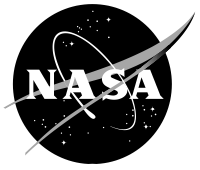
Software Design Document
for the
Core Flight System Test Framework Tool

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

Change Record

Revision	*Date*	*Originator*	*Description*
N/A	Aug 2021	Tam Ngo & Tao Wu	Initial draft
N/A	Mar 2022	Tam Ngo	Baseline

Table of Contents

List of Tables

List of Figures

1 INTRODUCTION

The Software Design Document (SDD) details the Computer Software Units (CSUs), including their identities, attributes, static relationships, dynamic interactions and design requirements. The SDD includes enough structural and behavioral information sufficient to (1) specify the internal and external behavior of the primitive software components, and (2) be translated straightforwardly into the chosen programming language(s). The SDD includes information sufficient to enable sustaining engineering of the software by programmers other than the original developers. This includes the software structure, module definitions and functionality, high-level interface descriptions, threads of control, major data structures, and important algorithms.

This SDD defines the requirements and design of the software for the Core Flight System Test Framework tool.

1.1 Scope

This Software Design Document (SDD) documents the requirements and design of the Core Flight System Test Framework tool. This document includes the selected design of the software, including its decomposition into software components, their relationships, and the design of interfaces.

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 Spacecraft Software Engineering Branch. Change authority is the Software, Robotics and Simulation Division of the Johnson Space Center.

2 RELATED DOCUMENTATION

The following documents are referenced in this specification. Unless otherwise specified, the exact issue shown is the applicable version.

2.1 Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this SDD 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
JSC-61949	Advanced Exploration Systems (AES) Core Flight Software (cFS) Software Development Plan	Rev C / Mar 2021

2.2 Reference Documents

The following documents are referenced within this SDD. These documents do not form a part of this SDD and are not controlled by their reference herein.

Table 2-2: Reference Documents

Document Number	*Document Title*	*Revision / Release Date*
GSFC 582-2007-001	cFE Application Developer's Guide	Rel. 5.4 / Sep 2014
GSFC 582-2008-012	cFS Deployment Guide	Rel. 3.0 / Sep 2014
N/A	CTF User's Guide	Baseline / Mar 2022

2.3 Order of Precedence

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

3 CSCI-WIDE DESIGN DECISIONS

3.1 Core Flight System

The Core Flight System (cFS) was developed by NASA Goddard Spaceflight Center (GSFC) as a reusable and extendible framework for developing flight software. The cFS framework has been developed over many years and was first used by NASA on the Lunar Reconnaissance Orbiter (LRO) spacecraft mission in 2009. Since then it has been successfully reused for other spacecraft missions, and its reuse has substantially increased across other NASA centers. A key feature of this framework is a layered software architecture proven to be robust and resilient in adapting to new hardware platforms with minimal changes. The cFS framework is composed of software abstraction layers from the underlying hardware and operating system as shown in **Figure 3-1**. Each layer provides a set of Application Programming Interfaces (APIs) that hides the underlying implementation for various platforms; and the implementation can be changed independently, without affecting the other layers.

