 2.5.0.0 testing was performed using 2 different configurations. Each configuration required a separate compilation with changes to the PLATFORM_DEFINED configuration parameters. The SC 2.5.0.0 configurations are described below:

- Normal: SC compiled out of the box using the cFE default time (TAI).
- UTC Time: SC compiled with the SC_TIME_TO_USE parameter set to SC_USE_UTC.

The 5 main SC test procedures do the following:

Procedure	**Description**
sc_atsfunc	The purpose of this test is to verify that Absolute Time Sequences (ATS) execute and function properly.
sc_gencmds	The purpose of this test is to verify that the SC general commands execute and function properly.
sc_resetnocds	The purpose of this test is to verify that the SC application does not save any data across a reset (Application, Processor, or Power-On).
sc_rtsfunc	The purpose of this test is to verify that Relative Time Sequences (RTS) execute and function properly.
sc_stress	The purpose of this test is to verify that the SC application supports the execution of ATSs and RTSs simultaneously. Also, this test verifies several conditions regarding the maximum number of commands executing simultaneously and the priority of those commands.

The test procedures described in the table below are called by at least one of the test procedures above.

Procedure	**Description**
sc_700cmdats	This procedure creates an ATS table load image containing 700 valid commands for ATS B.
sc_appendfull	This procedure creates an ATS Append table load image containing a full table of valid commands.
sc_appoffend	This procedure creates an ATS Append table load image containing a full table of commands with the last command going over the end of the table buffer.
sc_atsoddbyte	This procedure creates an ATS table load image file containing two odd byte commands along with another command contained in the buffer. The

	command that follows each odd byte command must start on an even word boundary. This means that the buffer must pad to the next word.
sc_atsoffend	The purpose of this proc is to create an ATS table load file containing a full table of commands with the last command going over the end of the table buffer. Basically, the last command is incomplete.
sc_loadrts1	The purpose of this proc is to create a table load file for the first RTS table.
sc_loadrts2	The purpose of this proc is to create a table load file for the second RTS table.
sc_maxcmdats	The purpose of this proc is to create an ATS table load file containing one more command than the maximum number of commands allowed in an ATS.
sc_rtsoddbyte	This procedure creates a table load image file for RTS #3 containing two odd byte commands along with another command contained in the buffer. The command that follows each odd byte command must start on an even word boundary. This means that the buffer must pad to the next word.
sc_rtsoffend	The purpose of this proc is to create an RTS table load file containing a full table of commands with the last command going over the end of the table buffer. Basically, the last command is incomplete.
sc_start_apps	The purpose of this proc is to start the SC and TST_SC applications.

The cFS Deployment Guide contains the instruction for how to set up both the cFS Flight and Ground test environment. The testers use a cFS Test Account for each build test. This account runs ASIST and is setup to contain all the files needed to test the application. These files are extracted from MKS, the source repository tool. Included in these files are test utilities. These utilities can be located in 2 places depending upon whether they are “local” or “global” utilities. The local utilities are extracted into the working prc directory ($WORK/prc). The global utilities are pointed to by ASIST in the global area defined on the test system. Additional tools utilized by the test procedures are located in the $TOOLS directory. It is assumed that test procedures and the ASIST telemetry database used for testing is built using procedure and database templates

The following utilities were used during testing:

Name	**Description**
cfe_startup	Directive combines the "start_data_center", "open_tlm", and "open cmd <cpu>" ASIST startup commands.
close_data_center	Directive that closes the command and telemetry connection to the CPU being used.
create_tbl_file_from_cvt load_start_app	Procedure that creates a load file from the specified arguments and cvt Procedure to load and start a user application from the /s/opr/accounts/cfebx/apps/cpux directory.
load_table	Procedure that takes the specified file and transfers the file to the specified processor and then issues a TBL_LOAD command using the file.
tst_sc (version 2.4.0.0)	Test application required to test the SC application.
ut_pfindicate	Directive to print the pass fail status of a particular requirement number.
ut_runproc	Directive to formally run the procedure and capture the log file.
ut_sendcmd	Directive to send EVS commands Verifies command processed and command error counters.
ut_sendrawcmd	Send raw commands to the spacecraft. Verifies command processed and command error counters.
ut_setrequirements	A directive to set the status of the cFE requirements array.
ut_setupevents	Directive to look for multiple events and increment a value for each event to indicate receipt.
ut_tlmupdate	Procedure to wait for a specified telemetry point to update.
ut_tlmwait	Directive that waits for the specified telemetry condition to be met

