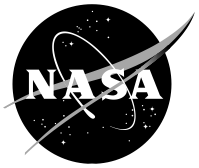
Software Test Plan and Procedures
for the
Core Flight System Test Framework Tool

Engineering Directorate
Software, Robotics and Simulation Division

Availability:
NASA & NASA contractor employees as required

March 2022
Baseline



National Aeronautics and
Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Verify this is the correct version before use

Change Record

Revision	*Date*	*Originator*	*Description*
N/A	Nov 2021	Tam Ngo	Initial draft
N/A	Mar 2022	Tam Ngo	Baseline

Table of Contents

1 INTRODUCTION 5
1.1 PURPOSE AND SCOPE 5
1.2 RESPONSIBILITY AND CHANGE AUTHORITY 5
2 APPLICABLE AND REFERENCE DOCUMENTS 5
2.1 APPLICABLE DOCUMENTS 5
2.2 REFERENCE DOCUMENTS 6
2.3 ORDER OF PRECEDENCE 6
3 TEST PLAN 6
3.1 TESTING ACTIVITIES 6
3.2 REQUIREMENT TRACEABILITY 6
4 VERIFICATION AND VALIDATION PROCESS 7
4.1 VERIFICATION AND VALIDATION MANAGEMENT RESPONSIBILITIES 7
4.2 VERIFICATION METHODS 7
4.2.1 Analysis 7
4.2.2 Inspection 7
4.2.3 Demonstration 8
4.2.4 Test 8
4.3 VALIDATION METHODS 8
4.4 CERTIFICATION PROCESS 9
4.5 ACCEPTANCE TESTING 9
5 CTF TEST PROCEDURES 9
5.1 ASSUMPTIONS, DEPENDENCIES AND CONSTRAINTS 9
5.1.1 Assumptions 9
5.1.2 Dependencies 9
5.1.3 Constraints 9
5.2 STATIC CODE ANALYSIS 9
5.2.1 Building the CTF static code analysis 9
5.2.2 Running the CTF static code analysis 9
5.2.3 Verifying the CTF static code analysis report against the provided analysis report 10
5.3 UNIT TESTING AND CODE COVERAGE 10
5.3.1 Building the CTF unit tests and code coverage 10
5.3.2 Running the CTF unit tests and code coverage 10
5.3.3 Verifying the CTF unit test results against the provided test results 10
5.4 FUNCTIONAL TESTING 11
5.4.1 Building the CTF functional tests 11
5.4.2 Running the CTF functional tests 11
5.4.3 Verifying the CTF functional test results against the provided test results 11
5.5 VERIFICATION TESTING 11
5.5.1 Setting up the test rig 11
5.5.2 Building CTF verification test 11
5.5.3 Running the CTF verification tests 11
5.5.4 Verifying the CTF verification test results against the provided test results 12
6 CTF CERTIFICATION ARTIFACTS 12

Verify this is the correct version before use

7 APPENDICES 13
7.1 ABBREVIATIONS AND ACRONYMS 13
7.2 DEFINITION OF TERMS 14
8 NOTES 16

List of Tables

Table 2-1: Applicable Documents 6
Table 2-2: Reference Documents 6
Table 3-1: Traceability Mapping 7
Table 5-1: 3rd-party Software Packages Used by CTF 9
Table 6-1: Certification Artifacts 12

1 INTRODUCTION

1.1 Purpose and Scope

This document defines the test plan and procedures for the "certifiable" version of the Core Flight System Test Framework (CTF) Tool. The intent of the "certifiable" version of the software is to provide the users with the necessary certification artifacts (i.e., documentations, test procedures, test suites and the expected test results) to perform a formal certification (i.e., the "run-for-record") of the CTF on a specific platform.

For clarification, "certified" software and "certifiable" software are defined as follow:

- "Certified" software would have already been verified and validated on a target platform with the documented specifications and configurations for the followings:
    - Processor - model, version, etc.
    - Operating system - vendor, version, kernel configurations & image, etc.
    - Hardware drivers - vendor, version, configurations, et.
    - Software configurations - per component, per build, per execution with all the required data files, etc.
    - Other factors as required by the mission/project

  Note: Any changes in the above after the official "run-for-record", regardless of the criticality of the changes, can invalidate the certification claim if certification is not performed again with the new changes.
- "Certifiable" software can become "certified" by using the provided certification artifacts to perform the official "run-for-record" with the user-specified configurations on the user-selected platform. Unlike "certified" software, "certifiable" software is not associated with any specific certification platform, configurations or builds, other than those that are used in development and testing of the software.