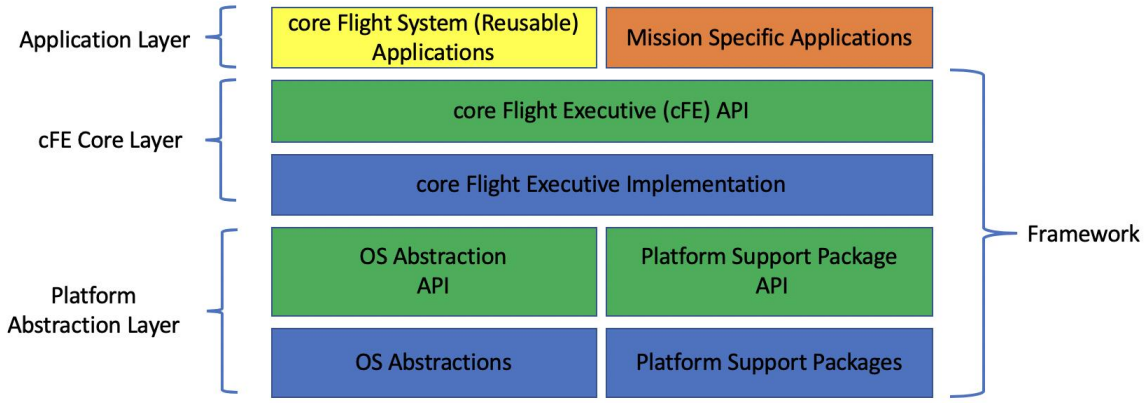


Figure 3-1: Core Flight System Architectural Layers

3.2 Core Flight System Test Framework

The Core Flight System Test Framework (CTF) is a ground software tool that provides cFS projects with the capability to develop and run automated verification tests. The CTF tool parses and executes JSON-based test scripts containing test instructions, while logging and reporting the test results. CTF utilizes a plugin-based architecture to allow users to extend CTF with new test instructions, external interfaces, and custom functionalities. CTF comes with a user-friendly, graphical editor that provides context-sensitive, auto-suggestions about a cFS system's command and telemetry definitions.

4 CSCI ARCHITECTURAL DESIGN

4.1 CSCI Components

Figure 4-1 depicts the file directory hierarchy of the CTF's code base.

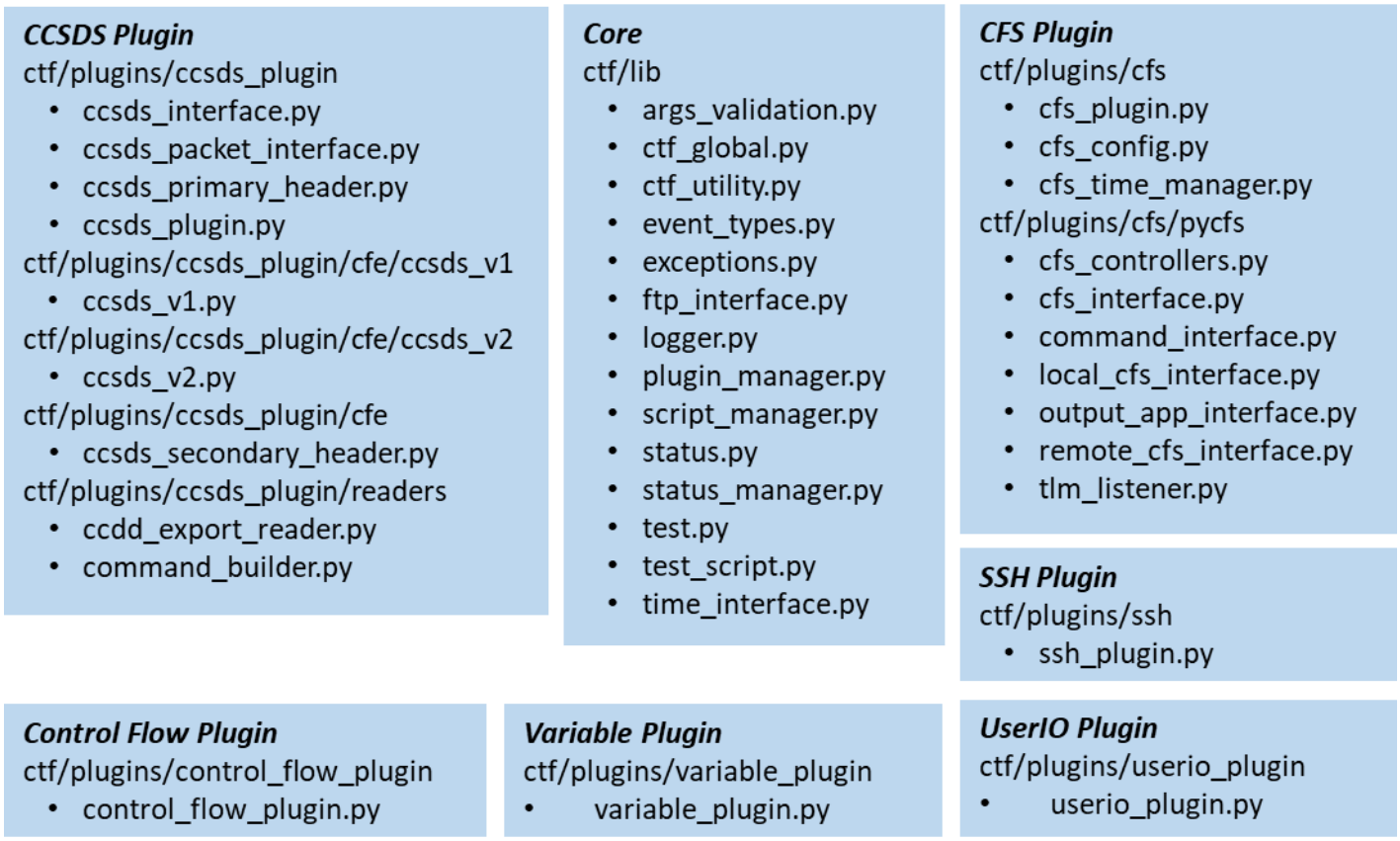


Figure 4-1. CTF File Structure

The followings list the Python source files and their purposes.