2.4 VERSION INFORMATION

Item	Version
SC Requirements	1.5
SC Application	2.5.0.0
TST_SC Application	2.4.0.0
CFE	6.5.0.0
ASIST	20.2
VxWorks	6.9

3 BUILD VERIFICATION TEST PREPARATION

3.1 SCENERIO DEVELOPMENT

No new scenarios were developed for SC 2.5.0.0 Build Verification Test. All scenarios are stored on the MKS server, in cFS-Repository SC test-and-ground directory within the Scenarios subdirectory. It should be noted that as SC requirements and BVT procedures evolve these scenarios are not updated to reflect any changes made.

3.2 PROCEDURE DEVELOPMENT AND EXECUTION

This build test was completed by running 5 test procedures. All 5 test procedures were modified as a result of SC 2.5.0.0 changes and are checked in to MKS at the conclusion of build testing. All test procedures were written using the STOL scripting language. The naming convention for files created by the test procedures was: scx_cpu<#>_<procedure name>_GMT.<ext>.

3.3 TEST PRODUCTS

Four log files were generated for every procedure that was run. They are defined as follows:

- Logs with the .loge extension list all events sent by the flight software
- Logs with the .logr extension list all requirements that passed validation by demonstration
- Logs with the .logp extension lists all prints that are generated by the test procedure
- Logs with the .logf extension lists everything from the other logs along with the steps in the test procedure
- Logs with the .logs extension lists the SFDU information (if applicable) contained in the full log.

A test summary report is developed in MKS for each procedure by the tester after build testing is completed. All test products are maintained on MKS in the cFS-Repository SC test-and-ground directory.

The SC 2.5.0.0 test results contain 2 sets of test products for the sc_atsfunc and sc_rtsfunc tests. One set for each of the test configurations described in Section 2.3 above.

4 BUILD VERIFICATION TEST EXECUTION

4.1 TESTBED OVERVIEW

SC FSW testing took place in the cFS FSW Development and Test Facility. A high level view of the cFS FSW Test Bed is shown in Figure 4-1. This facility is located in GSFC Building 23, Room N410. This facility consists of two ASIST workstations running ASIST version 20.2 and three MPC750 CPU boards running VxWorks. CPU1 is primarily used for development testing while CPU2 and CPU3 are used for build verification testing.

