A Project Document of the
ATC Application Programming Interface Working Group

DRAFT APIVS SW UM v01.01

Advanced Transportation Controller (ATC)
Application Programming Interface
Validation Suite (APIVS)
Software User Manual

November 11, 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
All Users of the APIVS Software

Prepared by:  Michael Gallagher, Intelight, Inc.
Douglas Crawford, Intelight, Inc.
Grant Gardner, Intelight, Inc.
Ralph W. Boaz, Pillar Consulting, Inc.
James A. Kinnard, Adaptive Solutions, Inc.

© Copyright 2015 AASHTO/ITE/NEMA. All rights reserved.
<table>
<thead>
<tr>
<th>DATE</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/27/15</td>
<td>Initial draft APIVS User Manual v01.00</td>
</tr>
<tr>
<td>11/11/15</td>
<td>Second draft APIVS User Manual v01.01</td>
</tr>
</tbody>
</table>
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 .............................................................................................................................. 5  
   1.1 Purpose .................................................................................................................................. 5  
   1.2 Scope ................................................................................................................................... 5  
   1.3 Document Organization ........................................................................................................ 5  

2  CONCEPT OF OPERATIONS ........................................................................................................... 5  

3  GETTING THE APIVS SOFTWARE SOURCE CODE ................................................................... 7  
   3.1 APIVS Software Source Code Organization ........................................................................ 7  

4  BUILDING THE APIVS SOFTWARE .............................................................................................. 8  
   5.1 Getting Started .................................................................................................................. 10  
   5.2 Command Line Interface (CLI) .......................................................................................... 11  
   5.3 Configuration File .............................................................................................................. 12  
   5.4 Using Virtual Loopback ..................................................................................................... 14  
   5.5 Using the APIVS Test Plan ............................................................................................... 15  

5  RUNNING THE APIVS SOFTWARE ............................................................................................. 10  

6  DEFINITIONS AND ACRONYMS ................................................................................................. 16  

7  REFERENCES ............................................................................................................................... 19  

A. APPENDICES ................................................................................................................................ 20  
   A.1 Sample APIVS Test Script ................................................................................................ 20  
   A.2 Test Conformance Report (XML Viewer) ........................................................................ 24  
   A.3 Test Conformance Report (XML File) ............................................................................. 26  
   A.4 Flat File Format ................................................................................................................. 28  
   A.5 APIVSXML Specification ................................................................................................... 29  

   1. Introduction .................................................................................................................. 33  
   1.1 Purpose ................................................................................................................... 33  
   1.2 Intended Audience .................................................................................................. 33  
   1.3 Scope ...................................................................................................................... 33  
   1.4 Conventions ............................................................................................................ 33  
   1.5 Definitions, Acronyms, and Abbreviations ............................................................... 34  
   1.6 References .............................................................................................................. 35  

   2. APIVSXML Schema Definition .................................................................................... 36  
   2.1 APIVSXML Syntax and Naming Convention Guidelines ........................................ 36  
   2.2 Validation Suite Architecture ................................................................................... 38  
   2.3 APIVSXML Schema ................................................................................................ 40  

   3. APIVSXML Quick Reference Guide ............................................................................. 68
1 INTRODUCTION

1.1 Purpose

This manual, the Advanced Transportation Controller (ATC) Application Programming Interface Validation Suite (APIVS) Software User Manual, describes how to acquire, build and deploy the API Validation Suite software (APIVS software). It has been developed for both the software developers that create programs for ATC units and end users that use ATC units to perform their operational tasks.

1.2 Scope

The APIVS software is an open-source software (OSS) implementation of a test suite designed to test the Reference Implementation of the ATC 5401 Application Programming Interface Standard (APIRI). The APIRI and APIVS software implementations are designed to operate on transportation controllers that conform to the ATC 5201 Advanced Transportation Controller Standard (ATC units). The APIVS software allows APIRI users to test the full suite of API interfaces on a given hardware platform in an automated way.

1.3 Document Organization

This document has been organized based on IEEE Std 1063-2001, IEEE Standard for Software User Documentation. The section titles have been updated for applicability to the APIVS project.

2 CONCEPT OF OPERATIONS

The APIVS software allows testing of the installed API software against the required functionality as specified in the ATC 5401 Standard. The testing and validation instructions are supplied in XML-format test script files, which are parsed and interpreted by the Validation Suite Engine (VSE) which forms the main run-time program of the APIVS software package.

The VSE is a Linux program and utilizes the functions defined in the ATC API Standard and implemented in the APIRI libraries and supported by the Front Panel Manager (FPM) and Field I/O Manager (FIOM).

The APIVS software package also includes a set of asynchronous and synchronous virtual loopback drivers for the Linux kernel, which are used to test serial port and front panel operation without the need for physical loopback devices. The package also includes several sample test configuration files which cover test scenarios for typical ATC API use cases, including FIO, FPUI and TOD library use.

Figures 1 shows the decomposition of the VSE software components in a functional diagram.
Figure 1. VSE Software Architecture
3 GETTING THE APIVS SOFTWARE SOURCE CODE

The APIVS Software source code is hosted on GitHub at the following URL:
- [https://github.com/apiriadmin/APIVS](https://github.com/apiriadmin/APIVS)

The entire source tree can be downloaded at the following URL as a zip file:
- [https://github.com/apiriadmin/APIVS/archive/master.zip](https://github.com/apiriadmin/APIVS/archive/master.zip)

The GitHub APIVS repository can also be forked or cloned using the GitHub client.

For more information see the following link for GitHub documentation:
- [https://help.github.com/articles/set-up-git/](https://help.github.com/articles/set-up-git/)

The APIVS source code is released under the GNU LESSER GENERAL PUBLIC LICENSE (LGPL) available here:
- [https://github.com/apiriadmin/APIVS/blob/master/LICENSE](https://github.com/apiriadmin/APIVS/blob/master/LICENSE)

3.1 APIVS Software Source Code Organization

The APIVS software source code is organized into folders as follows:

- `<root>`
  - LICENSE [APIVS license file]
  - Makefile [top-level makefile]
  - Readme.md [readme file]
  - Package.sh [Script to copy runtime VSE package to /usb_root]

- include/
  - Contains the C header files

- modules/
  - virtual-loopback-sync
    - Contains the C source code for the virtual synchronous loopback driver
  - virtual-loopback-async
    - tty0tty-1.2
      - Contains the C source code for the virtual asynchronous loopback driver
      - Virtual async loopback is supported through the tty0tty linux null modem emulator v1.2 utility

- samples/
  - Contains a sample test xml file and sample test xml output

- src/
  - Contains the C source code files

- support/
  - Contains the necessary support scripts from running the VSE from a USB

- tests/
  - Contains the set of predefined XML test definition files
4 BUILDING THE APIVS SOFTWARE

The APIVS software is built from a single top level makefile. The makefile accepts the standard “all” and “clean” targets. The development environment is required to have the gcc compiler for the target platform installed (version 4.x or higher).

Before running make, the environment must be setup to use the correct cross compiler for the target ATC platform.

An example environment setup script is included in /export-env.sh which can be enabled using the source command:

• source export-env.sh

Note that the variables in the export-env.sh script need to be modified to match the target ATC platform before running the source export-env.sh command.

The sample export-env.sh is configured for the Intelight controller as follows:

#!/bin/sh
export ARCH=powerpc
export PATH=/opt/intelight/toolchain/ctng-linux-eb8248-201305/bin/:$PATH
export LINUX_DIR=/ppc/eb8248/buildroot/output/build/linux-custom
export BSPDIR=/opt/intelight/toolchain/sdk-eb8248-1.02
export CC=powerpc-unknown-linux-uclibc-gcc
export AR=powerpc-unknown-linux-uclibc-ar

Where,
• ARCH = target ATC cpu architecture (powerpc)
• PATH = path to cross compile toolchain binaries
• CROSS_COMPILE = cross compile toolchain name
• LINUX_DIR = path to kernel build directory for fio and fpu kernel modules
• BSPDIR = path to vendor board support package
• CC = gcc compile
• AR = linux build ar

To build the APIVS software the following commands can be run from the root path:
• source export-env
• make

The APIVS software, including the asynchronous and synchronous loopback drivers, must be built for each target Engine Board and ATC Controller and may be included as part of the ATC vendor’s Linux distribution.
The APIRI and APIVS Test Plans require that the newly-built APIVS software be installed onto a USB flash drive. To copy the runtime VSE package to a USB flash drive, insert a blank drive into the primary USB port on the ATC and run the following script:

- source package.sh

Note that this script will first reformat the drive and will then create a new usb_root/ folder on the drive with the following structure and files set:

- `<root>`
  - `startup` [ATC auto-startup shell script]
  - `runAPIVS` [shell script which runs selected VSE test cases]
- `APIVS/`
  - `bin/`
    - `vse`
    - `virtual-loopback-sync.ko`
    - `fiodriver.ko`
    - `tty0tty.ko`
  - `lib/`
    - `libemfio.so.1`
    - `llibexpat.so.1`
    - `libvt100.so.1`
- `run/`
  - all VSE test, configuration and flat files
5  RUNNING THE APIVS SOFTWARE

5.1  Getting Started

The Validation Suite Engine (VSE) software is controlled through a command line interface (CLI). After the VSE software is built the VSE executable and configuration files can be packaged and copied to a USB to run on the DUT during boot.

