521-DDS-003

EARTH OBSERVING SYSTEM
DATA AND INFORMATION SYSTEM (EOSDIS)
TEST SYSTEM (ETS)
SIMULATED CCSDS TELEMETRY GENERATOR (SCTGEN)
DETAILED DESIGN SPECIFICATION

Volume 5

SIGNATURE DRAFT

October 1996

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

Earth Observing System Data and Information System (EOSDIS) Test System (ETS) Simulated CCSDS Telemetry Generator (SCTGEN) Detailed Design Specification

Volume 5

October 1996

Prepared by:

C. Mirchandani, ETS System Engineer
Lockheed-Martin Space Mission Systems

Quality Assured by:

C. Pepersack, Sr. Systems Engineer
SGT, Inc.

Reviewed by:

S. Hilinski, ETS Systems Engineer
ISI, Inc.

Concurred by:

W. Fuller, ETS Project Manager
Code 515.2, NASA/GSFC

Reviewed by:

C. Mirchandani, ETS Project Lead
Lockheed-Martin Space Mission Systems

Concurred by:

D. Lakins, ESDIS ETS Project Manager
ESDIS Project
Code 505, NASA/GSFC

Edited by:

L. Phillips, Publications Manager
RMS Technologies, Inc.

Approved by:

N. Speciale, Head
Microelectronic Systems Branch
Code 521, NASA/GSFC

Goddard Space Flight Center
Greenbelt, Maryland 20771

PREFACE

The ETS Simulated CCSDS Telemetry Generator (SCTGEN) Detailed Design Specification specifies the detailed system design for the ETS SCTGEN being developed by the Goddard Space Flight Center (GSFC) Microelectronic Systems Branch (MSB). This document is Volume 5 of a 5-volume set of documents that specify detailed design specifications for the Earth Observing System Data and Information System (EOSDIS) Test System (ETS).

This document will serve as the single source document for configuration control of the ETS SCTGEN system design, and will serve as the basis for design, implementation, and testing.

This document is under the configuration management of the Microelectronic Systems Branch Configuration Control Board (CCB). Any changes to this document shall be made by Documentation Change Notice (DCN), reflected in text by change bars, or by complete revision.

Requests for copies of this document, along with questions and proposed changes, should be addressed to:

Technical Publications Group, Code 520.9
Goddard Space Flight Center
Greenbelt, Maryland 20771

CHANGE INFORMATION PAGE

List of Effective Pages	
Page Number	**Issue**
Title	Original
Signature Page	Original
iii through viii	Original
1-1 through 1-5	Original
2-1 through 2-4	Original
3-1 through 3-26	Original
4-1 and 4-2	Original
5-1 through 5-10	Original
6-1 through 6-8	Original
AB-1	Original

Document History		
Issue	**Date**	**DCN No.**
Original	October 1996	N/A

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION 1-1
1.1 Overview 1-1
1.2 System Overview 1-1
1.3 Applicable Documents 1-2
1.4 Document Organization 1-3
1.5 Terminology 1-3
1.5.1 Definitions 1-3
1.5.2 Data Types and Formats 1-5

SECTION 2 - SYSTEM DESIGN OVERVIEW 2-1
2.1 Design Methodologies 2-1
2.2 Functional Overview 2-1
2.3 System Architecture 2-3
2.4 External Interfaces 2-4

SECTION 3 - USER INTERFACE 3-1
3.1 Overview 3-1
3.2 Top-Level Menu Descriptions 3-2
3.3 Typical Script and Data Generated 3-7

SECTION 4 - OPERATIONS SCENARIO 4-1
4.1 Overview 4-1
4.2 Test Data Generation Specification 4-1
4.3 Test Data Generation 4-2
4.4 Expected Results 4-2

SECTION 5 - DETAILED DESIGN 5-1
5.1 Configuration Items 5-1
5.2 ETS Layer Functions 5-2
5.3 CCSDS Layer Functions 5-2
5.4 Core Function Object Classes 5-3
5.4.1 Stream Function Class Description 5-3
5.4.2 Genstream Function Class Description 5-5
5.4.3 List Patterns Function Class Description 5-5
5.4.4 Consumer Function Class Description 5-6
5.4.5 Err List Function Class Description 5-7
5.4.6 List Devices Function Class Description 5-8
5.4.7 Exception Function Class Description 5-9

SECTION 6 - REQUIREMENTS TRACEABILITY MATRIX 6-1

ACRONYMS AND ABBREVIATIONS AB-1

TABLE OF CONTENTS (CONT'D)

TABLE

Table 2-1 External Interface Description....2-4

FIGURES

Figure 1-1 ETS System Architectural Overview....1-2
Figure 2-1 SCTGEN Functional Overview....2-1
Figure 2-2 ETS HRS Simulating High-Rate Data Flows from TGT to EDOS....2-2
Figure 2-3 ETS HRS Simulating EDOS Output Interface....2-2
Figure 2-4 ETS HRS Simulating DAAC Input Interface....2-3
Figure 2-5 ETS LRS Simulating Low-Rate Data Flows to/from EDOS....2-3
Figure 2-6 Architectural Features....2-4
Figure 2-7 SCTGEN External Interfaces....2-4
Figure 3-1 Navigation Chart for User Interface....3-1
Figure 3-2 Menu Requirements for User Interface....3-2
Figure 3-3 Main Graphical User Interface Panel....3-3
Figure 3-4 Data Scenario Menu Panel....3-4
Figure 3-5 CADU Data Scenario Graphical Illustration....3-5
Figure 3-6 Frame Formatting and Multiplexing Menu Panel....3-6
Figure 4-1 Operational Data Generation Scenario....4-1
Figure 5-1 Architectural Features....5-1
Figure 5-2 Version 2 CADU Return Link Data....5-2
Figure 5-3 CLTU Forward Link Data....5-3
Figure 5-4 Stream Function Object Class....5-4
Figure 5-5 Genstream Function Object Class....5-5
Figure 5-6 List Patterns Function Object Class....5-6
Figure 5-7 Consumer Function Object Class....5-7
Figure 5-8 ErrList Function Object Class....5-8
Figure 5-9 ListDevices Function Object Class....5-9
Figure 5-10 Exception Function Object Classes....5-10

SECTION 1
INTRODUCTION

1.1 OVERVIEW

This document specifies the detailed design for the Simulated Consultative Committee for Space Data Systems (CCSDS) Telemetry Generator (SCTGEN). SCTGEN is being developed as part of the Earth Observing System Data and Information System (EOSDIS) Test System (ETS) Project. SCTGEN's primary function is to generate telemetry data files in various formats according to user specifications. In conjunction with other ETS components, SCTGEN will support EOS ground system integration, testing, verification, and validation.

SCTGEN is a software package with a Graphical User Interface (GUI) that simulates CCSDS and non-CCSDS telemetry for forward and return link data streams. When used as a test tool for the EOSDIS, it will also simulate EOS Data and Operations System (EDOS)-generated data products, such as EDOS Data Units (EDU), Expedited Data Sets (EDS), and Production Data Sets (PDS). SCTGEN will also provide comprehensive error insertion capabilities. SCTGEN is an offline tool used to generate test data, and as such, will not present a schedule conflict with operational systems. However, to generate the required test data in time for testing, the user community will have to schedule adequate time to compose and generate the test data suite.

SCTGEN is being developed by Code 521, the Microelectronic Systems Branch (MSB), at Goddard Space Flight Center (GSFC). In addition to specific ETS simulation functions that are funded by the ETS Project, SCTGEN will also simulate other commonly used data formats, scenarios, and communication protocols as a generic data simulation tool. Generic efforts are funded by the MSB. All ETS Project needs and requirements pertaining to the same are listed in Section 6.

1.2 SYSTEM OVERVIEW

SCTGEN will be based on an architecture that provides a modular and flexible environment. This environment will support current data simulation requirements, and will also allow future development of new data formats and a range of data scenarios. It will run in a UNIX environment on the Control and Display (CDS) Workstation. High reusability is a basic design goal, with a perspective for code and data reuse. The user interface will be based on graphical representation, and will be enhanced with intelligence to alleviate the burdens involved in test data specification.

To support automated test operations, SCTGEN summarizes expected results for each test data file generated. Expected results will be stored in easily readable formats so that ETS data verification tools can read them in for comparison with actual test results.

Figure 1-1 illustrates the conceptual ETS system architecture; shaded boxes represent SCTGEN's role in the context of ETS.

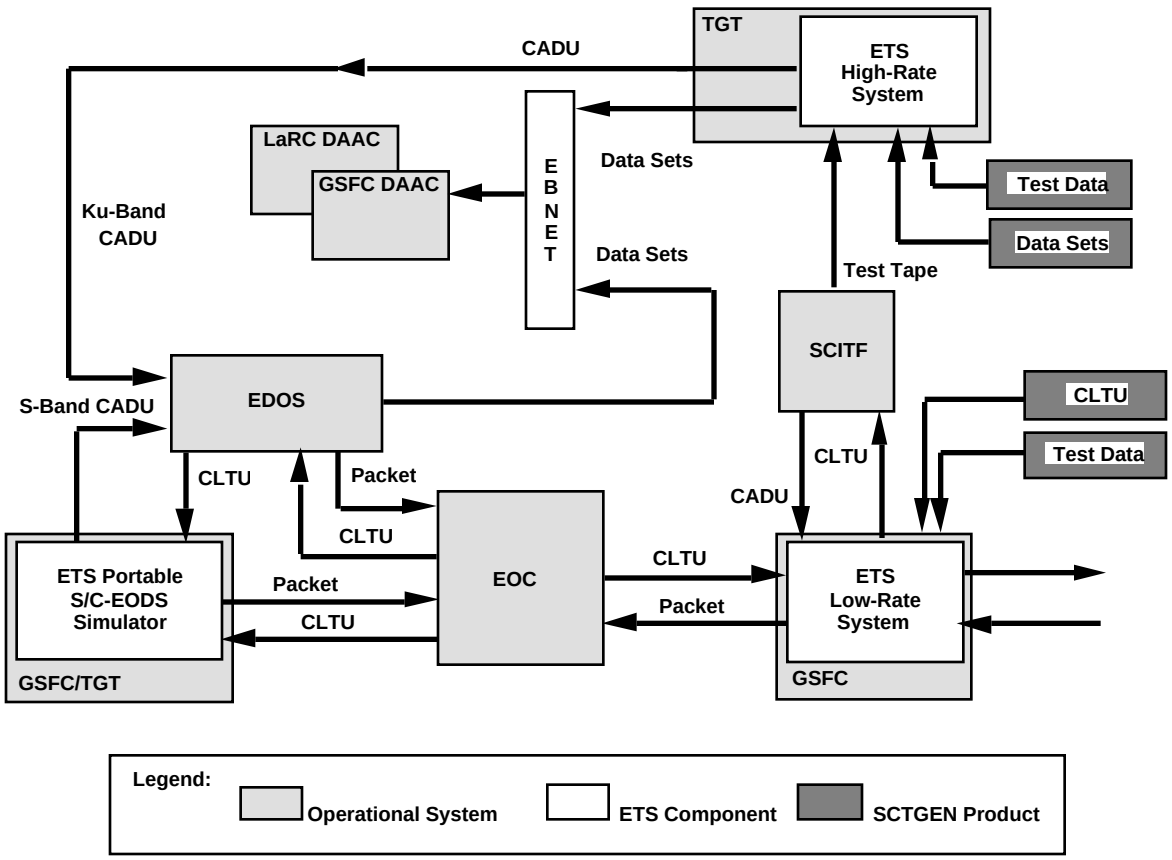

Figure 1-1. ETS System Architectural Overview

1.3 APPLICABLE DOCUMENTS

The following documents were used as references for the development of system requirements. They further clarify, support, and define SCTGEN objectives, and the information provided in this document.

a. Earth Observing System Data and Information System (EOSDIS) Test System (ETS) Functional and Performance Requirements, 515-4FRD/0294, March 1996.

b. Earth Observing System Data and Information System (EOSDIS) Test System (ETS) Operations Concept, 515-3OCD/0194, May 1995.

c. Data Format and Control Book for EOS-AM Spacecraft, Interface Control Document (ICD) 106, Martin Marietta Corporation, Astro Space, April 1996.

d. Consultative Committee for Space Data Systems, Recommendations for Space Data System Standards: Advanced Orbiting Systems (AOS), Networks and Data Links, CCSDS 701.0-B-2, November 1992.

e. Consultative Committee for Space Data Systems, Recommendations for Space Data System Standards: Telemetry Channel Coding, CCSDS 101.0-B-3, May 1992.

f. Consultative Committee for Space Data Systems, Recommendations for Space Data System Standards: Time Code Formats, CCSDS 301.0-B-2, April 1990.

g. Consultative Committee for Space Data Systems, Recommendations for Space Data System Standards: Packet Telemetry, CCSDS 102.0-B-3, November 1992.

h. EOS Data and Operations System (EDOS) - EOSDIS Backbone Network (EBnet) Interface Control Document, 510-ICD-EDOS/EBnet, February 1996.

i. Tracking and Data Relay Satellite System (TDRSS) Ground Terminal (TGT) - EDOS Interface Control Document.

j. EDOS Functional and Performance Specification, 560-EDOS-0202.0004, NASA/GSFC, December 18, 1992.

k. EDOS External Interface Control Document Data Format Control Document, July 1995.

l. High-Rate Data Test Equipment for EOS-AM Spacecraft Requirements Document, GSFC, NAS5-32500, Martin Marietta, March 1994.

1.4 DOCUMENT ORGANIZATION

This document is organized as follows:

a. Section 1—provides introduction.

b. Section 2—provides the system architecture and functional overview, and defines external interfaces as they pertain to SCTGEN.

c. Section 3—describes the high-level SCTGEN user interface.

d. Section 4—describes a typical operations scenario.

e. Section 5—provides a detailed design description of the system.

f. Section 6—contains traceability matrix between functional and performance requirements and system requirements.

1.5 TERMINOLOGY

1.5.1 DEFINITIONS

Data Format: data unit structure. Includes, but is not limited to, data unit size, description of fields within the data unit and any constraints therein, and format identification.

Data Layer: level of encapsulation of source data. Packet data layer implies that source data is encapsulated in CCSDS or non-CCSDS packets. Frame or Virtual Channel Data Unit (VCDU) data layer implies that source data is encapsulated within frames or VCDUs, etc.

Data Scenario: design of test data to be generated. Includes the order in which data units are multiplexed within each level, the order in which data units are multiplexed at the next level of encapsulation, etc. For example, the order of packets (as distinguished by Application Process Identifiers [APID]) within each Virtual Channel Identifier (VCID), and the frequency with which these VCIDs are encapsulated into Channel Access Data Units (CADU) and multiplexed to form a CADU test data product.

Data Segment: collection of time-ordered packets. A data stream may have many collections of time-ordered packets within the collection, but may be out of time order from collection to collection (for example, a real-time downlink and a playback dump).

Data Stream: serial stream of binary data contained in a test data product.

Data Unit: level of encapsulation of actual user data. For example, a packet data unit consists of a header, source data, and trailer; a frame data unit consists of a header, data unit zone, and trailer. The frame data unit zone may contain packet data units, or pure source data.

Specifications: user-defined parameters for the generation of test data. Includes data format, data scenario, level of encapsulation, special instructions for data manipulation (what, where, how many), error insertion, timecode deviation, gap insertion, and size of final product.

TestData: final product generated by SCTGEN. This could be a concatenated stream of bit stream data, packet data units, frame data units, block data units, etc., stored in file format, or as a binary stream on tape.

Data Patterns: data within the application data zone of a packet. This data can be any pattern, either defined by the user, or read in from a file.

Packet Stream: concatenated serial stream of multiplexed packets from a single virtual channel.

Bit Stream: single stream of data bits encapsulated in frames.

Frame Stream: concatenated serial stream of multiplexed frames that comprise a single data stream.

1.5.2 DATA TYPES AND FORMATS

Two basic data types are generated by SCTGEN: binary test data files and American Standard Code for Information Interchange (ASCII) summary (expected results) files. File formats include, but are not limited to, those used by ETS operationally and those used for internal testing.

a. Complete Format: complete test data stream saved as a binary data file with all data manipulations processed into it. No further modification is necessary when transmitting data in complete format.

b. Test Pattern Generator (TPGEN) Format: used in older and current Very Large Scale Integration (VLSI) systems produced by Code 521. It is a repeatable stream of test data, or base set, appended with a list of data manipulations, or updates. The base set and updates are saved as a binary data file. This format requires Code 521 hardware to perform updates to the base set prior to transmission.

c. Simulated Telemetry Generation (STGEN) Format: consists of two (or three for specific cases) files. The first file contains the complete test data stream; the second file contains a list of data manipulations, or updates. When a test data product consists of only a packet data stream, a third type of file contains a list of locations, or packet length marker, within the test data file where each packet ends. All three types within this format are saved as binary data files. This format requires Code 521 hardware to perform updates to the base set prior to transmission.

d. Expected Results: consists of a summary of expected statistics of the generated test data. It consists of statistics at the following levels of detail:

(1) Abstraction level, whereby SCTGEN will track certain types of data at certain locations within the test data, the values in those locations, and a flag that describes the state of the contents.

(2) First and second levels of detail, which may contain, for example, total statistics of the number of packets from each APID, errors in specific packets, type of errors, number of packets from that source that are dropped or missing (with explanation), and similar total statistics at the frame and/or block levels. These files are saved as ASCII data files.

SECTION 2
SYSTEM DESIGN OVERVIEW

2.1 DESIGN METHODOLOGIES

SCTGEN is a software system developed using object-oriented methodology and a Commercial Off-the-Shelf (COTS) product for diagramming techniques. The selected tool, Software Through Pictures (STP), Object Modeling Technique (OMT) diagramming techniques and rules, uses the Rambaugh approach to object-oriented design. In conjunction with the system design specifications for ETS, SCTGEN will reside on the CDS, which is the user interface for the configuration, command, and monitoring of all ETS elements. The top-level application that launches the various systems controlled by the CDS is the Menu Controller, which is described in the ETS Low-rate System and ETS High-rate System Detailed Design Specifications.

SCTGEN is written in the C++ programming language and uses COTS compilers that run on all UNIX platforms. SCTGEN will run under the Menu Controller and will be interfaced via the SCTGEN GUI. The GUI will be OSF Motif Style Guide-compliant.

2.2 FUNCTIONAL OVERVIEW

SCTGEN requires user-specified information to generate varied formats of test data for operational testing. Functional requirements are further abstracted to provide support of the ETS Project and data flows associated with the EOS AM-1. SCTGEN consists of the software system with a user interface (Figure 2-1).

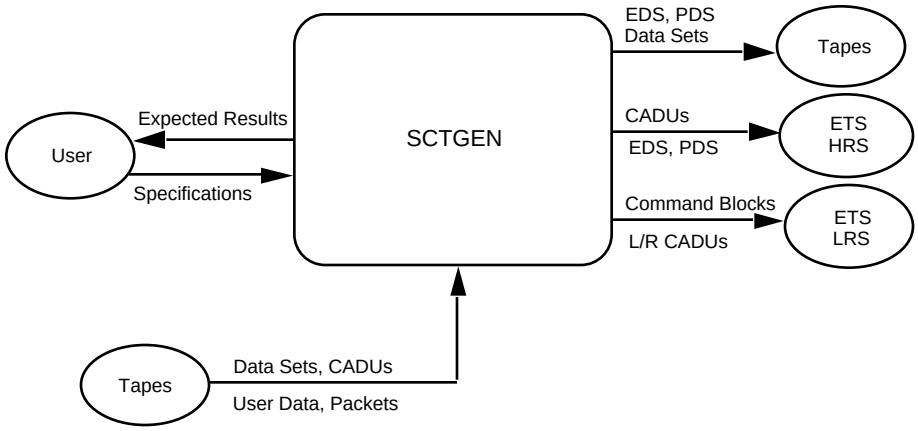

Figure 2-1. SCTGEN Functional Overview

Functionally, SCTGEN generates test data and scenarios in support of the configurations shown in Figures 2-2 through 2-5. In the configuration simulated in Figure 2-2, the ETS High-rate System (HRS) will simulate the Tracking and Data Relay Satellite System (TDRSS) Ground Terminal (TGT) interface to EDOS. SCTGEN will create test data as follows:

a. Format will be loaded directly onto an EOS Simulator Card within the HRS, which will send a serial stream of CADU at 150 Mbps.

b. Format will be stored directly on tape media, which will be transferred to the HRS for transmission as a serial stream of CADUs.

