A Project Document of the
ATC Application Programming Interface Working Group

ATC APIVS SDD v01.02

Software Design Description (SDD) for the Advanced Transportation Controller (ATC)
Application Programming Interface (API) Validation Suite Project

May 22, 2015

SDD in support of: USDOT Contract # DTFH61-11-D-00052, Task Order # T-13-003

For use by: Siva Narla, Chief Engineer and ITS Standards Manager
Institute of Transportation Engineers

George Chen and Douglas Tarico, Co-Chairs
ATC API Working Group

Ralph W. Boaz, Project Manager and Systems Engineer
ATC API Reference Implementation Project

Members of the ATC API Working Group

Consulting Team for the ATC API RI Project

Prepared by: Michael Gallagher, Intelight, Inc.
Douglas Crawford, Intelight, Inc.
Grant Gardner, Intelight, Inc.
Ralph W. Boaz, Pillar Consulting, Inc.

© Copyright 2015 AASHTO/ITE/NEMA. All rights reserved.
## CHANGE HISTORY

<table>
<thead>
<tr>
<th>DATE</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/04/14</td>
<td>Initial Draft Software Design Document (SDD) v01.00.</td>
</tr>
</tbody>
</table>
| 1/30/15  | Updated version to v1.01  
Added additional detail for all entity descriptions.  
Added more detailed overview description and usage information  
Added specific APIs that will be tested in each test scenario.               |
| 5/22/15  | Updated version to v1.02  
Added appendices. Enhanced graphics. Made terminology consistent throughout.  
Made changes based on comments gathered through API WG reviews/walkthroughs. |
NOTICE

Joint NEMA, AASHTO and ITE Copyright and
Advanced Transportation Controller (ATC)
Application Programming Interface (API) Working Group

These materials are delivered "AS IS" without any warranties as to their use or performance.

AASHTO/ITE/NEMA AND THEIR SUPPLIERS DO NOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THESE MATERIALS. AASHTO/ITE/NEMA AND THEIR SUPPLIERS MAKE NO WARRANTIES, EXPRESSED OR IMPLIED, AS TO NON-INFRINGEMENT OF THIRD PARTY RIGHTS, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AASHTO, ITE, NEMA, OR THEIR SUPPLIERS BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY CLAIM OR FOR ANY CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS ARISING FROM YOUR REPRODUCTION OR USE OF THESE MATERIALS, EVEN IF AN AASHTO, ITE, OR NEMA REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential, or special damages, or exclusion of implied warranties, so the above limitations may not apply to you.

Use of these materials does not constitute an endorsement or affiliation by or between AASHTO, ITE, or NEMA and you, your company, or your products and services.

If you are not willing to accept the foregoing restrictions, you should immediately return these materials.

ATC is a trademark of NEMA/AASHTO/ITE.
CONTENTS

1 INTRODUCTION................................................................................................................................. 6
  1.1 Purpose ........................................................................................................................................ 6
  1.2 Scope ......................................................................................................................................... 6
  1.3 Document Organization ......................................................................................................... 6
  1.4 Definitions of Terms and Acronyms ...................................................................................... 7
  1.5 References ............................................................................................................................ 10

2 APIVS VALIDATION TESTING OVERVIEW .................................................................................... 11
  2.1 APIVS Test Environment .................................................................................................... 11
  2.2 APIVS User Interface ......................................................................................................... 12
  2.3 APIVS Portability ................................................................................................................ 12

3 APIVS VIEWS ...................................................................................................................................... 14
  3.1 APIVS Decomposition Description .................................................................................... 14
    3.1.1 Module Decomposition .............................................................................................. 15
    3.1.2 Concurrent Process Decomposition ...................................................................... 16
    3.1.3 Data Decomposition ................................................................................................. 17
  3.2 APIVS Dependency Description .......................................................................................... 17
  3.3 APIVS Interface Description ............................................................................................... 17
    3.3.1 InputXMLParse Interface ....................................................................................... 17
    3.3.2 InterpretXML Interface .......................................................................................... 17
    3.3.3 Validate .................................................................................................................. 17
    3.3.4 OutputXMLReport Interface ................................................................................. 17
    3.3.5 EmulateFP Interface ............................................................................................... 18
    3.3.6 EmulateFIO Interface ............................................................................................ 18
    3.3.7 VirtualLoopbackAsync Interface ........................................................................ 18
    3.3.8 VirtualLoopbackSync Interface ........................................................................... 18
  3.4 APIVS Detailed Design ......................................................................................................... 18
    3.4.1 InputXMLParse Entity .............................................................................................. 18
    3.4.2 InterpretXML Entity ................................................................................................. 18
    3.4.3 Validate Entity ......................................................................................................... 19
    3.4.4 OutputXMLReport Entity ..................................................................................... 19
    3.4.5 EmulateFP Entity .................................................................................................... 20
    3.4.6 EmulateFIO Entity .................................................................................................. 20
    3.4.7 VirtualLoopbackAsync Entity ............................................................................ 21
    3.4.8 VirtualLoopbackSync Entity ............................................................................... 22

4 REQUIREMENTS TRACEABILITY ..................................................................................................... 23

APPENDICES ........................................................................................................................................... 28

APPENDIX A: SUMMARY CONFORMANCE REPORT ............................................................................ 29

APPENDIX B: DETAILED CONFORMANCE REPORT ............................................................................. 30

APPENDIX C: TEST CASE SCENARIOS ................................................................................................ 31
  C.1 FPUI_Every_Func Validation Test .................................................................................... 31
  C.2 FIO_Every_Func Validation Test ..................................................................................... 31
  C.3 TOD_Every_Func Validation Test .................................................................................. 31
  C.4 FPUIAPI_Scenario1 Validation Test ............................................................................. 31
  C.5 FPUIAPI_Scenario2 Validation Test ................................................................................ 31
  C.6 FIOAPI_Scenario1 Validation Test ................................................................................ 31
C.7  FIOAPI_Scenario2 Validation Test ................................................................................. 32
C.8  FIOAPI_Scenario3 Validation Test ................................................................................. 33
C.9  TODAPI_Scenario1 Validation Test ................................................................................. 34

APPENDIX D: VALIDATION SUITE ENGINE (VSE) EXECUTABLE COMMAND LINE OPTIONS .... 35
APPENDIX E: SYNTAX FOR FLAT FILES USED BY THE VSE .................................................... 37
1 INTRODUCTION

1.1 Purpose

This Software Design Description (SDD) is for the test software referred to as the Advanced Transportation Controller (ATC) Application Programming Interface (API) Validation Suite (APIVS) software. It is being developed as part of the “Reference Implementation of ATC 5401 Application Programming Interface (API) Standard Version 2” project funded by the United States Department of Transportation (USDOT) Contract Number DTFH61-11-D-00052, Work Order T-13003 (referred to as the APIRI Project). This SDD provides the software structure, software components, interfaces and data necessary for the subsequent implementation of the API Verification Suite software. It provides traceability from the requirements established in the APIVS Software Requirements Specification (SRS) v02.03 and the APIVS software design to ensure that each requirement is completely addressed. This SDD has been developed for:

a) The USDOT Intelligent Transportation Systems (ITS) Joint Program Office (JPO) who is sponsoring the work and requires the use of a formal software development process;

b) The consulting team contracted to develop the software described; and

c) The consultants, manufacturers, and public transportation professionals who participate in the API Working Group (WG) who provide domain expertise, quality assurance, testing assistance and the maintenance of the software; and

d) The transportation industry as a whole that will depend upon the software produced from this project to support operational programs on ATC controller equipment.

1.2 Scope

The ATC 5401 Standard defines a software platform that, when combined with the ATC O/S, forms a universal interface for application programs to run on ATC units. This API software allows application programs to be written so that they may run on any ATC unit regardless of the manufacturer. It also provides the capability for multiple application programs to run concurrently by sharing the fixed resources of the controller.

The APIRI project builds upon the ATC 5401 Standard to provide a fully functional and verified open source reference implementation (APIRI software). The APIRI software satisfies all of the ATC 5401 Standard functional requirements and can be leveraged by the user community (including manufacturers), with the goal of simplifying ATC application development.

This document describes the design the test software referred to as the Advanced Transportation Controller (ATC) Application Programming Interface (API) Validation Suite (APIVS) software.