In addition, 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.

The Validation Suite directory contains the Linux valsuite engine executable, its default configuration file, and the XML format validation test configuration files. The valsuite executable reads the XML format validation test configuration file(s) and writes an XML format result file.

The VSE accepts several command line options to provide the following functionality:

- Override the default name of the configuration file
- Specify the name of the selected XML test scenario files to process
- Process all of the XML test scenario files
- Override the default name of XML result file

Further details of the VSE command line options and its default configuration file format are contained in subsequent sections.
5.2 Command Line Interface (CLI)

The following command line parameters are supported by the VSE software. Bold parameters are required to run the VSE software.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Usage</th>
</tr>
</thead>
</table>
| L [1-3]       | Required. The conformance level of the output desired, where:  
   • 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 filename   | Optional. The path to a text file that specifies a series of VSE run configurable items (see Configuration File section below for configuration file format). If this file is omitted, default values are used. |
| -i input-file | Optional. The path to a text file for the input XML test configuration file to use. If -i is not present the input will be read from stdin. |
| -o output-file| Optional. 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. The 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. |
| -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. |

The following is an example of a full command line to run the VSE software.

5.3 Configuration File

The following options can be controlled through the configuration file which is passed as a command line argument to the VSE software. If no configuration is specified the default parameters will be used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Usage</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMLInputFilePath</td>
<td>Specifies the path to the APIVSXML input file.</td>
<td>../</td>
</tr>
<tr>
<td>XMLOutputFilePath</td>
<td>Specifies the path of where to place the output XML file.</td>
<td>../</td>
</tr>
<tr>
<td>SetFilePath</td>
<td>Specifies the path to the &lt;Set file=&quot;file&quot; /&gt; file path</td>
<td>../</td>
</tr>
<tr>
<td>ScreenHeight</td>
<td>Specifies the screen height. The configured value is available to the APIVSXML as the named constant &quot;#ROWS&quot;.</td>
<td>8</td>
</tr>
<tr>
<td>ScreenWidth</td>
<td>Specifies the screen width. The configured value is available to the APIVSXML as the named constant &quot;#COLUMNS&quot;.</td>
<td>40</td>
</tr>
<tr>
<td>FPUICompareFilePath</td>
<td>Specifies the path to the !VDCompare(file) file path.</td>
<td>../</td>
</tr>
<tr>
<td>FPUIInputFilePath</td>
<td>Specifies the path to the &lt;FPUIInput file=&quot;file&quot; /&gt; file path.</td>
<td>../</td>
</tr>
<tr>
<td>FPUIDumpFilePath</td>
<td>Specifies the path to the &lt;Dump dump=&quot;VD&quot; file=&quot;file&quot; /&gt; file path.</td>
<td>../</td>
</tr>
<tr>
<td>FPUIDumpFilePath</td>
<td>Specifies the path to the &lt;Dump dump=&quot;VD&quot; file=&quot;file&quot; /&gt; file path.</td>
<td>../</td>
</tr>
<tr>
<td>FPULoopbackDevice</td>
<td>Specifies the path to the VSE FPUI port to be used as the loopback port.</td>
<td>NULL (No Emulation)</td>
</tr>
<tr>
<td>FIOCompareFilePath</td>
<td>Specifies the path to the !CMDMSG(file,frame) file path.</td>
<td>../</td>
</tr>
<tr>
<td>FIOResponseFilePath</td>
<td>Specifies the path to the &lt;FIOResponse file=&quot;file&quot; /&gt; file path.</td>
<td>../</td>
</tr>
<tr>
<td>FIODumpFilePath</td>
<td>Specifies the path to the &lt;Dump dump=&quot;CMDMSG&quot; file=&quot;file&quot; /&gt; and &lt;Load load=&quot;CMDMSG&quot; file=&quot;file&quot; /&gt; file path.</td>
<td>../</td>
</tr>
<tr>
<td>FIOLoopbackDevice</td>
<td>Specifies the path to the VSE FIO port to be used as the loopback port.</td>
<td>NULL (No Emulation)</td>
</tr>
</tbody>
</table>
The following is a sample test configuration file:

```plaintext
# ATC 5401 API Reference Implementation Project
#
# Filename: VS_config_1.txt
# File Type: VSE configuration file
# Test Case: many
#
# Date      Revision    Description
# 10/21/15   1.0         initial release

XMLInputFilePath           = ./
XMLOutputFilePath           = ./
SetFilePath                 = ./

ScreenWidth                = 40
ScreenHeight               = 8

FPUICompareFilePath         = ./
FPUIInputFilePath           = ./
FPUIDumpFilePath            = ./
FPUILoopbackDevice          = /dev/sp6_loopback_b

FIOCmpareFilePath           = ./
FIOResponseFilePath         = ./
FIODumpFilePath             = ./
FIOLoopbackDevice           = /dev/sp5s_loopback_b
```
5.4 Using Virtual Loopback

The APIVS software uses virtual software based loopback drivers to complete various IO and front panel tests.

The lower layer asynchronous and synchronous serial drivers send and receive data to a physical device such as the ATC Front Panel or SDLC cabinet devices. In order to isolate validation testing of just the ATC API the valsuite engine depends on using emulators to emulate the behavior of these physical devices. The FIO and FPUI emulators are part of the valsuite engine executable. The emulators communicate with the ATC API using virtual loopback serial drivers.

The reference implementation includes loopback drivers for SP6 (asynchronous), and SP3, SP5, and SP8 (synchronous). The loopback driver will expose a pair of Linux char devices under /dev with an A side and B side. The emulator will connect to the B device (i.e. /dev/sp6_loopback_b). The ATC API drivers will connect to the A side (i.e. /dev/sp6_loopback_a).

To use the included virtual serial loopback drivers they must be specified in the APIVS configuration file FIOLoopbackDevice parameter as described later in the Configuration File section of this manual.

The serial driver selection for the FPUI ATC API is handled by specifying a command line argument when starting the FrontPanelManager process. Typically, the FrontPanelManager process is started with a single command line argument specifying the serial port connected to the physical front panel ("FrontPanelManager /dev/sp6"). For valsuite testing the FrontPanelManager process should be started with the virtual sp6 loopback driver ("FrontPanelManager /dev/sp6_loopback_a").

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 open /dev/spx_loopback_a instead of the physical serial driver /dev/spx.
5.5 Using the APIVS Test Plan

The Test Case Specifications (TCS) for the Advanced Transportation Controller (ATC) Application Programming Interface Validation Suite (APIVS), provides the specific test cases necessary to test the required features of the API Validation Suite Engine (VSE) application and its associated files.

All of the test cases included in the APIVS TCS document utilize a single hardware and software platform which is common to all tests. The compliance output files produced by all test cases also have a consistent format which allows pass/fail status to be easily ascertained.

Each test case execution produces an output compliance report (file) in XML format. This file contains an element (RunResult) toward the end of the file which indicates the overall completion status (PASS/FAIL) of the test.

```xml
<RunResult date="2015-10-01 12:33 PM EST" status="PASS" />
-OR-
<RunResult date="2015-10-01 12:33 PM EST" status="FAIL" />
```

If the test fails and the test execution was performed with detailed logging enabled, the file can be examined in more detail to determine the exact cause of the failure.

As this output XML file is textual, it can be viewed with a simple text editor such as Notepad. For a more structured view of the XML content in the file, there are many XML file viewing applications available, such as XML Notepad 2007, which can be downloaded via the Internet and installed on the test PC. There are also web sites available, such as http://www.xmlgrid.net, which provide customizable XML graphical views.