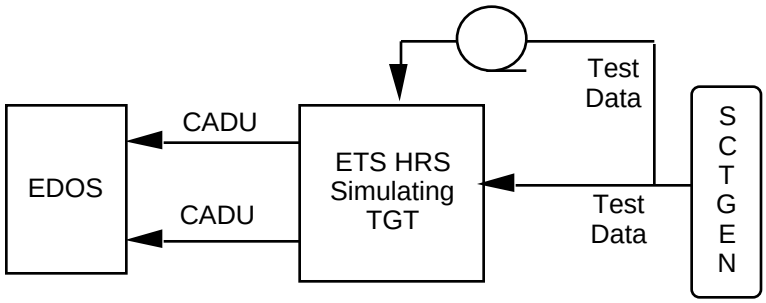

Figure 2-2. ETS HRS Simulating High-Rate Data Flows from TGT to EDOS

Test data generated by SCTGEN in this configuration can contain simulated user data, or actual user data provided via a recorded media.

Figure 2-3 illustrates SCTGEN simulating data flows from EDOS to a Distributed Active Archive Center (DAAC). SCTGEN will generate test data that will be fed to the HRS EDOS simulation configuration. This data can be generated and stored on tape media for later use, or, for smaller test data, loaded directly on local storage. SCTGEN can create EDOS data sets for transmission by the HRS, or a CADU stream for HRS processing, data set construction, and transmission. At test time, test data (in this case, data sets) will be transferred through the network interface to the DAAC.

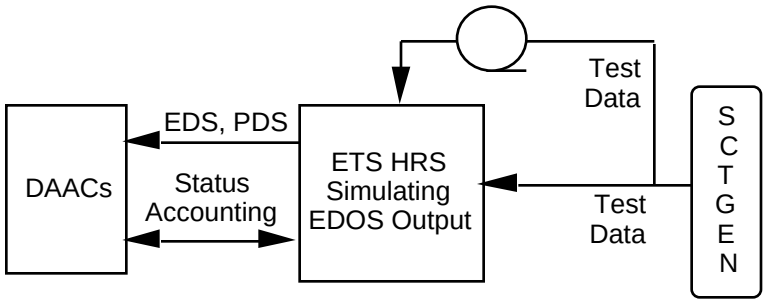

Figure 2-3. ETS HRS Simulating EDOS Output Interface

SCTGEN can also be used to support ETS test configurations in which ETS simulates a DAAC receiving EDOS-generated data sets (Figure 2-4). As part of data generation, SCTGEN will produce a set of expected results to be used for comparison in the post-test phase. SCTGEN will generate a CADU stream that will be transmitted to EDOS for data set processing, and an expected results file indicating data quality and accounting information, which can be used to check received EDOS data sets.

NOTE: Within this architecture, SCTGEN will also generate test data and scenarios for internally testing the subsystems developed and provided by the MSB.

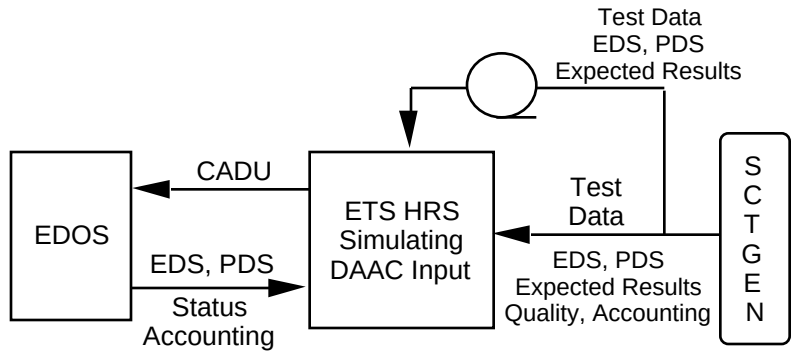

Figure 2-4. ETS HRS Simulating DAAC Input Interface

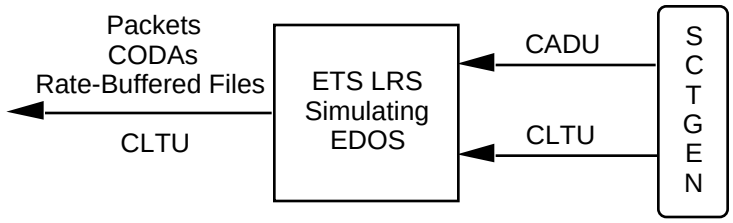

Figure 2-5. ETS LRS Simulating Low-Rate Data Flows to/from EDOS

2.3 SYSTEM ARCHITECTURE

SCTGEN consists of three layer functions (Figure 2-6). The User Interface implements the design input tool to capture the specification of the data to be generated. The interface, which will be a GUI, will ingest the user's requirement for a data scenario via key strokes and mouse clicks, and then generate the scripts necessary to run the lower layers of the system.

ETS Layer Functions implement ETS-specific processes to produce EDOS data products (i.e., EDSs, PDSs, EDUs, etc.). CCSDS Layer Functions are CCSDS-specific processes that follow CCSDS formats for return and forward link data generation and manipulation. Core Functions manipulate data in a generic manner. These processes can fill, build, and modify any field for any data structure.

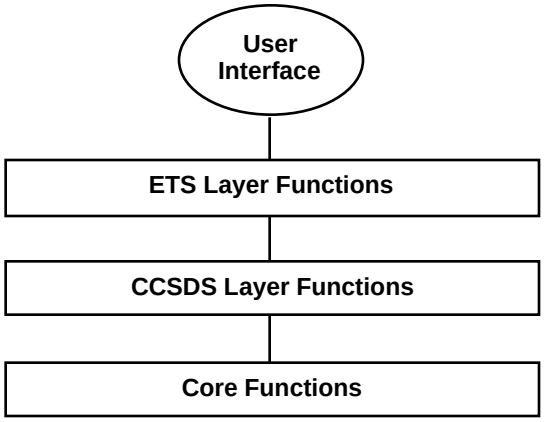

Figure 2-6. Architectural Features

2.4 EXTERNAL INTERFACES

Figure 2-7 illustrates the external interfaces to SCTGEN. All Input/Output (I/O) from SCTGEN is transferred as test data to storage devices.

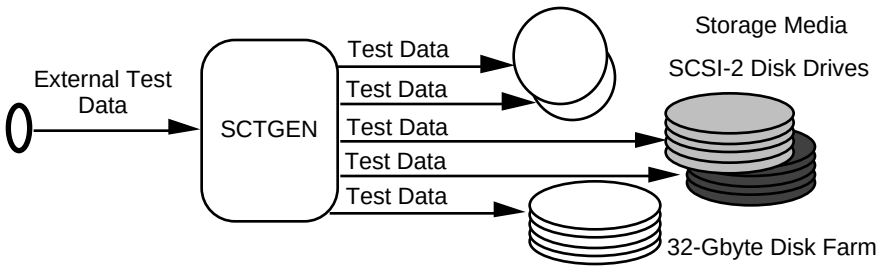

Figure 2-7. SCTGEN External Interfaces

SCTGEN will receive input test data and output final product test data into two types of storage media via a network interface, as defined in Table 2-1.

Table 2-1. External Interface Description

No.	Type	Protocol	Storage Media
1	Network interface	UNIX I/O protocols	Small Computer Systems Interface (SCSI)-2 disk drives, 32-Gbyte disk farm
2	Network interface	Transmission Control Protocol/Internet Protocol (TCP/IP), tape drive/disk storage write protocols, and TBS	Tape drives or disk storage

SECTION 3
USER INTERFACE

3.1 OVERVIEW

Figure 3-1 provides a user interface navigation diagram for the flow of the information needed for SCTGEN. The navigation diagram starts with a login screen that instantiates the interface.

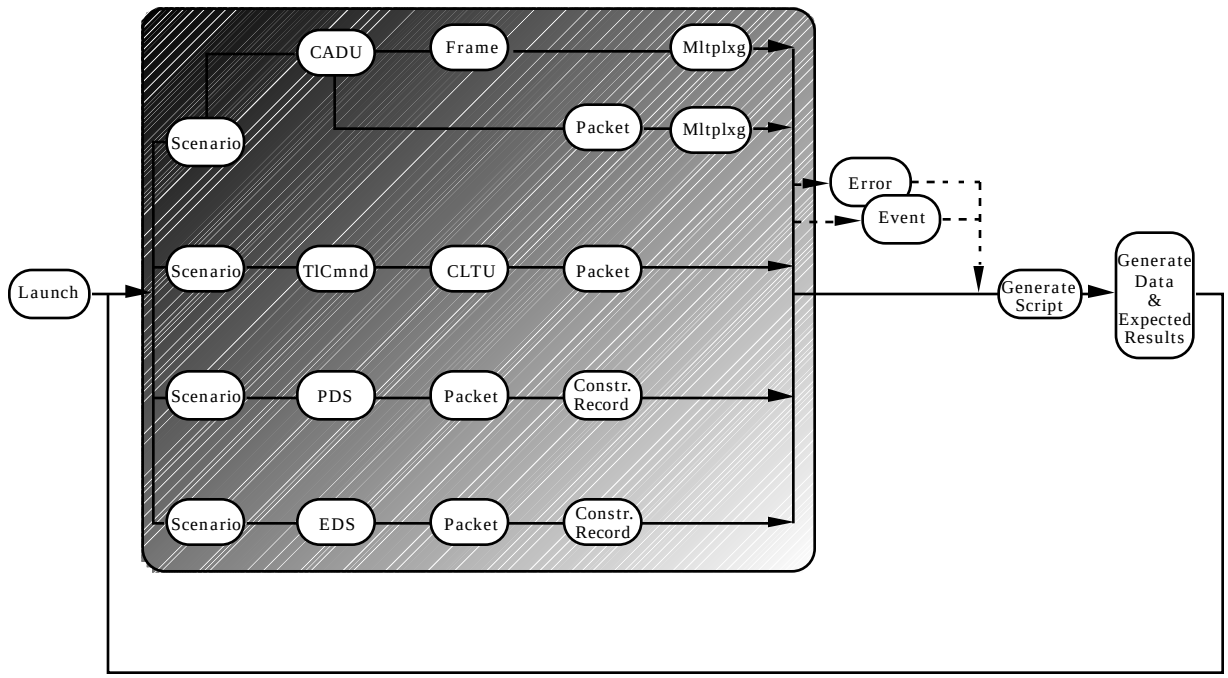

Figure 3-1. Navigation Chart for User Interface

SCTGEN needs user-specified information to generate varied formats of test data for operational testing. Functional requirements are further abstracted to provide the capability of not only supporting the ETS Project and the data flows associated with the EOS-AM and PM Spacecraft, but also for projected support of all types of known and predicted data formats used NASA-wide.

A GUI will be used with SCTGEN. This interface will translate the design of test data from paper to the computer. The GUI will generate a script that will initiate the generation of packets, VCDUs, CADUs, and data sets. The login menu for SCTGEN is envisaged to be called up from the Menu Controller, the top-level running application on the CDS. Once the login menu is displayed, the user will have options to initiate a data generation session.

Because this document describes the design philosophy for ETS from the EOS AM-1 requirements and the Interface Control Documents (ICD) listed in Section 1, only those options that are ETS-specific will be described. Figure 3-2 shows all the options that are available to the user for data generation using SCTGEN.

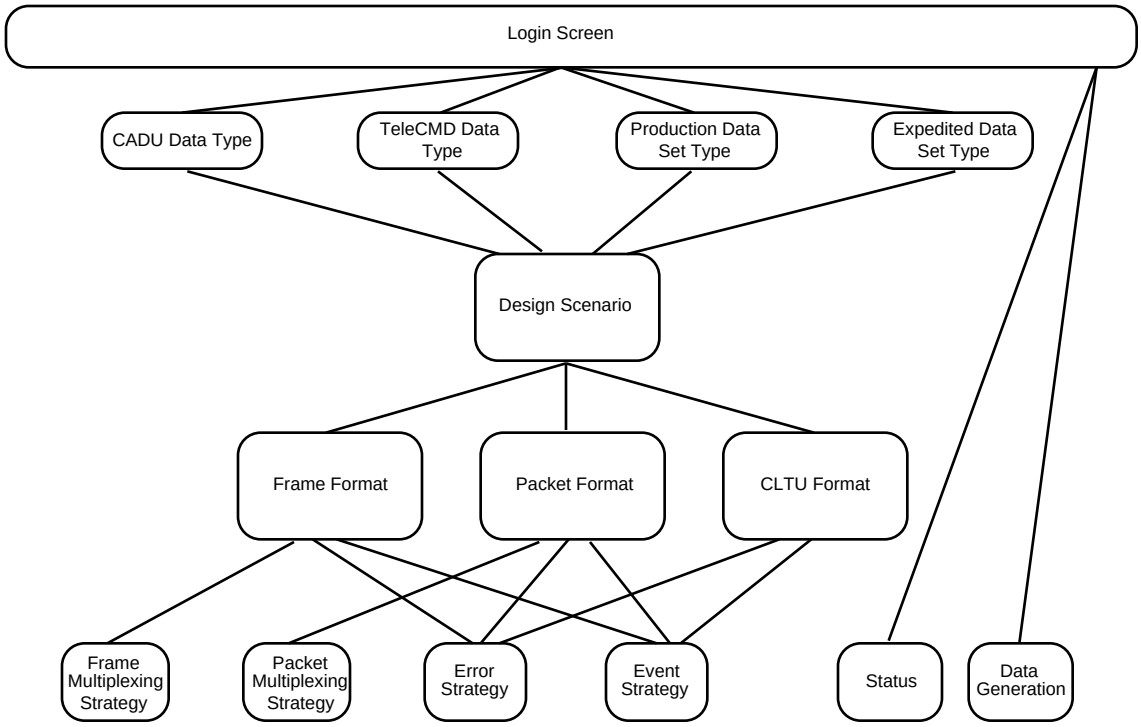

Figure 3-2. Menu Requirements for User Interface

3.2 TOP-LEVEL MENU DESCRIPTIONS

The SCTGEN GUI will run on the CDS, and as such, will utilize the X-Windows System and the Open Software Foundation (OSF) Motif toolkit. The main panel or window will allow the user to select the SCTGEN option. Available options are:

a. Generate Script.

b. Generate Output.

c. Insert Error.

However, each option is available only after test data has been designed and formatted as per the user's needs. Figure 3-3 provides the first menu of the SCTGEN system; the following paragraph describes one test data development sequence.

Figure 3-3. Main Graphical User Interface Panel

The first step is to develop a scenario for data generation. Clicking on the scenario icon will open a window that accepts the user input. The example illustrated in Figure 3-4 shows the input for the generation of a return link CADU stream. The information is the SCID, VCID, and APID. An additional piece of information is packet length.

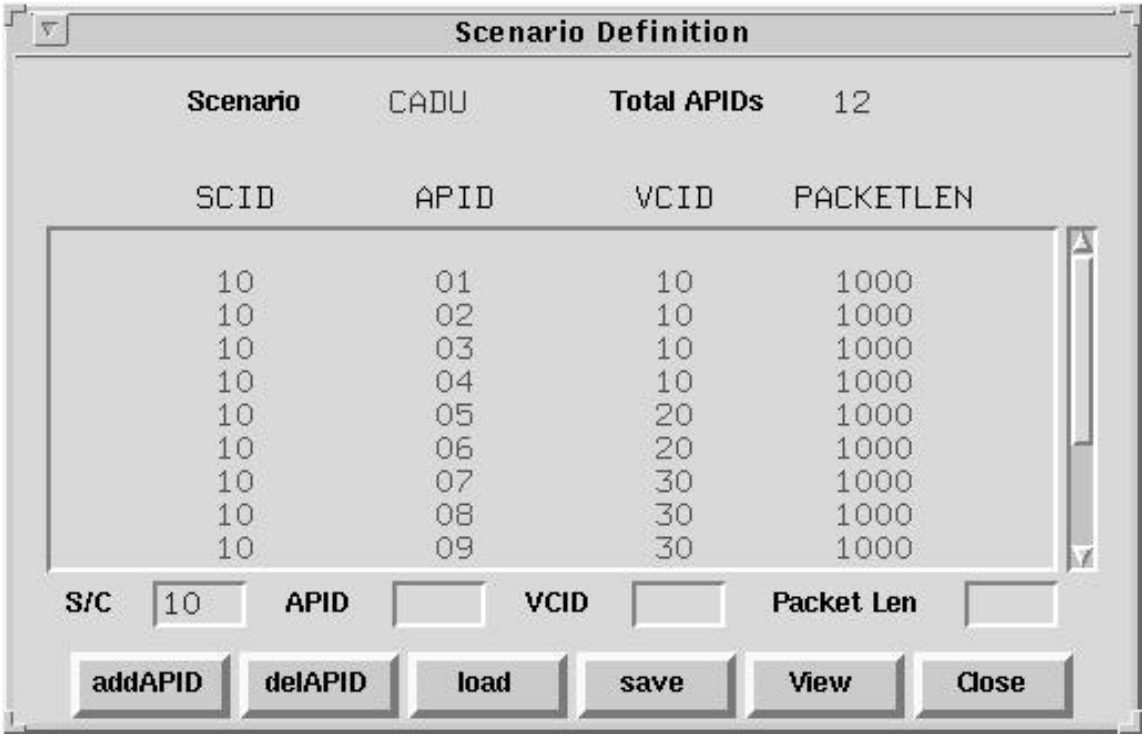

Figure 3-4. Data Scenario Menu Panel

A graphical description of the scenario is provided in Figure 3-5. From this point, each icon shown on the CADU stream menu will open additional detailed windows (for example, FrameMux will open the Frame Multiplexer Menu Panel, Figure 3-6, and will provide the user with the capability to generate the multiplexing strategy.) The user has the capability to interleave the different VCID streams entered in the data scenario. The strategy may be read from a pregenerated file called by the MUXtool File window, or may be entered by the user as a recurring pattern, or manually entered as a unique pattern.