1.3 Document Organization

This SDD provides a detailed description of the major ATC APIVS software. The design information has been organized based on IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions. The design is presented using the following concepts:

- Design Entity – An element of design that is structurally and functionally distinct from other elements and that is separately named and referenced. Sometimes more simply referred to as "entity."
- Design Entity Attribute – A characteristic or property of a design entity.
- Design View – A subset of design entity attribute information that is specifically suited to the needs of a software project activity.
Each major functional area is described in a separate section and organized by the following design views:

- Decomposition Description – This design view provides an entity breakdown for the library;
- Dependency Description – This design view highlights any dependencies on the software to be developed;
- Interface Description – This design view presents the high level interfaces of the software; and
- Detailed Design – This design view presents detailed information for each design entity.

This SDD may be updated during the implementation phase of the project to reflect design elements not accounted for in previous phases of development. A traceability matrix is included to ensure all APIVS SRS requirements are incorporated and linked to specific design elements.

### 1.4 Definitions of Terms and Acronyms

Table 1 provides definitions of terms and acronyms used in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AASHTO</td>
<td>American Association of State Highway and Transportation Officials</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>API Managers</td>
<td>API software that manages an ATC resource for use by concurrently running application programs.</td>
</tr>
<tr>
<td>API Utilities</td>
<td>API software not included in the API Managers that is used for configuration purposes.</td>
</tr>
<tr>
<td>APIRI Project</td>
<td>Entire project managed by this PMP including software, hardware and documentation.</td>
</tr>
<tr>
<td>APIRI Software</td>
<td>API Reference Implementation Software</td>
</tr>
<tr>
<td>APIVS Software</td>
<td>API Validation Suite Software</td>
</tr>
<tr>
<td>APIVSXML</td>
<td>APIVS Extensible Markup Language (XML) as defined by the <em>API Validation Suite APIVSXML Specification</em> (see Section 1.5 References). This version of XML includes elements for use with the APIVS software. APIVSXML is used to create test case specifications that are both human-readable and machine-readable. APIVSXML and XML are used synonymously within this document.</td>
</tr>
<tr>
<td>Application Program</td>
<td>Any program designed to perform a specific function directly for the user or, in some cases, for another application program. Examples of application programs include word processors, database programs, Web browsers and traffic control programs. Application programs use the services of a computer's O/S and other supporting programs such as an application programming interface.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programmer Interface</td>
</tr>
<tr>
<td>ATC</td>
<td>Advanced Transportation Controller</td>
</tr>
<tr>
<td>ATC Device Drivers</td>
<td>Low-level software not included in a typical Linux distribution that is necessary for ATC-specific devices to operate in a Linux O/S environment.</td>
</tr>
<tr>
<td>ATP</td>
<td>Authorization to Proceed</td>
</tr>
<tr>
<td>Board Support Package</td>
<td>Software usually provided by processor board manufacturers which provides a consistent software interface for the unique architecture of the board. In the case of the ATC, the Board Support Package also includes the O/S</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BSP</td>
<td>See Board Support Package</td>
</tr>
<tr>
<td>ConOps</td>
<td>Concept of Operations</td>
</tr>
<tr>
<td>CO</td>
<td>Contracting Officer</td>
</tr>
<tr>
<td>COR</td>
<td>Contract Officer’s Representative</td>
</tr>
<tr>
<td>COTM</td>
<td>Contract Officer’s Task Manager</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit. A programmable logic device that performs the instruction, logic and mathematical processing in a computer.</td>
</tr>
<tr>
<td>Device Driver</td>
<td>A software routine that links a peripheral device to the operating system. It acts like a translator between a device and the application programs that use it.</td>
</tr>
<tr>
<td>FHWA</td>
<td>Federal Highway Administration</td>
</tr>
<tr>
<td>FIO</td>
<td>Field Input and Output</td>
</tr>
<tr>
<td>FIOMAN</td>
<td>Field I/O Manager</td>
</tr>
<tr>
<td>FIOMSG</td>
<td>Field I/O Message Scheduler</td>
</tr>
<tr>
<td>FPMW</td>
<td>Front Panel Manager Window</td>
</tr>
<tr>
<td>FPUI</td>
<td>Front Panel User Interface</td>
</tr>
<tr>
<td>H/W</td>
<td>Hardware</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITE</td>
<td>Institute of Transportation Engineers</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transportation Systems</td>
</tr>
<tr>
<td>JC</td>
<td>Joint Committee</td>
</tr>
<tr>
<td>JPO</td>
<td>Joint Program Office</td>
</tr>
<tr>
<td>Linux</td>
<td>Low-level software that is freely available in the Linux community for use with common hardware components operating in a standard fashion.</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>The Unix-like operating system kernel that was begun by Linus Torvalds in 1991. The Linux Kernel provides general O/S functionality. This includes functions for things typical in any computer system such as file I/O, serial I/O, interprocess communication and process scheduling. It also includes Linux utility functions necessary to run programs such as shell scripts and console commands. It is generally available as open source (free to the public). The Linux Kernel referenced in this standard is defined in the ATC Controller Standard Section 2.2.5, Annex A and Annex B.</td>
</tr>
<tr>
<td>Loopback Driver</td>
<td>A virtual device driver that loops back the output ports to a device to the input ports from a device without actually going to through the physical device.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>Operational User</td>
<td>A technician or transportation engineer who uses the controller to perform its operational tasks.</td>
</tr>
<tr>
<td>O/S</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PMP</td>
<td>Project Management Plan</td>
</tr>
<tr>
<td>POP</td>
<td>Period of Performance</td>
</tr>
<tr>
<td>PRL</td>
<td>Protocol Requirements List</td>
</tr>
<tr>
<td>RI</td>
<td>Reference Implementation</td>
</tr>
<tr>
<td>RITA</td>
<td>Research and Innovative Technology Administration</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Clock</td>
</tr>
<tr>
<td>RTM</td>
<td>Requirements Traceability Matrix</td>
</tr>
<tr>
<td>SDD</td>
<td>Software Design Descriptions</td>
</tr>
<tr>
<td>SDO</td>
<td>Standards Development Organization</td>
</tr>
<tr>
<td>SE</td>
<td>Systems Engineer</td>
</tr>
<tr>
<td>SEP</td>
<td>Systems Engineering Process</td>
</tr>
<tr>
<td>SEMP</td>
<td>Systems Engineering Management Plan</td>
</tr>
<tr>
<td>SOW</td>
<td>Statement of Work</td>
</tr>
<tr>
<td>SPDD</td>
<td>Serial Port Device Driver</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirements Specification</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell. An encrypted network protocol for initiating text-based shell sessions.</td>
</tr>
<tr>
<td>S/W</td>
<td>Software</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Determined</td>
</tr>
<tr>
<td>TCS</td>
<td>Test Case Specification</td>
</tr>
<tr>
<td>Test Case Specification</td>
<td>For the purposes of the APIVS software, a TCS is a test case file written in APIVSXML.</td>
</tr>
<tr>
<td>TOD</td>
<td>Time of Day</td>
</tr>
<tr>
<td>TOPR</td>
<td>Task Order Proposal Request</td>
</tr>
<tr>
<td>TX</td>
<td>Transmission</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>USDOT</td>
<td>United States Department of Transportation</td>
</tr>
<tr>
<td>User Developer</td>
<td>A software developer that designs and develops programs for controllers.</td>
</tr>
<tr>
<td>VD</td>
<td>Virtual Display: the virtual front-panel display data maintained by the VSE during a test run.</td>
</tr>
<tr>
<td>VSE</td>
<td>Validation Suite Engine: the main executable program of the APIVS software.</td>
</tr>
<tr>
<td>Walkthrough</td>
<td>A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content.</td>
</tr>
<tr>
<td>WBS</td>
<td>Work Breakdown Structure</td>
</tr>
<tr>
<td>WG</td>
<td>Working Group</td>
</tr>
</tbody>
</table>
1.5 References

http://standards.ieee.org/index.html

http://standards.ieee.org/index.html

http://www.ite.org/standards/index.asp

Institute of Transportation Engineers, *API Validation Suite APIVSXML Specification v02.00*. ATC Joint Committee, 31 December 2010.
http://www.ite.org/standards/index.asp

Institute of Transportation Engineers, *Advanced Transportation Controller (ATC) Application Programming Interface (API) Validation Suite (APIVS) Concept of Operations (ConOps) v02.04*. ATC Joint Committee, 20 November 2014.
http://www.ite.org/standards/index.asp

http://www.ite.org/standards/index.asp

Institute of Transportation Engineers, *ATC 5401 Application Programming Interface (API) Standard for the Advanced Transportation Controller (ATC) v02*. ATC Joint Committee, 15 September 2013.
http://www.ite.org/standards/index.asp

Institute of Transportation Engineers, *ATC APIRI PMP v01.03 Project Management Plan (PMP) for the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation Project*. ATC Joint Committee, 5 November 2014.
http://www.ite.org/standards/index.asp

http://www.ite.org/standards/index.asp

Institute of Transportation Engineers, *User Comment Draft ATC 5201 Advanced Transportation Controller (ATC) Standard Version 06.10*. ATC Joint Committee, 30 July 2012.
http://www.ite.org/standards/index.asp


2 APIVS VALIDATION TESTING OVERVIEW

The APIVS software is made up of an executable application program, a configuration file and test case specifications (TCSs). The main executable program is the Validation Suite Engine (VSE). The configuration file is used to set up the test environment for the tests defined by TCSs. The TCSs are written in XML format which are parsed and interpreted by the VSE, invoking the API software tests, and validating results.