A naming convention has been established for all files associated with individual APIVS test cases:

```
C1nnn_xxxyyy.zzz
```

where,

- **C1** indicates a file associated with an APIVS test case; APIRI test cases will begin with C2
- **nnn** is the test case number (the last three digits from the Test Case Specification Identifier)
- **xxx** is an identifier for the specific file content:
  - **in** APIVSXML test script (XML format)
  - **log** conformance report (XML format)
  - **vd** Virtual Display compare file (VSE flat file format)
  - **fpi** Front Panel input file (VSE flat file format)
  - **cmd** SDLC command message file (VSE flat file format)
  - **rsp** SDLC response message file (VSE flat file format)
- **yyy** is an (optional) numeric identifier, generally for VSE flat files only
- **zzz** is the standard file type (txt or xml)

VSE configuration files, which can be shared between test cases, following the naming convention

```
VS_config_nnn.txt
```

where nnn is a number from 1-999 indicating the specific configuration file to be used.

The individual test case files can be found in the APIVS GitHub repository under the tests/ folder.

For the specifics cases tested by each test case file please review to the full APIVS TCS document.
## DEFINITIONS AND ACRONYMS

<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</td>
<td>API Reference Implementation (software)</td>
</tr>
<tr>
<td>APIRI Project</td>
<td>Entire project managed by this PMP including software, hardware and documentation.</td>
</tr>
<tr>
<td>APIVS</td>
<td>API Validation Suite (software and fixture)</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>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>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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 document is defined in the ATC 5201 Standard, Appendix A and Appendix B.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not Applicable</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 Document or 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>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</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>S/W</td>
<td>Software</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Determined</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>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>
7 REFERENCES

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

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 v01.01 Project Management Plan (PMP) for the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation Project*. ATC Joint Committee, 3 January 2014.
http://www.ite.org/standards/index.asp

Institute of Transportation Engineers, *ATC APIRI SEMP v01.01 Systems Engineering Management Plan (SEMP) for the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation Project*. ATC Joint Committee, 3 January 2014.
http://www.ite.org/standards/index.asp
A. APPENDICES

A.1 Sample APIVS Test Script

```xml
<xml version="1.0" encoding="utf-8"/>
<APIVSXML>
<!--
ATC 5401 API Reference Implementation Project
Filename: C2001d_in.xml
File Type: APIVSXML test script (XML format)
Test Case: APIRI.TCS.2001d
Description: Demonstrate FPUI API function testing
-->

<Define var="$fpui_handle" type="fpui_handle"/>
<Define var="$errno" type="int"/>
<Define var="$returnCode" type="int"/>
<Define var="$flags" type="int"/>
<Define var="$regname" type="char []" size="%16"/>
<Define var="$write_buf" type="char []" size="%64"/>
<Define var="$write_cnt" type="ssize_t"/>
<Define var="$row" type="int"/>
<Define var="$column" type="int"/>
<Define var="$abort_mode" type="boolean"/>
<Define var="$failed" type="boolean"/>

<!--
subroutines for setup, teardown, error logging
-->
<SetUp name="FPUI_Init_Variables">
  <Set var="$returnCode" value="%-1"/>
  <Set var="$errno" value="%0"/>
</SetUp>

<Subroutine name="Error_Handler">
  <If expression="$errno EQ %1">
    <Then>
      <Print>EPERM: Operation not permitted</Print>
    </Then>
  </If>
  <If expression="$errno EQ %2">
    <Then>
      <Print>ENOENT: No such file or directory</Print>
    </Then>
  </If>
  <If expression="$errno EQ %3">
    <Then>
      <Print>ESRCH: No such process</Print>
    </Then>
  </If>
  <If expression="$errno EQ %4">
    <Then>
      <Print>EFAULT: Argument list too long</Print>
    </Then>
  </If>
  <If expression="$errno EQ %5">
    <Then>
      <Print>EACCES: Bad file number</Print>
    </Then>
  </If>
  <If expression="$errno EQ %6">
    <Then>
      <Print>EBADF: Bad file number</Print>
    </Then>
  </If>
  <If expression="$errno EQ %7">
    <Then>
      <Print>EINVAL: Invalid argument</Print>
    </Then>
  </If>
  <If expression="$errno EQ %8">
    <Then>
      <Print>EOPNOTSUPP: Operation not supported</Print>
    </Then>
  </If>
  <If expression="$errno EQ %9">
    <Then>
      <Print>ENOTBLK: Block special file needed</Print>
    </Then>
  </If>
  <If expression="$errno EQ %10">
    <Then>
      <Print>ENOTTY: Inappropriate ioctl for device</Print>
    </Then>
  </If>
  <If expression="$errno EQ %11">
    <Then>
      <Print>ENXIO: No such device or file</Print>
    </Then>
  </If>
</Subroutine>
```

APIVS UserManual_0101_151111a.docx
<Then><Print>EAGAIN: Try again</Print></Then></If>
<If expression="$errno EQ %12">
<Then><Print>ENOMEM: Out of memory</Print></Then></If>
<If expression="$errno EQ %13">
<Then><Print>EACCES: Permission denied</Print></Then></If>
<If expression="$errno EQ %14">
<Then><Print>EFAULT: Bad address</Print></Then></If>
<If expression="$errno EQ %15">
<Then><Print>ENOTBLK: Block device required</Print></Then></If>
<If expression="$errno EQ %16">
<Then><Print>EBUSY: Device or resource busy</Print></Then></If>
<If expression="$errno EQ %17">
<Then><Print>ENOMEM: Out of memory</Print></Then></If>
<If expression="$errno EQ %18">
<Then><Print>ENOTDIR: Not a directory</Print></Then></If>
<If expression="$errno EQ %19">
<Then><Print>ENOTBLK: Block device required</Print></Then></If>
<If expression="$errno EQ %20">
<Then><Print>EBUSY: Device or resource busy</Print></Then></If>
<If expression="$errno EQ %21">
<Then><Print>ENOMEM: Out of memory</Print></Then></If>
<If expression="$errno EQ %22">
<Then><Print>ENOTDIR: Not a directory</Print></Then></If>
<If expression="$errno EQ %23">
<Then><Print>EINVAL: Invalid argument</Print></Then></If>
<If expression="$errno EQ %24">
<Then><Print>ENFILE: File table overflow</Print></Then></If>
<If expression="$errno EQ %25">
<Then><Print>EMFILE: Too many open files</Print></Then></If>
<If expression="$errno EQ %26">
<Then><Print>ENOTTY: Not a typewriter</Print></Then></If>
<If expression="$errno EQ %27">
<Then><Print>ETXTBSY: Text file busy</Print></Then></If>
<If expression="$errno EQ %28">
<Then><Print>EFBIG: File too large</Print></Then></If>
<If expression="$errno EQ %29">
<Then><Print>ENOSPC: No space left on device</Print></Then></If>
<If expression="$errno EQ %30">
<Then><Print>ESPIPE: Illegal seek</Print></Then></If>
<If expression="$errno EQ %31">
<Then><Print>EROFS: Read-only file system</Print></Then></If>
<If expression="$errno EQ %32">
<Then><Print>EMFILE: Too many files opened</Print></Then></If>
<If expression="$errno EQ %33">
<Then><Print>EIO: Input/output error</Print></Then></If>
<If expression="$errno EQ %34">
<Then><Print>ERANGE: Math argument out of range</Print></Then></If>

<Format var="$errno"/>
<Set var="$failed" value="#TRUE"/>
<If expression="$abort_mode EQ #TRUE">
<Then><Abort status="FAIL"/></Then></If>
</Subroutine>

<!--
subroutines to call API functions and verify return status
-->
<Subroutine name="fpui_close" description="Call fpui_close()">
    <Function funcName="fpui_close"
        return="$returnCode"
        errno="$errno"
        p1="$fpui_handle" />
    <If expression="$returnCode EQ %-1">
        <Then>
            <Print>Function fpui_close() failed</Print>
            <Call ref="Error_Handler" />
        </Then>
    </If>
</Subroutine>

<Subroutine name="fpui_open" description="Call fpui_open()">
    <Function funcName="fpui_open"
        return="$fpui_handle"
        errno="$errno"
        p1="$flags"
        p2="$regname" />
    <If expression="$fpui_handle EQ %-1">
        <Then>
            <Print>Function fpui_open() failed</Print>
            <Call ref="Error_Handler" />
        </Then>
    </If>
</Subroutine>

<Subroutine name="fpui_write_string_at" description="Call fpui_write_string_at()">
    <Function funcName="fpui_write_string_at"
        return="$write_cnt"
        errno="$errno"
        p1="$fpui_handle"
        p2="$write_buf"
        p3="$row"
        p4="$column" />
    <If expression="$write_cnt EQ %-1">
        <Then>
            <Print>Function fpui_write_string_at() failed</Print>
            <Call ref="Error_Handler" />
        </Then>
    </If>
</Subroutine>
<!--
  test case(s)
 -->
<TestCase name="Case_C2001d" description="C2001d: Demonstrate FPUI API function testing">
  <Set var="$abort_mode" value="#TRUE"/>
  <Set var="$failed" value="#FALSE"/>