The development of this "certifiable" software product followed the NPR 7150.2 requirements for a class C, non-safety-critical software product, as documented in the cFS Certification Software Development Plan. Hence, the certification artifacts that come with a "certifiable" version of this product could be used to formally certify it as class C, non-safety-critical software, as defined by the NPR 7150.2.

1.2 Responsibility and Change Authority

This document is prepared in accordance with EA-WI-025, "GFE Flight Project Software and Firmware Development". The responsibility for the development of this document lies with the Engineering Directorate Software, Robotics, & Simulation Division (SR&SD), Spacecraft Software Engineering Branch/ER6. Change authority is the Software, Robotics and Simulation Division of the Johnson Space Center.

2 APPLICABLE AND REFERENCE DOCUMENTS

2.1 Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this VDD to the extent specified herein.

Table 2-1: Applicable Documents

Document Number	*Document Title*	*Revision / Release Date*
NPR 7150.2	NASA Software Engineering Procedural Requirements	Rev C / Aug 2019
EA-WI-025	GFE Flight Project Software and Firmware Development	Rev D / Sep 2013
GP-10021	cFS Certification Software Development Plan	Baseline / May 2020

2.2 Reference Documents

The following documents are reference documents utilized in the development of this VDD. These documents do not form a part of this VDD and are not controlled by their reference herein.

Table 2-2: Reference Documents

Document Number	*Document Title*	*Revision / Release Date*
N/A	CTF Software Requirements Specification	Baseline / Mar 2022
N/A	CTF Software Design Document	Baseline / Mar 2022
N/A	CTF Software User's Guide	Baseline / Mar 2022

2.3 Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence.

3 TEST PLAN

CTF development and testing were done on a PC platform running CentOS 7.0. Regression testing was done with Gitlab Continuous Integration. All tests were built with the default configurations defined for the CTF.

3.1 Testing Activities

The following activities were performed against the code base of the CTF tool:

a. Static code analysis
b. Peer code inspection via peer reviews
c. Code coverage unit testing
d. Requirement verification testing

All items above have been perform by the CTF development team. Any of them can be repeated by the users using the instructions described in the subsequent sections.

3.2 Requirement Traceability

Table 3-1 lists the traceability between requirements, design and test cases.

Table 3-1: Traceability Mapping

Traceability Direction	*Traceability Matrix*
Requirements to Design Elements	CTF_RTM_r2d
Design Elements to Requirements	CTF_RTM_d2r
Requirements to Test Cases	CTF_RTM_r2t
Test Cases to Requirements	CTF_RTM_t2r

4 VERIFICATION AND VALIDATION PROCESS

4.1 Verification and Validation Management Responsibilities

The CTF development team is responsible for certifying the tool's capabilities as class C, non-safety critical software. The users are responsible for the development and testing of the CTF test scripts used to verify and validate their system requirements.

4.2 Verification Methods

Each software requirement is assigned a method for verification in the requirement verification matrix of this document. Four methods are used to satisfy verification requirements: inspection, analysis, demonstration, and test, or a combination of these. As the specification is developed, each requirement is analyzed and assigned a method that is documented in the verification matrix. This matrix thus provides the foundation for all further verification planning for that item. The assigned method may prove invalid as the design matures. In this case, the verification method will be updated in the verification matrix through normal document control channels, with the appropriate approvals.

4.2.1 Analysis

Analysis is a verification method utilizing techniques and tools such as math models, prior test data, simulations, analytical assessments, etc. Verification by similarity is acceptable if the subject article is similar or identical in design, manufacturing process, and quality control to another article that has been previously verified to an equivalent or more stringent criteria.

Examples of software verification by analysis include analyzing the test data or results post test execution.

4.2.2 Inspection

Inspection is a method of verification of software code or hardware physical characteristics that determine compliance without the use of special laboratory equipment, procedures, test support items, or services. Inspection often uses visual methods to verify compliance with design requirements. A record of the inspection (TPS, weld inspection record, etc.) is required.