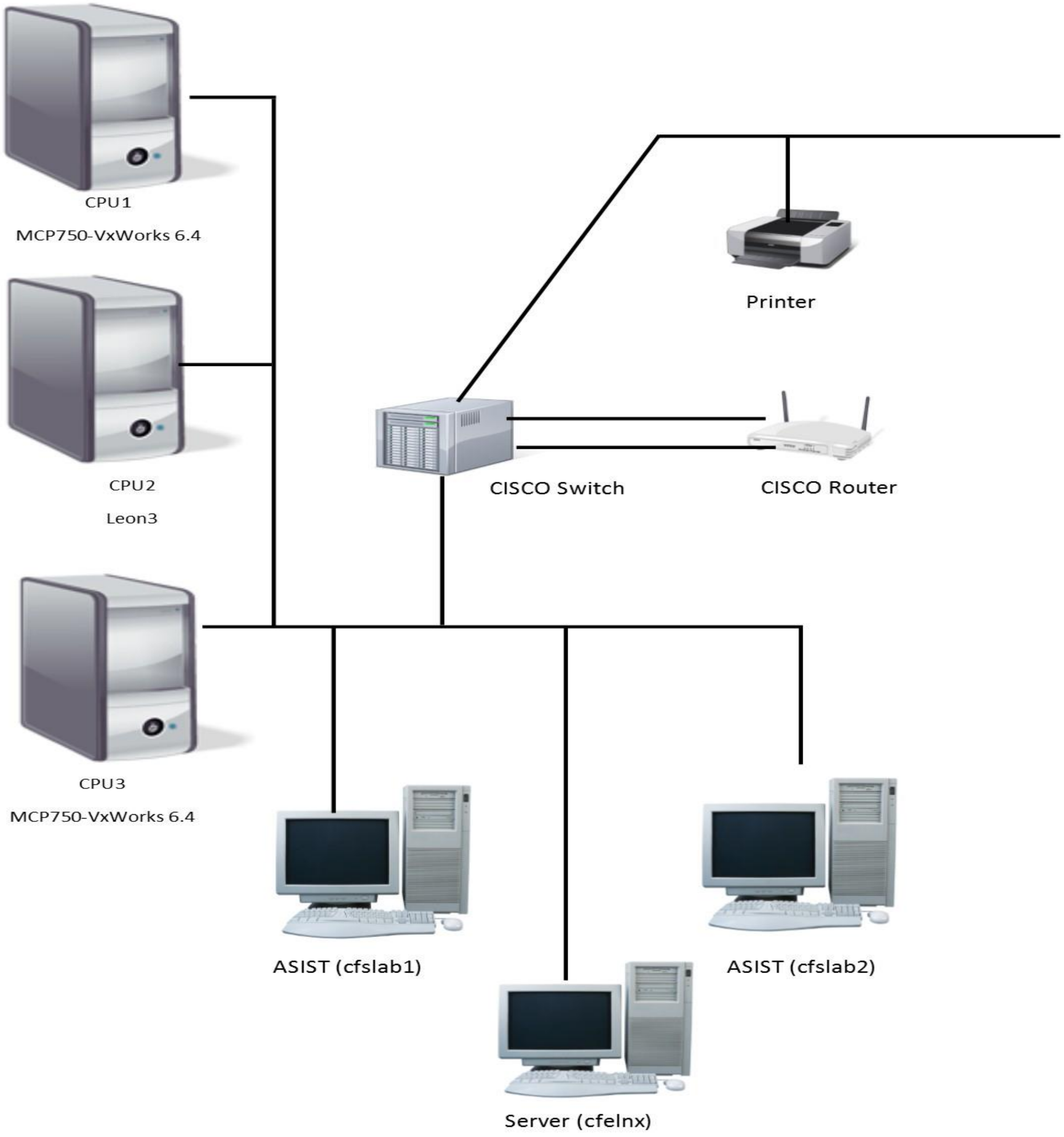

Figure 4-1 cFS FSW Development and Testing Facility

4.2 REQUIREMENTS VERIFICATION MATRIX

	Stored Command (SC)
Requirements Tested Passed	71
Requirements Tested Failed	0
Requirements Tested Partially	0
Total Tested	71
Deferred	0
Total	71

4.3 REQUIREMENTS PARTIALLY TESTED

No requirements were partially tested.

4.4 REQUIREMENTS/FUNCTIONALITY DEFERRED

No requirements/functionality was deferred.

4.5 REQUIREMENTS/FUNCTIONALITY DEFERRED FOR MISSION TESTING

No requirements/functionality was deferred to mission testing.

5 BUILD VERFICATON TEST RESULTS

5.1 OVERALL ASSESSMENT

During this build test of the SC Application the software behaved as expected with several problems still outstanding from previous testing.

Below is a summary of the results:

- 60 requirements passed via demonstration.
- 11 requirements were validated by analysis.
- 6 existing DCRs were verified.