  <!-- open a connection to the Front Panel API -->
  <Set var="$flags" value="#O_RDWR"/>
  <Set var="$flags" value="#O_NONBLOCK" operation="add"/>
  <Set var="$regname" value="@C2001d"/>
  <Call ref="fpui_open" setUp="FPUI_Init_Variables"/>

  <!-- select our test application for display by FPM -->
  <FPUIInput file="@Cxxxx_key0.txt" description="Provide keypress [0] input"/>

  <!-- clear our screen and confirm that it is clear -->
  <Call ref="fpui_clear" setUp="FPUI_Init_Variables"/>
  <Dump dump="VD" file="@C2001d_vd_clr" />
  <If expression="!VDCompare(@C2001d_vd_clr) EQ #FALSE">
    <Then>
      <Print>The display is NOT clear</Print>
      <Dump dump="VD" force="#TRUE" file="@C2001d_vd_clrfail"/>
      <Abort status="FAIL"/>
    </Then>
  </If>

  <!-- write a text string to the display at Row 3, Column 5 -->
  <Set var="$row" value="%3"/>
  <Set var="$column" value="%5"/>
  <Set var="$write_buf" value="@Demonstration of FPUI API"/>
  <Call ref="fpui_write_string_at" setUp="FPUI_Init_Variables"/>

  <!-- grab our screen and confirm the write -->
  <Dump dump="VD" file="@C2001d_vd_00" />
  <If expression="!VDCompare(@C2001d_vd_00) EQ #FALSE">
    <Then>
      <Print>The display is NOT correct</Print>
      <Dump dump="VD" force="#TRUE" file="@C2001d_vd_00_fail"/>
      <Abort status="FAIL"/>
    </Then>
  </If>

  <!-- close our connection to the Front Panel API -->
  <Call ref="fpui_close" setUp="FPUI_Init_Variables"/>
</TestCase>

<!--
  test suite(s)
 -->
<TestSuite name="All_Test_Cases" description="Run All Test Cases">
  <TestCaseRef ref="Case_C2001d"/>
</TestSuite>
</APIVSXML>
A.2 Test Conformance Report (XML Viewer)
A.3 Test Conformance Report (XML File)

```xml
<?xml version="1.0" encoding="utf-8" ?>
<ApilVsRun date="2015-10-27 10:19 AM PDT" configuration="/VS_config_1.txt" input="/C2001d_in.xml" output="/C2001d_log.xml" testSuite="ALL_TESTS" level="trace" >
  <Define lineNumber="15" date="2015-10-27 10:19 AM PDT" var="$fpui_handle" type="fpui_handle" />
  <Define lineNumber="16" date="2015-10-27 10:19 AM PDT" var="$errno" type="int" />
  <Define lineNumber="17" date="2015-10-27 10:19 AM PDT" var="$ReturnCode" type="int" />
  <Define lineNumber="18" date="2015-10-27 10:19 AM PDT" var="$flags" type="int" />
  <Define lineNumber="19" date="2015-10-27 10:19 AM PDT" var="$regname" type="char []" size="%16" />
  <Define lineNumber="20" date="2015-10-27 10:19 AM PDT" var="$write_buf" type="char []" size="%64" />
  <Define lineNumber="21" date="2015-10-27 10:19 AM PDT" var="$write_cnt" type="size_t" />
  <Define lineNumber="22" date="2015-10-27 10:19 AM PDT" var="$row" type="int" />
  <Define lineNumber="23" date="2015-10-27 10:19 AM PDT" var="$column" type="int" />
  <Define lineNumber="24" date="2015-10-27 10:19 AM PDT" var="$abort_mode" type="boolean" />
  <Define lineNumber="25" date="2015-10-27 10:19 AM PDT" var="$failed" type="boolean" />
  <TestSuite lineNumber="224" date="2015-10-27 10:19 AM PDT" name="Run All Test Cases" description="All_Test_Cases" operation="" />
    <TestCase lineNumber="177" date="2015-10-27 10:19 AM PDT" var="$abort_mode" type="char []" size="%16" />
    <Set lineNumber="178" date="2015-10-27 10:19 AM PDT" var="$errno" operation="equal" value="%TRUE" />
    <Set lineNumber="179" date="2015-10-27 10:19 AM PDT" var="$returnCode" operation="equal" value="%TRUE" />
    <Set lineNumber="180" date="2015-10-27 10:19 AM PDT" var="$flags" operation="add" value="%O_NONBLOCK" />
    <Call lineNumber="181" date="2015-10-27 10:19 AM PDT" ref="fpui_open" >
      <Set lineNumber="30" date="2015-10-27 10:19 AM PDT" name="FPUI_Init_Variables" >
        <Set lineNumber="31" date="2015-10-27 10:19 AM PDT" var="$returnCode" operation="equal" value="%TRUE" />
        <Set lineNumber="32" date="2015-10-27 10:19 AM PDT" var="$errno" operation="equal" value="%TRUE" />
      </Set>
    </Call>
    <Subroutine lineNumber="140" date="2015-10-27 10:19 AM PDT" name="fpui_open" description="Call fpui_open()" >
      <Function lineNumber="141" date="2015-10-27 10:19 AM PDT" funcName="fpui_open" returnType="fpui_handle" errno="errno" p1="flags" />
      <If lineNumber="146" date="2015-10-27 10:19 AM PDT" expression="fpui_handle EQ %1-1" evaluatesTo="FALSE/ELSE" >
        <If />
      </If>
    </Subroutine>
  </TestSuite>
  <TestSuite lineNumber="224" date="2015-10-27 10:19 AM PDT" name="Run All Test Cases" description="All_Test_Cases" operation="" />
    <TestCase lineNumber="177" date="2015-10-27 10:19 AM PDT" var="$number" type="char []" size="%16" />
    <Set lineNumber="178" date="2015-10-27 10:19 AM PDT" var="$number" operation="equal" value="%TRUE" />
    <Set lineNumber="179" date="2015-10-27 10:19 AM PDT" var="$number" operation="equal" value="%FALSE" />
    <Set lineNumber="180" date="2015-10-27 10:19 AM PDT" var="$number" operation="equal" value="%TRUE" />
    <Call lineNumber="181" date="2015-10-27 10:19 AM PDT" ref="fpui_open" >
      <Set lineNumber="30" date="2015-10-27 10:19 AM PDT" name="FPUI_Init_Variables" >
        <Set lineNumber="31" date="2015-10-27 10:19 AM PDT" var="$returnCode" operation="equal" value="%TRUE" />
        <Set lineNumber="32" date="2015-10-27 10:19 AM PDT" var="$errno" operation="equal" value="%TRUE" />
      </Set>
    </Call>
    <Subroutine lineNumber="140" date="2015-10-27 10:19 AM PDT" name="fpui_open" description="Call fpui_open()" >
      <Function lineNumber="141" date="2015-10-27 10:19 AM PDT" funcName="fpui_open" returnType="fpui_handle" errno="errno" p1="flags" />
      <If lineNumber="146" date="2015-10-27 10:19 AM PDT" expression="fpui_handle EQ %1-1" evaluatesTo="FALSE/ELSE" >
        <If />
      </If>
    </Subroutine>
  </TestSuite>
  <TestSuite lineNumber="224" date="2015-10-27 10:19 AM PDT" name="Run All Test Cases" description="All_Test_Cases" operation="" />
    <TestCase lineNumber="177" date="2015-10-27 10:19 AM PDT" var="$number" type="char []" size="%16" />
    <Set lineNumber="178" date="2015-10-27 10:19 AM PDT" var="$number" operation="equal" value="%TRUE" />
    <Set lineNumber="179" date="2015-10-27 10:19 AM PDT" var="$number" operation="equal" value="%FALSE" />
    <Set lineNumber="180" date="2015-10-27 10:19 AM PDT" var="$number" operation="equal" value="%TRUE" />
    <Call lineNumber="181" date="2015-10-27 10:19 AM PDT" ref="fpui_open" >
      <Set lineNumber="30" date="2015-10-27 10:19 AM PDT" name="FPUI_Init_Variables" >
        <Set lineNumber="31" date="2015-10-27 10:19 AM PDT" var="$returnCode" operation="equal" value="%TRUE" />
        <Set lineNumber="32" date="2015-10-27 10:19 AM PDT" var="$errno" operation="equal" value="%TRUE" />
      </Set>
    </Call>
    <Subroutine lineNumber="140" date="2015-10-27 10:19 AM PDT" name="fpui_open" description="Call fpui_open()" >
      <Function lineNumber="141" date="2015-10-27 10:19 AM PDT" funcName="fpui_open" returnType="fpui_handle" errno="errno" p1="flags" />
      <If lineNumber="146" date="2015-10-27 10:19 AM PDT" expression="fpui_handle EQ %1-1" evaluatesTo="FALSE/ELSE" >
        <If />
      </If>
    </Subroutine>
  </TestSuite>
</ApilVsRun>
```