- ctf/lib/args_validation.py

This source file provides the helper utilities to validate arguments and data used by CTF. For example, it provides the method to get the number of errors encountered during arguments validation and the method to verify whether a given directory exists on disk.

- ctf/lib/ctf_global.py
  This source file provides the CTF global state information for CTF Plugins. The test info object is accessible by all plugins. The script reader populates the test header info and other values for plugins.
- ctf/lib/ctf_utility.py
  This source file defines the utility functions for the CTF core and plugins, such as path expanding and variable setting, etc.
- ctf/lib/event_types.py
  This source file defines the CTF Test Instruction class.
- ctf/lib/exceptions.py
  This source file defines the CTF Exception classes, including CtfTestError, CtfConditionError, CtfParameterError.
- ctf/lib/ftp_interface.py
  This source file defines the FtpInterface classes, providing functionality to connect/disconnect to remote FTP server, upload/download files, create folder on server. Two implementations are provided: ftputil for the SSH interface and ftplib for the SP0 interface.
- ctf/lib/logger.py
  This source file contains the functions for CTF logging of configurations and initializations.
- ctf/lib/plugin_manager.py
  This source file contains a CTF core component that manages the CTF plugins. The PluginManager class reads the plugins packages/modules that contain inherited Plugin classes. It includes methods to initialize / reload / shutdown plugins, and methods to find the plugin instance which can execute a given instruction.
- ctf/lib/script_manager.py
  This source file defines the ScriptManager class, which adds and manages all loaded CTF test scripts. It also runs all added scripts, updates the status packets, and ensures plugins are reloaded between scripts, if needed.
- ctf/lib/status.py
  This source file lists the status messages to be sent out by CTF during a test run.
- ctf/lib/status_manager.py
  This source file defines the StatusManager class that establishes a status stream with the current test suite status. The status packets are sent over a UDP socket over the specified port. The clients that listen on that port will receive periodic CTF status messages during a test execution.
- ctf/lib/test.py
  This source file defines the Test class. It is a core component of CTF library. The class represents a CTF Test Case, and includes important methods like test execution, test verification and command dispatch.

- ctf/lib/test_script.py
  This source file defines the TestScript class. It loads and validates the input CTF test scripts, manages execution of loaded test scripts, and generates the test results after test execution.
- ctf/lib/time_interface.py
  This source file defines the TimeInterface base class for the custom plugins to implement their own time managers.
- ctf/lib/reader/JSON_script_reader.py
  This source file defines the JSONScriptReader class. It provides methods to parse the contents of a CTF JSON test script and to resolve imports.
- ctf/plugins/ccsds_plugin/cfe/ccsds_v1/ccsds_v1.py
  This source file defines the CCSDS v1 message types used by cFE. It includes the definitions of CcsdsV1PrimaryHeader, CcsdsV1Packet, CcsdsV1CmdPacket, CcsdsV1TlmPacket classes.
- ctf/plugins/ccsds_plugin/cfe/ccsds_v2/ccsds_v2.py
  This source file defines the CCSDS v2 message types used by cFE. It includes definitions of CcsdsV2ExtendedHeader, CcsdsV2PrimaryHeader, CcsdsV2Packet, CcsdsV2CmdPacket, CcsdsV2PrimaryHeader classes.
- ctf/plugins/ccsds_plugin/cfe/ccsds_secondary_header.py
  This source file defines the secondary header structure types that are shared by the CCSDS V1 and V2 plugins. It includes the definitions of CcsdsSecondaryCmdHeader, CcsdsSecondaryTlmHeader classes.
- ctf/plugins/ccsds_plugin/readers/ccdd_export_reader.py
  This source file defines the CCDDExportReader class. It implements the CCSDS interface to read JSON CCSDS data from CCDD, and creates dictionaries mapping names to Python types and values for CTF.
- ctf/plugins/ccsds_plugin/readers/command_builder.py
  This source file provides the functions for building CCSDS command messages. It contains the CommandArg, CommandCode, CommandMessage classes definition, and a few helper functions to construct CommandArg, CommandCode, CommandMessage objects.
- ctf/plugins/ccsds_plugin/ccsds_interface.py
  This source file defines the CCSDSInterface class. It provides an interface and partial implementation for a CCSDS reader to process CCSDS data from the directory/files into dynamic type definitions. The method of parsing the data is left to a subclass.
- ctf/plugins/ccsds_plugin/ccsds_packet_interface.py
  This source file contains the abstract types that provide the interfaces for packet and header implementations, including the CcsdsPacketType, CcsdsPacketInterface classes.
- ctf/plugins/ccsds_plugin/ccsds_primary_header.py
  This source file implements the CCSDS primary header structure that is standard for all CCSDS messaging.
- ctf/plugins/ccsds_plugin/ccsds_plugin.py