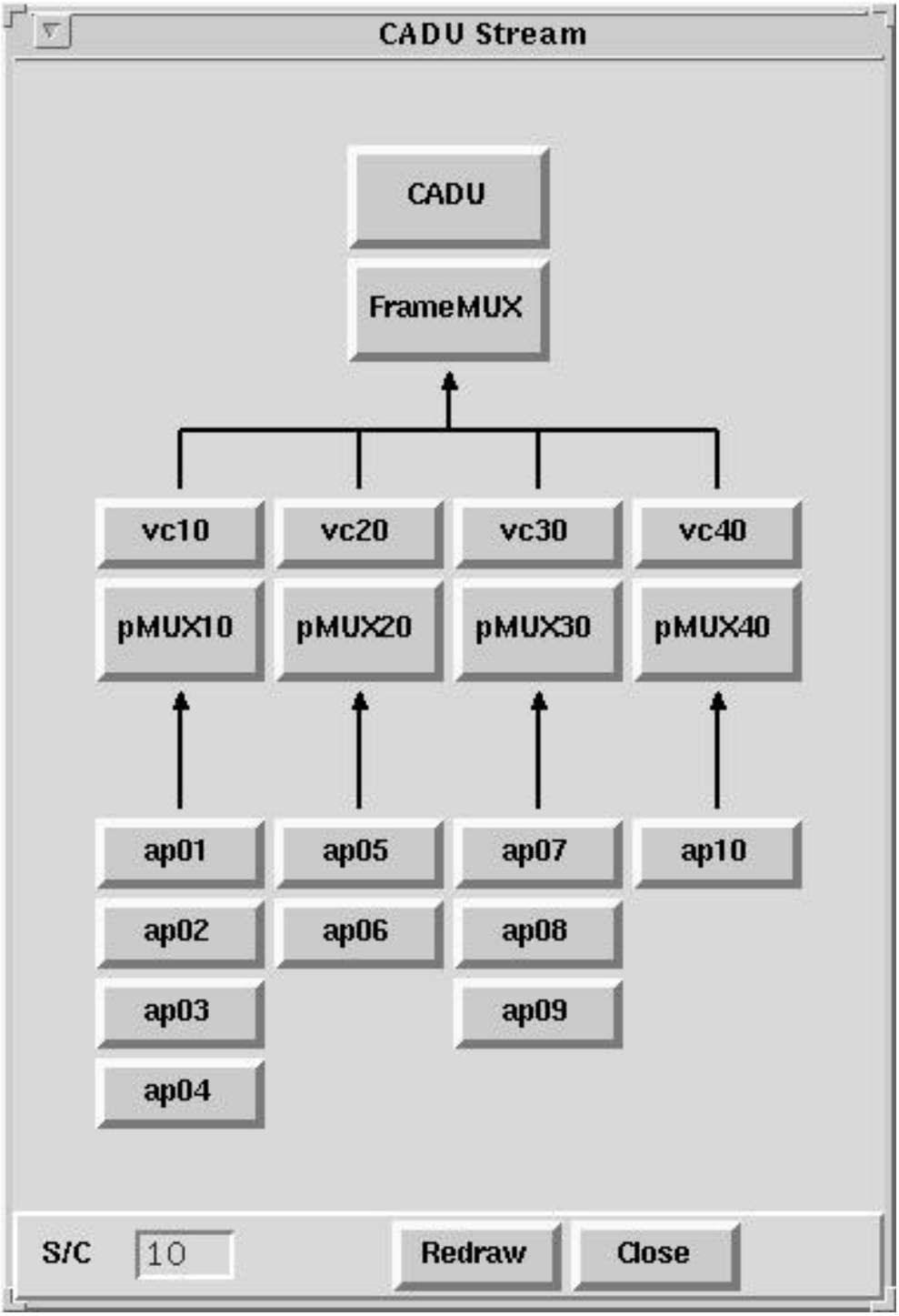

Figure 3-5. CADU Data Scenario Graphical Illustration

Clicking on the Frame icon in Figure 3-4 will open the Frame Formatting and Multiplexing Menu Panel shown in Figure 3-6. In this panel, the user can click on the VCIDs selected in the scenario and describe the different fields within the frames that constitute the VCID stream within the

CADU stream. Each VCID stream is so defined. In a similar manner, clicking on the pMUXxx icon directly under a specific VCID icon will open a menu panel that will provide the capability to interleave the APIDs contained within the VCID stream. The Packet icon in Figure 3-3 will enable the user to define packet headers. The top strip that shows script, output, and error are pull-down menus that provide the capabilities to view and generate scripts, generate test data, and provide menu panels for error insertion. The first run of the data generation will provide the option for the type of data, test data (CADUs), pseudo-data (self-check to verify that generated data matches design), data set (EDSs, PDSs, Rate-buffered Files [RBF]), and expected results (quality and accounting report).

The GUI for SCTGEN will be delivered with the final build, and will undergo modifications and optimization as the system is being developed.

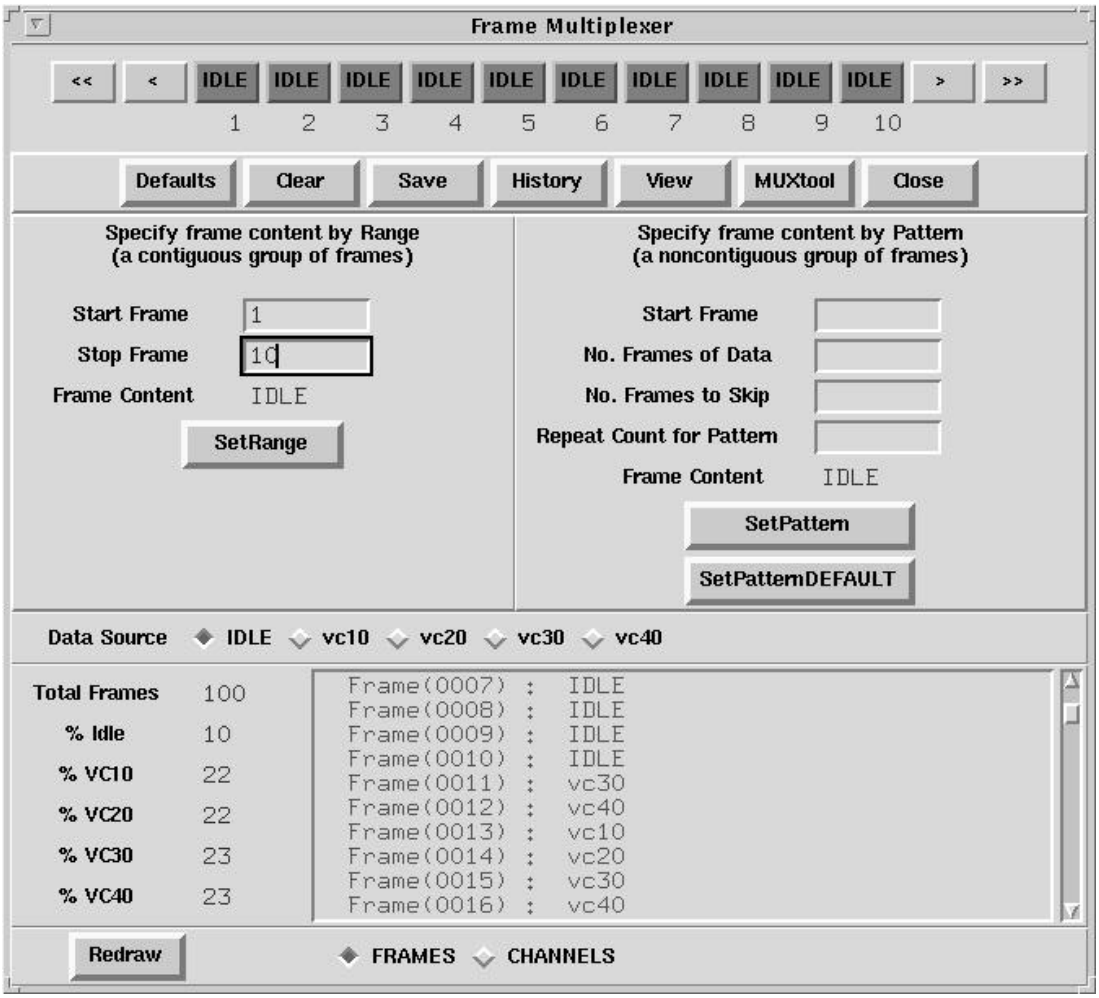

Figure 3-6. Frame Formatting and Multiplexing Menu Panel

3.3 TYPICAL SCRIPT AND DATA GENERATED

A typical script needed to run the SCTGEN engine for test data generation is shown below. This script was created using a standard UNIX editor, and will be generated by the GUI when it is completed.

This SCTGEN script will create 1,000 CADUs, and will be written to a plain, binary file called cadulist.dat. This scenario models the following situation:

```
vc      apid    pktLength

17      256     332
17      257     340
17      258     128
18      259     784
18      260     392
18      261     128
23      262     964
23      263     944
30      265     332
30      266     340
30      267     128
41      320     780
41      321     580
41      322     560
41      323     572
41      324     571
42      67      642
42      68      642
42      69      276
11      11      64
2       frame service
```

The packet length includes the 6-byte primary header. All packets will have a secondary header with CCSDS unsegmented time in the secondary header location. It will consist of the optional pfield, 2 bytes of coarse time, and 3 bytes of fine time, which makes a 6-byte secondary header. Each packet is filled with a constant pattern, which will be different for each application ID. Finally, a checksum in the last byte of each packet, which is the sum of all previous bytes modulo 256, could be used in post-processing to verify proper packet construction.

All frames (CADUs) are 1024 bytes long, and are Reed-Solomon encoded with interleave 4 and codeword length 255. Each frame has a 4-byte sync pattern of 0x1acffc1d. (This combination of interleave and codeword length forces 1020-byte frames. The sync pattern makes it 1024 bytes.) The OCF field (Command Link Control Word [CLCW]), insert zone, or any inversion or pseudo-noise encoding is not implemented.

The script shows a number of streams. Each one constructs or handles a unit (packet, frame, etc.). Although the list shows the streams from CADU makers down to packet makers, the order is not important, and can be arranged in any order. Streams are linked using a *stream* or *inStream* field. For example, the vc17 stream links to a vc17 mux, which interleaves packets from the ap256, ap257, and ap258 streams.

A stream definition consists of a dot string, such as *stream.mux.top* followed by an argument list. The individual fields in the dot string are called paths. The *stream* path identifies a mechanism that makes or handles units such as packets. The second path identifies the kind of stream. These are predefined names, such as *mux* or *cadu*. Case is important. The third path is a user-selected label that uniquely identifies the stream. No other stream may have the same name. The label is used to link streams.

The *main* line is mandatory. The single argument *c()* identifies which of the three CCSDS basic scenarios are to run. In this example, the *cadu* scenario has been selected. Each basic scenario has its own collection of unique streams.

There must be one output defined. In the example, output is a plain binary file. Output is writing up to 1,000 records to the *out* device, and it is getting records from the stream named *main*. There are several different output formats that could have been used; the device line allows the selection of different output devices. In the example, a substitute device, the null device, has been commented out. The null device discards all records, and is used to test the script and obtain an expected results file before running the real scenario.

```
main c(cadu)
output.plain device(out) inStream(top) max(1000)
device.file.out name(cadulist.dat) access(w)
device.null.out name(cadulist.dat) access(w)
```

A mux stream merges a unit from multiple input streams into one output stream. The following mux stream interleaves CADUs from virtual channel CADU streams. This example is taken from an actual test. The user wanted to create an interleave pattern for the first 100 frames and to have it repeat thereafter. The mux setup is more complicated, and the user may have instead selected to have the mux use an input file.

The first 14 frames out of this mux will be (by vcid): idle, 17, 17, 30, 30, 18, 18, 42, 42, 17, 17, 30, 30, 41. This mux consists of a number of recurrent patterns. For example, line 3 shows that vcid 17 starts with the second output frame and repeats once. This means that output frames 2 and 3 will be from input stream *vc17*. The *span(100)* argument specifies that it repeat every 100 frames, so vc17 will occur again at frames 102 and 103, 202 and 203, etc. Line 4 specifies that frames 4 and 5 will come from vcid 30; line 5 specifies frames 6 and 7 will come from vcid 18. The remaining *recur* lines define VCIDs for the remaining frames in the scenario.

The second line defines an idle frame. (There are other idle definitions in the list.) It specifies that the first frame is idle, and so are 101, 201, 301, etc. The idle argument, which is zero, is the bit length of the idle frame. Because all idle frames are the same fixed size, SCTGEN ignores the value, but it must exist.

The first line has three fields. *er(1)* turns on expected results output. The expected results file will contain unit IDs (frame numbers) and corresponding VCIDs for all 1000 frames. By default, this is off.

default(vc17) specifies that the mux use input stream *vc17* if a unit ID is undefined. For example, the mux would use the default if an input stream for the 100th frame has not been specified. It would also use the default if an input stream ran out of frames. For example, if *vc30* was programmed to make only 10 frames, the mux would get frames from the default for *vc30* after it used up the 10 frames. If not specified as default, or if the default itself ran out of units, the mux would terminate upon its next attempt to get a unit from the default.

idle(vc17) specifies from which stream to get idle units. Usually, it does not matter which stream provides because they all can make idles. However, if a CADU stream that used an insert zone was selected, all idles from that stream would also have an insert zone. For this example, no CADU stream is using the insert zone, so any CADU stream will be sufficient.

```
stream.mux.top idle(vc17) er(1) default(vc17)
stream.mux.top.recur idle(0) start(1) repeat(0) span(100)
stream.mux.top.recur stream(vc17) start(2) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(4) repeat(1) span(100)
stream.mux.top.recur stream(vc18) start(6) repeat(1) span(100)
stream.mux.top.recur stream(vc42) start(8) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(10) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(12) repeat(1) span(100)
stream.mux.top.recur stream(vc41) start(14) repeat(0) span(100)
stream.mux.top.recur stream(vc18) start(15) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(17) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(19) repeat(1) span(100)
stream.mux.top.recur stream(vc41) start(21) repeat(1) span(100)
stream.mux.top.recur stream(vc23) start(23) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(25) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(27) repeat(1) span(100)
stream.mux.top.recur stream(vc18) start(29) repeat(1) span(100)
stream.mux.top.recur stream(vc42) start(31) repeat(1) span(100)
stream.mux.top.recur stream(vc41) start(33) repeat(0) span(100)
stream.mux.top.recur stream(vc30) start(34) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(36) repeat(1) span(100)
stream.mux.top.recur stream(vc23) start(38) repeat(1) span(100)
stream.mux.top.recur stream(vc18) start(40) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(42) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(44) repeat(1) span(100)
stream.mux.top.recur stream(vc41) start(46) repeat(1) span(100)
stream.mux.top.recur stream(vc23) start(48) repeat(0) span(100)
stream.mux.top.recur stream(vc42) start(49) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(51) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(53) repeat(1) span(100)
stream.mux.top.recur stream(vc18) start(55) repeat(1) span(100)
stream.mux.top.recur stream(vc23) start(57) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(59) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(61) repeat(1) span(100)
stream.mux.top.recur stream(vc11) start(63) repeat(0) span(100)
stream.mux.top.recur idle(0) start(64) repeat(1) span(100)
stream.mux.top.recur stream(vc11) start(65) repeat(1) span(100)
stream.mux.top.recur stream(vc18) start(66) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(68) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(70) repeat(1) span(100)
stream.mux.top.recur stream(vc42) start(72) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(74) repeat(3) span(100)
stream.mux.top.recur stream(vc17) start(78) repeat(1) span(100)
stream.mux.top.recur stream(vc18) start(80) repeat(1) span(100)
stream.mux.top.recur stream(vc42) start(82) repeat(1) span(100)
stream.mux.top.recur idle(0) start(84) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(85) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(87) repeat(1) span(100)
stream.mux.top.recur stream(vc41) start(89) repeat(0) span(100)
stream.mux.top.recur stream(vc18) start(90) repeat(2) span(100)
stream.mux.top.recur stream(vc30) start(93) repeat(1) span(100)
stream.mux.top.recur stream(vc17) start(95) repeat(1) span(100)
stream.mux.top.recur stream(vc30) start(97) repeat(0) span(100)
stream.mux.top.recur stream(vc2) start(98) repeat(2) span(100)
```

This stream makes CADUs for virtual channel 17. There are similar streams for the other virtual channels below. Path service has been selected. The spacecraft ID is 42. Data is to be Reed-Solomon encoded with interleave 4 and codeword length 255. The length is interleave * codeword length + sync length (1024 is this example).

The second *vc17* line specifies to the stream how to fill its data region. If an insert zone or an operational control field (OCF or CLCW) was specified, similar lines will be needed for those regions. The *vc17* data region is filled using type C, which is the consumer loader type. It fills the region with units from another stream. In this case, packets are filled from the *pmux17* input stream, which is a mux for three packet streams. The loader will load packets back-to-back, splitting them between CADUs if necessary.

If the input stream runs out of packets, the consumer loader, by default, will put a constant fill pattern of 0xC9 into the data region to fill any remaining unused bytes. The field *lastUnit(I)* does just that. If the input stream runs out of packets, the consumer loader will request idle packets. This will only happen in the last CADU produced by this stream. The loader and the stream will terminate after providing this last CADU.

The *ERcomposition(1)* field is an optional field that causes the consumer loader to write unit composition information to the expected results file. By default it is off because it can produce lots of output. When on, the expected results file will show each CADU and the identity of all packets in it. It will also show the first header pointer value and the number of bytes per packet fragment when a CADU contains part of a split packet.

```
stream.cadu.vc17 service(P) vcid(17) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc17.region.data type(C) ERcomposition(1) inStream(pmux17) -
.        lastUnit(I)
```

This mux stream interleaves the packets from three packet streams for input to *vc17*. Note that this mux uses a *range* specification. In this example, this means that the first 10 packets (1-10) come from *ap256*. In addition, packet 500 will also come from *ap256*. The other streams use a recurrent pattern. The mux will get packets from *ap257* for packets 9-16, 19-26, 29-36, etc. It will get packets from *ap258* for packets 17-18, 27-28, 37-38, etc. Note that some unit IDs, such as 9, are in more than one list.

To resolve conflicts, a mux searches through the list in a specific order and uses the stream from the first match that it finds. A mux always searches through range definitions before it searches through recurrent ones. To avoid confusion, list ranges before recurs. Within ranges and recurs, a mux searches them in the order they are listed in the script. If it still has not found a match, it uses the default; if that fails (it does not exist or it ran out of units), the mux terminates. In the example, this mux gets packets 9, 10, and 500 from *ap256* because they are range lists, and it uses ranges over recurs.

```
stream.mux.pmux17 default(ap256) idle(ap256)
stream.mux.pmux17.range stream(ap256) uid0(1) uid1(10)
stream.mux.pmux17.range stream(ap256) uid0(500) uid1(500)
stream.mux.pmux17.recur stream(ap257) start(9) repeat(7) skip(2)
           stream.mux.pmux17.recur stream(ap258) start(17) repeat(2) skip(8)
```

This stream makes packets for application ID 256. Each packet is 332 bytes long, which includes the primary header length. It was decided to put a 6-byte secondary header in each packet, and a checksum at the end of each packet. The checksum is the sum of all bytes in the packet modulo 256. Each packet has a data region and a secondary header region, so lines must be included to tell the packet stream how to fill them.

The data region is filled with a constant pattern of 0x11. Type F is a constant pattern loader. For the secondary header region, using the CCSDS unsegmented timecode. The timecode will increment by the step from packet to packet.

```
stream.pkt.ap256 appid(256) length(332) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap256.region.data type(F) pattern(0x11)
stream.pkt.ap256.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
          fineBytes(3) stepFine(200)
```

This stream makes packets for application ID 257.

```
stream.pkt.ap257 appid(257) length(340) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap257.region.data type(F) pattern(0x22)
stream.pkt.ap257.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
           fineBytes(3) stepFine(200)
```

This stream makes packets for application ID 258. In this stream, the 30th packet will be made a 40-byte idle packet.

```
stream.pkt.ap258 appid(258) length(128) 2hdr(1) 2hdrLength(6) checksum(1) -
               idleEvent(ap258idle)
stream.pkt.ap258.region.data type(F) pattern(0x33)
stream.pkt.ap258.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
               fineBytes(3) stepFine(200)
event.value.ap258idle.range uid0(30) uid1(30) v(40)
```

This stream makes CADUs for virtual channel 18, and its layout is almost identical to that of *vc17*. In this example, however, it will demonstrate how to introduce errors at the frame and packet levels.