APIVS SW UM v01.01
<Set lineNumber="203" date="2015-10-27 10:19 AM PDT" var="column" operation="equal" value="%5" />
<Set lineNumber="204" date="2015-10-27 10:19 AM PDT" var="write_buf" operation="equal" value="Demonstration of FPUI API" />
<Call lineNumber="205" date="2015-10-27 10:19 AM PDT" ref="fpui_write_string_at" >
<html>
<SetUp lineNumber="30" date="2015-10-27 10:19 AM PDT" name="FPUI_Init_Variables" >
<Set lineNumber="31" date="2015-10-27 10:19 AM PDT" var="returnCode" operation="equal" value="%-1" />
<Set lineNumber="32" date="2015-10-27 10:19 AM PDT" var="errno" operation="equal" value="%0" />
</SetUp>
<Subroutine lineNumber="154" date="2015-10-27 10:19 AM PDT" name="fpui_write_string_at" description="Call fpui_write_string_at()" >
<Function lineNumber="155" date="2015-10-27 10:19 AM PDT" funcName="fpui_write_string_at" return="write_cnt" errno="errno" >
pl="fpui_handle" p2="write_buf" p3="row" p4="column" />
<If lineNumber="162" date="2015-10-27 10:19 AM PDT" expression="!VDCompare(@C2001d_vd_00) EQ #FALSE" evaluatesTo="FALSE/ELSE" >
</If>
</Function>
</Subroutine>
<If lineNumber="209" date="2015-10-27 10:19 AM PDT" expression="!VDCompare(#C2001d_vd_00) EQ #FALSE" evaluatesTo="FALSE/ELSE" >
</If>
<Call lineNumber="218" date="2015-10-27 10:19 AM PDT" ref="fpui_close" >
<html>
<SetUp lineNumber="30" date="2015-10-27 10:19 AM PDT" name="FPUI_Init_Variables" >
<Set lineNumber="31" date="2015-10-27 10:19 AM PDT" var="returnCode" operation="equal" value="%-1" />
<Set lineNumber="32" date="2015-10-27 10:19 AM PDT" var="errno" operation="equal" value="%0" />
</SetUp>
<Subroutine lineNumber="127" date="2015-10-27 10:19 AM PDT" name="fpui_close" description="Call fpui_close()" >
<Function lineNumber="128" date="2015-10-27 10:19 AM PDT" funcName="fpui_close" return="returnCode" errno="errno" >
pl="fpui_handle" />
<If lineNumber="132" date="2015-10-27 10:19 AM PDT" expression="returnCode EQ %-1" evaluatesTo="FALSE/ELSE" >
</If>
</Function>
</Subroutine>
</Call>
</TestSuite>

<RunResult date="2015-10-27 10:19 AM PDT" status="PASS" />
</ApiVsRun>
A.4 Flat File Format

The following specifies the format of the “flat files” used by the Validation Suite.

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,
- The sequence “\>” is translated as a single 0x3e (>) character,
- 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 and 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, the destination is padded with 0x00. If the destination is shorter than the converted binary form, the converted binary form is truncated to fit the destination size.
API Validation Suite
APIVSXML Specification

Version: 2.0
Save Date: 2015-11-11
Author: Thomas E. Gauger

File Name: APIVSXML.doc
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010/12/31</td>
<td>2.0</td>
<td>Accepted all changes to revision 1.4. Updated based upon actual implementation of APIVSXML.</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/18</td>
<td>1.4</td>
<td>Accepted all changes made to revision 1.3. Corrected some formatting issues, correct minor problems with examples.</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/17</td>
<td>1.3</td>
<td>Changes made during development, improved Validation Suite Diagram, added &lt;Abort&gt; element, correct minor typos, added “Hello World” example, added TOD example, added FPUI example, added supported for O_FLAG constants, added strlen() macro, added FIO example, fixed names for FIO_PORT constants, corrected understanding of RQMSG printing and processing, added #FIO_BIT_TEST() macro.</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/06</td>
<td>1.2</td>
<td>Changes following design review</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/08/01</td>
<td>1.1</td>
<td>Additions to constant values</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
<tr>
<td>2010/07/30</td>
<td>1.0</td>
<td>Initial Released Version</td>
<td>Thomas E. Gauger nFocal</td>
</tr>
</tbody>
</table>
# Table of Contents

1. **Introduction** ........................................................................................................................................ 33
   1.1 Purpose ............................................................................................................................................... 33
   1.2 Intended Audience ............................................................................................................................. 33
   1.3 Scope .................................................................................................................................................. 33
   1.4 Conventions ....................................................................................................................................... 33
   1.5 Definitions, Acronyms, and Abbreviations ...................................................................................... 34
   1.6 References ........................................................................................................................................... 35

2. **APIVSXML Schema Definition** .......................................................................................................... 36
   2.1 APIVSXML Syntax and Naming Convention Guidelines .......................................................... 36
   2.1.1 Element Naming Convention ............................................................................... 36
   2.1.2 Attribute Naming Convention ............................................................................... 36
   2.1.3 API Defined Function Names ............................................................................... 36
   2.1.4 API Defined Variable Types ................................................................................ 36
   2.1.5 API Defined Constants and VS Defined Constants ............................................. 37
   2.1.6 String Constants ....................................................................................................... 37
   2.1.7 Numeric Constants ................................................................................................ 37
   2.1.8 Macro Definitions .................................................................................................. 37
   2.2 Validation Suite Architecture ............................................................................................. 38
   2.2.1 Validation Suite Description ................................................................................. 38
   2.2.2 Validation Suite Structure .................................................................................... 38
   2.3 APIVSXML Schema .......................................................................................................... 40
   2.3.1 APIVSXML Attributes ......................................................................................... 40
   2.3.2 Predefined API Values ....................................................................................... 48
   2.3.3 APIVSXML Elements ......................................................................................... 53

3. **APIVSXML Quick Reference Guide** .................................................................................................. 68

4. **APIVSXML Usage Examples** ........................................................................................................... Error! Bookmark not defined.
List of Figures

Figure 1 - Validation Test Suite Structure 38

List of Tables

Table 1. Definitions 34
Table 2. Reference Documents 35
1. INTRODUCTION

1.1 Purpose
This document describes the APIVSXML schema, syntax, semantics and usage for the API Validation Suite project. The API Validation Suite APIVSXML is the methodology by which test cases are defined for the validation suite and is used by the validation suite software to drive API validation testing.

The APIVSXML describes a programming language that is used to construct a Validation Suite (VS) for the ATC API. As such, it must contain many of the same elements of any other programming language: variables, defined constants, actions, sequences of functions, loops, if-then-else processing, subroutine calls and error handling -- to name a few.

1.2 Intended Audience
This document is of a technical nature. It is intended for members of the cross-functional project team involved in the design, implementation, maintenance, and testing of the API Validation Suite project.

1.3 Scope
This document defines the schema, syntax, semantics and usage of the API Validation Suite APIVSXML that is used to define the actual validation suite test cases and is used by the API Validation Suite VSE to drive these tests.