This source file defines the CCSDSPlugin class that provides the CCSDS validation for CTF. It implements a method to send every command code (with empty payload) to cFS target found in the MID map one by one.

- ctf/plugins/cfs/pycfs/cfs_controllers.py
  This source file defines the CfsController class. When the cFS plugin registers a target, a cFS controller object is instantiated. It will establish telemetry/command interfaces. The CCSDS message definitions are parsed to build the MID map. After the cFS plugin receives a test instruction, the controller handles all lower-level functionality beneath the plugin. It will shut down the target at the end of the test script or after receiving ShutdownCfs instruction.
- ctf/plugins/cfs/pycfs/cfs_interface.py
  This source file defines the lower-level CfsInterface base class to communicate with cFS.
- ctf/plugins/cfs/pycfs/command_interface.py
  This source file defines the CommandInterface class. It provides the methods to send CCSDS messages from the CTF to cFS on the specified UDP socket.
- ctf/plugins/cfs/pycfs/local_cfs_interface.py
  This source file defines the LocalCfsInterface class. It is a derived class of the CommandInterface class, providing lower-level interface to communicate with cFS locally.
- ctf/plugins/cfs/pycfs/output_app_interface.py
  This source file contains the OutputManager and ToApi classes. The OutputManager is a base class that each output class must inherit from. ToApi is one such inherited class.
- ctf/plugins/cfs/pycfs/remote_cfs_interface.py
  This source file defines the RemoteCfsInterface class that provides the lower-level interface to communicate with cFS remotely over SSH.
- ctf/plugins/cfs/pycfs/tlm_listener.py
  This source file defines the TlmListener class that connects and manages a given UDP port connection. It implements a method to read in the next packet in telemetry stream.
- ctf/plugins/cfs/cfs_config.py
  This source file defines the CfsConfig class. It interprets the cFS target sections in the loaded INI config file. The config file can have multiple cFS targets defined, and each target section contains the needed and/or overriden configuration items for that target.
- ctf/plugins/cfs/cfs_plugin.py
  This source file implements the CfsPlugin class. It is a CTF core component. The class defines the command map with the CTF test instructions for cFS, including the functionality to register, build, start and shutdown cFS, as well as to send command, check telemetry and events, etc. The cFS plugin takes many default values from the CTF config file. The section [cfs] defines the defaults for all cFS targets and is always required.
- ctf/plugins/cfs/cfs_time_manager.py
  This source file defines the CfsTimeManager, a derived class of the TimeInterface class. When initialized by the cFS plugin, CfsTimeManager overrides the default CTF time manager (OS Time). It implements a serialized telemetry receiver as CTF instructions are “waiting” for previous instructions to complete.