There are two error specifications in this CADU stream. In the first, plan to invert bit 2 (a bit in the spacecraft ID field) in the second CADU. In the second, plan to invert 5 bits (bits 100-104) in CADUs 5, 105, 205, 305, and 405.

The *convey* field defines how and when the error is applied. When convey is on, the error is conveyed to every receiving stream in the pipeline. For example, if a conveyable error is present in a packet, the receiving frame stream and other receivers will behave as if the error were introduced at that level. Note that the stream applies the error after it has encoded the unit when convey is on. If a bit error is put in a packet and piped in a frame stream that was Cyclical Redundancy Check (CRC) encoding frames, the affected frame would have a CRC error because the stream applied the error after it encoded the frame. If the frame was then piped into a NASA Communications (Nascom) block stream, that Nascom block would also have a CRC error. In all cases, the streams apply the error after encoding when convey is on.

When convey is off, the stream that applies the error hides it from all receiving streams. This means the error is applied before encoding occurs. The error may be thought of as an instrument error, as opposed to a transmission error. For example, if a nonconveyable error is put in a packet, CRC or Reed-Solomon errors will not be seen in any frames containing that packet.

In summary, when convey is on all receiving streams reflect the error, and the error is applied after encoding. When convey is off, only the original stream shows the error, and the error is applied before encoding.

In this example, the spacecraft ID error in the second CADU is nonconveyable, which means it is applied before Reed Solomon encoding. The target processor should detect a spacecraft ID error. The other errors to bits 100-104 in the other five frames are conveyable, so errors are applied after Reed Solomon encoding. This means that the target processor should see Reed Solomon errors, and they are correctable. The target processor should correct the errors in the five frames. The label field in both errors is an information tag that the stream writes to the expected results file.

```
stream.cadu.vc18 service(P) vcid(18) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc18.region.data type(C) ERcomposition(1) inStream(pmux18) -
               lastUnit(I)
stream.cadu.vc18.error.flip convey(0) label(spidError) event(ev0) -
                 startbit(2) bits(1)
event.unit.ev0.range uid0(2) uid1(2)

stream.cadu.vc18.error.flip convey(1) label(rsError) event(ev1) -
                 startbit(100) bits(5)
event.unit.ev1.recur start(5) skip(100) occur(5)

stream.mux.pmux18 default(ap259) idle(ap256)
stream.mux.pmux18.recur stream(ap259) start(1) repeat(5) span(13)
stream.mux.pmux18.recur stream(ap260) start(7) repeat(5) span(13)
stream.mux.pmux18.recur stream(ap261) start(13) repeat(0) span(13)

stream.pkt.ap259 appid(259) length(784) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap259.region.data type(F) pattern(0x44)
stream.pkt.ap259.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
               fineBytes(3) stepFine(200)
```

The following packet stream makes packets for application ID 260. In this example, the second packet is dropped, which should cause a gap in the target processor. The expected results file will also note a gap. Notice that event *ev0*, is shared, which was defined above to introduce a CADU error. Events can be shared.

```
stream.pkt.ap260 appid(260) length(392) 2hdr(1) 2hdrLength(6) checksum(1) -
                drop(ev0)
stream.pkt.ap260.region.data type(F) pattern(0x55)
stream.pkt.ap260.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
               fineBytes(3) stepFine(200)
```

The following packet stream makes packets for application ID 261. Value 1 will be deposited into the packet data length field for packets 10-12. The error is nonconveyable, which means it is hidden from the CADU. In general, packet errors should not be conveyed to a CADU stream that is Reed-Solomon encoding frames because the target processor will probably remove them when the Reed-Solomon decoder detects and corrects the errors. The user will receive Reed-Solomon corrected frames instead of planned errors.

```
stream.pkt.ap261 appid(261) length(128) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap261.region.data type(F) pattern(0x66)
stream.pkt.ap261.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)
stream.pkt.ap261.error.set convey(0) label(length) event(ap261e) -
                              startbit(32) bits(16) v(1)
event.unit.ap261e.range uid0(10) uid1(12)

stream.cadu.vc23 service(P) vcid(23) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc23.region.data type(C) ERcomposition(1) inStream(pmux23) -
              lastUnit(I)

stream.mux.pmux23 default(ap262) idle(ap256)
stream.mux.pmux23.recur stream(ap262) start(1) skip(1)
stream.mux.pmux23.recur stream(ap263) start(2) skip(1)

stream.pkt.ap262 appid(262) length(964) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap262.region.data type(F) pattern(0x77)
stream.pkt.ap262.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)
```

The following packet stream creates packets for application ID 263. It will create only 100 packets. The mux will use the default after this stream provides 100 packets.

```
stream.pkt.ap263 appid(263) length(944) 2hdr(1) 2hdrLength(6) checksum(1) -
                max(100)
stream.pkt.ap263.region.data type(F) pattern(0x88)
stream.pkt.ap263.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.cadu.vc30 service(P) vcid(30) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc30.region.data type(C) ERcomposition(1) inStream(pmux30) -
              lastUnit(I)

stream.mux.pmux30 default(ap265) idle(ap256)
stream.mux.pmux30.recur stream(ap265) start(1) repeat(3) span(17)
stream.mux.pmux30.recur stream(ap266) start(5) repeat(3) span(17)
stream.mux.pmux30.recur stream(ap265) start(9) repeat(3) span(17)
stream.mux.pmux30.recur stream(ap266) start(13) repeat(1) span(17)
stream.mux.pmux30.recur stream(ap267) start(15) repeat(0) span(17)
stream.mux.pmux30.recur stream(ap266) start(16) repeat(1) span(17)

stream.pkt.ap265 appid(265) length(332) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap265.region.data type(F) pattern(0x99)
stream.pkt.ap265.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.pkt.ap266 appid(266) length(340) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap266.region.data type(F) pattern(0xaa)
stream.pkt.ap266.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)
```

```
stream.pkt.ap267 appid(267) length(128) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap267.region.data type(F) pattern(0xbb)
stream.pkt.ap267.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.cadu.vc41 service(P) vcid(41) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc41.region.data type(C) ERcomposition(1) inStream(pmux41) -
              lastUnit(I)

stream.mux.pmux41 default(ap320) idle(ap256)
stream.mux.pmux41.recur stream(ap320) start(1) repeat(0) skip(4)
stream.mux.pmux41.recur stream(ap321) start(2) repeat(0) skip(4)
stream.mux.pmux41.recur stream(ap322) start(3) repeat(0) skip(4)
stream.mux.pmux41.recur stream(ap323) start(4) repeat(0) skip(4)
stream.mux.pmux41.recur stream(ap324) start(5) repeat(0) skip(4)

stream.pkt.ap320 appid(320) length(780) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap320.region.data type(F) pattern(0xcc)
stream.pkt.ap320.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.pkt.ap321 appid(321) length(580) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap321.region.data type(F) pattern(0xdd)
stream.pkt.ap321.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.pkt.ap322 appid(322) length(560) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap322.region.data type(F) pattern(0xee)
stream.pkt.ap322.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.pkt.ap323 appid(323) length(572) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap323.region.data type(F) pattern(0xff)
stream.pkt.ap323.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.pkt.ap324 appid(324) length(571) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap324.region.data type(F) pattern(0xa5)
stream.pkt.ap324.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.cadu.vc42 service(P) vcid(42) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc42.region.data type(C) ERcomposition(1) inStream(pmux42) -
              lastUnit(I)

stream.mux.pmux42 default(ap67) idle(ap256)
stream.mux.pmux42.recur stream(ap67) start(1) repeat(2) skip(4)
stream.mux.pmux42.recur stream(ap68) start(4) repeat(2) skip(4)
stream.mux.pmux42.recur stream(ap69) start(7) repeat(0) skip(6)

stream.pkt.ap67 appid(67) length(642) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap67.region.data type(F) pattern(0x67)
stream.pkt.ap67.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.pkt.ap68 appid(68) length(642) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap68.region.data type(F) pattern(0x68)
stream.pkt.ap68.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)
```

```
stream.pkt.ap69 appid(69) length(276) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap69.region.data type(F) pattern(0x69)
stream.pkt.ap69.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)

stream.cadu.vc11 service(P) vcid(11) spid(42) frameSync(0x1acffc1d) -
                 RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc11.region.data type(C) ERcomposition(1) inStream(ap11) -
              lastUnit(I)

stream.pkt.ap11 appid(11) length(64) 2hdr(1) 2hdrLength(6) checksum(1)
stream.pkt.ap11.region.data type(F) pattern(0x88)
stream.pkt.ap11.region.2hdr type(tccuc) stepSeconds(0) coarseBytes(2) -
              fineBytes(3) stepFine(200)
```

This stream makes CADUs using a frame service (Virtual Channel Access [VCA]/VCDU) for virtual channel 2. A 12-byte insert zone is used in every CADU. The data region will be filled with a random pattern. The insert zone will be filled with the user pattern 0x00faf30502ffd30403fc1303. However, an 8-bit sequence number will be put in the first byte of the insert zone.

```
stream.cadu.vc2 service(V) vcid(2) spid(42) frameSync(0x1acffc1d) -
                insertZone(12) RSencode(1) RSinterleave(4) RScodeLength(255)
stream.cadu.vc2.region.data type(A)
stream.cadu.vc2.region.insertZone type(U) wrap(0) format(H)
stream.cadu.vc2.region.insertZone.data s0(0x00) s1(0xfa) s2(0xf3) s3(0x05)
stream.cadu.vc2.region.insertZone.data s0(0x02) s1(0xff) s2(0xd3) s3(0x04)
stream.cadu.vc2.region.insertZone.data s0(0x03) s1(0xfc) s2(0x13) s3(0x03)
stream.cadu.vc2.task.sequence event(1) startbit(80) bits(8)
```

The resulting data generated when the script was run is shown below. Test data contains CADUs from eight VCIDs with packets in VCIDs 11, 17, 18, 23, 30, 41, and 42, and frame-level data in VCID 2. Note that the entire data stream is not replicated—only the first two and a part of the third to show the product. The first CADU is for FILL; the other two CADUs are for VCID17, which is packet service. The synchronization pattern used in this test case was 1ACFFC1D.

```
00000000:  1ACFFC1D 4ABF0000 0000C9C9 C9C9C9C9  "....J..........."
00000010:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000020:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000030:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000040:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000050:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000060:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000070:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000080:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000090:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000000A0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000000B0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000000C0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000000D0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000000E0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000000F0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000100:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000110:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000120:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000130:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000140:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000150:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000160:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000170:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000180:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000190:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000001A0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000001B0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000001C0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000001D0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
```

```
000001E0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000001F0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000200:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000210:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000220:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000230:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000240:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000250:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000260:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000270:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000280:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000290:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000002A0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000002B0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000002C0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000002D0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000002E0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
000002F0:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000300:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000310:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000320:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000330:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000340:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000350:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000360:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000370:  C9C9C9C9 C9C9C9C9 C9C9C9C9 C9C9C9C9  "................"
00000380:  19EC0000 C0E59A9A 5D20A1A1 2DFA8585  "........] ..-..."
00000390:  C1DB9B9B D2C23E3E 7280D6D6 3F53E3E3  "......>>r...?S.."
000003A0:  6D005656 5F21CACA E23DFEFE F4A65757  "m.VV_!...=....WW"
000003B0:  85CAD3D3 5F888585 81F56F6F DC5C1313  "...._.....oo.\.."
000003C0:  D9AE0404 B7371313 17636F6F E3348585  ".....7...coo.4.."
000003D0:  5718D3D3 D98B5757 7FA0FEFE F789CACA  "W.....WW........"
000003E0:  44295656 BFD3E3E3 AF5DD6D6 9F8F3E3E  "D)VV.....]....>>"
000003F0:  7A609B9B 17C08585 423FA1A1 61449A9A  "z`......B?..aD.."
00000400:  1ACFFC1D 4A910000 00000000 0900C000  "....J..........."
00000410:  01451700 00000000 11111111 11111111  ".E.............."
00000420:  11111111 11111111 11111111 11111111  "................"
00000430:  11111111 11111111 11111111 11111111  "................"
00000440:  11111111 11111111 11111111 11111111  "................"
00000450:  11111111 11111111 11111111 11111111  "................"
00000460:  11111111 11111111 11111111 11111111  "................"
00000470:  11111111 11111111 11111111 11111111  "................"
00000480:  11111111 11111111 11111111 11111111  "................"
00000490:  11111111 11111111 11111111 11111111  "................"
000004A0:  11111111 11111111 11111111 11111111  "................"
000004B0:  11111111 11111111 11111111 11111111  "................"
000004C0:  11111111 11111111 11111111 11111111  "................"
000004D0:  11111111 11111111 11111111 11111111  "................"
000004E0:  11111111 11111111 11111111 11111111  "................"
000004F0:  11111111 11111111 11111111 11111111  "................"
00000500:  11111111 11111111 11111111 11111111  "................"
00000510:  11111111 11111111 11111111 11111111  "................"
00000520:  11111111 11111111 11111111 11111111  "................"
00000530:  11111111 11111111 11111111 11111111  "................"
00000540:  11111111 11111111 11111111 11111111  "................"
00000550:  11111111 11111155 0900C001 01451700  ".......U.....E.."
00000560:  000000C8 11111111 11111111 11111111  "................"
00000570:  11111111 11111111 11111111 11111111  "................"
00000580:  11111111 11111111 11111111 11111111  "................"
00000590:  11111111 11111111 11111111 11111111  "................"
000005A0:  11111111 11111111 11111111 11111111  "................"
000005B0:  11111111 11111111 11111111 11111111  "................"
000005C0:  11111111 11111111 11111111 11111111  "................"
000005D0:  11111111 11111111 11111111 11111111  "................"
000005E0:  11111111 11111111 11111111 11111111  "................"
000005F0:  11111111 11111111 11111111 11111111  "................"
00000600:  11111111 11111111 11111111 11111111  "................"
00000610:  11111111 11111111 11111111 11111111  "................"
00000620:  11111111 11111111 11111111 11111111  "................"
00000630:  11111111 11111111 11111111 11111111  "................"
00000640:  11111111 11111111 11111111 11111111  "................"
00000650:  11111111 11111111 11111111 11111111  "................"
```

```
00000660:  11111111 11111111 11111111 11111111  "................"
00000670:  11111111 11111111 11111111 11111111  "................"
00000680:  11111111 11111111 11111111 11111111  "................"
00000690:  11111111 11111111 11111111 11111111  "................"
000006A0:  1111111E 0900C002 01451700 00000190  "........E......."
000006B0:  11111111 11111111 11111111 11111111  "................"
000006C0:  11111111 11111111 11111111 11111111  "................"
000006D0:  11111111 11111111 11111111 11111111  "................"
000006E0:  11111111 11111111 11111111 11111111  "................"
000006F0:  11111111 11111111 11111111 11111111  "................"
00000700:  11111111 11111111 11111111 11111111  "................"
00000710:  11111111 11111111 11111111 11111111  "................"
00000720:  11111111 11111111 11111111 11111111  "................"
00000730:  11111111 11111111 11111111 11111111  "................"
00000740:  11111111 11111111 11111111 11111111  "................"
00000750:  11111111 11111111 11111111 11111111  "................"
00000760:  11111111 11111111 11111111 11111111  "................"
00000770:  11111111 11111111 11111111 11111111  "................"
00000780:  1B4EF700 308E0BFF E351EAAE C7451E2F  ".N..0....Q...E./"
00000790:  EEF53373 767DA91A 50E97D5D 905415BB  "..3sv}..P.}].T.."
000007A0:  178EA7C4 F69EF661 2DF205F9 05405F31  ".......a-....@_1"
000007B0:  B7DF2250 2971BF23 BDB5B677 229FED2F  "..."P)q.#...w"../"
000007C0:  B3434B97 F7CA7729 C5F532D9 CA0EA597  ".CK...w)..2....."
000007D0:  C4101469 6AB59DE5 116B04E1 A5FE327F  "...ij....k....2."
000007E0:  CCA2B68E 7052148A 0FD72166 4EF9FBB0  "....pR....!fN..."
000007F0:  6D93B931 055094FD 04EC3968 364270DE  "m..1.P....9h6Bp."
00000800:  1ACFFC1D 4A910000 01000070 11111111  "....J......p...."
00000810:  11111111 11111111 11111111 11111111  "................"
```

The following paragraphs describe the expected results file that is generated when *tccsds* is run with the script *caduSample.script*. This file is designed to be interpreted by software, so it is not very user-friendly. It is also not sorted because the events are written chronologically. It is easier to read when run through a sort utility.

The information is arranged into three columns. Column 1 is a keyword to the type of information that follows. Column 2 is a stream ID and indicates the stream that provided the information. Stream IDs are greater than zero, but occasionally a zero will be seen in the stream ID field. This means a nonstream mechanism, such as the output class, reported the information. Column 3 contains the information, which depends on information type.

The *A* type shows the stream type and stream name associated with a stream ID. For example, stream ID 1 is the mux stream *top*. All stream IDs are identified. NOTEs usually appear before *A* lines, but have been rearranged here for convenience.

```
A     1     mux    top
A     2     mux    pmux17
A     3     mux    pmux18
A     4     mux    pmux23
A     5     mux    pmux30
A     6     mux    pmux41
A     7     mux    pmux42
A     8     cadu   vc17
A     9     cadu   vc18
A     10    cadu   vc23
A     11    cadu   vc30
A     12    cadu   vc41
A     13    cadu   vc42
A     14    cadu   vc11
A     15    cadu   vc2
A     16    pkt    ap256
A     17    pkt    ap257
A     18    pkt    ap258
A     19    pkt    ap259
A     20    pkt    ap260
A     21    pkt    ap261
A     22    pkt    ap262
A     23    pkt    ap263
A     24    pkt    ap265
A     25    pkt    ap266
A     26    pkt    ap267
A     27    pkt    ap320
A     28    pkt    ap321
A     29    pkt    ap322
A     30    pkt    ap323
A     31    pkt    ap324
A     32    pkt    ap67
A     33    pkt    ap68
A     34    pkt    ap69
A     35    pkt    ap11
```

NOTE provides additional information about a stream. Not all streams show NOTE information.

```
NOTE   8      CADU spid=42 vcid=17 size=1024 service=P RS=4,255
NOTE   9      CADU spid=42 vcid=18 size=1024 service=P RS=4,255
NOTE   10     CADU spid=42 vcid=23 size=1024 service=P RS=4,255
NOTE   11     CADU spid=42 vcid=30 size=1024 service=P RS=4,255
NOTE   12     CADU spid=42 vcid=41 size=1024 service=P RS=4,255
NOTE   13     CADU spid=42 vcid=42 size=1024 service=P RS=4,255
NOTE   14     CADU spid=42 vcid=11 size=1024 service=P RS=4,255
NOTE   15     CADU spid=42 vcid=2 size=1024 service=V RS=4,255 insertZone
NOTE   16     appid=256 pktlength=332 hdr2Length=6
NOTE   17     appid=257 pktlength=340 hdr2Length=6
NOTE   18     appid=258 pktlength=128 hdr2Length=6
NOTE   19     appid=259 pktlength=784 hdr2Length=6
NOTE   20     appid=260 pktlength=392 hdr2Length=6
NOTE   21     appid=261 pktlength=128 hdr2Length=6
NOTE   22     appid=262 pktlength=964 hdr2Length=6
NOTE   23     appid=263 pktlength=944 hdr2Length=6
NOTE   24     appid=265 pktlength=332 hdr2Length=6
NOTE   25     appid=266 pktlength=340 hdr2Length=6
NOTE   26     appid=267 pktlength=128 hdr2Length=6
NOTE   27     appid=320 pktlength=780 hdr2Length=6
NOTE   28     appid=321 pktlength=580 hdr2Length=6
NOTE   29     appid=322 pktlength=560 hdr2Length=6
NOTE   30     appid=323 pktlength=572 hdr2Length=6
NOTE   31     appid=324 pktlength=571 hdr2Length=6
NOTE   32     appid=67 pktlength=642 hdr2Length=6
NOTE   33     appid=68 pktlength=642 hdr2Length=6
NOTE   34     appid=69 pktlength=276 hdr2Length=6
NOTE   35     appid=11 pktlength=64 hdr2Length=6
```

Lines beginning with *c* are unit composition lines. They may not appear unless the script turns on *ERcomposition* for a stream. A composition line shows which units are inside another unit.