1.4 Conventions
- Square brackets ([ ]) indicate to-be-determined information which will be resolved before this document is finalized. For example, “The minimum hard disk space is [TBD] MB.”
- Mono-spaced type indicates functions, variables, parameters, statements, commands, and other operating system or programming language constructs. For example, “Use the dir command to list the files and directories.”
- Italicized text (substitution) indicates a substitution parameter. For example, “Use the dir path command to list the files and directories in the given path.”
- Bold type indicates user interface elements or APIVSXML syntax (windows, dialog boxes, menus, and controls). For example, “Press the OK button.” or “the print attribute …”
- Vertical bars (|) are used to indicate a path through a hierarchical user interface. For example, “Select File | Open… to open a file.”
1.5 Definitions, Acronyms, and Abbreviations

The following terms are defined in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APIVSXML</td>
<td>Application Programming Interface Validation Suite eXtensible Markup Language</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange – Character set utilized</td>
</tr>
<tr>
<td>ATC</td>
<td>Advanced Traffic Controller</td>
</tr>
<tr>
<td>FIO</td>
<td>Field I/O. A library of the ATC API.</td>
</tr>
<tr>
<td>FIOD</td>
<td>Field I/O Device</td>
</tr>
<tr>
<td>FPUI</td>
<td>Front Panel User Interface. A library of the ATC API.</td>
</tr>
<tr>
<td>I/O, IO</td>
<td>Input / Output</td>
</tr>
<tr>
<td>IPC</td>
<td>Inter-Process Communications</td>
</tr>
<tr>
<td>TOD</td>
<td>Time Of Day. A library of the ATC API.</td>
</tr>
<tr>
<td>VD</td>
<td>Virtual Display – Result of VT100 emulator processing of FPUI output.</td>
</tr>
<tr>
<td>VS</td>
<td>Validation Suite</td>
</tr>
<tr>
<td>VSE</td>
<td>Validation Suite Engine</td>
</tr>
<tr>
<td></td>
<td>Software that utilizes the APIVSXML to validate a manufactures API.</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
</tbody>
</table>

Table 1. Definitions
1.6 References

The following may be referenced in this document. These documents, of the issue in effect on the date of this specification, form a part of this specification to the extent specified herein. In the event of a conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered a superseding requirement.

<table>
<thead>
<tr>
<th>Source Document</th>
<th>ID Tag</th>
<th>Author</th>
<th>Date</th>
<th>Document Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIVSXML.xsd</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td></td>
</tr>
<tr>
<td>API VS Design Specification</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td></td>
</tr>
<tr>
<td>HW_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>Hello World APIVSXML example, found in etc directory</td>
</tr>
<tr>
<td>TOD_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>TOD library APIVSXML example, found in etc directory</td>
</tr>
<tr>
<td>FPUI_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>FPUI library APIVSXML example, found in etc directory</td>
</tr>
<tr>
<td>FIO_Example.xml</td>
<td></td>
<td>Thomas E. Gauger</td>
<td>20101231</td>
<td>FIO library APIVSXML example, found in etc directory</td>
</tr>
</tbody>
</table>

Table 2. Reference Documents
2. APIVSXML SCHEMA DEFINITION

This section defines the schema for the APIVSXML. This schema defines all tags, elements and attributes of the APIVSXML. The document APIVSXML.xsd, defined in the references table, describes the actual XML schema used by the Validation Suite Engine (VSE). This document does not attempt to teach XML or XSD syntax or usage. It is assumed that the reader is familiar with XML and XSD, their general syntax and rules for defining XML and XSD documents. This document rather describes the specifics of the schema for the APIVSXML and defines elements described in the XSD document that defines this schema. A good simple explanation of XML and its constructs may be found in the references table above.

2.1 APIVSXML Syntax and Naming Convention Guidelines

This section describes the syntax and naming conventions used in defining the APIVSXML. These guidelines are present to ensure consistency in the way the language is defined to improve readability, usability and maintainability. Consistency and orthogonality are important.

2.1.1 Element Naming Convention

Element names, tags, shall use the “Title” or “BumpyName” syntax. An example is:

<TestCase> … </TestCase>

2.1.2 Attribute Naming Convention

Attributes shall be named using a lower case string. If an attribute name contains more than one word, the attribute shall be named using “lowerCaseTitle” or “bumpyName” syntax. For instance:

<TestCase description="This is a test case." />

2.1.3 API Defined Function Names

All referenced ATC API function names will match their definition in the ATC API. For instance:

<Function name="fio_register" />

2.1.4 API Defined Variable Types

In the course of processing the API, it is necessary to define variables that can be initialized for a function call or where returned values can be stored and referenced by subsequent function calls or expressions. All such variables must be preceded by the ‘$’ character when referenced. The actual variable name may be in upper or lower case and is case sensitive. The variable $ABC is not the same as variable $abc. It is highly recommended that the user be consistent in defining any such variables. A variables scope is global to a run of the VSE. There is no practical limit to the number of variables that may be defined. All variables must be defined before they are used. All referenced API variable types must match their type definition in the ATC API. If the types mismatch, the VSE will output an error message and will terminate.

For instance:

<Define type="unsigned char" var="$tmpChar" /> <!-- Defines the variable unsigned char $tmpChar -->
2.1.5 API Defined Constants and VS Defined Constants

The API and the Validation Suite define several enumerated values and defined values. These values must be referenced by the VS test cases. All such values must be preceded by the ‘#’ character when referenced. This syntax denotes a named numeric constant. The actual value string, for API defined constants, is consistent with what is defined by the API. For instance:

```xml
<Function p1="#FIO_SP3" />
```

2.1.6 String Constants

The API and the Validation Suite require character strings to be defined and referenced. These values must be referenced by the VS test cases. All such values must be preceded by the ‘@’ character when referenced. This syntax denotes a character string constant. For instance:

```xml
<Function p1="@This is a character string" />
```

2.1.7 Numeric Constants

The API and the Validation Suite require numeric constants to be defined and referenced. These values must be referenced by the VS test cases. All such values must be preceded by the ‘%’ character when referenced. This syntax denotes a numeric constant. For instance:

```xml
<Function p1="%25" />
```

2.1.8 Macro Definitions

The APIVSXML defines several macros used in processing. These macros must be referenced by the VS test cases. All such macros must be preceded by the ‘!’ character when referenced. This syntax denotes a macro. For instance:

```xml
<If expression="!'FIO_BIT_TEST($array,$bit) NE #TRUE" />
```
2.2 Validation Suite Architecture

This section describes the overall Validation Suite (VS) Architecture that the APIVSXML satisfies.

2.2.1 Validation Suite Description

The VS is used to validate a manufactures implementation of the ATC API. XML is utilized because it is both human readable and is usable by a computer program. The APIVSXML is the mechanism by which the VS is defined in a human readable format. The APIVSXML is subsequently used by the Validation Suite Engine (VSE) to actually execute the validation tests that have been described to validate a manufacturer’s implementation of the ATC API. The APIVSXML is easily extensible to include new validation tests as they are identified.

2.2.2 Validation Suite Structure

The following diagram illustrates the structure of the Validation Suite (VS).

Figure 1 - Validation Test Suite Structure

The validation test suite is defined in an APIVSXML document. There may be multiple APIVSXML documents. Which validation test suite to execute is specified as an argument to the VSE at run-time.
Each validation test suite consists of a series of validation test cases. These validation test cases are described and defined in the APIVSXML document. Each validation test case is described by a validation test case set up, validation program and validation test case tear down sequence. A validation program consists of a sequence of APIVSXML statements (elements), arranged in such a way as to validate the ATC API feature being tested.

The “statements” are really APIVSXML elements, that like any other programming language, allows for the execution of all ATC API function interfaces and the validation of all output parameters, function return values, validation of Linux errno, events that should be “seen” or encountered, and observation of current state versus expected state. The APIVSXML is an open ended “programming” language that allows for countless combinations of processing and validation to be performed. The APIVSXML allows for easily updating the test suite as new validation tests are identified.
2.3 APIVSXML Schema

This section discusses the APIVSXML schema defined in the APIVSXML.xsd document.

2.3.1 APIVSXML Attributes

This section describes all attributes that are supported and referenced by the various APIVSXML elements described in section 0 below.

2.3.1.1 Attributes Common to All Elements

This section contains attributes that may be used in any and all APIVSXML elements.

2.3.1.1.1 Attribute: description

The description attribute may be used to add descriptive text to any element. The value associated with this attribute is a free-form text string. This descriptive text is output by the Validation Suite Engine when an element is added to the output XML, during the processing of a validation suite.

2.3.1.2 Attributes that are context specific

The attributes listed in this section are context specific and may only be used with the correct associated APIVSXML element.