- ctf/plugins/control_flow_plugin/control_flow_plugin.py
  This source file implements the ControlFlowPlugin class that provides the functionality of CTF control flow looping. It defines two test instructions: "BeginLoop" and "EndLoop". "BeginLoop" instruction creates a loop entry point. The loop is identified by a unique label and loop condition is defined in parameter "conditions" as a list of variables, the associated comparison operations, and values. "EndLoop" instruction create a loop exit point. It must match a "BeginLoop" instruction with the same label.
- ctf/plugins/ssh/ssh_plugin.py
  This source file contains the SshPlugin class and SshConfig, SshController helper classes. The SshPlugin class provides the remote and local shell command execution capability for CTF. It implements the test instructions such as "SSH_RunRemoteCommand", "SSH_PutFile", "SSH_GetFile", "SSH_PutFTP".
- ctf/plugins/userio_plugin/userio_plugin.py
  This source file defines the UserIOPlugin class. It implements the test instruction to allow user to pause the testing. The user must confirm to continue the testing for a safety critical task. CTF waits until a user instructs to continue or to abort the testing.
- ctf/plugins/variable_plugin/variable_plugin.py
  This source file defines the VariablePlugin class. This class allows the users to set, read and test variables defined in the JSON test scripts. It implements the test instructions such as "SetUserVariable", "SetUserVariableFromTlm" and "CheckUserVariable".

4.2 Concept of Execution

The CTF test goes through 3 execution phases: Initialization, Operation and Termination.

4.2.1 Initialization Phase

Figure 4-2 depicts the execution flow of CTF's Initialization phase. Before CTF runs test scripts, it goes through the sequence of steps listed below to complete its initialization phase. If a critical error occurs during the execution of these steps, CTF immediately transitions to its Termination phase.

1. CTF processes the command line arguments for the INI config file and the test scripts. If no INI config file is provided, it will use the default config file. The test scripts could be a JSON test file or a directory containing multiple JSON test files. CTF will read the INI file for config sections, which will be utilized by the plugins, the script manager and the logger.
2. CTF walks through the plugin path, loads and initializes all the available plugins. For the cFS plugin, during its initialization, it reads the provided CCDD JSON files to build a MID dictionary to interpret CCSDS messages.
3. CTF initializes the test status manager and initializes a UDP socket for sending test execution statuses to the CTF Editor.
4. CTF initializes the script manager, parses the test files, and validates all the test functions and instructions defined in the test files.

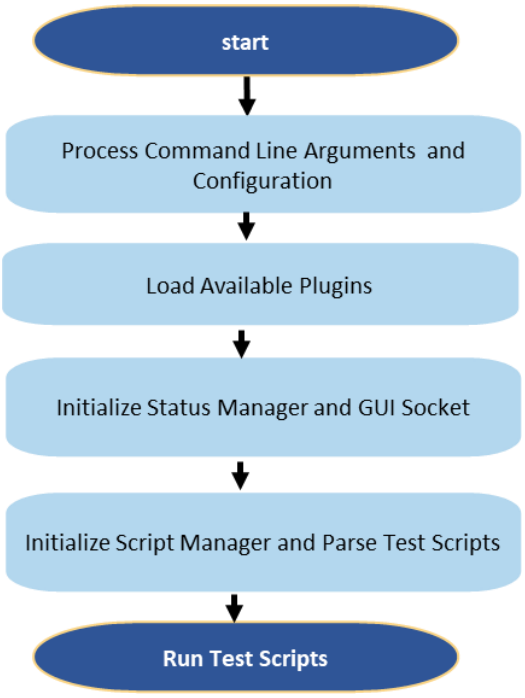


Figure 4-2. CTF Initialization Flow

4.2.2 Operation Phase

Figure 4-3 depicts the execution flow of CTF's Operation phase. Upon a successful Initialization phase, CTF enters its Operation phase to process and execute the test scripts. The test scripts include one or more JSON test files based on the command-line arguments. And each test file can have multiple test instructions.

CTF processes the test files sequentially until it completes all test files unless a critical test instruction failure occurs. (Critical test instructions are defined in INI config file). For each test file, CTF resets the script status before the first test instruction, then executes all test instructions in the order they are defined. During the Initialization phase, all loaded plugins register their supported instructions to CTF. So CTF plugin manager could search and find the appropriate plugins to execute the test instructions.

The instruction execution states are either "Pass" or "Fail". If it fails and it is critical, CTF will transition to the Termination phase. Otherwise, CTF logs the instructions' status and sends the updates to the CTF GUI periodically. After completing all instructions in one test script file, CTF logs the test script summary, then processes the next test file. The process repeats until all test files are executed.