The following line was provided by stream #8. Under the *A* lines, stream #8 is the CADU maker *vc17* for virtual channel 17. NOTE shows the CADU is 1024 bytes long, Reed Solomon encoded, and uses the path service. *uid=1* shows that this is unit ID #1, which is the first CADU this stream made. The first header pointer is zero, so the first packet began with the first byte of the data region. This CADU contains packets 16.1, 16.2, and 16.3. 16 is the stream ID, which is the packet maker for application ID 256. (See 16 in the *A* list.) From the NOTE for 16, it is seen that these packets are 332 bytes long including all headers. The numbers to the right of the point are the unit numbers. Therefore, this line shows that CADU #1 for vc17 contains packets 1, 2, and 3 from stream 16, which is the packet generator *ap256*. The *last=220* shows that the last packet did not fit into the CADU, so only 220 bytes of it are in this frame; the rest are in the next one.

```
c      8      uid=1 fhp=0 16.1 16.2 16.3 last=220
```

This is the second frame (uid=2) for stream #8, *vc17*. The first header pointer is 112, which accounts for the remaining 112 bytes of packet 16.3. This CADU contains packets 3-6 from stream 16, *ap256*. The last packet did not completely fit, so only 108 bytes are in this frame.

```
c      8      uid=2 fhp=112 16.3 16.4 16.5 16.6 last=108

c      11     uid=1 fhp=0 24.1 24.2 24.3 last=220
c      11     uid=2 fhp=112 24.3 24.4 25.1 25.2 last=100
c      9      uid=1 fhp=0 19.1 19.2 last=100
```

This is the second unit of stream 9, which makes it the second CADU of *vc18*. It contains the last 684 bytes of packet #2 and the first 200 bytes of packet #3 from stream 19, which is packet generator *ap259*. The *err* line shows that CADU #2 has an error. The error label is *spidError*. In the script, this is a nonconveyable error that was applied to the spacecraft ID.

```
c       9       uid=2 fhp=684 19.2 19.3 last=200
err     9       uid=2 spidError

c       13      uid=1 fhp=0 32.1 32.2 last=242
c       13      uid=2 fhp=400 32.2 32.3 last=484
c       8       uid=3 fhp=224 16.6 16.7 16.8 last=328
c       8       uid=4 fhp=4 16.8 16.9 16.10 17.1 last=216
c       11      uid=3 fhp=240 25.2 25.3 25.4 last=304
c       11      uid=4 fhp=36 25.4 24.5 24.6 24.7 last=184
c       12      uid=1 fhp=0 27.1 28.1 last=104
c       9       uid=3 fhp=584 19.3 19.4 last=300
c       9       uid=4 fhp=484 19.4 19.5 last=400
c       11      uid=5 fhp=148 24.7 24.8 25.5 25.6 last=64
```

The *mux* line is an optional report that a mux provides. The expected results output was turned on for the *top* mux in the script. Stream #1 is the *top* mux. The information shows how the CADUs were interleaved. The *uid0=1* fields shows the output unit number for the first unit identified on this line. Unit 1.1/I is the first unit output by this mux. *1.1/I* shows this is the first CADU made by stream #1, which is the *top* mux stream. Because mux streams do not make units, this must be an idle frame. */I* confirms it because this means idle. Output units 2 and 3 are CADUs 1 and 2 from stream 8, which is *vc17*. This is followed by CADUs 1 and 2 from stream 11 and CADUs 1 and 2 from stream 9. Note *9.2/E*, which means CADU 2 from stream 19 had an error.

```
mux     1       uid0=1 1.1/I 8.1-8.2 11.1-11.2 9.1 9.2/E 13.1-13.2 8.3-8.4 11.3-11.4
12.1 9.3-9.4

c       11      uid=6 fhp=276 25.6 26.1 25.7 25.8 last=140
c       8       uid=5 fhp=124 17.1 17.2 17.3 17.4 last=80
c       8       uid=6 fhp=260 17.4 17.5 17.6 last=284
c       12      uid=2 fhp=476 28.1 29.1 last=408
c       12      uid=3 fhp=152 29.1 30.1 31.1 last=160
c       10      uid=1 fhp=0 22.1 last=884
c       10      uid=2 fhp=80 22.1 23.1 last=804
c       11      uid=7 fhp=200 25.8 24.9 24.10 24.11 last=20
c       11      uid=8 fhp=312 24.11 24.12 25.9 last=240
c       8       uid=7 fhp=56 17.6 18.1 18.2 17.7 17.8 last=232
c       8       uid=8 fhp=108 17.8 17.9 17.10 17.11 last=96
```

CADU #5 from stream 9, *vc18*, has an error. It is the Reed-Solomon correctable error that was put into the script.

```
c       9       uid=5 fhp=384 19.5 19.6 last=500
err     9       uid=5 rsError
```

In the script, the second packet from packet maker *ap260* is dropped, as shown by the expected results file. Note that this also caused a gap in the sequence count. The count was 2, 1 was expected, so the gap was 1. If bit errors are inserted in a packet sequence number, this will also cause gaps to appear in the report.

```
drop    20      uid=2
gap     20      uid=3 actual=2 expected=1 gap=1

c       9       uid=6 fhp=284 19.6 20.1 20.3 last=208
c       13      uid=3 fhp=158 32.3 33.1 33.2 last=84
c       13      uid=4 fhp=558 33.2 33.3 last=326
c       12      uid=4 fhp=411 31.1 27.2 last=473
c       11      uid=9 fhp=100 25.9 25.10 25.11 25.12 last=104
```

The following mux line for the *top* mux shows CADU interleaving starting with the 17th CADU out of the mux.

```
mux    1      uid0=17 11.5-11.6 8.5-8.6 12.2-12.3 10.1-10.2 11.7-11.8 8.7-8.8 9.5/E
9.6 13.3-13.4 12.4

c      11     uid=10 fhp=236 25.12 24.13 24.14 last=316
c      8      uid=9 fhp=244 17.11 17.12 17.13 last=300
c      8      uid=10 fhp=40 17.13 17.14 16.11 18.3 17.15 last=44
c      10     uid=3 fhp=140 23.1 22.2 last=744
c      10     uid=4 fhp=220 22.2 23.2 last=664
c      9      uid=7 fhp=184 20.3 20.4 20.5 last=308
c      9      uid=8 fhp=84 20.5 20.6 20.7 21.1 last=16
c      11     uid=11 fhp=16 24.14 24.15 24.16 25.13 last=204
c      11     uid=12 fhp=136 25.13 25.14 26.2 25.15 last=280
c      8      uid=11 fhp=296 17.15 17.16 17.17 last=248
c      8      uid=12 fhp=92 17.17 17.18 17.19 17.20 last=112
c      12     uid=5 fhp=307 27.2 28.2 last=577
c      12     uid=6 fhp=3 28.2 29.2 30.2 last=321
c      10     uid=5 fhp=280 23.2 22.3 last=604
c      13     uid=5 fhp=316 33.3 34.1 32.4 last=292
c      13     uid=6 fhp=350 32.4 32.5 last=534
c      11     uid=13 fhp=60 25.15 25.16 24.17 24.18 last=152
c      11     uid=14 fhp=180 24.18 24.19 24.20 25.17 last=40
c      8      uid=13 fhp=228 17.20 17.21 17.22 last=316
mux    1      uid0=34 11.9-11.10 8.9-8.10 10.3-10.4 9.7-9.8 11.11-11.12 8.11-8.12
12.5-12.6 10.5 13.5-13.6 11.13-11.14
c      8      uid=14 fhp=24 17.22 16.12 16.13 17.23 last=196
c      9      uid=9 fhp=112 21.1 19.7 last=772
c      9      uid=10 fhp=12 19.7 19.8 19.9 last=88
c      10     uid=6 fhp=360 22.3 23.3 last=524
c      10     uid=7 fhp=420 23.3 22.4 last=464
c      11     uid=15 fhp=300 25.17 25.18 25.19 last=244
c      11     uid=16 fhp=96 25.19 25.20 24.21 24.22 last=116
c      8      uid=15 fhp=144 17.23 17.24 17.25 17.26 last=60
c      8      uid=16 fhp=280 17.26 17.27 17.28 last=264
c      14     uid=1 fhp=0 35.1 35.2 35.3 35.4 35.5 35.6 35.7 35.8 35.9 35.10 35.11
35.12 35.13 35.14 last=52
c      14     uid=2 fhp=12 35.14 35.15 35.16 35.17 35.18 35.19 35.20 35.21 35.22 35.23
35.24 35.25 35.26 35.27 35.28 last=40
c      9      uid=11 fhp=696 19.9 19.10 last=188
c      11     uid=17 fhp=216 24.22 24.23 24.24 25.21 last=4
c      11     uid=18 fhp=336 25.21 25.22 26.3 25.23 last=80
c      8      uid=17 fhp=76 17.28 17.29 17.30 16.14 last=128
```

(Deleted lines in this area to shorten this file.)

```
c      8      uid=130 fhp=260 17.272 17.273 17.274 last=284
c      10     uid=38 fhp=692 22.18 23.18 last=192
c      10     uid=39 fhp=752 23.18 22.19 last=132
c      9      uid=87 fhp=520 19.66 20.62 last=364
c      9      uid=88 fhp=28 20.62 20.63 20.64 20.65 last=72
c      11     uid=141 fhp=16 25.180 24.181 24.182 24.183 last=204
c      11     uid=142 fhp=128 24.183 24.184 25.181 25.182 last=84
c      8      uid=131 fhp=56 17.274 17.275 17.276 17.277 last=148
mux    1      uid0=525 11.137-11.138 8.127-8.128 9.85-9.86 13.53-13.54 12.39 11.139-
11.140 8.129-8.130 10.38-10.39 9.87-9.88 11.141-11.142
c      8      uid=132 fhp=192 17.277 17.278 16.59 18.21 last=20
c      12     uid=40 fhp=577 28.12 29.12 last=307
c      12     uid=41 fhp=253 29.12 30.12 31.12 last=59
c      10     uid=40 fhp=832 22.19 23.19 last=52
c      13     uid=55 fhp=240 33.34 33.35 33.36 last=2
c      13     uid=56 fhp=640 33.36 34.12 last=244
c      11     uid=143 fhp=256 25.182 26.23 25.183 25.184 last=160
```

```
c       11      uid=144 fhp=180 25.184 24.185 24.186 24.187 last=40
c       8       uid=133 fhp=108 18.21 17.279 17.280 17.281 last=96
c       8       uid=134 fhp=244 17.281 17.282 17.283 last=300
c       9       uid=89 fhp=320 20.65 20.66 20.67 last=172
err     21      uid=11 length
c       9       uid=90 fhp=220 20.67 21.11 19.67 last=536
c       10      uid=41 fhp=none 23.19 last=884
c       10      uid=42 fhp=8 23.19 22.20 last=876
c       11      uid=145 fhp=292 24.187 24.188 25.185 last=260
c       11      uid=146 fhp=80 25.185 25.186 25.187 25.188 last=124
c       8       uid=135 fhp=40 17.283 17.284 17.285 17.286 last=164
c       8       uid=136 fhp=176 17.286 16.60 16.61 17.287 last=44
c       14      uid=11 fhp=56 35.139 35.140 35.141 35.142 35.143 35.144 35.145 35.146
35.147 35.148 35.149 35.150 35.151 35.152 last=60
mux     1       uid0=544 8.131-8.132 12.40-12.41 10.40 13.55-13.56 11.143-11.144 8.133-
8.134 9.89-9.90 10.41-10.42 11.145-11.146 8.135-8.136
c       14      uid=12 fhp=4 35.152 35.153 35.154 35.155 35.156 35.157 35.158 35.159
35.160 35.161 35.162 35.163 35.164 35.165 35.166 last=48
c       9       uid=91 fhp=248 19.67 19.68 last=636
c       11      uid=147 fhp=216 25.188 24.189 24.190 24.191 last=4
c       11      uid=148 fhp=328 24.191 24.192 25.189 last=224
c       8       uid=137 fhp=296 17.287 17.288 17.289 last=248
c       8       uid=138 fhp=92 17.289 17.290 17.291 17.292 last=112
c       13      uid=57 fhp=32 34.12 32.37 32.38 last=210
c       13      uid=58 fhp=432 32.38 32.39 last=452
c       11      uid=149 fhp=116 25.189 25.190 26.24 25.191 last=300
c       11      uid=150 fhp=40 25.191 25.192 24.193 24.194 last=172
c       11      uid=151 fhp=160 24.194 24.195 24.196 25.193 last=60
c       11      uid=152 fhp=280 25.193 25.194 25.195 last=264
c       8       uid=139 fhp=228 17.292 17.293 17.294 last=316
c       8       uid=140 fhp=24 17.294 16.62 16.63 17.295 last=196
c       9       uid=92 fhp=148 19.68 19.69 last=736
c       9       uid=93 fhp=48 19.69 19.70 19.71 last=52
c       13      uid=59 fhp=190 32.39 33.37 33.38 last=52
mux     1       uid0=563 14.11 1.564-1.565/I 14.12 9.91 11.147-11.148 8.137-8.138 13.57-
13.58 11.149-11.152 8.139-8.140 9.92-9.93
c       13      uid=60 fhp=590 33.38 33.39 last=294
c       11      uid=153 fhp=76 25.195 25.196 24.197 24.198 last=136
c       8       uid=141 fhp=144 17.295 17.296 17.297 17.298 last=60
c       8       uid=142 fhp=280 17.298 17.299 17.300 last=264
c       12      uid=42 fhp=512 31.12 27.13 last=372
c       9       uid=94 fhp=732 19.71 19.72 last=152
c       9       uid=95 fhp=632 19.72 20.68 last=252
c       9       uid=96 fhp=140 20.68 20.69 20.70 last=352
c       11      uid=154 fhp=196 24.198 24.199 24.200 25.197 last=24
c       11      uid=155 fhp=316 25.197 25.198 26.25 25.199 last=100
c       8       uid=143 fhp=76 17.300 17.301 17.302 16.64 last=128
c       8       uid=144 fhp=204 16.64 16.65 17.303 17.304 last=8
c       11      uid=156 fhp=240 25.199 25.200 24.201 last=304
mux     1       uid0=582 13.59-13.60 1.584-1.585/I 11.153 8.141-8.142 12.42 9.94-9.96
11.154-11.155 8.143-8.144 11.156 15.16-15.18
c       8       uid=145 fhp=332 17.304 17.305 17.306 last=212
c       8       uid=146 fhp=128 17.306 17.307 17.308 17.309 last=76
c       11      uid=157 fhp=28 24.201 24.202 24.203 24.204 last=192
c       11      uid=158 fhp=140 24.204 25.201 25.202 25.203 last=64
c       9       uid=97 fhp=40 20.70 20.71 20.72 20.73 last=60
```

The following shows an error that was put into the packet length field of *ap261*. It was not conveyable, so the following CADU composition line shows 21.12 as being without error.

```
err     21      uid=12 length
c       9       uid=98 fhp=332 20.73 21.12 19.73 last=424

c       13      uid=61 fhp=348 33.39 34.13 32.40 last=260
c       13      uid=62 fhp=382 32.40 32.41 last=502
c       8       uid=147 fhp=264 17.309 17.310 16.66 last=280
c       8       uid=148 fhp=52 16.66 16.67 17.311 17.312 last=160
c       11      uid=159 fhp=276 25.203 25.204 24.205 last=268
c       11      uid=160 fhp=64 24.205 24.206 24.207 24.208 last=156
c       12      uid=43 fhp=408 27.13 28.13 last=476
c       9       uid=99 fhp=360 19.73 19.74 last=524
c       9       uid=100 fhp=260 19.74 19.75 last=624
c       11      uid=161 fhp=176 24.208 25.205 25.206 26.26 last=28
c       11      uid=162 fhp=100 26.26 25.207 25.208 24.209 last=104
c       8       uid=149 fhp=180 17.312 17.313 17.314 17.315 last=24
mux     1       uid0=601 1.601/I 8.145-8.146 11.157-11.158 9.97-9.98 13.61-13.62 8.147-
8.148 11.159-11.160 12.43 9.99-9.100 11.161-11.162
c       8       uid=150 fhp=316 17.315 17.316 17.317 last=228
c       12      uid=44 fhp=104 28.13 29.13 30.13 last=220
c       12      uid=45 fhp=352 30.13 31.13 last=532
c       10      uid=43 fhp=88 22.20 23.20 last=796
c       10      uid=44 fhp=148 23.20 22.21 last=736
c       11      uid=163 fhp=228 24.209 24.210 24.211 last=324
c       11      uid=164 fhp=8 24.211 24.212 25.209 25.210 last=204
c       8       uid=151 fhp=112 17.317 17.318 16.68 16.69 last=100
c       8       uid=152 fhp=232 16.69 17.319 17.320 last=312
c       9       uid=101 fhp=160 19.75 19.76 last=724
c       9       uid=102 fhp=60 19.76 19.77 19.78 last=40
c       13      uid=63 fhp=140 32.41 32.42 33.40 last=102
c       13      uid=64 fhp=540 33.40 33.41 last=344
c       12      uid=46 fhp=39 31.13 27.14 28.14 last=65
c       11      uid=165 fhp=136 25.210 25.211 25.212 24.213 last=68
c       11      uid=166 fhp=264 24.213 24.214 24.215 last=288
c       8       uid=153 fhp=28 17.320 17.321 17.322 17.323 last=176
c       8       uid=154 fhp=164 17.323 17.324 17.325 17.326 last=40
c       10      uid=45 fhp=228 22.21 23.21 last=656
mux     1       uid0=619 8.149-8.150 12.44-12.45 10.43-10.44 11.163-11.164 8.151-8.152
9.101-9.102 13.63-13.64 12.46 11.165-11.166 8.153-8.154
c       10      uid=46 fhp=288 23.21 22.22 last=596
c       9       uid=103 fhp=744 19.78 20.74 last=140
c       9       uid=104 fhp=252 20.74 20.75 20.76 last=240
c       11      uid=167 fhp=44 24.215 24.216 25.213 25.214 last=168
c       11      uid=168 fhp=172 25.214 26.27 25.215 25.216 last=244
c       8       uid=155 fhp=300 17.326 16.70 16.71 last=252
c       8       uid=156 fhp=80 16.71 17.327 17.328 17.329 last=124
c       12      uid=47 fhp=515 28.14 29.14 last=369
c       12      uid=48 fhp=191 29.14 30.14 31.14 last=121
c       10      uid=47 fhp=368 22.22 23.22 last=516
c       13      uid=65 fhp=298 33.41 33.42 last=586
c       13      uid=66 fhp=56 33.42 34.14 32.43 last=552
c       11      uid=169 fhp=96 25.216 24.217 24.218 24.219 last=124
c       11      uid=170 fhp=208 24.219 24.220 25.217 25.218 last=4
c       8       uid=157 fhp=216 17.329 17.330 17.331 last=328
c       8       uid=158 fhp=12 17.331 17.332 17.333 17.334 last=192
c       9       uid=105 fhp=152 20.76 20.77 20.78 last=340
c       9       uid=106 fhp=52 20.78 20.79 21.13 19.79 last=312
```