2.3.1.2.1 Attribute: name

The name attribute is used to label various APIVSXML elements. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. The name space is specific to the APIVSXML element being named. This means that it is possible to have a Test Suite

2.3.1.2.2 Attribute: ref

The ref attribute is used to reference various APIVSXML elements that are named using the name attribute. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. All name attribute values must be unique and must be defined before being referenced.
2.3.1.2.3 Attribute: funcName

The **funcName** attribute is used to specify the specific ATC API function that is to be called by the VSE. Valid values for **funcName** are:

- fpui_apiver
- fpui_clear
- fpui_clear_tab
- fpui_close
- fpui_close_aux_switch
- fpui_compose_special_char
- fpui_del_keymap
- fpui_display_special_char
- fpui_get_auto_repeat
- fpui_get_auto_scroll
- fpui_get_auto_wrap
- fpui_get_backlight
- fpui_get_character_blink
- fpui_get_cursor
- fpui_get_cursor_blink
- fpui_get_cursor_pos
- fpui_get_focus
- fpui_get_keymap
- fpui_get_led
- fpui_get_reverse_video
- fpui_get_underline
- fpui_get_window_attr
- fpui_get_window_size
- fpui_home
- fpui_open
- fpui_open_aux_switch
- fpui_poll
- fpui_read
- fpui_read_aux_switch
- fpui_read_char
- fpui_read_string
- fpui_refresh
- fpui_reset_all_attributes
- fpui_reset_keymap
- fpui_set_auto_repeat
- fpui_set_auto_scroll
- fpui_set_auto_wrap
- fpui_set_backlight
- fpui_set_backlight_timeout
- fpui_set_character_blink
- fpui_set_cursor
- fpui_set_cursor_blink
- fpui_set_cursor_pos
- fpui_set_emergency
• fpui_set_keymap
• fpui_set_led
• fpui_set_reverse_video
• fpui_set_tab
• fpui_set_underline
• fpui_set_window_attr
• fpui_write
• fpui_write_at
• fpui_write_char
• fpui_write_char_at
• fpui_write_string
• fpui_write_string_at

• fio_apiver
• fio_deregister
• fio_fiod_channel_map_count
• fio_fiod_channel_map_get
• fio_fiod_channel_map_set
• fio_fiod_channel_reservation_get
• fio_fiod_channel_reservation_set
• fio_fiod_cmos_dark_channel_get
• fio_fiod_cmos_dark_channel_set
• fio_fiod_cmos_fault_get
• fio_fiod_cmos_fault_set
• fio_fiod_deregister
• fio_fiod_disable
• fio_fiod_enable
• fio_fiod_frame_notify_deregister
• fio_fiod_frame_notify_register
• fio_fiod_frame_read
• fio_fiod_frame_schedule_get
• fio_fiod_frame_schedule_set
• fio_fiod_frame_size
• fio_fiod_inputs_filter_get
• fio_fiod_inputs_filter_set
• fio_fiod_inputs_get
• fio_fiod_inputs_trans_get
• fio_fiod_inputs_trans_read
• fio_fiod_inputs_trans_set
• fio_fiod_mmu_flash_bit_get
• fio_fiod_mmu_flash_bit_set
• fio_fiod_outputs_get
• fio_fiod_outputs_reservation_get
• fio_fiod_outputs_reservation_set
• fio_fiod_outputs_set
• fio_fiod_register
• fio_fiod_status_get
• fio_fiod_status_reset
• fio_fiod_ts_fault_monitor_get
• fio_fiod_ts_fault_monitor_set
• fio_fiod_ts1_volt_monitor_get
• fio_fiod_ts1_volt_monitor_set
• fio_fiod_wd_deregister
• fio_fiod_wd_heartbeat
• fio_fiod_wd_register
• fio_fiod_wd_reservation_get
• fio_fiod_wd_reservation_set
• fio_hm_deregister
• fio_hm_fault_reset
• fio_hm_heartbeat
• fio_hm_register
• fio_query_fiod
• fio_query_frame_notify_status
• fio_register

• tod_cancel_onchange_signal
• tod_cancel_tick_signal
• tod_get
• tod_get_dst_info
• tod_get_dst_state
• tod_get_timesrc
• tod_get_timesrc_freq
• tod_request_onchange_signal
• tod_request_tick_signal
• tod_set
• tod_set_dst_info
• tod_set_dst_state
• tod_set_timesrc
2.3.1.2.4 Attribute: type

The `type` attribute is used to specify the type of variable to be created in a `<Define>` element. Valid values for `type` are:

- char
- char []
- unsigned char
- unsigned char []
- boolean
- ssize_t
- int
- unsigned int
- fpui_handle
- O_FLAGS
- FIO_APP_HANDLE
- FIO_DEV_HANDLE
- FIO_VERSION
- FIO_VIEW
- FIO_CHANNEL_MAP
- FIO_CMU_DC_MASK
- FIO_CMU_FSA
- FIO_NOTIFY
- FIO_FRAME_SCHD
- FIO_INPUT_FILTER
- FIO_INPUTS_TYPE
- FIO_TRANS_STATUS
- FIO_TRANSBUFFER
- FIO_MMU_FLASH_BIT
- FIO_PORT
- FIO_DEVICE_TYPE
- FIO_FIOD_STATUS
- FIO_TS_FM_STATE
- FIO_TS1_VM_STATE
- FIO_NOTIFY_INFO
- struct timeval
- dst_info_t

2.3.1.2.5 Attribute: var

The `var` attribute is used to reference a variable that was created using `<Define>`. Since this is a reference, the variable must appear is `$variable`.

2.3.1.2.6 Attribute: value

The `value` attribute is used to set a value to a `$variable` in a `<Set>` element.

2.3.1.2.7 Attribute: size

The `size` attribute is used to specify the number of occurrences of `type` for this `var` in a `<Define>`.
2.3.1.2.8 Attribute: operation

The operation attribute is used to specify what is to be done in a <Set> element. Valid values for operation are:

- (not present) – A set var equal to value is performed.
- add – value is added to var
- subtract – value is subtracted from var

2.3.1.2.9 Attribute: member

The member attribute is used to reference a member of a structure that was created using <Define>. The member name is given as the value of the attribute.

2.3.1.2.10 Attribute: signal

The signal attribute is used to specify the Linux signal that is to be handled. Either a variable, constant or defined constant may be referenced.

2.3.1.2.11 Attribute: action

The action attribute is used to specify what is to be done with the signal in question. Valid values for action are:

- enable
- disable

2.3.1.2.12 Attribute: return

The return attribute is used to reference a variable that was created using <Define>, which is used to place the return value of a <Function>. Since this is a reference, the variable must appear is $variable.

2.3.1.2.13 Attribute: errno

The errno attribute is used to reference a variable that was created using <Define>, which is used to place the errno value of a <Function> into. Since this is a reference, the variable must appear is $variable.

2.3.1.2.14 Attribute: p1

The p1 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear is $variable.

2.3.1.2.15 Attribute: p2

The p2 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear is $variable.

2.3.1.2.16 Attribute: p3

The p3 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear is $variable.

2.3.1.2.17 Attribute: p4

The p4 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear is $variable.

2.3.1.2.18 Attribute: p5

The p5 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear is $variable.
2.3.1.2.19 Attribute: p6
The p6 attribute is used to reference a variable, constant or defined constant that is to be passed to a <Function>. Since this is a reference, a variable must appear is $variable.

2.3.1.2.20 Attribute: expression
The expression attribute is used to define a boolean expression to be evaluated. References to variables must appear as $variable, references to numeric constants must appear as %number, references to named constants must appear as #constant, references to strings must appear as @string, and references to macros must appear as !macro().

2.3.1.2.21 Attribute: setUp
The setUp attribute is used to reference a <SetUp> element. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. All <SetUp> name attribute values must be unique and must be defined before they are referenced.

2.3.1.2.22 Attribute: tearDown
The tearDown attribute is used to reference a <TearDown> element. In so doing, an element can be defined once and referenced and re-used multiple times like a subroutine call in a programming language. All <TearDown> name attribute values must be unique and must be defined before they are referenced.

2.3.1.2.23 Attribute: status
The status attribute is used to specify special PASS / FAIL notification in the log file. Valid values for status are:

- FAIL – Causes a -1 (non-conformance) to be returned to the calling environment.
- PASS – Causes a 0 (conformance) to be returned to the calling environment. This is the default status, unless “FAIL” is specified.

2.3.1.2.24 Attribute: time
The time attribute is used to specify a time in seconds. The value of the attribute must be a numeric constant, integer variable or named constant.

2.3.1.2.25 Attribute: file
The file attribute is used to specify the input file to be used a.

2.3.1.2.26 Attribute: frame
The frame attribute is used to specify the request frame to apply FIO input to.

2.3.1.2.27 Attribute: level
The level attribute is used to specify the output processing level that the APIVSXML element in question is to be processed at. Only some elements contain this attribute. Valid values for level are:

- always – Always process this element regardless of output level.
- conformance – Only process this element when running at the conformance level.
- summary – Only process this element when running at the conformance or summary level. This is the default level if this attribute is not given for an element that uses this attribute.
- trace – Process this element only when running at the trace level
2.3.1.2.28 Attribute: dump
The dump attribute is used to specify the type of output for a <Dump> element. Valid values for dump are:

- VD – A dump of the Virtual Display (VD) is performed.
- CMDMSG – A dump of a FIO Command Message is performed.