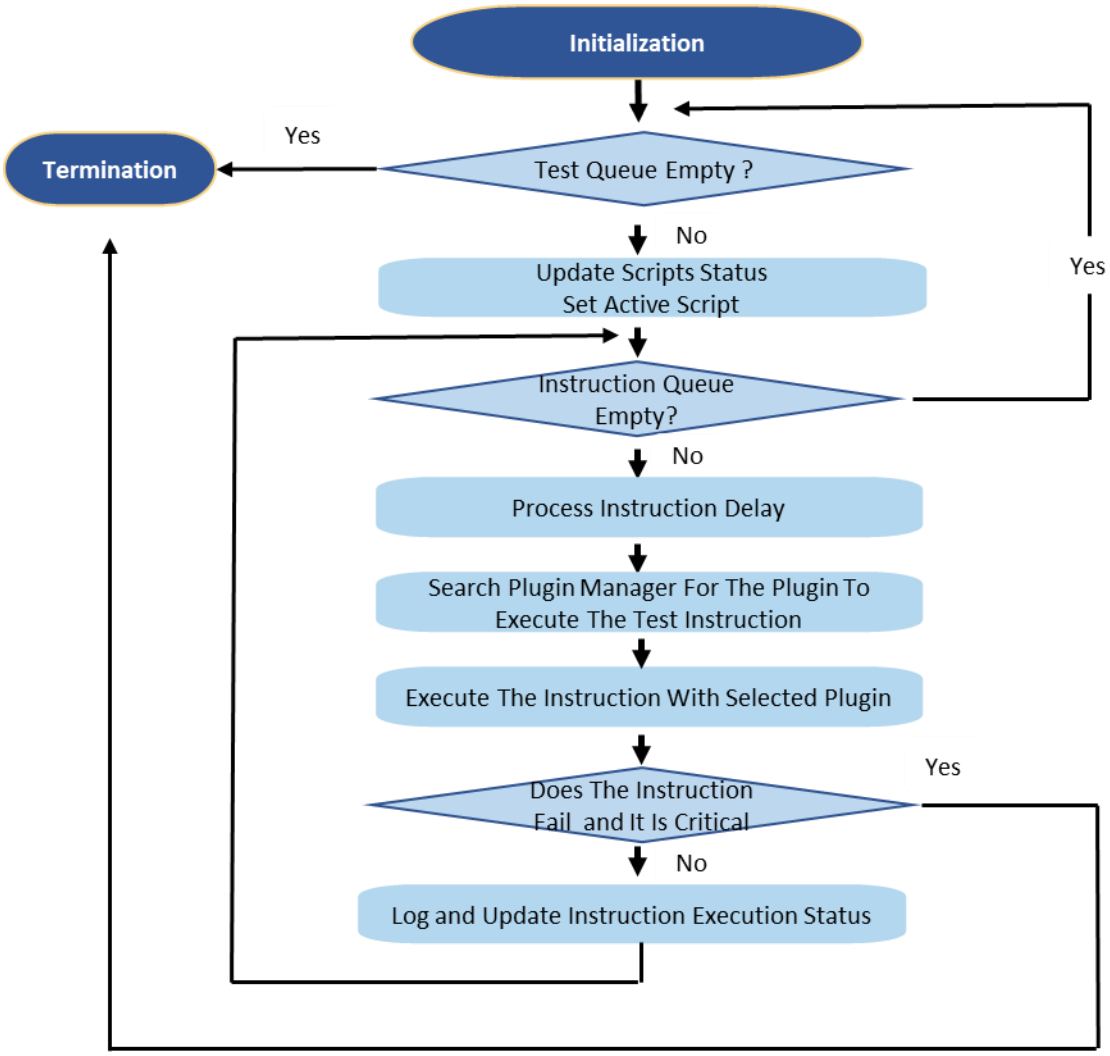


Figure 4-3. CTF Operational Flow

4.2.3 Termination Phase

CTF transitions to the Termination phase when it

1. encounters a critical error during the Initialization or Operational phase, or
2. completes the execution of all test scripts, or
3. is directed by the CTF user to terminate.

In the Termination phase, CTF will do the followings:

1. log the tests summary, and
2. calls all the loaded plugins' shutdown method to clear the test environment, and
3. shutdowns the cFS target that it is testing with.

4.3 Interface Design

Figure 4-4 depicts the CTF architecture. It composes of the CTF core modules and plugins. The test instructions are executed by the plugins, and CTF allows the users to extend the functionality with new test instructions by adding new plugins. During a test, the CTF Core searches all the loaded plugins and selects the appropriate plugins to execute the test instructions.

Before a test, the CTF Core parses the input files. During a test execution, it generates the log files and periodically sends the test statuses to CTF GUI for display. It interfaces with the cFS instances through their Command Ingest (CI) and Telemetry Output (TO) applications.