The following shows a Reed-Solomon error applied to a CADU stream, which the Reed-Solomon decoder should correct.

```
err     9       uid=106 rsError

mux     1       uid0=638 10.45-10.46 9.103-9.104 11.167-11.168 8.155-8.156 12.47-12.48
10.47 13.65-13.66 11.169-11.170 8.157-8.158 9.105
c       10      uid=48 fhp=428 23.22 22.23 last=456
c       10      uid=49 fhp=508 22.23 23.23 last=376
c       11      uid=171 fhp=336 25.218 25.219 25.220 last=208
c       11      uid=172 fhp=132 25.220 24.221 24.222 24.223 last=88
c       8       uid=159 fhp=148 17.334 16.72 16.73 17.335 last=72
c       8       uid=160 fhp=268 17.335 17.336 17.337 last=276
c       14      uid=13 fhp=16 35.166 35.167 35.168 35.169 35.170 35.171 35.172 35.173
35.174 35.175 35.176 35.177 35.178 35.179 35.180 last=36
c       14      uid=14 fhp=28 35.180 35.181 35.182 35.183 35.184 35.185 35.186 35.187
35.188 35.189 35.190 35.191 35.192 35.193 35.194 last=24
c       9       uid=107 fhp=472 19.79 19.80 last=412
c       11      uid=173 fhp=244 24.223 24.224 25.221 last=308
c       11      uid=174 fhp=32 25.221 25.222 26.28 25.223 25.224 last=44
c       8       uid=161 fhp=64 17.337 17.338 17.339 17.340 last=140
c       8       uid=162 fhp=200 17.340 17.341 17.342 18.22 last=4
c       13      uid=67 fhp=90 32.43 32.44 32.45 last=152
```

(Deleted lines in this area to shorten this file.)

```
c       8       uid=204 fhp=204 16.91 17.430 17.431 last=204
c       12      uid=61 fhp=391 28.18 29.18 last=493
c       12      uid=62 fhp=67 29.18 30.18 31.18 last=245
c       10      uid=61 fhp=384 23.28 22.29 last=500
c       13      uid=85 fhp=48 34.18 32.55 32.56 last=194
c       13      uid=86 fhp=448 32.56 32.57 last=436
c       11      uid=221 fhp=168 25.282 25.283 25.284 24.285 last=36
c       11      uid=222 fhp=296 24.285 24.286 24.287 last=256
c       8       uid=205 fhp=0 17.432 17.433 17.434 last=204
c       8       uid=206 fhp=136 17.434 17.435 17.436 17.437 last=68
c       9       uid=137 fhp=208 20.99 20.100 20.101 last=284
c       9       uid=138 fhp=108 20.101 20.102 20.103 last=384
c       10      uid=62 fhp=464 22.29 23.29 last=420
c       10      uid=63 fhp=524 23.29 22.30 last=360
c       11      uid=223 fhp=76 24.287 24.288 25.285 25.286 last=136
c       11      uid=224 fhp=204 25.286 26.36 25.287 25.288 last=212
```

This composition line includes an idle unit. *18.30.I* indicates that packet 30 from stream 18 was idle. Occasionally, idle units will have unit ID 0. Idle units are the only units that may have unit ID 0.

```
c      8      uid=207 fhp=272 17.437 18.29 18.30.I 17.438 17.439 last=104

c      8      uid=208 fhp=236 17.439 17.440 17.441 last=308
c      14     uid=17 fhp=0 35.222 35.223 35.224 35.225 35.226 35.227 35.228 35.229
35.230 35.231 35.232 35.233 35.234 35.235 last=52
mux    1      uid0=844 8.203-8.204 12.61-12.62 10.61 13.85-13.86 11.221-11.222 8.205-
8.206 9.137-9.138 10.62-10.63 11.223-11.224 8.207-8.208
c      14     uid=18 fhp=12 35.235 35.236 35.237 35.238 35.239 35.240 35.241 35.242
35.243 35.244 35.245 35.246 35.247 35.248 35.249 last=40
c      9      uid=139 fhp=8 20.103 21.17 19.103 last=748
c      11     uid=225 fhp=128 25.288 24.289 24.290 24.291 last=92
c      11     uid=226 fhp=240 24.291 24.292 25.289 last=312
c      8      uid=209 fhp=32 17.441 17.442 17.443 17.444 last=172
c      8      uid=210 fhp=168 17.444 17.445 18.31 18.32 17.446 last=120
c      13     uid=87 fhp=206 32.57 33.55 33.56 last=36
c      13     uid=88 fhp=606 33.56 33.57 last=278
c      11     uid=227 fhp=28 25.289 25.290 25.291 25.292 last=176
c      11     uid=228 fhp=164 25.292 24.293 24.294 24.295 last=56
c      11     uid=229 fhp=276 24.295 24.296 25.293 last=276
c      11     uid=230 fhp=64 25.293 25.294 26.37 25.295 25.296 last=12
c      8      uid=211 fhp=220 17.446 17.447 17.448 last=324
c      8      uid=212 fhp=16 17.448 17.449 17.450 17.451 last=188
c      9      uid=140 fhp=36 19.103 19.104 19.105 last=64
c      9      uid=141 fhp=720 19.105 19.106 last=164
c      13     uid=89 fhp=364 33.57 34.19 32.58 last=244
```

(Deleted lines in this area to shorten this file.)

```
c      8      uid=239 fhp=312 17.506 17.507 17.508 last=232
mux    1      uid0=974 11.253-11.256 8.235-8.236 9.156-9.157 13.99-13.100 1.984-
1.985/I 11.257 8.237-8.238 12.70 9.158-9.160 11.258-11.259
c      8      uid=240 fhp=108 17.508 17.509 16.105 16.106 last=104
c      11     uid=260 fhp=60 24.334 24.335 24.336 25.333 last=160
```

The *Stat* line shows counts by stream.

```
Stat   16     total=106 errors=0
Stat   17     total=509 errors=0
Stat   18     total=33 regular=32 idles=1 errors=0
Stat   2      total=648 regular=647 idles=1 errors=0
Stat   8      total=290 regular=240 idles=50 errors=0
Stat   24     total=336 errors=0
Stat   25     total=333 errors=0
Stat   26     total=41 errors=0
Stat   5      total=710 errors=0
Stat   11     total=260 errors=0
Stat   19     total=120 errors=0
Stat   20     total=115 errors=0 dropped=1 gaps=1
Stat   21     total=19 errors=3
Stat   3      total=254 errors=0
Stat   9      total=160 errors=3
Stat   32     total=66 errors=0
Stat   33     total=63 errors=0
Stat   34     total=21 errors=0
Stat   7      total=150 errors=0
Stat   13     total=100 errors=0
Stat   27     total=21 errors=0
Stat   28     total=20 errors=0
Stat   29     total=20 errors=0
Stat   30     total=20 errors=0
Stat   31     total=20 errors=0
Stat   6      total=101 errors=0
Stat   12     total=70 errors=0
Stat   22     total=33 errors=0
Stat   23     total=32 errors=0
Stat   4      total=65 errors=0
Stat   10     total=70 errors=0
Stat   35     total=277 errors=0
Stat   14     total=20 errors=0
Stat   15     total=30 errors=0
Stat   1      total=1000 regular=950 idles=50 errors=0
```

A final note provided by the output class. 1000 records were written to a plain output file.

```
NOTE   0      Output. File. Records=1000
```

SECTION 4
OPERATIONS SCENARIO

This section describes a typical operations scenario for SCTGEN, and outlines the sequence of steps needed to ingest user-specified information and start data generation.

4.1 OVERVIEW

SCTGEN is an offline support system for ETS; no operation of SCTGEN is time-critical so as to prevent the rest of ETS from functioning. However, it is imperative that SCTGEN functionality provide the test data necessary for ETS to function; basic components are the user, test data generation specification, and the system.

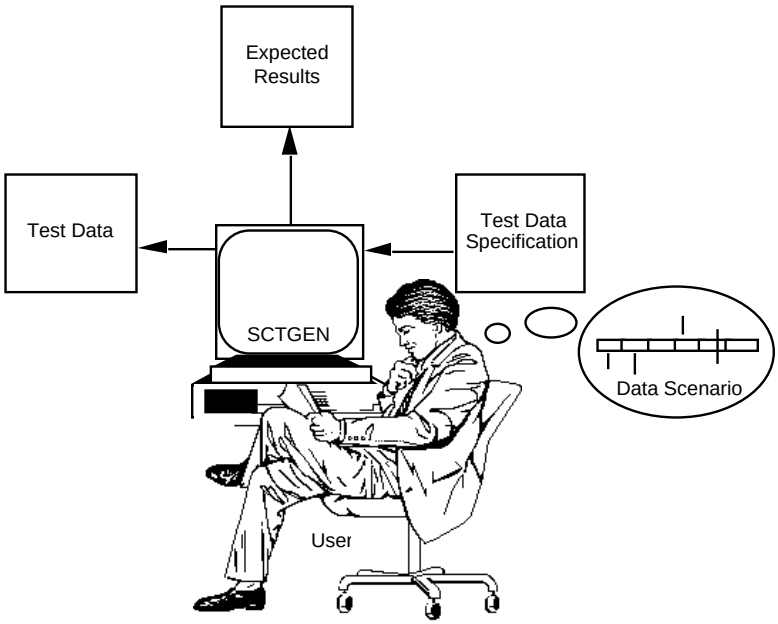

Figure 4-1. Operational Data Generation Scenario

4.2 TEST DATA GENERATION SPECIFICATION

The user will devise data scenarios that will exercise ETS functions dependent on its output product. The user will design the mix of sensor/source data, the merging of this data so as to simulate a multiplexed data stream, and the anomalies that this data stream must contain. This test data generation specification is entered into the system using the keyboard. These specifications can be entered by evoking the GUI that will query the user to enter the size, type, quantity, etc., of the test data to be generated, or through a script generator.

The next step in the specification process is to insert error locations, by type and value, into the data. Errors and causes of dropped data are injected at each layer or level of test data generation. Packet-level errors are inserted at the packet data specification level; frame-level errors are inserted at frame data specification level. Once the test data specifications are entered into the

system, the user will generate a "pseudo-test data" file, whereby expected results are generated to verify the data scenario.

4.3 TEST DATA GENERATION

Once the user is satisfied that expected results match the devised scenario for testing, the user will then generate test data. This is done by targeting a file name and location where this data is to be stored. The maximum test data size is storage-limited to 25 Gbytes. The format of test data is selected by the user to be any one of the formats described in Section 2 (i.e., LVGEN, TPGEN, STGEN, or any one of the SCTGEN formats). The type of data, be it command blocks or CADUs, will be designated as the highest layer of data encapsulation.

If test data is CADUs with Version 1 packets, the sequence of test data generation is as follows:

a. First, the user will select the total number of frames to be generated, for example 100,000. The user will then assign a specific number of APIDs and VCIDs to these frames, specifying the number of packets per APID per VCID.

b. Second, the user will devise the frame multiplexing design to generate the test data (i.e., VCID 1 [10 CADUs] to be followed by VCID 2 and 3 in an alternating pattern, and so on). Within the VCID stream, the user will devise the multiplexing strategy based on packet rates or user-selected frequencies. Once this is done, the user will either generate the pseudo-test data file and view data to selectively insert errors, or insert errors before the first generation cycle and generate the errored test data.

In the former option, once test data is generated the user will devise the frame-level modifications to simulate frame-level errors, and packet-level errors to simulate packet errors. Once all frame- and packet-level modifications are entered, the user will regenerate the test data.

4.4 EXPECTED RESULTS

Based on errors inserted in test data, SCTGEN will generate a summary of expected statistics. Statistics will be generated at different levels of detail. In the first level, abstraction, SCTGEN will keep track of certain types of data at certain locations within test data; the values in those locations and a flag to describe the state of the contents. For example, if the location contains the position of the packet in the stream of packets from a specific (APID) source, the value in that location may contain the errored packet length of the packet, and the flag may describe the type of error.

The second level is more descriptive and may contain total statistics of the number of packets from each APID, errors in specific packets, type of errors, number of packets from that source that are dropped or are missing, and the reason for the same.

These expected results will be generated at data layers (packets, frames). However, only errors inserted within a data layer will have a reliability of being correct. Errors inserted at the higher level of encapsulation may cause errors at the lower level, which may or may not be identified in the expected results. To ensure that the data scenario meets the expectation of the test case developer, the SCTGEN generates "pseudo-test data file" expected results are printed out for the designed test data without actually generating test data. This will enable the designer to make modifications to data scenarios before actually generating data.

For EDOS-specific data, expected results will reflect the requirements documented in the appropriate ICD to generate EDUs and associated EDU headers. The EDU header will, at a minimum, contain the information described in Sections 2 and 3, and the previous paragraphs.

SECTION 5
DETAILED DESIGN

This section describes the software Configuration Items (CI) that comprise the SCTGEN system. SCTGEN is being developed using object-oriented modeling techniques; modules are discussed as classes and subclasses.

5.1 CONFIGURATION ITEMS

The CIs that will meet the summarized ETS requirements levied on SCTGEN are functionally described as follows:

a. Generate CCSDS return link data consisting of:

   (1) CCSDS Version 2 Coded Virtual Channel Data Units (CVCDU) and CADUs.

   (2) CCSDS Version 1 packets.

b. Generate EDOS data products consisting of:

   (1) EDUs consisting of CCSDS Version 1 packets with EDOS service headers.

   (2) EDSs consisting of time-ordered packets sorted by APID on a per-session basis.

   (3) PDSs consisting of time-ordered, overlap deleted, quality annotated packets sorted by APID on a user-selected time period.

c. Simulate errors in the generated or user-provided CCSDS return link data or EDOS data products.

d. Generate an expected results summary of generated test data and inserted errors.

e. SCTGEN will store all generated data on disk and tape storage media.

The detailed requirements analysis and description that were the basis for this design are provided in the SCTGEN System Requirements Document. CIs are described in layer functions as organized in the architectural overview in Section 2 and illustrated in Figure 5-1.

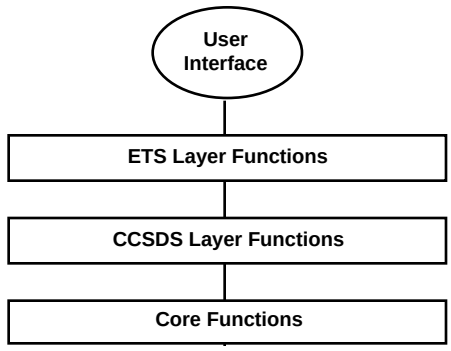

Figure 5-1. Architectural Features

5.2 ETS LAYER FUNCTIONS

This functional layer will manipulate artifacts such as headers and trailers for the data, and are used to produce ETS-specific products. For example, an expedited data set consists of packets sorted by APID with a service header that identifies the type of data set and describes the data in accordance with descriptions specified in the EDOS-EGS ICD. This set of functions will generate EOS service headers for each EDOS data unit, data set, and rate-buffered file.

5.3 CCSDS LAYER FUNCTIONS

At this layer, software CIs are described in terms of object modules. Figure 5-2 illustrates the functional flow of CCSDS return link data generation. The diagram is hierarchically drawn from left to right, but data generation is from right to left.

The object class v1Packet creates packets for a v1PacketStream. A packet stream is built for each path, or in terms of the spacecraft, for each APID. There are many such streams, which are generated for a VCID and SCID designated channel. More than one stream of packets feeds a single channel. Thus, a packet stream for APID2 and a packet stream for APID4 may feed channel VCID2, SCID42. As the second from right object class in the figure shows, these packet streams are multiplexed by a core function to generate a multiplexed stream of packets, one for each uniquely designated VCID, SCID channel. Many such streams of multiplexed packet streams are further multiplexed to generate a single CADU stream for a unique SCID. This stream of CADUs is a single object, and is stored as a test data file of CADUs.

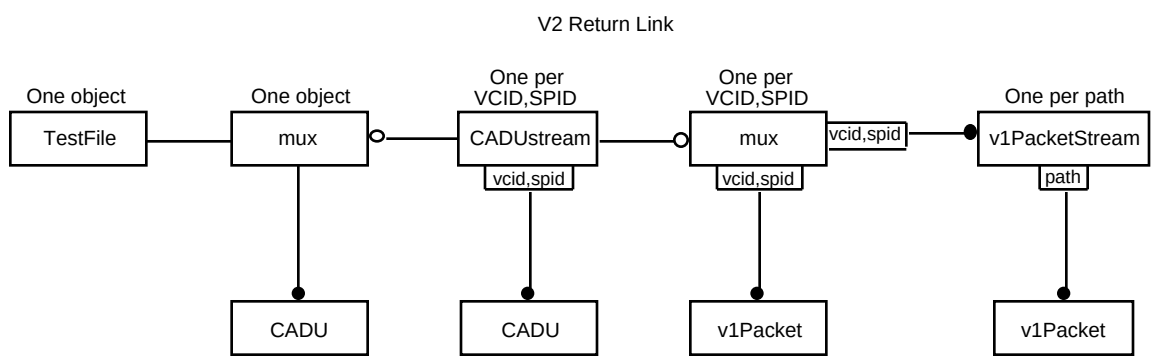

Figure 5-2. Version 2 CADU Return Link Data

Similarly, Figure 5-3 illustrates the functional flow of CCSDS forward link data generation. In this case, the formats and recommendations specified in the standards for CCSDS forward link data are followed to produce a test data file of CLTUs. Many v1Packets, which in this case are TC packets, are multiplexed to create a TC packet stream.

Many TC packet streams are multiplexed to create a telecommand telemetry frame stream, one per VCID for an SCID. Many telecommand telemetry frame streams, from more than one VCID, are multiplexed to produce a CLTU stream. The CLTU data stream is saved as a test data file of CLTUs.

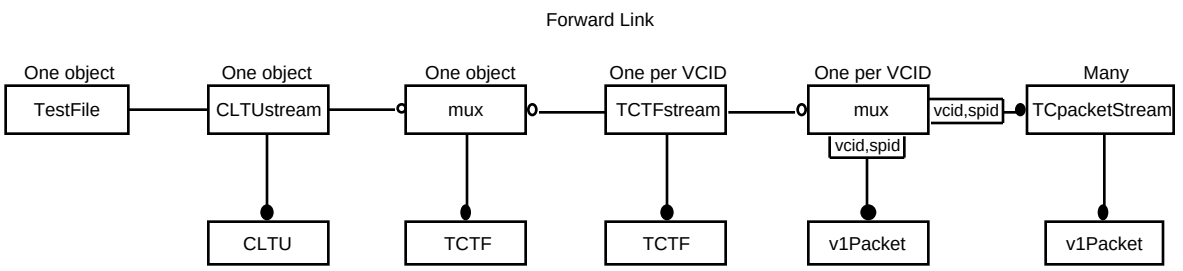

Figure 5-3. CLTU Forward Link Data

5.4 CORE FUNCTION OBJECT CLASSES

Core classes are toolkits that CCSDS classes use to create and process units such as packets and frames. The following core function object class diagrams are OMT diagrams that were developed using the commercial development package *Software Through Pictures*. The following definitions aid in explaining the functions and products from core classes:

a. Stream: a stream can insert a random, fixed, or user-specified pattern into any data region. It can also fill a region by using a file or input from another stream. Streams can create units, or read complete units from files.

b. Fillers: stream tools that fill data regions in units. For example, a packet stream uses a TigPattern tool to fill the packet's data region.