5.2 PROCEDURE DESCRIPTION

Procedure	**Description**	**Requirements tested**
sc_atsfunc	The purpose of this test is to verify that Absolute Time Sequences (ATS) execute and function properly.	SC1002, SC1004, SC1005, SC2000, SC2000.1, SC2000.2, SC2000.3, SC2000.4, SC2001, SC2004, SC2007, SC2007.1, SC2008, SC2008.1, SC2008.2, SC2008.3, SC2008.4, SC2008.5, SC2008.6, SC2008.7, SC2009, SC3000, SC3000.1, SC3000.3, SC3000.3.1, SC3000.3.2, SC3001, SC3001.1, SC3002, SC3002.1, SC3002.2, SC3002.3, SC3002.4, SC3003, SC3003.1, SC3003.2, SC3003.2.1, SC3003.2.2, SC3003.3, SC3003.4, SC3004, SC3005, SC8000, SC9000
sc_gencmds	The purpose of this test is to verify that the SC general commands execute and function properly.	SC1000, SC1001, SC1002, SC1004, SC1005, SC8000, SC9000
sc_resetnocds	The purpose of this test is to verify that the SC application does not save any data across a reset (Application, Processor, or Power-On). This test should NOT be executed if the configuration parameter indicating Save Critical Data is set by the Mission.	SC1004, SC2000.2, SC2000.3, SC2001, SC2002.2, SC2002.3, SC2003, SC3000, SC4000, SC4000.1, SC4004, SC8000, SC9000, SC9004, SC9005
sc_rtsfunc	The purpose of this test is to verify that Relative Time Sequences (RTS) execute and function properly.	SC1002, SC1004, SC1005, SC2002, SC2002.1, SC2002.2, SC2002.3, SC2003, SC2005, SC2006, SC4000, SC4000.1, SC4000.2, SC4001, SC4001.1, SC4001.2, SC4001.3, SC4001.3.1, SC4001.4, SC4002, SC4003, SC4004, SC4005, SC4005.1, SC8000, SC9000. SC9004

Procedure	Description	Requirements tested
sc_stress	The purpose of this test is to verify that the SC application supports the execution of ATSs and RTSs simultaneously. Also, this test verifies several conditions regarding the maximum number of commands executing simultaneously and the priority of those commands.	SC1004, SC2000.2, SC2000.3, SC2001, SC2002, SC2002.2, SC2002.3, SC2003, SC2005, SC2005.1, SC2005.2, SC3000, SC3001, SC4000, SC4000.1, SC4003, SC4004, SC8000, SC9000, SC9004

5.3 ANALYSIS REQUIREMENTS VERIFICATION

There were 11 requirements verified using analysis.

Requirement	Requirement Text	Analysis
SC2000.1	ATS commands were executed every second with the proper delay as specified in the loaded table	Step 2.6 of the atsfunc test procedure verifies this requirement. The ATS commands executed with the proper delay between them as specified in the table load file. The load file contained a 5 second delay between the commands.
SC2000.4	This requirement was tested using normal time (TAI) and UTC time. For the test, the LeapSeconds were jammed to be 5 less than the current value. In normal time, there was no affect. In UTC time, 6 commands executed in the same second.	Steps 5.4 thru 5.6 of the atsfunc test procedure verify this requirement. When TAI (CFE_DEFAULT) time was used, a jam of the LeapSeconds did not have any effect on the ATS execution. When using UTC time, there were 6 commands that executed in the same second when the LeapSeconds was jammed from 32 to 27 seconds.
SC2002.1	RTS commands executed every second as specified in the loaded table.	Step 2.7 of the rtsfunc test procedure verifies this requirement. The RTS commands executed every 1 second as specified in the loaded table.
SC2004	ATS commands were executed in the proper “time” order rather than the order loaded in the table.	Step 2.6 of the atsfunc test procedure verifies this requirement. The ATS commands were executed in the proper "time" order rather than the order loaded in the table. The table load contained command 2 (SC_NOOP), command 1 (TBL_NOOP), and command 3 (EVS_NOOP).