In order to automate this testing, a capability needs to exist which brings the outputs of the functioning API software back to the inputs so that proper operation can be verified. This “loopback” capability could be done with some limitations using physical loopback cables connected to the various external serial inputs and outputs of the ATC unit. This design of the APIVS software incorporates a virtual loopback capability where the loopback occurs internal to the ATC unit. This includes a set of asynchronous and synchronous virtual loopback drivers for the Linux kernel which replace like drivers on the ATC Engine Board for the purposes of testing. This allows the testing to be more comprehensive and facilitate the testing for the test personnel.

2.1 APIVS Test Environment

In order to perform a consistent software validation of an API software implementation, the software under test (API software) must be isolated (to the extent possible) from other software or systems that may unpredictably influence its operation. The Engine Board based architecture specified in the ATC 5201 Standard is ideal for this purpose by isolating the computational components and the software environment of the controller unit from other components of the controller unit. The APIVS test environment is shown in Figure 1. It consists of an ATC unit and a personal computer (PC). The PC interface is necessary to load test software, initiate tests, and extract results. It is possible that the PC can also serve in the operation of some tests. Details of the operation of the test environment and tests are to be described using test procedures.

![Figure 1. API test environment uses a personal computer connected to the console port of the controller.](image)

The layered software environment for the APIVS software (see Figure 2) is similar to the layered organization of the ATC software described in Section 2.1 of the ATC APIVS Software Requirements Specification (see Section 1.5, References). In this case, the user is now a Tester which may be a User Developer, Test Engineer or Test Technician. The APIVS software resident on the Engine Board exercises the API software and records results. During testing, the APIVSLoopbackDrivers replace the standard Board Support Package (BSP) Layer asynchronous and synchronous drivers so that the API software can be tested within the controller without the need of external cabling or other devices.
2.2 APIVS User Interface

The APIVS software has a command line user interface for the VSE. The user must be able to open a Linux shell on the controller which can either be a serial connection on SP4 or a SSH connection over IP. The APIVS directory contains the VSE executable, its default configuration file, and the XML format TCS. The VSE reads the TCS and writes an XML format result file.

The VSE executable accepts several command line options to provide the following functionality:
1. Override the default name of the FIO emulator configuration file;
2. Override the default name of the FPUI emulator configuration file;
3. Specify the name(s) of one or more selected XML test scenario files to process;
4. Process all of the XML test scenario files;
5. Override the default name of XML result file;
6. Repeat execution a specified number of times or indefinitely; and
7. Stop on first error or continue running remaining tests on error.

The VSE executable returns a value of 0 to indicate a pass or conformance condition for the test run. The VSE executable returns a value of -1 to indicate a failure or non-conformance condition for the test run.

A detailed description of the VSE executable command line options and the VSE configuration file are found in Appendix D. A detailed description of using APIVXML to create a TCS is found in the API Validation Suite APIVXML Specification (see Section 1.5, References).

The executable VSE program is easily incorporated into Linux shell scripts. Multiple instances may be arranged to run concurrently from the command line or from shell scripts.

2.3 APIVS Portability

The VSE executable must be compiled for each target engine board and may be included as part of the ATC vendor’s Linux distribution. The APIVS loopback drivers must also be compiled for each target engine board and may also be bundled as part of the ATC vendor’s Linux distribution. All APIVS software
programs are written in the C Programming Language and are designed to be compatible with the uClibc C library.
3 APIVS VIEWS

The following design views show decomposition, dependency, interface and detail aspects of the proposed design for the APIVS software.

3.1 APIVS Decomposition Description

Figure 3 shows the architecture of the APIVS software. The arrows show the general flow of information. The items in the Application Layer and the ATC Board Support Package Layer are being developed under the scope of this SDD. The items in the API Software Layer represent the API software under test. The architecture allows the APIVS software to invoke test cases which exercise the API software and compare the output of the API software to a known result.

![APIVS Software Architecture Diagram]

Although most of the interfaces in Figure 3 actually have information flowing both directions, a simplified description of the data flow within the APIVS software can be thought of as follows:

a) The TCS and the Expected Result Flat Files (see Appendix E for the syntax used in these flat files) are read by the VSE.

b) The VSE interacts with the API software via the API Software Libraries.

c) The API libraries interact with the managers which allow the Front Panel and Field I/O devices to be shared across applications.

d) The managers send out commands through serial drivers as if they were going to physical devices. In the case of the APIVS software, VirtualLoopbackAsync and VirtualLoopbackSync, redirect the commands normally sent to the ATC unit’s Front Panel and Field I/O devices back to the VSE.

e) The VSE emulates operation of the Front Panel and Field I/O devices internally and allows the
comparison of the API software as specified by the TCS to the expected result of the test case specified by the flat file.
f) An API Conformance Report is generated according to options set by the Tester.

3.1.1 Module Decomposition

Figure 4 provides further decomposition of the APIVS software into design entities. The arrows show the general flow of information between the entities. The subsections below describe their function.

![Diagram showing APIVS Software Architecture Decomposition into Design Entities]

3.1.1.1 InputXMLParse Entity

This design entity is responsible for parsing XML from the TCS and passing each element parsed for interpretation.

3.1.1.2 InterpretXML Entity
This design entity provides functionality to translate parsed XML elements into ATCAPI library function calls and to validate calling parameters and syntax.

3.1.1.3 Validate Entity

This design entity provides comparison and checking functionality in the validation of field I/O command message frames and front panel escape sequences generated by the ATCAPI library calls.

3.1.1.4 OutputXMLReport Entity

This design entity provides functions to generate and record results to the XML-format conformance report file.

3.1.1.5 EmulateFP Entity

This design entity provides a full emulation of the ATC front panel hardware device.

3.1.1.6 EmulateFIO Entity

This design entity provides emulation of the ATC cabinet field I/O devices.

3.1.1.7 VirtualLoopbackAsync Entity

This design entity comprises a Linux kernel module which provides a virtual asynchronous loopback serial connection.

3.1.1.8 VirtualLoopbackSync Entity

This design entity comprises a Linux kernel module which provides a virtual synchronous loopback serial connection.

3.1.2 Concurrent Process Decomposition

3.1.2.1 VSE

This entity is the main application program for the APIVS software performing the key functions. It also invokes concurrent threads EmulateFP and EmulateFIO.

3.1.2.2 EmulateFP

This entity runs as a separate concurrent thread from the main VSE program in order to handle front panel related serial port events.

3.1.2.3 EmulateFIO

This entity runs as a separate concurrent thread from the main VSE program in order to handle field I/O related serial port events.

3.1.2.4 VirtualLoopbackAsync

This entity is a Linux kernel device driver module and depends on functionality provided by the Linux kernel system libraries.
3.1.2.5  VirtualLoopbackSync

The VirtualLoopbackSync Entity is a Linux kernel device driver module and depends on functionality provided by the Linux kernel system libraries.

3.1.3  Data Decomposition

3.1.3.1  VirtualDisplayTable Entity

This data entity comprises a storage structure for a virtual representation of the appearance of the ATC front panel display. Access to this data entity is shared within the VSE.

3.1.3.2  FIOMessageTable Entity

This data entity comprises a table of ATC FIO command frame data and corresponding response frame data. Access to this data entity is shared within the VSE.

3.2  APIVS Dependency Description

The VSE depends on a functioning implementation of API software. The InterpretXML entity depends on the functionality of the FIO, FPUI and TOD libraries provided as part of the platform API software.