There are seven basic core function object classes, as follows:

a. Stream Function.

b. Genstream Function.

c. ListPatterns Function.

d. Consumer Function.

e. ErrList Function.

f. ListDevices Function.

g. Exception Function.

5.4.1 STREAM FUNCTION CLASS DESCRIPTION

TigStream is the base class for all streams (see Figure 5-4). The TigDictionary class is used by many classes to obtain the script setup input by the SCTGEN user.

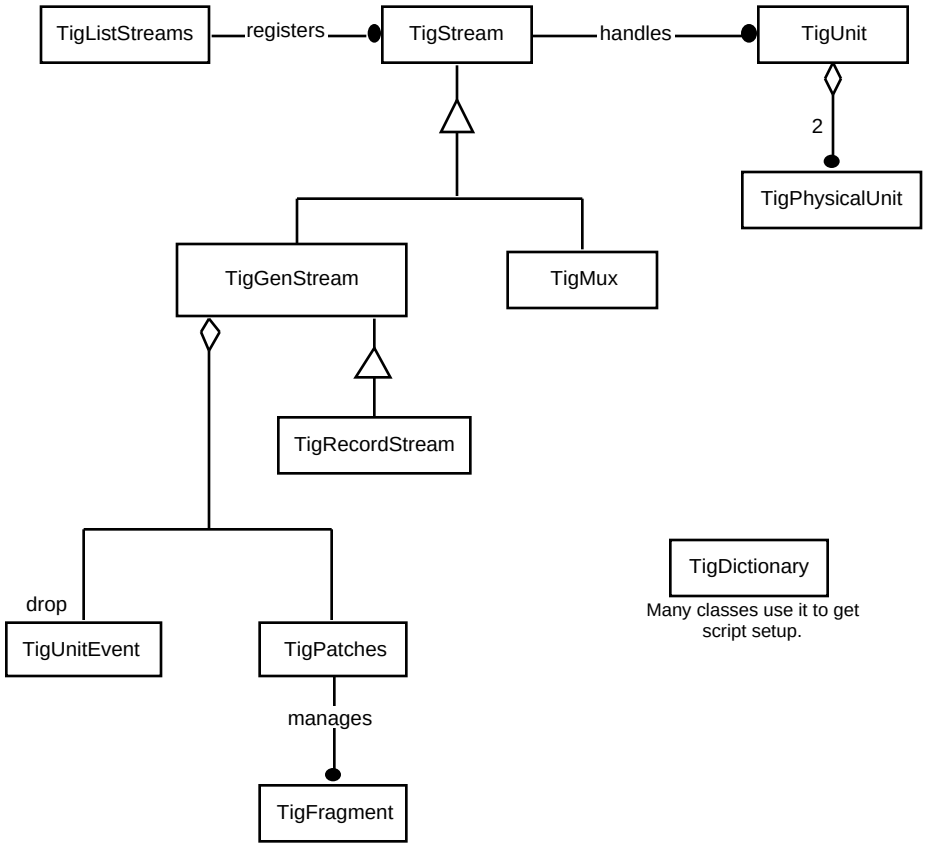

Figure 5-4. Stream Function Object Class

a. TigGenStream is the base class for all streams that create units such as packet maker or frame maker.

b. TigUnit is the base class for all units such as packets or frames.

c. TigPhysical Unit is a subclass of TigUnit that contains the actual unit memory.

d. TigMux is a special stream that merges several input streams into one output stream.

e. TigRecordStream is a special stream that creates fixed size units from an input stream that could produce variable-length streams.

f. TigPatches is a special class that streams use to overlay unit regions with poor quality data (part of error insertion).

g. TigFragment is one overlay in TigPatches.

h. TigUnitEvent is a utility class that tells a stream when to perform a function (i.e., drop units or insert errors).

i. TigDictionary is a data base class that contains all script setup information.

5.4.2 GENSTREAM FUNCTION CLASS DESCRIPTION

TigGenStream is the base class for all streams that create units such as packet maker or frame maker (see Figure 5-5).

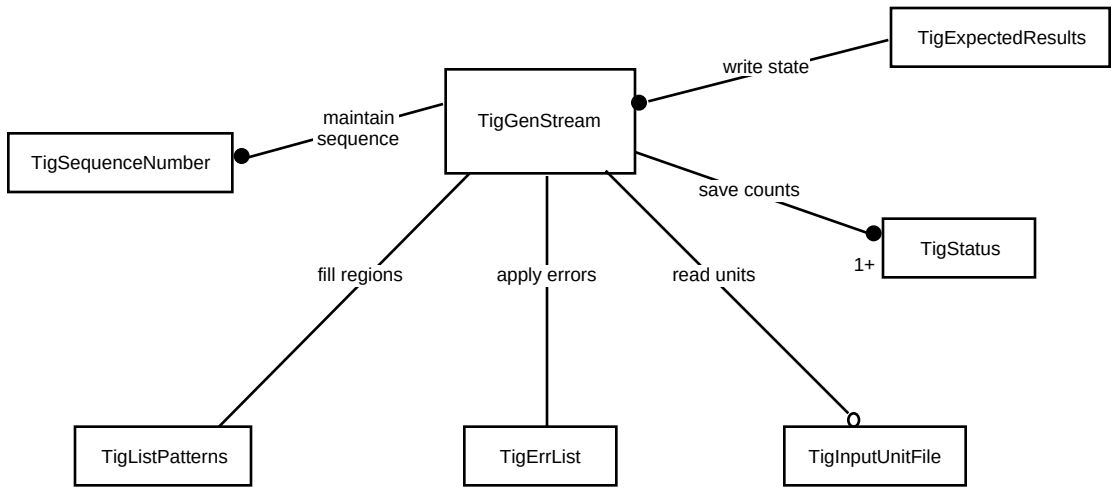

Figure 5-5. Genstream Function Object Class

a. TigStatus contains stream counts such as number of units and number of units with errors.

b. TigInputUnitFile is used by some streams to obtain units from a file, rather than creating them.

c. TigSequenceNumber is a utility class that maintains a sequence number.

d. TigErrList is a list of all errors that a stream inserts into a unit.

e. TigListPatterns is a list class that contains all pattern classes that a stream uses to fill a unit's data regions.

f. TigPatterns is the base class for all regions.

g. TigExpectedResults: all streams send expected results information to this class to be written to a text file.

5.4.3 LIST PATTERNS FUNCTION CLASS DESCRIPTION

TigListPatterns is a list class that contains all pattern classes that a stream uses to fill a unit's data regions; TigPatterns is the base class for all region fillers (see Figure 5-6). Subclasses within this object class are:

a. TigRandomPattern is a region filler class that fills a region with a random sequence.

b. TigFixedPattern is a region filler class that fills a region with a fixed pattern.

c. TigUserPattern is a region filler class that fills a region with a user-provided pattern.

d. TigStepPattern is a region filler class that fills a region with a step or ramp pattern.

e. TigRawFilePattern is a region filler class that fills a region from a user file.

f. TigConsumerPattern is a region filler class that fills a region with units from an input stream (packets into frames, for example).

g. TigRegion is a utility class that some pattern classes use to manage a data region.

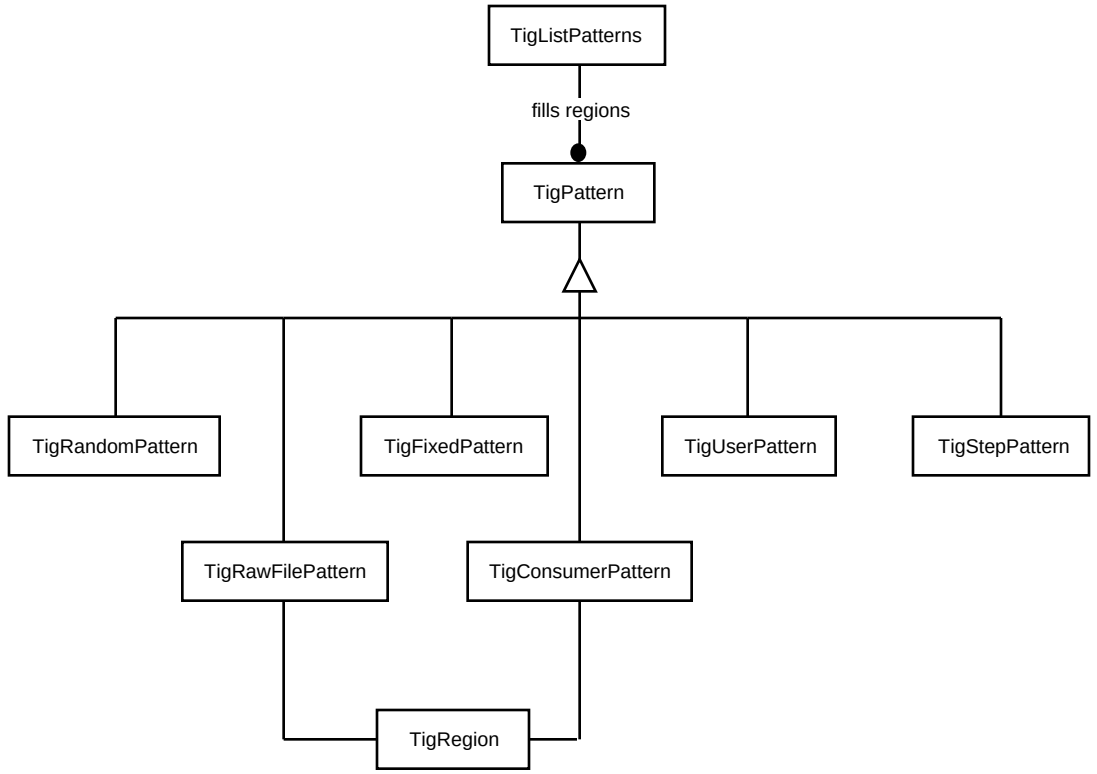

Figure 5-6. List Patterns Function Object Class

5.4.4 CONSUMER FUNCTION CLASS DESCRIPTION

TigConsumerPattern (Figure 5-7) is a region filler class that fills a region with units from an input stream (packets into frames, for example).

a. TigStream is the base class for all streams.

b. TigGenStream is the base class for all streams that create units such as a frame or packet maker.

c. TigUnit is the base class for all units such as packet or frames.

d. TigPatchesis a special class that streams use to overlay unit regions with poor quality data (part of error insertion).

e. TigHistory is a subclass of TigConsumerPattern that records which input units are put into each output region.

f. TigRegion is a utility class that some pattern classes use to manage a data region.

g. TigBitSlip is a subclass of TigConsumerPattern that simulates bit slip errors.

h. TigSlip controls bit slip in one direction—long or short.

i. TigUnitEvent is a utility class that tells a stream when to perform a function (i.e., drop units or insert errors).

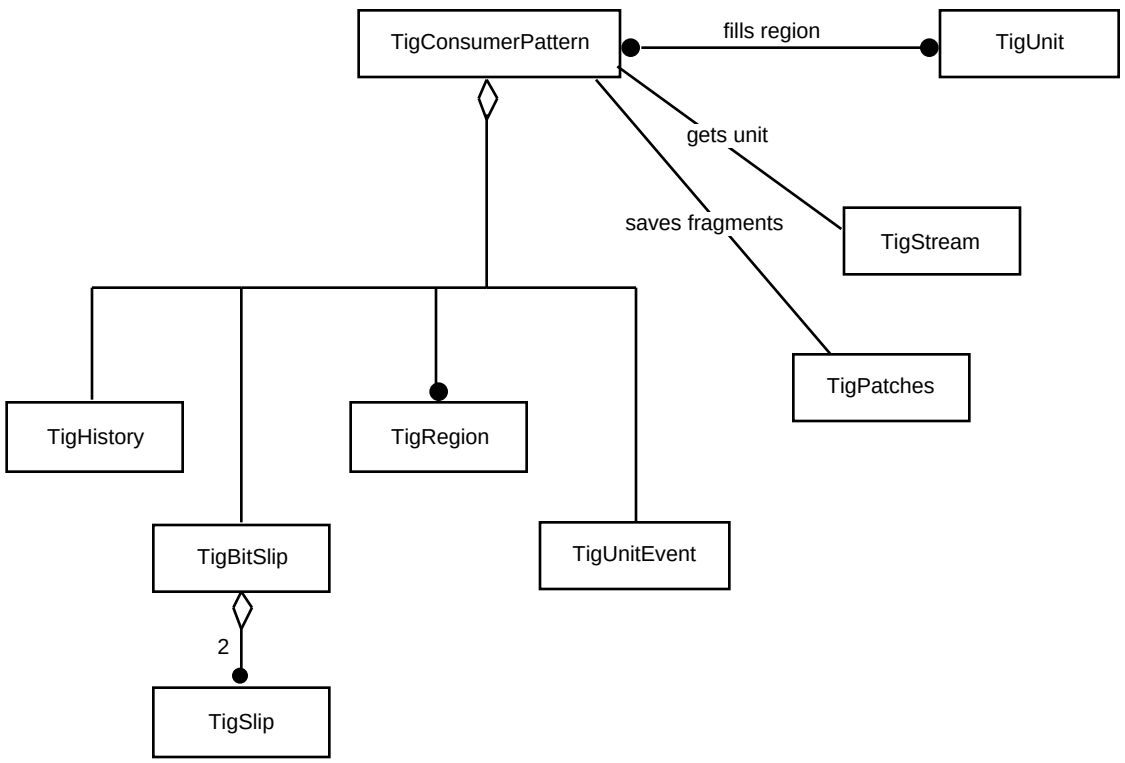

Figure 5-7. Consumer Function Object Class

5.4.5 ERR LIST FUNCTION CLASS DESCRIPTION

TigErrList is a list of all errors that a stream inserts into a unit; TigErrors is the base class for a single error class (see Figure 5-8). All derived error classes specify a starting bit and a number of bits.

a. TigFlipBitsError flips all bits within the specified range.

b. TigSetValuaError deposits up to a 32-bit value within a unit.

c. TigAddValueError adds up to a 32-bit value to any 32 bit number within a unit.

d. TigFlipMaskError flips “care” bits specified by a 32-bit mask within a unit.

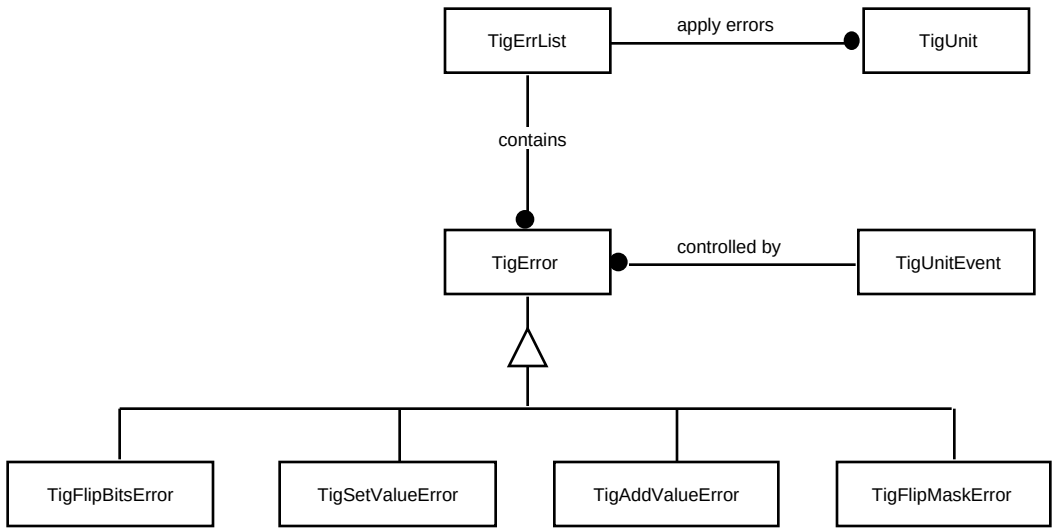

Figure 5-8. ErrList Function Object Class

5.4.6 LIST DEVICES FUNCTION CLASS DESCRIPTION

TigListDevices and all TigDevices are the base classes from which all input and output device classes are derived (Figure 5-9).

a. TigUpdateFile creates the two files required by the Simulator Card.

b. TigInputUnitFile is a class that some streams use to obtain units from a file, rather than creating them.

c. TigFileDevice is a device class that reads or writes to a disk file.

d. TigOutputUnitFile is a class that programs may use to create a file of units.

e. TigPlainUnits is a type of input unit file class where units are all fixed size with no special annotation.

f. TigFancyUnits is a type of input unit file class where units have special descriptive annotation.

g. TigOutPlainUnits creates a plain unit file.

h. TigOutFancyUnits creates a more complicated unit file, which contains unit annotation with each unit.

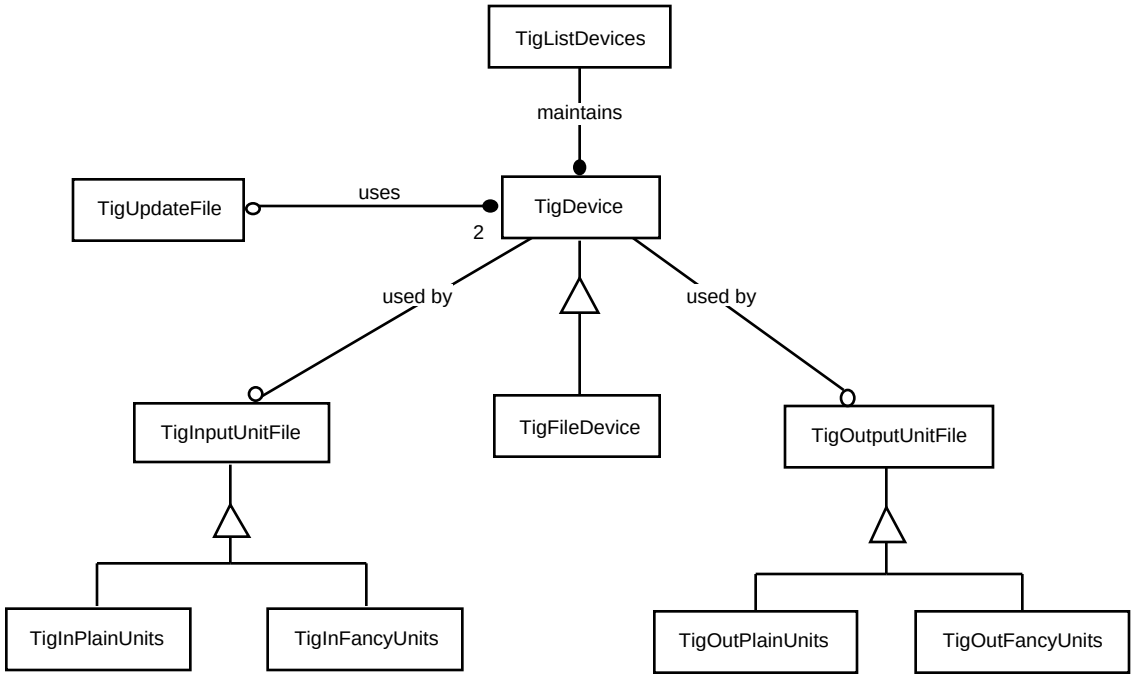

Figure 5-9. ListDevices Function Object Class

5.4.7 EXCEPTION FUNCTION CLASS DESCRIPTION

TigException is a utility class that reports an error and aborts a program (Figure 5-10). All derived class are specific errors.

a. TigFailOpen.

b. TigFailScript.

c. TigFailNoField.

d. TigFailRange.

e. TigFailBadChoice.