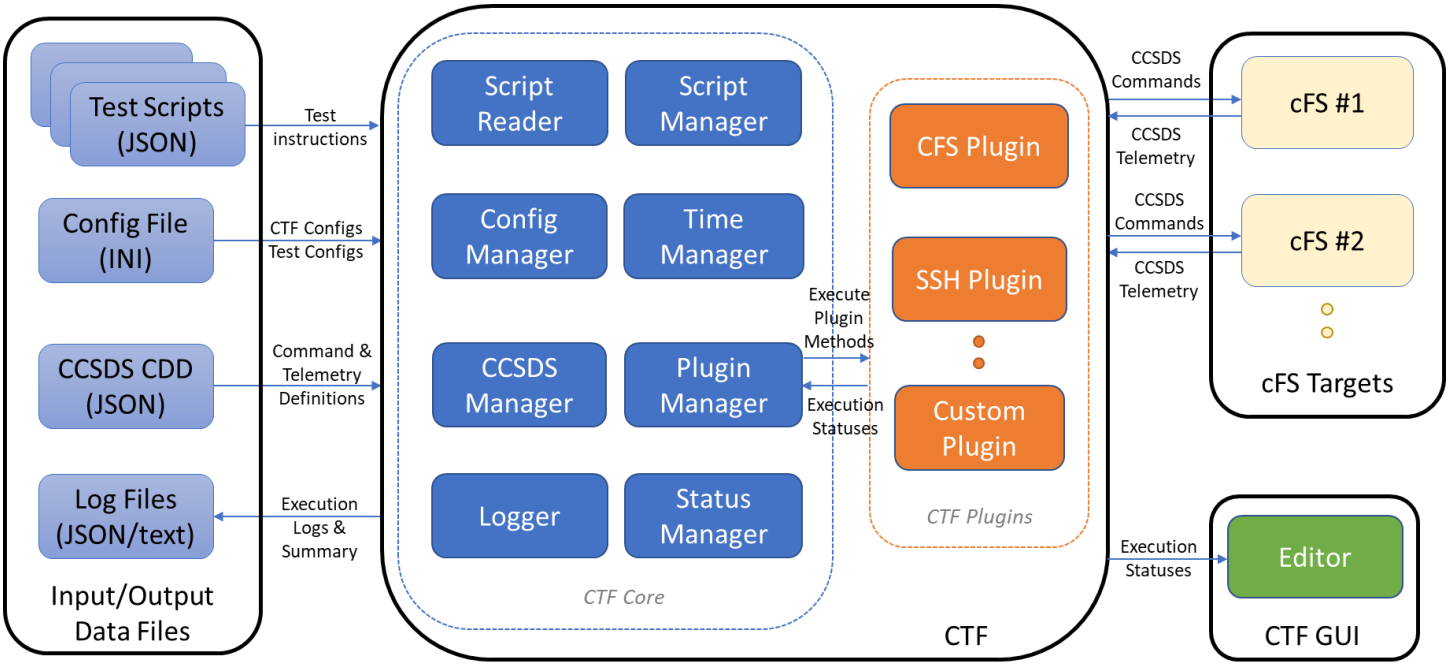


Figure 4-4. CTF Architecture Overview

4.3.1 Internal Interfaces

The interface between CTF core and the plugins is a "command_map" dictionary consisting of a collection of key-value pairs. The key of the pair is instruction name; and the value of the pair is a tuple of the associated function and arguments types. The customized plugins must inherit from the base Plugin class and must implement the "command_map" dictionary. The instruction name should be unique for all plugins. Below is an example from the Example Plugin.

```
{
   "TestCommand": (self.test_command, [ArgTypes.string] * 2),
   "TestVerifyCommand": (self.test_verify_command, []),
   "TestSharedLibraryCommand": (self.test_shared_library, [])
}
```

During the Initialization phase, CTF will walk through the plugin path directory, and load and initiate all the available plugins. To execute an instruction, the Plugin Manager selects the plugin whose "command_map"

dictionary contains that test instruction name. Then CTF executes the plugin's method to process that test instruction.

4.3.2 External Interfaces

The users must provide the test scripts, test configuration, and CCSDS message definition files. These files should follow certain format so that CTF could understand and run them.