Requirement	Requirement Text	Analysis
SC2005	The maximum commands per second were executed as specified by the configuration parameter.	Step 2.19 of the rtsfunc and 2.17 of the stress test procedures verify this requirement. RTS #3 contained 6 commands (3 SC_NOOP and 3 ES_NOOP) that execute each second for 2 consecutive seconds. RTS #8 contained 4 commands (TBL_NOOP) that executed each second for 3 seconds. The test log shows that the max number of commands (8) executed in any one second.
SC2005.1	ATS commands were executed before RTS commands and the RTS commands were properly deferred.	Step 2.17 of the stress test procedure verifies this requirement. ATS B was loaded with the maximum commands per second (8) and executed for 6 seconds. The first ATS command started RTS #5 which contained 3 commands (TO_NOOP, CI_NOOP and TST_SC_NOOP) that executed for 6 consecutive seconds. The ATS commands executed first followed by the deferred RTS commands until both sequences completed.
SC2006	The higher priority RTSs commands were executed first followed by the lower priority RTSs commands.	Step 2.19 of the rtsfunc test procedure verifies this requirement. RTS #3 contained 6 commands (3 SC_NOOP and 3 ES_NOOP) that execute each second for 2 consecutive seconds. RTS #8 contained 4 commands (TBL_NOOP) that executed each second for 3 seconds. The test log shows that all RTS #3 commands were executed in each second along with 2 of RTS #8 commands. The remaining RTS #8 commands executed when RTS #3 completed.

Requirement	Requirement Text	Analysis
SC3002.4	The Switch command was executed immediately after it was dispatched.	Step 4.27 of the atsfunc test procedure waits for the switch command to execute. However, in this test run, the Switch command executed prior to getting to this step. The ATS purposely contained a switch command to an empty ATS in order to generate an error event. The error event message was captured which indicated that the switch command executed immediately since it was contained in an ATS.
SC4001	RTS commands were dispatched in the order they were loaded in the table.	Step 2.6 and 2.7 of the rtsfunc test procedure verify this requirement. Three commands were loaded into RTS #2. The command order in the table was SC_NOOP, TBL_NOOP and EVS_NOOP. The RTS commands were dispatched in the correct order.
SC4001.1	The delay was interpreted properly.	Step 2.6 and 2.7 of the rtsfunc test procedure verify this requirement. The delay prior to the first command was 5 seconds, followed by a 1 second delay between the other commands. The test log shows these delays.
SC4001.2	The delay for the first command was relative to the Start RTS command.	Step 2.6 and 2.7 of the rtsfunc test procedure verify this requirement. The first command in the RTS table was to delay for 5 seconds. The log file of the test shows that the first command executed approximately 5 seconds after the RTS was started.

5.4 DCRS

No new DCRs were generated during SC 2.5.0.0 testing.

5.4.1 DCRs Verified

The following DCRs were verified during testing:

DCR	Description	Test Method	Test Approach
4243	SC – increments ground command error counter when internal command to report housekeeping has invalid packet length	Demonstration	Step 2.7 of the sc_gencmds test procedure sends an invalid length housekeeping command and verifies that the CMDEC does not increment.
4245	Incorrect message on ATS abort	Inspection	The incorrect message did not get generated. This fix was included with DCR 6323.
4246	Incomplete Event Message is SC	Inspection	The incomplete message did not get generated due to the fact that the underlying SB error could not be simulated. This fix was included with DCR 6323
6323	Several minor improvements/fixes needed in SC	Inspection	The stated changes were verified by inspecting the code submitted with this DCR.
145721	SC - Integrate Babelfish Ticket Fixes	Demonstration	No compiler errors/warnings were generated during the make process.
145832	SC: CFE_EVS_SendEvent Format Warnings	Demonstration	No compiler warnings were generated during the make process.

5.4.2 Outstanding DCRs

DCR	Description	State
3854	ATS and RTS status information is not saved across a reset.	Submitted
4092	Startup time with a large number of RTS tables can be extensive. The startup time on MMS with 320 RTS tables is nearly 2 minutes.	On Hold
4120	SC - Add Trick Simulation Support (JSC Request)	Submitted
4139	SC does not allow duplicate command numbers in an ATS.	Submitted
4161	Add ATS filename to housekeeping telemetry	Submitted

5.5 NOTES

It should be noted that integration testing is the ultimate verification of the SC applications performance in a system-like scenario.

APPENDIX A - RTTM

The SC Build 2.5.0.0 RTTM can be found on the MKS server, in cFS-Repository SC test-and-ground directory results folder.