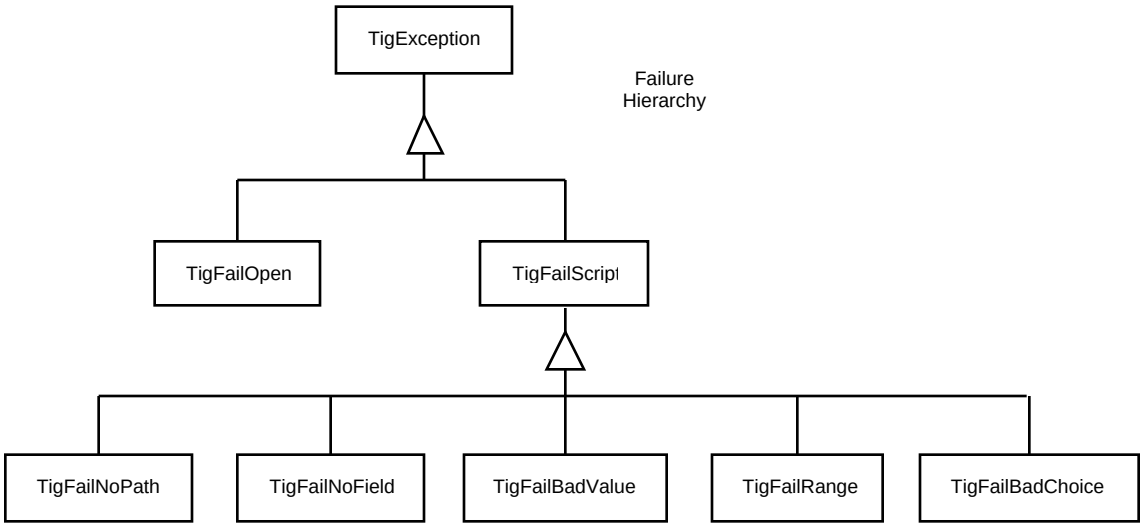

Figure 5-10. Exception Function Object Classes

SECTION 6
REQUIREMENTS TRACEABILITY MATRIX

The high-level requirements levied on SCTGEN are based on the configurations that SCTGEN will support for ETS. These configurations require that SCTGEN generate test data to support the data flows that are described in Section 1. An asterisk (*) signifies a new requirement. The SCTGEN requirements are derived from the following ETS requirements.

SCTGEN Rqmt. No.	SCTGEN Requirement Description	F&P Rqmt. No.
3.1.1.1	SCTGEN shall be based on the UNIX environment.	Derived
3.1.1.2	SCTGEN shall be portable to different UNIX platforms.	Derived
3.1.1.3	SCTGEN shall generate test data to simulate forward/return link telemetry conforming to CCSDS formats and protocols at each data layer, as defined in the applicable CCSDS documents listed in Section 1.4.	3.1.1
3.1.1.4	SCTGEN shall generate test data to simulate EOSDIS internal data products as required by the ETS Project and specified in this document.	3.1.1
3.1.1.5	SCTGEN shall generate a summary of expected results for each test data file generated.	3.3.3
3.1.1.6	SCTGEN shall generate a test data file (Complete format) consisting of the complete data stream inclusive of all designed sequencing options, timecode updating options and user-specified errors.	3.2.1
3.1.1.7	SCTGEN shall generate test data (STGEN format) consisting of the complete data stream with a user-specified update file containing sequencing options, timecode updating options, and error location, quantity and type information.	3.2.1
3.1.1.8	SCTGEN shall provide the capability to create test data specification files.	3.1.1, 3.2.1, 5.1.7, 5.1.8
3.1.1.9	SCTGEN shall have the capability to save test data specifications in a file.	3.1.1, 3.2.1, 5.1.7, 5.1.8
3.1.1.10	SCTGEN shall provide a browse capability for test data specification files.	3.1.1, 3.2.1, 5.1.7, 5.1.8
3.1.1.11	SCTGEN shall allow users to edit test data specification files.	3.1.1, 3.2.1, 5.1.7, 5.1.8

3.1.1.12	SCTGEN shall allow users to edit existing test data files without changing data formats.	3.1.1, 3.2.1, 5.1.7, 5.1.8
3.1.1.13	SCTGEN shall allow users to halt and resume test data generation.	3.1.1, 3.2.1, 5.1.7, 5.1.8
3.1.1.14	SCTGEN shall allow users to create a summary of expected results file pertaining to a test data specification without actually generating test data.	3.1.1, 3.1.2
3.1.1.15	SCTGEN shall provide quality and accounting statistics during test data generation, as specified in the EOS-AM Spacecraft Requirements Document listed in Section 1.4.	3.3.3
3.1.2.1	SCTGEN shall generate test data with CCSDS day segmented timecode formats.	3.1.5, 3.1.15, 3.5.1, 3.5.2
3.1.2.2	SCTGEN shall generate test data with non-CCSDS PB-5C timecode formats.	3.1.5, 3.1.15, 3.5.2
3.1.2.3	SCTGEN shall allow users to specify epoch time for each simulated test data stream with January 1, 1958 as default.	3.1.8, 3.5.2
3.1.2.4	SCTGEN shall have the capability to specify starting time and time increment/time decrement for any source—identified by any combination of SCID, VCID, and APID in each generated test data stream controlled by: a) User-specified packet/EDU data rate. b) Correlated to a specified packet/EDU data rate.	3.1.1, 3.5.1, 3.5.2
3.1.2.5	SCTGEN shall have the capability to generate test data with user-specified encoding at VCDU/transfer frame data and headers.	3.1.5
3.2.1.1	SCTGEN shall simulate CCSDS Version 2 VCDUs, CVCDUs, and CADUs.	3.1.1, 3.2.1, 5.2.1
3.2.1.2	SCTGEN shall simulate CCSDS Version 1 source packets.	3.1.5, 3.2.1, 5.2.1
3.2.1.3	SCTGEN shall allow users to specify VCDU data fields as data patterns, packet streams, a bit stream, or frame streams.	3.1.8, 3.5.1, 3.5.3
3.2.1.4	SCTGEN shall allow users to specify any field in a VCDU primary header.	3.1.5, 3.5.1
3.2.1.5	SCTGEN shall allow users to specify any field in a VCDU trailer.	3.1.5

3.2.1.6	SCTGEN shall allow users to specify any field in a packet primary header.	3.1.5, 3.5.1
3.2.1.7	SCTGEN shall allow users to specify any field in a packet secondary header, as defined by the EOS AM-1 mission.	3.1.5, 3.5.1
3.2.1.8	SCTGEN shall allow users to specify packet data fields as data patterns.	3.1.8, 3.5.1, 3.5.2
3.2.1.9	SCTGEN shall allow users to assign any CCSDS-defined value to an APID field.	3.1.5, 3.1.14
3.2.1.10	SCTGEN shall allow users to assign any CCSDS-defined value to a VCID field.	3.1.5, 3.1.14
3.2.1.11	SCTGEN shall allow users to assign any CCSDS-defined value to an SCID field.	3.1.5, 3.1.14
3.2.1.12	SCTGEN shall allow users to select incrementing or decrementing packet sequence counts in single or multiple steps.	3.1.5, 3.1.8
3.2.1.13	SCTGEN-generated data patterns in the TF, VCDU, and packet data fields can be one of the following: a) Fixed 8-, 16-, or 32-bit data pattern. b) Step pattern with user-specified step increment/decrement. c) Pseudorandom pattern, repeated as a fixed pattern on repetition. d) User-provided data.	3.1.8, 3.1.20, 3.5.3, 3.1.24(*)
3.2.1.14	SCTGEN shall provide the option to generate a check sum byte for the data field of each data unit and insert this check sum byte into the data field as the last byte.	3.1.7, 3.1.8
3.2.1.15	SCTGEN shall simulate fill VCDUs with any user-defined 16-bit pattern.	3.1.8, 3.1.11
3.2.1.16	SCTGEN shall simulate fill Multiplexing Protocol Data Units (M_PDU) with any user-defined 16-bit pattern.	3.1.8, 3.1.11
3.2.1.17	SCTGEN shall simulate fill packets with any user-defined 16-bit pattern.	3.1.8, 3.1.11
3.2.1.18	SCTGEN shall allow users to select data polarity (true/inverted).	3.1.5
3.2.2.1	SCTGEN shall simulate the following EDOS data products as defined in the EDOS External ICD DFCD: a) EDOS Data Unit (EDU). b) Expedited Data Set (EDS). c) Production Data Set (PDS). d) Rate-buffered Data File (RDF).	3.1.2, 3.3.1, 3.3.2, 3.5.15, 5.3.1, 5.5.18, 5.5.19

3.2.2.2	SCTGEN shall simulate headers of any EDOS data products.	3.1.9, 3.1.14, 3.1.15, 3.3.1, 3.3.2, 3.5.1, 3.5.2
3.2.2.3	SCTGEN shall provide users with the capability to specify the contents of any field within the headers of any EDOS data products.	3.1.5, 3.1.9, 3.1.14, 3.1.15,
3.2.2.4	SCTGEN shall simulate the data contents of any EDOS data products.	3.1.8, 3.1.9
3.2.2.5	SCTGEN shall simulate the trailers of any EDOS data products.	3.1.5, 3.1.9
3.2.2.6	SCTGEN shall provide users with the capability to specify the contents of any field within the trailers of any EDOS data products.	3.1.5, 3.1.9
3.2.2.7	SCTGEN shall provide the capability to simulate multiple TDRSS Service Session (TSS) PDSs.	5.3.3
3.2.2.8	SCTGEN shall provide the capability to simulate 24-hour PDSs.	3.1.20, 3.3.1, 5.3.1, 5.3.3
3.2.3.1	SCTGEN shall have the capability to simulate test data streams containing unique, unreplicated data patterns up to a maximum size as specified in this document.	3.1.9, 5.2.1, 5.3.1, 5.3.2, 5.3.3
3.2.3.2	SCTGEN shall allow users to specify multiple data segments within a data stream, with each data segment having its own data characteristics of sequence numbers and increment/decrement, start/end timecode, and source identified by any combination of SCID, VCID, and APID.	3.1.8, 3.1.9, 3.1.10, 3.5.2, 5.5.16, 5.5.17
3.2.3.3	SCTGEN shall have the capability to simulate packets of the same APID in multiple virtual channels.	3.1.1, 3.1.14
3.2.3.4	SCTGEN shall allow users to specify packet sequence count, limited by the size and number of packets in the test data, and timecode starting values and update intervals for these counts in each data segment within the same data stream.	3.1.10, 3.1.15, 3.5.2
3.2.3.5	SCTGEN shall allow users to specify TF or VCDU sequence count, limited by the size and number of frames in the test data, and timecode starting values and update intervals for these counts in each data segment within the same data stream.	3.5.3, 5.5.16

3.2.3.6	SCTGEN shall have the capability to simulate duplicate data units.	Derived
3.2.3.7	SCTGEN shall have the capability to sequentially simulate multiple TDRSS service sessions.	5.3.3
3.2.3.8	SCTGEN shall have the capability to merge user-provided test data in CADUs from user-assigned virtual channels with SCTGEN-generated test data of other virtual channels into a single test data file.	3.1.8, 3.2.1, 3.5.3 3.1.24(*)
3.2.3.9	SCTGEN shall have the capability to merge user-provided test data in packets from user-assigned APIDs with SCTGEN-generated test data of other APIDs into a single test data file.	3.1.8, 3.2.1, 3.4.3, 3.5.3, 3.1.24(*)
3.3.1.1	SCTGEN shall allow users to change/modify/manipulate any values at specified data layers in the simulated test data.	3.1.8, 3.1.9
3.3.1.2	SCTGEN shall allow users to change/modify/manipulate any values at specified locations and/or specified frequencies within the test data.	3.1.8, 3.1.9
3.3.2.1	SCTGEN shall allow users to insert errors at the bit level in any of the following CADU fields: a) Synchronization pattern. b) Any field in the VCDU primary header. c) Any field in the VCDU trailer. d) VCDU data field. e) Reed-Solomon data field. f) Reed-Solomon check symbols. g) First header pointer in M_PDU header.	3.1.8, 3.1.9
3.3.2.2	SCTGEN shall simulate the error condition signifying missing or dropped CADUs.	3.1.8, 3.1.9
3.3.2.3	SCTGEN shall simulate bit slips up 3 bits per CADU in the forward/backward direction in test data.	3.1.9, 3.1.10
3.3.2.4	SCTGEN shall simulate CADU synchronization search, check, lock, and flywheel scenarios.	3.1.9
3.3.2.5	SCTGEN shall simulate correctable Reed-Solomon errors in CADUs.	3.1.9
3.3.2.6	SCTGEN shall simulate uncorrectable Reed-Solomon errors in CADUs.	3.1.9
3.3.3.1	SCTGEN shall allow users to insert errors at the bit level in any of the following source data unit fields: a) Any field in source data unit primary header. b) Any field in source data unit secondary header. c) Source data unit data field.	3.1.9
3.3.3.2	SCTGEN shall simulate the error condition signifying missing or dropped source data units.	3.1.9, 3.1.10
3.3.3.3	SCTGEN shall simulate source data unit length errors.	3.1.9

3.3.4.1	SCTGEN shall allow users to insert errors at the bit level in any of the following EDOS data product fields: a) Any field in the EDOS data product header. b) EDOS data product data field. c) Any field in the EDOS data product trailer.	3.1.9
3.3.4.2	SCTGEN shall simulate the error condition signifying missing or dropped EDOS data products.	3.1.9, 3.1.10
3.3.4.3	SCTGEN shall simulate EDOS data product length errors.	3.1.9
3.3.5.1	SCTGEN shall simulate length errors in user-provided telemetry.	3.1.9, 3.1.19, 3.1.24(*)
3.3.5.2	SCTGEN shall be able to insert errors at the bit level in any of the following fields of a user-provided telemetry by sequence location of a specific CADU/frame: a) Synchronization pattern. b) Any field in the user-provided VCDU/frame primary header. c) Any field in the user-provided VCDU/frame secondary header. d) Any field in the user-provided VCDU/frame trailer. e) User-provided VCDU/frame data field. f) Reed-Solomon data field. g) Reed-Solomon check symbols. h) First header pointer in M_PDU header.	3.1.9, 3.1.19, 3.1.24(*)
3.3.5.3	SCTGEN shall simulate the error condition signifying missing or dropped user-provided VCDUs/frames.	3.1.9, 3.1.10, 3.1.19, 3.1.24(*)
3.3.5.4	SCTGEN shall simulate bit slips up to 3 bits per user-provided VCDU/frame in the forward/backward direction in the test data.	3.1.9, 3.1.19, 3.1.24(*)
3.3.5.5	SCTGEN shall simulate user-provided VCDU/frame synchronization search, check, lock, and flywheel scenarios.	3.1.9, 3.1.19, 3.1.24(*)
3.3.5.6	SCTGEN shall simulate correctable Reed-Solomon errors in user-provided VCDU/frames.	3.1.9, 3.1.19, 3.1.24(*)
3.3.5.7	SCTGEN shall simulate uncorrectable Reed-Solomon errors in user-provided VCDU/frames.	3.1.9, 3.1.19, 3.1.24(*)
3.4.1.1	SCTGEN shall allow users access via American National Standards Institute (ANSI) standard keyboard and mouse functions.	3.1.2

3.4.1.2	SCTGEN shall allow users to execute scripts and other command-level inputs to generate test data products at any level of encapsulation; packets, frames, or blocks.	3.1.5, 3.1.6, 3.1.7, 3.1.8, 3.1.9
3.4.1.3	SCTGEN shall allow users to execute scripts and other command-level inputs to verify, validate, and monitor the status of test data generation.	3.1.5, 3.1.6, 3.1.7, 3.1.8, 3.1.9
3.4.1.4	SCTGEN shall have the capability of reading in user-provided data in the form of raw sensor data, telemetry packets, telemetry CADUs/frames, telemetry blocks, and data sets from magnetic tape media or disk.	3.1.12, 3.1.19
3.4.1.5	SCTGEN shall have the capability of downloading user-provided data electronically from local area networks.	3.1.2
3.4.1.6	SCTGEN shall have the capability of ingesting user-provided data in binary format.	3.1.2, 3.1.12
3.4.2.1	SCTGEN shall output generated test data (binary-compatible) in the following formats: a) File containing a serial stream of data. b) File containing a base set of serial stream of data with an update file.	3.1.2, 3.1.12, 5.2.1, 5.3.1
3.4.2.2	SCTGEN shall store generated test data files in the following physical media: a) System disks. b) Remotely mounted disk system. c) 4-mm DAT.	3.1.2
3.4.2.3	SCTGEN shall support data formats as required by the EOS Simulator Card data transmission hardware.	Derived
4.1	SCTGEN shall generate test data as a CADU serial stream up to 25 Gbytes in size.	5.2.1, 5.3.1
4.2	SCTGEN shall generate test data simulating data from at least 8 SCIDs.	5.3.1
4.3	SCTGEN shall generate test data simulating data from at least 64 VCIDs.	5.1.4, 5.5.16
4.4	SCTGEN shall generate test data simulating data from at least 512 APIDs.	5.1.3

4.5	SCTGEN shall generate test data simulating data with at least 60 data segments from a specific APID per test scenario.	Derived
4.6	SCTGEN shall generate test data simulating timecode with a resolution of at least 1 microsecond.	5.5.1, 5.5.2, 5.5.3
4.7	SCTGEN shall generate data sets up to a maximum of 25 Gbytes in size, consisting of multiple files each no more than 2 Gbytes in length within 6 hours.	Derived
4.8	SCTGEN shall generate a CADU serial stream of up to 25 Gbytes within 6 hours.	Derived

ACRONYMS AND ABBREVIATIONS

Term	Definition
ANSI	American National Standards Institute
APID	Application Process Identifier
ASCII	American Standard Code for Information Interchange
CADU	Channel Access Data Unit
CCB	Configuration Control Board
CCSDS	Consultative Committee for Space Data Systems
CDS	Control and Display
CI	Configuration Item
CLCW	Command Link Control Word
COTS	Commercial Off-the-Shelf
CRC	Cyclical Redundancy Check
CVCDU	Coded Virtual Channel Data Unit
DAAC	Distributed Active Archive Center
DCN	Documentation Change Notice
EBnet	EOSDIS Backbone Network
EDOS	EOS Data and Operations System
EDS	Expedited Data Set
EDS	Expedited Data Sets
EDU	EDOS Data Unit
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
ESDIS	Earth Science Data and Information System
ETS	EOSDIS Test System
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HRS	High-rate System
I/O	Input/Output
ICD	Interface Control Document
MSB	Microelectronic Systems Branch
M_PDU	Multiplexed Protocol Data Unit
Nascom	NASA Communications
OMT	Object Modeling Technique
OSF	Open Software Foundation
PDS	Production Data Set
RDF	Rate-Buffered Data File
SCID	Spacecraft Identifier
SCSI	Small Computer System Interface
STGEN	Simulated Telemetry Generation
STP	Software Through Pictures
TCP/IP	Transmission Control Protocol/Internet Protocol
TDRSS	Tracking and Data Relay Satellite System
TGT	TDRSS Ground Terminal
TPGEN	Test Pattern Generator
TSS	TDRSS Service Session
VCA	Virtual Channel Access
VCDU	Virtual Channel Data Unit
VCID	Virtual Channel Identifier
VLSI	Very Large Scale Integration