Examples of software verification by inspection include verification of expected events from the test output during execution or from test logs post execution.

4.2.3 Demonstration

Demonstration is a qualitative method of verification that evaluates the properties of the subject end item. Demonstration is used with or without special test equipment or instrumentation to verify required characteristics such as operational performance, human engineering features, service and access features, transportability, and displayed data. A formal record of demonstration is required.

One example of software verification by demonstration is verifying displays are legible when viewed through a helmet visor.

4.2.4 Test

Test is a method of verification wherein formal project hardware, software and firmware requirements (performance, environment, etc.) are verified by measurement or functional test during or after the controlled application of functional and/or environmental stimuli. These measurements may require the use of laboratory equipment, recorded data, procedures, test support items, or services.

Examples include vibration testing and thermal/vacuum testing. Different types of testing that may be included in this section are qualification, acceptance, system integration, program integration, major ground testing, orbital flight demonstrations, prelaunch checkout, and on-orbit checkout.

Note that when the results of any testing are intended for use in the formal verification of the project's products (i.e., verification of requirements listed in Appendix C), the test must be performed on controlled hardware/software, using only controlled and approved GSE, and must be documented (Task Performance Sheets, Discrepancy Reports).

4.2.4.1 Qualification Testing

Qualification testing is performed on units that are identical to the flight articles but are not intended for flight, i.e., a qualification unit. The purpose of qualification tests is to ensure the design of the project's deliverables meet the environment (thermal, pressure, radiation, vibration, etc.) requirements imposed on the deliverable. These tests may exceed the expected induced environment levels. Qualification testing proves that an end item's design is adequate to meet the environment specification requirements. This testing will include functional tests before and after exposure to the test environment to determine the success or failure of the test. Depending on the project requirements, this may also include functional and performance tests being conducted during the environment tests.

Note that qualification testing of cFS products shall be performed by the project in the context of the overall software system that the project is responsible for.

4.3 Validation Methods

Validation is performed to ensure that, regardless of any specific set of requirements, the customer/sponsor will be satisfied with the product provided. Validation ensures the purpose of a system is not lost in the specification of the detailed requirements. Validation provides a means to confirm with the customer/sponsor to determine whether the product will meet their expectations. When performed early in the project, validation prevents wasted time and effort resulting from misunderstandings and wrong assumptions.

The same four methods can be used to perform validation: inspection, analysis, demonstration, and test, or a combination of these. The validation activities are negotiated with the customer/sponsor and are documented in a validation matrix.

Note that validation of the cFS products shall be performed by the project in the context of the overall software system that the project is responsible for.

4.4 Certification Process

The certification of cFS products shall be performed by the users in accordance with their software certification process.

4.5 Acceptance Testing

Acceptance testing of cFS products shall be performed by the users in accordance with their software acceptance process.

5 CTF TEST PROCEDURES

5.1 Assumptions, Dependencies and Constraints

5.1.1 Assumptions

None.

5.1.2 Dependencies

The CTF tool uses the 3rd-party software packages as listed in **Table 5-1** below.

Table 5-1: 3rd-party Software Packages Used by CTF

5.1.3 Constraints

None.

5.2 Static code analysis

The static code analyzer tool, ***pylint***, was used to perform static code analysis on the CTF code base against the PEP 8 Style Guide for Python code. Any deviation from the coding standard is documented in the configuration file, ***.pylintrc***, which is also included in the release. Since the CTF source code is available, the users can opt to re-run static code analysis using the users' preferred analysis tool and coding standards.

5.2.1 Building the CTF static code analysis

N/A.

5.2.2 Running the CTF static code analysis

The execution steps are captured in a shell script for ease of execution and maintenance.

1. Ensure that the CTF environment is activated, i.e., the (**pythonEnv3**) prompt should be shown in terminal.

2. Execute the following command from the CTF top-level directory, where the script resides,

```
$  ./run_tests.sh sca
```

Note that to display the script usage, the users can execute ***run_tests.sh*** script without command line argument.

5.2.3 Verifying the CTF static code analysis report against the provided analysis report

The generated analysis report, ***runtests_output/sca/ctf_sca.log***, can be verified against the provided analysis report.

See Table 6-1 below for the file name and location of the provided static code analysis report.