The EmulateFIO design entity depends on a synchronous serial port loopback device connection (e.g. VirtualLoopbackSync) and several operating system functions provided by the Linux kernel and ATC BSP.

The EmulateFP process design entity depends on an asynchronous serial port loopback device connection (e.g. VirtualLoopbackAsync) and operating system functions provided by the Linux kernel.

The InputXMLParse and OutputXMLReport design entities depend on functions provided by the libexpat open source XML parsing library along with file system interface functions provided by the Linux kernel.

3.3  APIVS Interface Description

The interfaces discussed in the following subsections refer to those that connect the APIVS software to the API software and to the elements external to the VSE application program.

3.3.1  InputXMLParse Interface

The InputXMLParse entity makes use of the Linux file system interface to open and read the XML format TCS files.

3.3.2  InterpretXML Interface

The InterpretXML design entity makes use of the functional interface provided by the API libraries for FIO, FPUI and TOD.

3.3.3  Validate

The validate design entity makes use of the Linux file system interface to open and read the flat files containing the expected results for the test case.

3.3.4  OutputXMLReport Interface
The OutputXMLReport entity makes use of the Linux file system interface to open and write XML-format output of the conformance report for the latest validation test run.

3.3.5 **EmulateFP Interface**

The EmulateFP entity makes use of the Linux I/O interface provided by the asynchronous serial device configured as the loopback port for FPUI front panel emulation (i.e. VirtualLoopbackAsync).

3.3.6 **EmulateFIO Interface**

The EmulateFIO entity makes use of the Linux I/O interface provided by the synchronous serial device configured as the loopback port for FIO device emulation (i.e. VirtualLoopbackSync).

3.3.7 **VirtualLoopbackAsync Interface**

The VirtualLoopbackAsync entity is a Linux Device Driver module and presents its interface via a pair of character type devices, supporting the same interface as a standard Linux serial port device.

3.3.8 **VirtualLoopbackSync Interface**

The VirtualLoopbackSync entity is a Linux Device Driver module and presents its interface via a pair of character type devices, supporting the same interface as that defined in sections B5.3 and B6.3 of the ATC 5201 Standard.

3.4 **APIVS Detailed Design**

3.4.1 **InputXMLParse Entity**

The InputXMLParse design entity uses functions from the open source libexpat library for XML parsing to recognize the test instructions TCS and to translate the instructions into function calls for InterpretXML.

The following are the libexpat library functions which are used to provide the XML parser service:

- XML_ParserCreate()
- XML_SetUserData()
- XML_SetElementHandler()
- XML_SetCharacterDataHandler()
- XML_Parse()
- XML_ParserFree()

The InputXMLParse entity provides the following functions to decode the XML tag elements when called by the XML parser service on each opening and closing tag in the XML file:

- pCode_start()
- pCode_end()
- pCode_parseOpen()
- pCode_parseClose()


3.4.2 **InterpretXML Entity**

The InterpretXML design entity performs calls to the API library functions using function names and parameters as translated from XML tag elements in the TCS files.

Function call syntax, parameters and return codes are validated for conformance with the ATC 5401 Standard using a look-up table of function templates.

**Table 2. Function Template Look-up Table with Example Data.**
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function Pointer</th>
<th>Return Type</th>
<th>Num Args</th>
<th>Var Types</th>
<th>Arg Directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>“fpui_apiver”</td>
<td>fnFpuiApiver</td>
<td>VAR_PCHAR</td>
<td>2</td>
<td>VAR_FPUIH, VAR_INT</td>
<td>FUNC_IN, FUNC_IN</td>
</tr>
<tr>
<td>“fio_query_fiod”</td>
<td>fnFioQueryFiod</td>
<td>VAR_INT</td>
<td>3</td>
<td>VAR_FAPPH, VAR_FPORT, VAR_FDEVT</td>
<td>FUNC_IN, FUNC_IN, FUNC_IN</td>
</tr>
<tr>
<td>“tod_set_timesrc”</td>
<td>fnTodSetTimesrc</td>
<td>VAR_INT</td>
<td>1</td>
<td>VAR_INT</td>
<td>FUNC_IN</td>
</tr>
</tbody>
</table>

Where:
- fnFpuiApiver is a function pointer template corresponding to the fpui_apiver library function;
- fnFioQueryFiod is a function pointer template corresponding to the fio_query_fiod library function;
- fnTodSetTimesrc is a function pointer template corresponding to the tod_set_timesrc library function;
- VAR_PCHAR is an enumerated type representing a character string type;
- VAR_INT is an enumerated type representing an integer type;
- VAR_FPUIH is an enumerated type representing a fpui_handle type;
- VAR_FAPPH is an enumerated type representing a FIO_APP_HANDLE type;
- VAR_FPORT is an enumerated type representing a FIO_PORT type;
- VAR_FDEVT is an enumerated type representing a FIO_DEVICE_TYPE type;
- FUNC_IN is an enumerated type representing that the corresponding function argument is an input;

Each table entry corresponds to one of the API library functions, and contains that function’s attributes such that validation of the completeness and correctness may be performed.

Logic and comparison operations are also translated and performed as instructed in the TCS files.

### 3.4.3 Validate Entity

The Validate design entity performs comparison of Field I/O command frame content stored in the FIOMessageTable data entity against flat files of expected content.

The Validate entity also performs comparison of front panel display data stored in the VirtualDisplayTable data entity against flat files of expected data.

The input to the Validate entity is the character data to be compared and a file pointer to the flat file to be compared against.

The output of the Validate entity is a Boolean result indicating if the two inputs are equal.

### 3.4.4 OutputXMLReport Entity

The OutputXMLReport design entity contains output stream processing functionality to generate the XML format validation report containing results of the corresponding test run.

The format of the report file XML allows display at increasing detail level from a basic pass/fail result status for the test overall, to examining the parameters and result status at each stage of the validation test. Examples of such output are included in Appendix A.
The output functionality is called at various points during the processing of the TCS file instructions with output verbosity according to the current logging level.

### 3.4.5 EmulateFP Entity

The EmulateFP design entity is a front panel device emulator which opens the asynchronous serial port at the opposite end of a serial loopback from that of the API library FPUI Front Panel Manager.

Incoming escape sequences or ASCII characters are read and a virtual display representation of the actual front panel appearance is maintained accordingly.

Corresponding response sequences are sent as required.

#### 3.4.5.1 Virtual Display

The VSE supports an ATC front panel emulator to interpret the escape sequence commands sent by the ATCAPI implementation and will construct a VirtualDisplayTable data entity.

**Table 4. VirtualDisplayTable Data Structure**

<table>
<thead>
<tr>
<th>pText</th>
<th>pAttribs</th>
</tr>
</thead>
<tbody>
<tr>
<td>display text content for row 1</td>
<td>display attribute content for row 1</td>
</tr>
<tr>
<td>display text content for row 2</td>
<td>display attribute content for row 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>display text content for last row</td>
<td>display attribute content for last row</td>
</tr>
</tbody>
</table>

This data entity may be interrogated and compared against the expected contents of the front panel display at any point in time, as specified by the APIVSXML.

Expected displays are kept in human-readable flat files that may be referenced by the APIVSXML.

#### 3.4.5.2 Input Generation

The VSE supports the ability to generate input to the FPM as if the user had actually pressed a function key on the physical ATC front panel.

The generation of this input is under the control of the APIVSXML.

The input to be “generated” and passed to the FPM is stored in human-readable flat files that may be referenced by `<FPULInput>` elements in the TCS file.

### 3.4.6 EmulateFIO Entity

The EmulateFIO design entity is a command/response frame FIO device emulator which opens the synchronous serial port at the opposite end of a serial loopback from that of the API library FIO driver.

Incoming command frames are read and stored via the FIOMessageTable data entity and corresponding response frames are sent as required.
Table 3. FIOMessageTable Data Structure

<table>
<thead>
<tr>
<th>cmdType</th>
<th>cmdRcvd</th>
<th>maxCmdSize</th>
<th>lastCmdBuf</th>
<th>lastCmdSize</th>
<th>responseEntry</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ_MOD_STATUS_CMD</td>
<td>RQ_MOD_STATUS_CMD_SIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where:
- `cmdType` is an integer type containing the command frame identifier;
- `cmdRcvd` is a Boolean type set true when the corresponding command frame has been received by EmulateFIO;
- `maxCmdSize` is an integer type containing the maximum frame data storage required;
- `lastCmdBuf` is a buffer containing the data of the last command frame of this type received by EmulateFIO;
- `lastCmdSize` is the actual size of the last command frame of this type received by EmulateFIO;
- `RQ_MOD_STATUS_CMD` is an example defined constant representing the command frame identifier for the “Request Module Status” command, i.e. 49;
- `RQ_MOD_STATUS_CMD_SIZE` is an example defined constant representing the maximum frame size of the “Request Module Status” command, i.e. 2; and
- `responseEntry` is a pointer to a data structure representing the corresponding response frame to be sent by EmulateFIO that is populated by the InterpretXML entity according to instructions in the TCS.