APPENDIX B - COMMAND, TELEMETRY, AND EVENTS VERIFICATION MATRIX

Command	**Test Procedure(s)**	**Notes/Comments**
SC_NOOP	sc_gencmds	
SC_ResetCtrs	sc_gencmds	
SC_StartATS	sc_atsfunc, sc_resetcds, sc_resetnocds, sc_stress	
SC_StopATS	sc_atsfunc, sc_stress	
SC_StartRTS	sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_StopRTS	sc_rtsfunc, sc_stress	
SC_DisableRTS	sc_rtsfunc	
SC_EnableRTS	sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_SwitchATS	sc_atsfunc	
SC_JumpATS	sc_atsfunc	
SC_ContinueATS	sc_atsfunc	
SC_AppendATS	sc_atsfunc	
SC_StartRTSGroup	sc_rtsfunc	Controlled via configuration parameter
SC_StopRTSGroup	sc_rtsfunc	Controlled via configuration parameter
SC_DisableRTSGroup	sc_rtsfunc	Controlled via configuration parameter
SC_EnableRTSGroup	sc_rtsfunc	Controlled via configuration parameter

Telemetry	**Test Procedure(s)**	**Notes/Comments**
SC_ATSNumber	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_ATPState	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_ContATSFlag	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_ATPCmdNumber	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	

SC_SwitchPend	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_ActiveRTSs	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_NextRTS	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_CMDEC	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_CMDPC	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_RTSActvCtr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_RTSActvErr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_ATSCmdCtr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_ATSErrCtr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_RTSCmdCtr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_RTSErrCtr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	

SC_LastATSErr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_LastATSCmdErr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_LastRTSErr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_LastRTSCmdErr	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_AppendATSID	atsfunc	
SC_AppendCount	atsfunc	
SC_AppendSize	atsfunc	
SC_AppendLoads	atsfunc	
SC_FreeBytes[2]	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_NextRTSTime	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_NextATSTime	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_RTSExeStatus[RTS_Tables/16]	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
SC_RTSDisableStatus[RTS_Tables/16]	sc_atsfunc, sc_gencmds, sc_resetcds, sc_resetnocds, sc_rtsfunc, sc_stress	
Table Telemetry		
SC_ATP_State		
SC_ATSNum		
SC_CmdNum		
SC_TimIdx		
SC_SwitchFlag		

SC_ATSCMD_Status[SC_MAX_ATS_CMDS]	sc_atsfunc	
SC_ATSInfo_Table[2].UsageCtr		
SC_ATSInfo_Table[2].CmdCtr		
SC_ATSInfo_Table[2].ATSSize		
SC_ATS_DATA[SC_ATS_BUFF_SIZE]	sc_atsfunc, sc_resetcds, sc_resetnocds, sc_stress	
SC_RTP_ActiveCtr		
SC_RTP_NextRTS		
SC_RTSINFO_Table[RTS_Tables].Status		
SC_RTSINFO_Table[RTS_Tables].DisabledFlag		
SC_RTSINFO_Table[RTS_Tables].CmdCtr		
SC_RTSINFO_Table[RTS_Tables].CmdErrCtr		
SC_RTSINFO_Table[RTS_Tables].NextCmdTime		
SC_RTSINFO_Table[RTS_Tables].NextCmd		
SC_RTSINFO_Table[RTS_Tables].UsageCtr		
SC_RTS_Data[SC_RTS_BUFF_SIZE]	sc_resetcds, sc_resetnocds, sc_rtsfunc , sc_stress	

Id	**Event Message**	**Test Procedure(s)**	**Notes/Comments**
1	SC_APP_EXIT_ERR_EID		
2	SC_LEN_ERR_EID	sc_atsfunc, sc_gencmds, sc_rtsfunc	
3	SC_INIT_SB_CREATE_ERR_EID		
4	SC_INIT_SB_SUBSCRIBE_HK_ERR_EID		
5	SC_INIT_SB_SUBSCRIBE_1HZ_ERR_EID		
6	SC_INIT_SB_SUBSCRIBE_CMD_ERR_EID		
9	SC_INIT_INF_EID	sc_atsfunc, sc_gencmds, sc_resetnocds, sc_rtsfunc, sc_stress	
10	SC_REGISTER_RTS_TBL_NO_CDS_ERR_EID		
11	SC_REGISTER_ATS_TBL_NO_CDS_ERR_EID		
16	SC_REGISTER_RTS_INFO_TABLE_ERR_EID		
17	SC_REGISTER_RTS_CTRL_BLK_TABLE_ERR_EID		
18	SC_REGISTER_ATS_INFO_TABLE_ERR_EID		
19	SC_REGISTER_ATS_CTRL_BLK_TABLE_ERR_EID		
20	SC_REGISTER_ATS_CMD_STATUS_TABLE_ERR_EID		
21	SC_RTS_LOAD_COUNT_INFO_EID	sc_atsfunc, sc_gencmds, sc_resetnocds, sc_rtsfunc, sc_stress	
23	SC_STARTATS_CMD_INF_EID	sc_atsfunc, sc_resetnocds, sc_stress	
24	SC_STARTATS_CMD_NOT_LDED_ERR_EID		