5.3 Unit testing and code coverage

The CTF code base comes with source code for unit testing that can be built and run on a Linux platform. All external interfaces are stubbed so that unit tests can be executed as a stand-alone program. This is accomplished with the use of ***pytest***, a Python test framework for building scalable tests.

To get code coverage, CTF unit tests were built and run on a Linux platform with ***pytest-cov***, a pytest plugin for measuring code coverage.

5.3.1 Building the CTF unit tests and code coverage

N/A.

5.3.2 Running the CTF unit tests and code coverage

The execution steps are captured in a shell script for ease of execution and maintenance.

1. Ensure that the CTF environment is activated, i.e., the (**pythonEnv3**) prompt should be shown in terminal.

2. Execute the following command from the CTF top-level directory, where the script resides,

```
$  ./run_tests.sh utc
```

When prompted "**Are you sure you want to continue connecting (yes/no)?**", press **<Enter>**.

5.3.3 Verifying the CTF unit test results against the provided test results

The generated test and coverage results, ***./runtests_output/utc/ctf_ut_results.log*** and ***./runtests_output/utc/ctf_ut_coverage.pdf***, can be verified against the provided test results. Additional information about the test execution artifacts can also be found in the directory, ***runtests_output/utc***.

See Table 6-1 below for the file name and location of the provided unit test results and code coverage results.

5.4 Functional testing

5.4.1 Building the CTF functional tests

N/A.

5.4.2 Running the CTF functional tests

1. Ensure that the CTF environment is activated, i.e., the (**pythonEnv3**) prompt should be shown in terminal.
2. Execute the following command and follow the usage instruction:

```
$  ./run_tests.sh ft
```

When prompted "**Please Enter 'Y' to continue...**", enter '**Y**'.

5.4.3 Verifying the CTF functional test results against the provided test results

The generated test result summary in ***runtests_output/ft/ft_run/results_summary.txt*** can be verified against the provided test results. Additional information about the test execution artifacts can also be found in the directory, ***runtests_output/ft***.

See Table 6-1 below for the file name and location of the provided functional test results.

5.5 Verification testing

The descriptions of the CTF test cases are captured in the Verification Test Procedures. These test cases are implemented as CTF test scripts in JSON format. They can be found in the ***vv_tests*** sub-directory of the CTF code base. The users can opt to implement the test cases using the user's preferred verification framework based on the test description in the Test Procedures.

5.5.1 Setting up the test rig

CTF can run on any platform running CentOS 7.0.

5.5.2 Building CTF verification test

N/A.

5.5.3 Running the CTF verification tests

1. Ensure that the CTF environment is activated, the (**pythonEnv3**) prompt should be shown in terminal.
2. Execute the following command and follow the usage instruction:

```
$  ./run_tests.sh vv
```

5.5.4 Verifying the CTF verification test results against the provided test results

The generated test summaries, ***runtests_output/vv/vv_run_tc/results_summary.txt*** and ***runtests_output/vv_run_ci/results_summary.txt***, can be verified against the provided test summaries. Additional information about the test execution artifacts can also be found in the directory, ***runtests_output/vv***.

See Table 6-1 below for the file name and location of the provided verification test results.

6 CTF CERTIFICATION ARTIFACTS

Table 6-1 lists the available certification artifacts for the CTF tool.

Table 6-1: Certification Artifacts

Testing Type	*Artifact Description*	*Artifact Location*
Static code analysis	Static code analysis report	Artifacts Release package: ctf_sca.log
Unit testing with code coverage	Unit test source code and build file	ctf/unit_test/*
	Unit test results	Artifacts Release package: ctf_ut_results.log
	Code coverage results	Artifacts Release package: ctf_ut_coverage.pdf
Functional testing	Functional tests	ctf/functional_tests/*
	Functional test results	Artifacts Release package: ctf_ft_results.log
Verification testing	Verification test descriptions	Artifacts Release package: CTF_STP_procs.pdf
	CTF verification tests	ctf/ctf_test/*
	Verification test results	Artifacts Release package: ctf_vv_results.zip