This entity executes as a concurrent thread alongside the main VSE program thread.

### 3.4.6.1 Field I/O Command Message Processing / Emulating

The VSE supports a Field I/O Command Message Processor / Emulator to interpret and process Command Messages sent by the FIOM.

This processor maintains the last Command Message received, of each Command Message Frame type, so that comparison and validation can be performed by the VSE as instructed by the APIVSXML. This Command Message Frame may be interrogated and compared against the expected Command Message Frame at any point in time, as specified by the APIVSXML.

Expected Command Message Frames are kept in human-readable flat files that may be referenced by the APIVSXML.

### 3.4.6.2 Field I/O Response Message Processing

The VSE supports the ability to generate Field I/O Response Messages to the FIOM as if an FIOD was actually attached to the system.

The generation of this input is under the control of the APIVSXML. The input to be “generated” and passed to the FIOM is stored in human-readable flat files that may be referenced by the APIVSXML.

### 3.4.7 VirtualLoopbackAsync Entity

This VirtualLoopbackAsync design entity comprises a Linux kernel driver module which presents a pair of asynchronous serial port devices which are looped back internally such that data written to one port is received on the other.
The module has configuration parameters for port naming purposes. By default the device names "/dev/sp6_loopback_a" and "/dev/sp6_loopback_b" are used.

The serial driver selection for the FPUI library and FPM is handled by specifying a command line argument when starting the Front Panel Manager process. Typically, the Front Panel Manager process is started with a single command line argument specifying the serial port connected to the physical front panel ("FrontPanelManager /dev/sp6"). For APIVS testing, the Front Panel Manager process should be started with the virtual sp6 loopback driver ("FrontPanelManager /dev/sp6_loopback_a"). The paired loopback device name "/dev/sp6_loopback_b" should be specified in the VSE program configuration file “FPUILoopbackDevice” parameter described in section A5.

Each loopback port device conforms to standard Linux serial port interface requirements and those of the ATC 5201 Standard.

3.4.8 VirtualLoopbackSync Entity

The VirtualLoopbackSync design entity comprises a Linux kernel driver module which presents pairs of synchronous serial port devices which are looped back internally such that data written to one port is received on the other. Each port device conforms to the ATC SPxs synchronous driver interface requirements of sections B5.3 and B6.3 of the ATC 5201 Standard, including the Kernel Level Interface global functions. The naming of these global functions will differ from the standard names in order to avoid conflicting with such standard functions which may be built-in to the kernel on a conformant ATC platform. This implementation will name the global functions with the prefix "vs_", e.g. "vs_sdlc_kernel_open()".

FIO API implementations which do not support the Kernel Level Interface global functions, but which interact directly with user-space serial port devices, will need to be configurable to use alternate port names which match those of the VirtualLoopbackSync driver, as described above, for validation testing purposes with virtual loopback.

The ATC APIRI field I/O driver will be configurable for compatibility with the APIVS function names. The serial driver selection for the FIO ATC API is handled by specifying a Linux module option for the fiodriver.ko kernel module. When the fiodriver.ko kernel module is loaded the "fio.loopback=1" module parameter should be specified. When this option is enabled fiodriver.ko will use the modified global function names as described above.

The paired loopback device name "/dev/spxs_loopback_b" should be specified in the VSE program configuration file "FIOLoopbackDevice” parameter described in section A5.
4 REQUIREMENTS TRACEABILITY

Table 5 shows the relationship between the requirements from the APIVS SRS and the design presented in this SDD document.