25	SC_STARTATS_CMD_NOT_IDLE_ERR_EID	sc_atsfunc	
26	SC_STARTATS_CMD_INVLD_ID_ERR_EID	sc_atsfunc	
27	SC_STOPATS_CMD_INF_EID	sc_atsfunc, sc_stress	
28	SC_STOPATS_NO_ATS_INF_EID		
29	SC_ATS_SKP_ALL_ERR_EID		
30	SC_ATS_ERR_SKP_DBG_EID	sc_atsfunc, sc_resetnocds, sc_stress	
31	SC_SWITCH_ATS_CMD_INF_EID	sc_atsfunc	
32	SC_SWITCH_ATS_CMD_NOT_LDED_ERR_EID		
33	SC_SWITCH_ATS_CMD_IDLE_ERR_EID	sc_atsfunc	
34	SC_ATS_SERVICE_SWTCH_INF_EID	sc_atsfunc	
35	SC_SERVICE_SWITCH_ATS_CMD_LDED_ERR_EID		
36	SC_ATS_SERVICE_SWITCH_IDLE_ERR_EID		
37	SC_ATS_INLINE_SWTCH_INF_EID		
38	SC_ATS_INLINE_SWTCH_NOT_LDED_ERR_EID	sc_atsfunc	
39	SC_JUMPATS_CMD_STOPPED_ERR_EID	sc_atsfunc	
40	SC_JUMP_ATS_INF_EID	sc_atsfunc	
41	SC_JUMPATS_CMD_NOT_ACT_ERR_EID	sc_atsfunc	
42	SC_CONT_CMD_ERR_EID		
43	SC_CONT_CMD_DEB_EID	sc_atsfunc	
44	SC_ATS_CHKSUM_ERR_EID	sc_atsfunc	
45	SC_ATS_ABT_ERR_EID	sc_atsfunc	
46	SC_ATS_DIST_ERR_EID		
47	SC_ATS_MSMTCH_ERR_EID		
48	SC_ATS_SKP_ERR_EID		
49	SC_RTS_DIST_ERR_EID		
50	SC_RTS_CHKSUM_ERR_EID	sc_rtsfunc	
51	SC_RESET_DEB_EID	sc_gencmds	
52	SC_NOOP_INF_EID	sc_atsfunc, sc_gencmds, sc_resetnocds, sc_rtsfunc, sc_stress	
59	SC_RTS_INVLD_MID_ERR_EID		
60	SC_RTS_LEN_ERR_EID	sc_rtsfunc	
61	SC_RTS_LEN_BUFFER_ERR_EID	sc_rtsfunc	
62	SC_RTS_LEN_TOO_LONG_ERR_EID		
63	SC_MID_ERR_EID		
64	SC_INVLD_CMD_ERR_EID	sc_gencmds	
65	SC_GET_ADDRESS_RTS_INFO_ERR_EID		
66	SC_GET_ADDRESS_RTS_CTRL_BLCK_ERR_EID		
67	SC_GET_ADDRESS_ATS_INFO_ERR_EID		