7 APPENDICES

7.1 Abbreviations and Acronyms

Term	*Definition*
API	Application Programming Interface
CBCS	Computer Based Control System
cFE	Core Flight Executive
cFS	Core Flight System
ES	cFE Executive Services
EVS	cFE Event Services
GFE	Government Furnished Equipment
GSFC	Goddard Space Flight Center
ICD	Interface Control Document
ISR	Interrupt Service Routine
JSC	Johnson Space Center
LRO	Lunar Reconnaissance Orbiter
MDT	SCH_TT Message Definition Table
Msg	Message
PMP	Project Management Plan
SB	cFE Software Bus
SBNG	Software Bus Network for Gateway cFS application
SCH_TT	Time-Triggered Ethernet Scheduler cFS application
SDD	Software Design Document
SDP	Software Development Plan
SDT	SCH_TT Schedule Definition Table
SRS	Software Requirements Specification
SR&SD	JSC Engineering Directorate Software, Robotics & Simulation Division
SSET	JSC Spacecraft Software Engineering Team
STP	Software Test Plan
SUG	Software User's Guide
TTE	Time-Triggered Ethernet
TTE_LIB	Time-Triggered Ethernet cFS Library
TTE_MGR	Time-Triggered Ethernet Manager cFS application
VDD	Version Description Document

7.2 Definition of Terms

Terms	*Definition*
V&V Terms	
Certification	The audit process by which the body of evidence that results from the verification activities presented are provided to the appropriate certifying authority to indicate all requirements are met.
Deviation	Written authorization issued "before the fact" to develop a product that departs from established requirements.
HSI1	Hardware/software integration (HSI) that is performed prior to PDR. This testing establishes confidence that the hardware and software design concepts are adequate to meet functional interfaces.
HSI2	Hardware/software integration that is performed prior to CDR on engineering unit or DVTU hardware. This testing establishes confidence that the hardware and software detailed designs meets requirements.
Validation	The process that ensures a system meets the customer/sponsor's expectations for intended use. Unique validation activities may not be required if validation is satisfied through verification or acceptance testing activities.
Verification	A formal process, using the method of test, analysis, inspection or demonstration, to confirm that a system and its hardware and software components satisfy all specified performance and operational requirements.
Waiver	Written authorization to temporarily accept an item that departs from a particular performance or design requirement of a specification, drawing, or other contract document. The authorization is granted for a specific number of items and/or a specific period of time. The item(s) is/are considered suitable for use "as is" for a specified period of time or quantity of items, until reworked by approved method..
Types of V&V Methods	
Test	A method of verification wherein formal project requirements (performance, environment, etc.) are verified by measurement or functional test during or after the controlled application of functional and/or environmental stimuli. These measurements may require the use of laboratory equipment, recorded data, procedures, test support items, or specialized software.
Analysis	A verification method utilizing techniques and tools such as math models, prior test data, simulations, analytical assessments, etc. Verification by similarity is acceptable if the subject article is similar or identical in design, manufacturer, manufacturing process, and quality control to another article that has been previously verified to equivalent or more stringent criteria.
Inspection	A method of verification of physical characteristics that determines compliance without the use of special laboratory equipment, procedures, test support items, or services. Inspection uses standard methods such as visual gauges, etc. to verify compliance with design requirements.

Demonstration	A qualitative method of verification that evaluates the properties of the subject end item. Demonstration is used with or without special test equipment or instrumentation to verify required characteristics such as operational performance, human engineering features, service and access features, transportability, and displayed data.
Testing Levels	
Development Testing	
Qualification Testing	
Acceptance Testing	
System Level Testing	
End-to-End Testing	
Types of Test Articles	
Prototype Unit	The breadboard, generic component or developmental assembly of hardware and software that roughly performs the basic functions of the engineering unit but is not fully functional equivalent. This unit is used for proof of concept testing of the preliminary design.
Engineering Unit	The hardware, firmware, and software unit that is functionally equivalent to the qualification unit, but not necessarily form and fit equivalent. This unit is used for proof of concept testing of the detailed design. It may be used for software verification credit after CDR with quality controls as defined in the Software Development Plan.
Design Verification Test Unit (DVTU)	The hardware, firmware, and software unit which is form, fit and functional equivalent to the flight unit, but may not be manufactured using the exact flight parts. This unit is used for design proof of concept.
Qualification Unit	
Flight Unit	
Proto-flight Unit	
Ground Support Equipment	

8 NOTES

None.