<table>
<thead>
<tr>
<th>Category</th>
<th>SRS Req ID</th>
<th>SRS Requirement Description</th>
<th>SDD Design Narrative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Source Software (OSS) Environment</td>
<td>3.1</td>
<td>No Cost</td>
<td>All APIVS software shall be free of licensing fees and available for free download.</td>
</tr>
<tr>
<td></td>
<td>3.2</td>
<td>Open Source</td>
<td>All APIVS software source code shall be available under open source licensing terms.</td>
</tr>
<tr>
<td></td>
<td>3.3</td>
<td>ITE Approved License</td>
<td>All APIVS software source code shall be available under open source licensing terms approved by ITE.</td>
</tr>
<tr>
<td>Unrestricted Use</td>
<td>3.4</td>
<td>Unrestrictive Use by Users</td>
<td>All APIVS software shall be available under open source licensing terms which allow unrestricted use.</td>
</tr>
<tr>
<td>Redistribution</td>
<td>3.5</td>
<td>Redistribution of Modified Source Code</td>
<td>All APIVS software source code shall be available under open source licensing terms (Gnu Public License) which require modifications and derived works to be distributed under the same terms.</td>
</tr>
<tr>
<td></td>
<td>3.6</td>
<td>No Cost Redistributed APIVS Source Code</td>
<td>All APIVS software source code shall be available under the terms of the Gnu Public License (GPL).</td>
</tr>
<tr>
<td>Testing Environment</td>
<td>3.7</td>
<td>Testing Environment</td>
<td>The APIVS software package is designed to run on an ATC 5201 Standard conforming controller. See Section 2.1.</td>
</tr>
<tr>
<td>C Programming Language</td>
<td>3.8</td>
<td>C Programming Language</td>
<td>All APIVS software programs are written in the C Programming Language and are designed to be compatible with the uClibc C library. See Section 2.3.</td>
</tr>
<tr>
<td>Source Code Quality</td>
<td>3.9</td>
<td>C Source Code Quality</td>
<td>All APIVS source code is written to follow the styling practices of the Linux kernel and GNU standard library source code.</td>
</tr>
<tr>
<td>Extensible</td>
<td>3.10</td>
<td>XML Scripting Language</td>
<td>The APIVS software allows validation tests to be configured using XML files. Details of the APIVSXML language are found in API Validation Suite APIVSXML Specification (see Section 1.5, References).</td>
</tr>
<tr>
<td></td>
<td>3.11</td>
<td>Interpreted Test Scripts</td>
<td>XML TCS files may be established and interpreted by the APIVS software at run-time without requiring recompilation, as described in Section 2.2.</td>
</tr>
<tr>
<td>Selectable Tests</td>
<td>3.12</td>
<td>Run All Tests</td>
<td>The VSE executable may be configured to run through all tests supplied in the distribution by use of the command line switch enabling this option, as described in Section 2.2.</td>
</tr>
<tr>
<td></td>
<td>3.13</td>
<td>Run Selected Tests</td>
<td>The VSE executable may be configured to run one or more individual tests from the full set of tests supplied in the distribution, as described in Section 2.2.</td>
</tr>
<tr>
<td>Category</td>
<td>SRS Req ID</td>
<td>SRS Requirement Description</td>
<td>SDD Design Narrative</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------</td>
<td>----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Continuous Loop</td>
<td>3.14</td>
<td>Continuous Loop</td>
<td>The VSE executable may be configured to run each test load once or continuously, as described in Section 2.2.</td>
</tr>
<tr>
<td>Pass / Fail Indications</td>
<td>3.15</td>
<td>Conformance Indication</td>
<td>The VSE executable returns a value of 0 to indicate a pass or conformance condition for the test load run as stated in Section 2.2.</td>
</tr>
<tr>
<td></td>
<td>3.16</td>
<td>Nonconformance Indications</td>
<td>The VSE executable returns a value of -1 to indicate a failure or non-conformance condition for the test run as stated in Section 2.2.</td>
</tr>
<tr>
<td>Logging Option</td>
<td>3.17</td>
<td>Detailed Log</td>
<td>When configured for verbose output, the APIVS software returns a detailed conformance report. An example of the detailed form of log file when opened with an XML viewer is shown in Appendix B.</td>
</tr>
<tr>
<td></td>
<td>3.18</td>
<td>Summary Result</td>
<td>The format of the conformance report, as described in Section 3.4.4, allows interactive viewing of test results at increasing levels of detail. An example of the summary form of the log file when opened with an XML viewer is shown in Appendix A.</td>
</tr>
<tr>
<td></td>
<td>3.19</td>
<td>Output Options</td>
<td>The format of the conformance report, as described in Section 3.4.4, allows interactive viewing of test results at increasing levels of detail. Examples of different output options are shown in Appendix A and B.</td>
</tr>
<tr>
<td></td>
<td>3.20</td>
<td>XML Output Files</td>
<td>The XML format of the conformance report, as described in Section 3.4.4, allows interactive viewing of test results at increasing levels of detail. An example of this format is shown in Appendix B.</td>
</tr>
<tr>
<td>API FPUI Library C Functions Completeness Testing</td>
<td>3.21.1</td>
<td>FPUI Library C Function Present</td>
<td>A TCS file &quot;FPUI_every_func.xml&quot; is provided in the APIVS software distribution which includes calls to all FPUI library functions and indicates any absent functions in the conformance report.</td>
</tr>
<tr>
<td></td>
<td>3.21.2</td>
<td>FPUI Library C Function Conforming Arguments</td>
<td>A TCS file &quot;FPUI_every_func.xml&quot; is provided in the APIVS software distribution which includes calls to all FPUI library functions and indicates any function argument or parameter type non-conformance in the conformance report. Function parameter checking is performed by the InterpretXML entity as described in Section 3.4.2.</td>
</tr>
<tr>
<td>API FPUI Library C Functions Correctness Testing</td>
<td>3.21.3</td>
<td>FPUI Library C Function Error Checking</td>
<td>The VSE program includes the validation test interpreter entity InterpretXML, described in Section 3.4.2, which validates the return value of each FPUI library function as it is called. The extent of error code checking depends on the TCS making invalid function calls and comparing the resultant returned codes against expected values.</td>
</tr>
<tr>
<td>Category</td>
<td>SRS Req ID</td>
<td>SRS Requirement Description</td>
<td>SDD Design Narrative</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------</td>
<td>------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>API Front Panel Manager Software Testing</strong></td>
<td>3.21.6</td>
<td>Front Panel Manager Window Testing</td>
<td>A TCS file “FPUI_Scenario1.xml” is provided in the APIVS software distribution and described in Appendix C, which calls multiple FPUI library functions in a scenario which validates typical use of the Front Panel Manager Window specified in Section 3.1.1.1 of ATC 5401 Standard, checking each function call for correctness and validating the resultant controller behavior against expected results.</td>
</tr>
<tr>
<td><strong>API Utility Software Testing</strong></td>
<td>3.21.7</td>
<td>ATC Configuration Window Testing</td>
<td>A TCS file “FPUI_Scenario2.xml” is provided in the APIVS software distribution and described in Appendix C, which calls multiple FPUI library functions in a scenario which validates typical use of the ATC Configuration Window specified in Section 3.2.1 of ATC 5401 Standard, checking each function call for correctness and validating the resultant controller behavior against expected results.</td>
</tr>
<tr>
<td><strong>API FIO Library C Functions Completeness Testing</strong></td>
<td>3.22.1</td>
<td>FIO Library C Function Present</td>
<td>A TCS file “FIO_Every_Func.xml” is provided in the APIVS software distribution which includes calls to all FIO library functions and indicates any absent functions in the conformance report.</td>
</tr>
<tr>
<td>Category</td>
<td>SRS Req ID</td>
<td>SRS Requirement Description</td>
<td>SDD Design Narrative</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------</td>
<td>-----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td><strong>API FIO Library C Functions Correctness Testing</strong></td>
<td>3.22.2</td>
<td>FIO Library C Function Conforming Arguments</td>
<td>A TCS file “FIO_every_func.xml” is provided in the APIVS software distribution which includes calls to all FIO library functions and indicates any function argument or parameter type non-conformance in the conformance report. Function parameter checking is performed by the InterpretXML entity as described in Section 3.4.2.</td>
</tr>
<tr>
<td><strong>3.22.3</strong></td>
<td></td>
<td>FIO Library C Function Error Checking</td>
<td>The VSE program includes the validation test interpreter entity InterpretXML, described in Section 3.4.2, which validates the return value of each FIO library function as it is called. The extent of error code checking depends on the TCS making invalid function calls and comparing the resultant returned codes against expected values.</td>
</tr>
<tr>
<td><strong>3.22.4</strong></td>
<td></td>
<td>FIO Library C Function Argument Boundary Checking</td>
<td>The VSE program includes the validation test interpreter entity InterpretXML, described in Section 3.4.2, which validates the argument type and bounds of each FIO library function as it is called. The extent of function parameter range checking depends on the TCS making invalid function calls and comparing the resultant returned codes against expected values.</td>
</tr>
<tr>
<td><strong>3.22.5</strong></td>
<td></td>
<td>FIO Library Composite Testing</td>
<td>A TCS file “FIO_Scenario1.xml” is provided in the APIVS software distribution and described in Appendix C, which calls multiple FIO library functions in a typical use case scenario, checking each function call for correctness and validating the resultant controller behavior against expected results.</td>
</tr>
<tr>
<td><strong>API FIO Manager Software Testing</strong></td>
<td>3.22.6</td>
<td>Field I/O Manager Testing</td>
<td>Several TCS files are provided in the APIVS software distribution and described in Appendix C, which establish typical Field I/O Manager scenarios for cabinet I/O device control and communication, and which validate the resultant controller behavior against expected results.</td>
</tr>
<tr>
<td><strong>API TOD Library C Functions Completeness Testing</strong></td>
<td>3.23.1</td>
<td>TOD Library C Function Present</td>
<td>A TCS file “TOD_Every Func.xml” is provided in the APIVS software distribution which includes calls to all TOD library functions and indicates any absent functions in the conformance report.</td>
</tr>
<tr>
<td>3.23.2</td>
<td></td>
<td>TOD Library C Function Conforming Arguments</td>
<td>A TCS file “TOD_every_func.xml” is provided in the APIVS software distribution which includes calls to all TOD library functions and indicates any function argument or parameter type non-conformance in the conformance report.</td>
</tr>
<tr>
<td>Category</td>
<td>SRS Req ID</td>
<td>SRS Requirement Description</td>
<td>SDD Design Narrative</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------</td>
<td>--------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>API TOD Library C Functions Correctness Testing</td>
<td>3.23.3</td>
<td>TOD Library C Function Error Checking</td>
<td>The VSE program includes the validation test interpreter entity InterpretXML, described in Section 3.4.2, which validates the return value of each TOD library function as it is called. The extent of error code checking depends on the TCS making invalid function calls and comparing the resultant returned codes against expected values.</td>
</tr>
<tr>
<td></td>
<td>3.23.4</td>
<td>TOD Library C Function Argument Boundary Checking</td>
<td>The VSE program includes the validation test interpreter entity InterpretXML, described in Section 3.4.2, which validates the argument type and bounds of each TOD library function as it is called. The extent of function parameter range checking depends on the TCS making invalid function calls and comparing the resultant returned codes against expected values.</td>
</tr>
<tr>
<td></td>
<td>3.23.5</td>
<td>TOD Library Composite Testing</td>
<td>A TCS file “TOD_Scenario1.xml” is provided in the APIVS software distribution and described in Appendix C, which calls multiple TOD library functions in a typical use case scenario, checking each function call for correctness and validating the resultant controller behavior against expected results.</td>
</tr>
<tr>
<td>Multiple and Concurrent Applications</td>
<td>3.24</td>
<td>Multiple and Concurrent Applications</td>
<td>The APIVS software supports the ability for multiple instances of the VSE program to run concurrently, as described in Section 2.2, in order to test multiple client support of the API libraries.</td>
</tr>
</tbody>
</table>
APPENDIX A: SUMMARY CONFORMANCE REPORT

The figure below provides an example of the summary form of the conformance report when opened with a XML viewer.
APPENDIX B: DETAILED CONFORMANCE REPORT

The figure below provides an example of the detailed form of the conformance report when opened with a XML viewer.