CTF obtains the CCSDS message definitions by parsing a set of files that adhere to a JSON schema to define the command and telemetry message structures. These files can be created manually or automatically using the CCDD tool. Examples of such file can be found in the "***ccdd***" directory of the "***CFS_Sample_Workspace***" that comes with the CTF release under the "***external***" directory.

The configuration file contains the configuration options for each CTF plugin. To run CTF with a selected configuration, add the `--config_file <path_to_config_file>` as the command line argument. If no file is provided, CTF will use the default config file.

Each plugin can define one or more sections with specific configuration values as follows:

```
[some_plugin_config]
some_field = true
some_other_field = xyz
```

See CTF User's Guide for more details about the content of the configuration file.

The test scripts files need to have the following properties:

- **test_number**: a unique string identifier for the test script.
- **test_name**: a short descriptive name for the tests in the file. The name should be descriptive but concise such that the users can understand what is being tested.
- **requirements**: the list of requirement IDs that this test script is verifying.
- **description**: a detailed description of the test script.
- **import**: the list of test functions that can be imported from other JSON files.
- **functions**: user-defined test functions that can be used by the test script, described below.
- **tests**: list of test instructions. Multiple instructions can be grouped into a test case. A test script can have multiple test cases.

See CTF User's Guide for more details about the content of the test scripts.

CTF interacts with cFS instances via 2 UDP ports: one for sending commands to the cFS, and one for receiving telemetry and events from the cFS. CTF and cFS exchange messages that are conformed to the CCSDS message format.

CTF and CTF GUI are 2 independent applications. The users can run CTF without running GUI and vice versa. If "`--port`" argument is passed via the command line arguments, CTF will open a UDP socket and periodically sends tests statuses through it. The CTF GUI could listen to the port, receive the tests statuses, and update its display accordingly. It is worth to note that this is a one-way communication, i.e., CTF GUI does not send CTF any messages.

5 CSCI DETAILED DESIGN

This section of the document is generated from Doxygen comments in the application's source code. The section contains the declarations of macros, variables and data structures, the function definitions, along with their inputs, outputs and limitations. The generated document also includes the application's configurable parameters and their values. For more details, see Section_5. If this document comes from the NASA-only release of CTF, see NASA-only Section_5.

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.

6 BI-DIRECTIONAL TRACEABILITY MATRIX

This section documents the mapping of the software requirements to the design elements and from the design elements to the software requirements. The CTF has three execution phases as defined in section 4: Initialization, Operations, and Termination. The execution phases are the application design elements.

6.1 From Requirement to Design Element

For more details, see Requirements to Design Elements table.

6.2 From Design Element to Requirement

For more details, see Design Elements to Requirements table.

7 APPENDICES

7.1 Abbreviations and Acronyms

Term	*Definition*
AES	Advanced Exploration Systems
API	Application Programming Interface
CDR	Critical Design Review
cFE	Core Flight Executive
cFS	Core Flight System
CI	Command Ingest cFS Application
CMD	Command
CR	Change Request
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
CTF	cFS Test Framework tool
EA	Engineering Directorate Mail Code
FSW	Flight Software
GFE	Government Furnished Equipment
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HK	Housekeeping Telemetry, or cFS Housekeeping Application
ICD	Interface Control Document
ISR	Interrupt Service Routine
JSC	Johnson Space Center
MD	Memory Dwell cFS Application
MDT	Message Definition Table
MID	Message Identifier
NASA	National Aeronautics and Space Administration
NPR	NASA Procedural Requirements
OS	Operating System
OSAL	Operating System Abstraction Layer
PDR	Preliminary Design Phase
PMP	Project Management Plan
PSP	Platform Support Package
PTRS	Project Technical Requirements Specification
SCH	Scheduler cFS Application
SCH_TT	Time-Triggered Scheduler cFS Application
SB	cFE Software Bus
SDD	Software Design Document
SDP	Software Development Plan
SDT	Schedule Definition Table

Term	*Definition*
SW, S/W	Software
TBD	To Be Determined
TBL	Table
TDM	Time-Division Multiplexing
TLM	Telemetry
TO	Telemetry Output cFS Application
TTE	Time-Triggered Ethernet
TTE_MGR	Time-Triggered Manager cFS Application
V&V	Verification and Validation
V&VD	Verification and Validation Document
VDD	Version Description Document
WI	Work Instruction

7.2 Definition of Terms

None.

8 NOTES

None.