68	SC_GET_ADDRESS_ATS_CTRL_BLCK_ERR_EID		
69	SC_GET_ADDRESS_ATS_CMD_STAT_ERR_EID		
70	SC_GET_ADDRESS_RTS_ERR_EID		
71	SC_GET_ADDRESS_ATS_ERR_EID		
72	SC_STARTRTS_CMD_DBG_EID	sc_stress	
73	SC_RTS_START_INF_EID	sc_atsfunc, sc_gencmds, sc_resetnocds, sc_rtsfunc, sc_stress	
74	SC_STARTRTS_CMD_INVLD_LEN_ERR_EID	sc_rtsfunc	
75	SC_STARTRTS_CMD_NOT_LDED_ERR_EID	sc_rtsfunc	
76	SC_STARTRTS_CMD_DISABLED_ERR_EID	sc_resetnocds, sc_rtsfunc	
77	SC_STARTRTS_CMD_INVALID_ERR_EID	sc_rtsfunc	
78	SC_STOPRTS_CMD_INF_EID	sc_rtsfunc, sc_stress	
79	SC_STOPRTS_CMD_ERR_EID	sc_rtsfunc	
80	SC_DISABLE_RTS_DEB_EID	sc_rtsfunc	
81	SC_DISRTS_CMD_ERR_EID	sc_rtsfunc	
82	SC_ENABLE_RTS_DEB_EID	sc_resetnocds, sc_rtsfunc, sc_stress	
83	SC_ENARTS_CMD_ERR_EID	sc_rtsfunc	
84	SC_RTS_LNGTH_ERR_EID		
85	SC_RTS_CMD_LNGTH_ERR_EID		
86	SC_RTS_COMPL_INF_EID	sc_atsfunc, sc_gencmds, sc_resetnocds, sc_rtsfunc, sc_stress	
87	SC_ATS_COMPL_INF_EID	sc_atsfunc, sc_stress	
88	SC_JUMP_ATS_SKIPPED_DBG_EID	sc_atsfunc	
90	SC_REGISTER_APPEND_INFO_TABLE_ERR_EID		
91	SC_GET_ADDRESS_APPEND_INFO_ERR_EID		
92	SC_GET_ADDRESS_APPEND_ERR_EID		
93	SC_REGISTER_APPEND_TBL_NO_CDS_ERR_EID		
97	SC_UPDATE_APPEND_EID	sc_atsfunc	
98	SC_APPEND_CMD_INF_EID	sc_atsfunc	
99	SC_APPEND_CMD_ARG_ERR_EID	sc_atsfunc	
100	SC_APPEND_CMD_TGT_ERR_EID	sc_atsfunc	
101	SC_APPEND_CMD_SRC_ERR_EID	sc_atsfunc	
102	SC_APPEND_CMD_FIT_ERR_EID	sc_atsfunc	
103	SC_VERIFY_ATS_EID	sc_atsfunc, sc_resetnocds, sc_stress	
104	SC_VERIFY_ATS_NUM_ERR_EID	sc_atsfunc	
105	SC_VERIFY_ATS_END_ERR_EID		
106	SC_VERIFY_ATS_PKT_ERR_EID	sc_atsfunc	
107	SC_VERIFY_ATS_BUF_ERR_EID	sc_atsfunc	
109	SC_VERIFY_ATS_DUP_ERR_EID	sc_atsfunc	

110	SC_VERIFY_ATS_MPT_ERR_EID	sc_atsfunc	
111	SC_TABLE_MANAGE_ID_ERR_EID		
112	SC_TABLE_MANAGE_RTS_ERR_EID		
113	SC_TABLE_MANAGE_ATS_ERR_EID		
114	SC_TABLE_MANAGE_APPEND_ERR_EID		
115	SC_STARTRTSGRP_CMD_INF_EID	sc_rtsfunc	
116	SC_STARTRTSGRP_CMD_ERR_EID	sc_rtsfunc	
117	SC_STOPRTSGRP_CMD_INF_EID	sc_rtsfunc	
118	SC_STOPRTSGRP_CMD_ERR_EID	sc_rtsfunc	
119	SC_DISRTSGRP_CMD_INF_EID	sc_rtsfunc	
120	SC_DISRTSGRP_CMD_ERR_EID	sc_rtsfunc	
121	SC_ENARTSGRP_CMD_INF_EID	sc_rtsfunc	
122	SC_ENARTSGRP_CMD_ERR_EID	sc_rtsfunc	