The text below is the source XML of the conformance report shown previously.

```xml
<?xml version="1.0" encoding="utf-8"?>

<ApiVsRun date="1970-02-07 03:11 pm PST" input="./FPUI_Example.xml" output="stdout"
    testSuite="ALL_TESTS" level="summary">
  <TestSuite lineNumber="200" date="1970-02-07 03:11 pm PST" name="Sanity Test Suite"
      description="Collection of Sanity Test Cases">
    <TestCase lineNumber="157" date="1970-02-07 03:11 pm PST" name="FPUI_Sanity"
      description="Basic write string FPUI check">
      <Call lineNumber="160" date="1970-02-07 03:11 pm PST" ref="FPUI_Open"
        description="fpui_open: Open FPUI interface">
        <Function lineNumber="49" date="1970-02-07 03:11 pm PST" funcName="fpui_open"
          description="Open the FPUI interface" return="$fpui_handle" errno="$errno"
          p1="O_RDWR" p2="FPUI Sanity" />
        <Print lineNumber="56" date="1970-02-07 03:11 pm PST">
          The returned fpui_handle is [\$fpui_handle:hex]
        </Print>
      </Call>
      <Call lineNumber="163" date="1970-02-07 03:11 pm PST" ref="FPUI_Clear"
        description="fpui_clear: Open FPUI interface">
        <Function lineNumber="75" date="1970-02-07 03:11 pm PST" funcName="fpui_clear"
          description="Clear the display" return="$returnCode" errno="$errno"
          p1="fpui_handle" />
    </TestCase>
  </TestSuite>
</ApiVsRun>
```
APPENDIX C: TEST CASE SCENARIOS

These test case scenarios are to be turned into XML test case specifications and provided as part of the deliverable APIVS software.

C.1 FPUI_Every(Func Validation Test

This file contains an example of a call to every FPUI library function.

C.2 FIO_Every(Func Validation Test

This file contains an example of a call to every FIO library function.

C.3 TOD_Every(Func Validation Test

This file contains an example of a call to every TOD library function.

C.4 FPUIAPI_Scenario1 Validation Test

Perform a test whereby a FPUIAPI client is registered which in turn performs a series of display functions to maintain a client virtual window display. The display appearance and keypad input is validated.

The following outlines the testing sequence of API calls:
1. Open the FPUI using fpui_open().
2. Clear the display to ensure we are starting from a known state using fpui_clear().
3. Compare the content of the display against the expected content, showing the new client application listed in the Front Panel Manager menu.
4. Generate the key-press sequence to select the new client virtual screen.
5. Write a string to the display at a known location using fpui_write_at().
6. Compare the content of the display against the expected content of the display.
7. Generate key-press input from the example input file.
8. Read the key-press input using fpui_read_char() and validate the input against expected values.
9. Close the FPUI using fpui_close().
10. Generate the key-press sequence to select the Front Panel Manager menu.
11. Compare the content of the display against the expected content, showing the new client is no longer listed in the Front Panel Manager menu.

C.5 FPUIAPI_Scenario2 Validation Test

Perform a test which simulates the sequence of key presses required to operate the System Configuration window and edit a configuration item. The operation of the FPUIAPI is validated and the configured item verified independently.

The following outlines the testing sequence of API calls:
1. Generate the key-press sequence to select the System Configuration screen.
2. Generate the key-press sequence to select the System Time screen.
3. Generate the key-press sequence to enter a sample date/time.
4. Verify that the operating system date/time has been changed as expected using the "date" command.

C.6 FIOAPI_Scenario1 Validation Test

Perform a test whereby a FIOAPI client is registered which then registers and enables a FIO332 device. Watchdog service functionality is validated; output and input functionality is validated.
The following outlines the testing sequence of API calls:

1. Register with the FIO library using `fio_register()`.
2. Register to use a FIO332 type FIOD on port FIO_SP5 using `fio_fiod_register()`.
3. Register with the watchdog monitor service, using `fio_fiod_wd_register()`.
4. Set the watchdog point to use, using `fio_fiod_wd_reservation_set()`.
5. Get the watchdog point to use, and ensure it is the same point as that which was set, using `fio_fiod_wd_reservation_get()`.
6. Set up the SDLC response message to be sent when a Set Outputs command message (frame #55) is sent by the FIOAPI.
7. Enable communications to the FIOD using `fio_fiod_enable()`.
8. Heartbeat the watchdog point using `fio_fiod_wd_heartbeat()`.
9. Observe the actual watchdog point, and ensure it has toggled, using `fio_fiod_outputs_get()`.
10. Observe the SDLC message that was sent and ensure the bit has toggled.
11. Heartbeat the watchdog point using `fio_fiod_wd_heartbeat()`.
12. Observe the actual watchdog point, and ensure it has toggled, using `fio_fiod_outputs_get()`.
13. Observe the SDLC message that was sent and ensure the bit has toggled.
14. Register the output point reservations for the output test set using `fio_fiod_outputs_reservation_set()`.
15. Get the output point reservations and ensure the output test set is reserved using `fio_fiod_outputs_reservation_get()`.
16. Set the test set of outputs using `fio_fiod_outputs_set()`.
17. Observe the actual output points set by the FIOAPI using `fio_fiod_outputs_get()`.
18. Observe the SDLC message that was sent and ensure the output test bits are set.
19. Set up the SDLC response message to be sent when a Get Inputs command message (frame #52) is sent by the FIO API.
20. Get the inputs and ensure that the appropriate test set bits are correct, using `fio_fiod_inputs_get()`.
21. Disable communications with the FIOD using `fio_fiod_disable()`.
22. Ensure that the test set of output reservations are cancelled by calling `fio_fiod_outputs_reservation_get()`.
23. Deregister with the watchdog monitor service using `fio_fiod_wd_deregister()`.
24. Deregister the FIOD used using `fio_fiod_deregister()`.
25. Deregister with the FIO library using `fio_deregister()`.

C.7 FIOAPI_Scenario2 Validation Test

Perform a test whereby a FIOAPI client is registered which then registers and enables a FIOCMU, a FIOOUT14SIU1, a FIOOUT6SIU4 and a FIOINSIU1 device. Watchdog service functionality is validated; output and input functionality is validated.

The following outlines the testing sequence of API calls:

1. Register with the FIO library using `fio_register()`.
2. Register to use a FIOCMU type FIOD on port FIO_SP5 using `fio_fiod_register()`.
3. Register to use a FIOOUT14SIU type FIOD on port FIO_SP5 using `fio_fiod_register()`.
4. Register to use a FIOOUT6SIU type FIOD on port FIO_SP5 using `fio_fiod_register()`.
5. Register to use a FIOINSIU1 type FIOD on port FIO_SP5 using `fio_fiod_register()`.
6. Register with the watchdog monitor service, using `fio_fiod_wd_register()`.
7. Set the watchdog point to use, using `fio_fiod_wd_reservation_set()`.
8. Get the watchdog point to use, and ensure it is the same point as that which was set, using `fio_fiod_wd_reservation_get()`.
9. Set up the SDLC response message to be sent when a Set Outputs command message (frame #55) is sent by the FIO API.
10. Enable communications to the FIODs using fio_fiod_enable().
11. Heartbeat the watchdog point using fio_fiod_wd_heartbeat().
12. Observe the actual watchdog point, and ensure it has toggled, using fio_fiod_outputs_get().
13. Observe the SDLC message that was sent and ensure the bit has toggled.
14. Heartbeat the watchdog point using fio_fiod_wd_heartbeat().
15. Observe the actual watchdog point, and ensure it has toggled, using fio_fiod_outputs_get().
16. Observe the SDLC message that was sent and ensure the bit has toggled.
17. Register the output point reservations for the output test set using fio_fiod_outputs_reservation_set().
18. Get the output point reservations and ensure the output test set is reserved using fio_fiod_outputs_reservation_get().
19. Set the test set of outputs using fio_fiod_outputs_set().
20. Observe the actual output points set by the FIOAPI using fio_fiod_outputs_get().
21. Observe the SDLC message that was sent and ensure the output test bits are set.
22. Set up the SDLC response message to be sent when a Get Inputs command message (frame #52) is sent by the FIO API.
23. Get the inputs and ensure that the appropriate test set bits are correct, using fio_fiod_inputs_get().
24. Disable communications with the FIOD using fio_fiod_disable().
25. Ensure that the test set of output reservations are cancelled by calling fio_fiod_outputs_reservation_get().
26. Deregister with the watchdog monitor service using fio_fiod_wd_deregister().
27. Deregister the FIOD used using fio_fiod_deregister().
28. Deregister with the FIO library using fio_deregister().

C.8 FIOAPI_Scenario3 Validation Test

Perform a test whereby a FIOAPI client is registered which in turn registers and enables a FIOMMU, a FIOTF1, a FIOTF2, a FIODR1 and a FIOTS2 device. Fault monitor functionality is validated; output and input functionality is validated.

The following outlines the testing sequence of API calls:
1. Register with the FIO library using fio_register().
2. Register to use a FIOMMU type FIOD on port FIO_SP3 using fio_fiod_register().
3. Register to use a FIOTF1 type FIOD on port FIO_SP3 using fio_fiod_register().
4. Register to use a FIOTF2 type FIOD on port FIO_SP3 using fio_fiod_register().
5. Register to use a FIODR1 type FIOD on port FIO_SP3 using fio_fiod_register().
6. Register to use a FIOTS2 type FIOD on port FIO_SP5 using fio_fiod_register().
7. Set the fault monitor state of the FIOTS2 device to FIO_TS_FM_ON using fio_fiod_ts_fault_monitor_set();
8. Get the fault monitor state from the FIOTS2 device and ensure it is set to FIO_TS_FM_ON, using fio_fiod_ts_fault_monitor_get().
9. Set up the SDLC response messages to be sent when a TF BIU Outputs/Inputs command message (frame #10, frame #11) is sent by the FIO API.
10. Enable communications to the FIODs using fio_fiod_enable().
11. Observe the SDLC message that was sent to the FIOTS2 device and ensure the fault monitor output bit is set.
12. Register the output point reservations for the output test set using fio_fiod_outputs_reservation_set().
13. Get the output point reservations and ensure the output test set is reserved using fio_fiod_outputs_reservation_get().
14. Set the test set of outputs using fio_fiod_outputs_set().
15. Observe the actual output points set by the FIOAPI using fio_fiod_outputs_get().
16. Observe the SDLC message that was sent and ensure the output test bits are set.
17. Set up the SDLC response message to be sent when a DR BIU Outputs/Inputs command message (frame #20) is sent by the FIO API.
18. Get the inputs and ensure that the appropriate test set bits are correct, using fio_fiod_inputs_get().
19. Disable communications with the FIOD using fio_fiod_disable().
20. Ensure that the test set of output reservations are cancelled by calling fio_fiod_outputs_reservation_get().
21. Deregister with the watchdog monitor service using fio_fiod_wd_deregister().
22. Deregister the FIOD used using fio_fiod_deregister().
23. Deregister with the FIO library using fio_deregister().

C.9 TODAPI_Scenario1 Validation Test

Perform a test which uses the TODAPI function calls to: select different clock sources for the operating system tick; change the timezone and daylight saving time settings; request and handle a signal on change of date/time. In each case the TODAPI functions are validated using alternate operating system means.

The following outlines the testing sequence of API calls:

1. Get the current time source using tod_get_timesrc().
2. Verify the value returned against that obtained from the /dev/tod device provided by the platform BSP.
3. Change the current time source using tod_set_timesrc().
4. Verify the change is registered via the /dev/tod device.
5. Get the current timezone and dst settings using tod_get().
6. Verify the values returned against those obtained from the operating system “date” command.
7. Change the timezone and dst settings using tod_set().
8. Verify the change is registered via the operating system “date” command.
9. Register a signal handler and a signal to be sent on any change of date/time using tod_request_onchange_signal().
10. Verify the signal is delivered after a change of date/time using the operating system "date" command.
APPENDIX D: VALIDATION SUITE ENGINE (VSE) EXECUTABLE COMMAND LINE OPTIONS


Where:
- vse – Is the name of the VSE executable
- -L [1-3] – (required) Is the conformance level of the output desired.
  - 1 – Conformance / non-conformance indication only, 2 – Conformance / non-conformance indication and summary results, 3 – Conformance / non-conformance indication, summary result and all logs and traces.
- -c configuration-file – (optional) Is a file that specifies a series of VSE run configurable items (see section Appendix A5 below for configuration file format). If this file is omitted, default values are used.
- -i APIVSXML-file – (optional) Is the path to the input XML test configuration file to use. If –i is not present the input will be read from stdin.
- -o output-file – (optional) Is the path of where to place the generated output XML file. If –o is not present, the output will be placed on stdout.
- -n test_suite_name – (optional) Is a specific test suite named in the input XML test configuration file that is to be run. If omitted, all test suites contained in the file will be run.
- -R count – Repeat test load count times, or indefinitely if count is 0.
- -H – Halt on error when running in Repeat mode.
- -C – (optional) If present, this command line argument indicates that this run of the VSE is to “capture” the Virtual Displays (VD) and the SDLC Command Messages (CMDMSG) to be used by future VSE Validation runs. This information, VD and CMDMSG, are captured into flat flats as specified by the configuration-file.

VSE Configurable Items Used in the VSE Configuration-File

<table>
<thead>
<tr>
<th>VSE Configurable Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLInputFilePath</td>
<td>Specifies the path to the APIVSXML input file. Default: ./</td>
</tr>
<tr>
<td>XMLOutputFilePath</td>
<td>Specifies the path of where to place the output XML file. Default: ./</td>
</tr>
<tr>
<td>SetFilePath</td>
<td>Specifies the path to the &lt;Set file=&quot;file&quot; /&gt; file path. Default: ./</td>
</tr>
<tr>
<td>ScreenHeight</td>
<td>Specifies the screen height. The configured value is available to the APIVSXML as the named constant “#ROWS”. Default: 8</td>
</tr>
<tr>
<td>ScreenWidth</td>
<td>Specifies the screen width. The configured value is available to the APIVSXML as the named constant “#COLUMNS”. Default: 40</td>
</tr>
<tr>
<td>FPUICompareFilePath</td>
<td>Specifies the path to the !VDCompare(file) file path. Default: ./</td>
</tr>
<tr>
<td>FPUIInputFilePath</td>
<td>Specifies the path to the &lt;FPUIInput file=&quot;file&quot; /&gt; file path. Default: ./</td>
</tr>
<tr>
<td>FPUIDumpFilePath</td>
<td>Specifies the path to the &lt;Dump dump=&quot;VD&quot; file=&quot;file&quot; /&gt; and &lt;Load load=&quot;VD&quot; file=&quot;file&quot; /&gt; file path. Default: ./</td>
</tr>
<tr>
<td>FPUILoopbackDevice</td>
<td>Specifies the path to the VSE FPUI port to be used as the loopback port. Default: NULL – This indicates perform no VT100 loopback emulation.</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FIOCompareFilePath</td>
<td>Specifies the path to the <code>!CMDMSG(file,frame)</code> file path. Default: <code>/</code></td>
</tr>
<tr>
<td>FIOResponseFilePath</td>
<td>Specifies the path to the <code>&lt;FIOResponse file=&quot;file&quot;/&gt;</code> file path. Default: <code>/</code></td>
</tr>
<tr>
<td>FIODumpFilePath</td>
<td>Specifies the path to the <code>&lt;Dump dump=&quot;CMDMSG&quot; file=&quot;file&quot;/&gt;</code> and <code>&lt;Load load=&quot;CMDMSG&quot; file=&quot;file&quot;/&gt;</code> file path. Default: <code>/</code></td>
</tr>
<tr>
<td>FIOLoopbackDevice</td>
<td>Specifies the path to the VSE FIO port to be used as the loopback port. Default: NULL – This indicates perform no FIO emulation.</td>
</tr>
</tbody>
</table>
APPENDIX E: SYNTAX FOR FLAT FILES USED BY THE VSE

The VSE supports human-readable, and modifiable, input and output files. For input, VSE variable set files, SDLC Command Message Load files, VD Load files, SDLC Response Message files, and FP Input files are supported. For output, VSE variable dumps, SDLC Command Message dumps, and VD dumps are supported.

For human-readable FP Input files, the following is supported:
- All lines starting with '#' are ignored as comments;
- The sequence "<ESC>" is translated as a single 0x1b (escape) character;
- The sequence "<NL>" is translated as a single 0x0a (new-line) character;
- The sequence "<CR>" is translated as a single 0x0d (carriage-return) character;
- The sequence "<TAB>" is translated as a single 0x09 (tab) character;
- The sequence "\" is translated as a single 0x5c (backslash) character;
- The sequence "<" is translated as a single 0x3c (<) character; and
- The sequence "0xNN" is translated as a sign 0xNN character, such as "0xFF" for a (DEL) character.

All other human-readable flat files allow for the use of '#' as a comment line. All other human-readable flat files support the input of binary hexadecimal data as a sequence such as:
- 0x2a 0x55 0x8f – A space must be present between values.

Upon reading an input to the VSE, this information is translated into the appropriate binary format internally. If the value converted to binary form is shorter than the destination variable, the destination variable is padded with 0x00. If the destination variable is shorter than the converted binary form, the converted binary form is truncated to fit the destination variable size.