2.3.1.2.29 Attribute: sequence
The sequence attribute is used to specify a sequence number to be appended to a dump file name in a <Dump> element.

2.3.1.2.30 Attribute: force
The force attribute is used to specify if a <Dump> should occur, regardless of the “capture” mode being used. This attribute is useful to ensure the capturing of <Dump> elements when a failure is present; otherwise <Dump> elements are only normally processed during “capture” mode. Valid values for force are:

- #TRUE – Always perform dump.
- #FALSE – (default) Only perform dump during capture mode.

2.3.1.2.31 Attribute: load
The load attribute is used to specify the type of input for a <Load> element. Valid values for load are:

- VD – A load of the Virtual Display (VD) is performed.
- CMDMSG – A load of a FIO Command Message is performed.

2.3.1.2.32 Attribute: flip
The flip attribute is used to specify the type of flip for a <Aux> element. Valid values for flip are:

- #ON – Flip the Auxiliary Switch On.
- #OFF – Flip the Auxiliary Switch Off.
2.3.2 Predefined API Values

This section describes all constants and macros that are supported and referenced by the various APIVSXML elements described in section 2.3.3 APIVSXML Elements below.

2.3.2.1 VSE Named Constants

This section describes named constants that are supported by the Valid Suite Engine (VSE). All named constants are referenced by the preceding a constant with a ‘#’ symbol.

2.3.2.1.1 #NULL
This constant is used to test / compare / set a pointer, returned by an ATC API or to be passed to the ATC API, the special value 0x00000000 or <null>.

2.3.2.1.2 #TRUE
This constant is used to test / compare / set a boolean to a value of 0x00000001 or <true>.

2.3.2.1.3 #FALSE
This constant is used to test / compare / set a boolean to a value of 0x00000000 or <false>.

2.3.2.1.4 #ON
This constant is used to test / compare / set a boolean to a value of 0x00000001 or <true>.

2.3.2.1.5 #OFF
This constant is used to test / compare / set a boolean to a value of 0x00000000 or <false>.

2.3.2.1.6 #ROWS
This constant is the value of the number of rows in Virtual Display set in the configuration file for this VSE run.

2.3.2.1.7 #COLUMNS
This constant is the value of the number of columns in Virtual Display set in the configuration file for this VSE run.

2.3.2.1.8 Errno Constants
The VSE supports the errno constants by referencing the decimal equivalent of the errno being utilized, utilizing the %number syntax.

2.3.2.1.9 SIGNAL Constants
The VSE supports the signal value constants:

- #FIO_SIGIO
- #SIGWINCH
- Other signals may be referenced using their decimal equivalent and the %number syntax.
2.3.2.10 O_FLAGS Constants

The VSE supports the Linux O_FLAGS constants. Values may be OR'ED together using `<Set operation="add" />` element. If a constant name is not supported by the VSE, the O_FLAGS value may be specified in using the decimal equivalent %number syntax. The constant values supported are:

- #O_RDWR
- #O_RDONLY
- #O_WRONLY
- #O_DIRECT
- #O_NONBLOCK

2.3.2.2 VSE Macros

This section defines macros that are supported by the VSE to allow processing of information that the VSE maintains or collects during the course of running an APIVSXML “program”. All macros are preceded by the ‘!’ character.

2.3.2.2.1 Generally Useful Macros

During the course of VSE processing, certain operations will make processing easier. The macros listed below are generally useful for VSE processing in general. The available macros are:

- `!strlen(char [])`
  This macro may be used to return the string length of a variable using type “char []”. The value returned is of type “int”.
- `!sizeof($var)`
  This macro may be used to return the number of elements of $var, as configured using the size attribute in a `<Define>`. The value returned is of type “int”.

2.3.2.2.2 FPUI Macros

During the course of processing FPUI API calls, it is necessary to compare the current state of the Virtual Display (VD) generated by the VT100 emulator against what the current VD is expected to contain. In order to accomplish this comparison, the following macro is defined:

- `!VDCompare(@expected_display_file)`
  This macro will return #TRUE if the current contents of the VD match the information contained in the “expected_display_file”. Otherwise, this macro returns #FALSE. The “@expected_display_file” is the path to the file to be used for comparison to the VD.

2.3.2.2.3 FIO Macros

During the course of processing FIO API calls, it is necessary to compare the last SDLC Command Message (CMDMSG) that was received on the FIO Loopback Port, for the given frame_number, against what the last CMDMSG was expected to be and contain. In order to accomplish this comparison, the following macro is defined:

- `!CMDMSGCompare(@expected_message_file,%frame_number)`
  This macro will return #TRUE if the last CMDMSG received on the FIO Loopback Port matches the information contained in “expected_message_file”. Otherwise, this macro returns #FALSE. The “@expected_message_file” is the path to the file to be used for comparison to the last CMDMSG received. During processing of output and input points, it is necessary to test bits in a bit array. The following macro supports testing bits in a bit array:

- `!FIO_BIT_TEST($array,%bit)`
This macro will return #TRUE if the bit is set in the bit array, otherwise #FALSE is returned.

### 2.3.2.3 ATC API Constants

This section describes constants that are support by the VSE that are defined by the ATC API itself.

#### 2.3.2.3.1 FPUI Constants

There are no constants that are just specific to the FPUI.

#### 2.3.2.3.2 FIO Constants

This section describes constants that are specific to the FIO.

##### 2.3.2.3.2.1 FIO_VERSION Constants

The constants that are valid for FIO_VERSION are:

- #FIO_VERSION_LIBRARY
- #FIO_VERSION_LKM

##### 2.3.2.3.2.2 FIO_VIEW Constants

The constants that are valid for FIO_VIEW are:

- #FIO_VIEW_APP
- #FIO_VIEW_SYSTEM

##### 2.3.2.3.2.3 FIO_CMU_DC_MASK Constants

The constants that are valid for FIO_CMU_DC_MASK are:

- #FIO_CMU_DC_MASK1
- #FIO_CMU_DC_MASK2
- #FIO_CMU_DC_MASK3
- #FIO_CMU_DC_MASK4

##### 2.3.2.3.2.4 FIO_CMU_FSA Constants

The constants that are valid for FIO_CMU_FSA are:

- #FIO_CMU_FSA_NONE
- #FIO_CMU_FSA_NON_LATCHING
- #FIO_CMU_FSA_LATCHING

##### 2.3.2.3.2.5 FIO_NOTIFY Constants

The constants that are valid for FIO_NOTIFY are:

- #FIO_NOTIFY_ONCE
- #FIO_NOTIFY_ALWAYS

##### 2.3.2.3.2.6 FIO_INPUTS_TYPE Constants

The constants that are valid for FIO_INPUTS_TYPE are:

- #FIO_INPUTS_RAW
- #FIO_INPUTS_FILTERED
2.3.2.3.2.7 FIO_TRANS_STATUS Constants
The constants that are valid for FIO_TRANS_STATUS are:

- #FIO_TRANS_SUCCESS
- #FIO_TRANS_FIOD_OVERRUN
- #FIO_TRANS_APP_OVERRUN

2.3.2.3.2.8 FIO_MMU_FLASH_BIT Constants
The constants that are valid for FIO_MMU_FLASH_BIT are:

- #FIO_MMU_FLASH_BIT_ON
- #FIO_MMU_FLASH_BIT_OFF

2.3.2.3.2.9 FIO_PORT Constants
The constants that are valid for FIO_PORT are:

- #FIO_SP3
- #FIO_SP5
- #FIO_SP8
2.3.2.3.2.10 FIO_DEVICE_TYPE Constants
The constants that are valid for FIO_DEVICE_TYPE are:

- #FIO332
- #FIOTS1
- #FIOTS2
- #FIOMMU
- #FIODR1
- #FIODR2
- #FIODR3
- #FIODR4
- #FIODR5
- #FIODR6
- #FIODR7
- #FIODR8
- #FIOTF1
- #FIOTF2
- #FIOTF3
- #FIOTF4
- #FIOTF5
- #FIOTF6
- #FIOTF7
- #FIOTF8
- #FIOMUX
- #FIWINI1
- #FIWINI2
- #FIWINI3
- #FIWINI4
- #FIWINI5
- #FIOUT6SIU1
- #FIOUT6SIU2
- #FIOUT6SIU3
- #FIOUT6SIU4
- #FIOUT14SIU1
- #FIOUT14SIU2

2.3.2.3.2.11 FIO_TS_FM_STATE Constants
The constants that are valid for FIO_TS_FM_STATE are:

- #FIO_TS_FM_ON
- #FIO_TS_FM_OFF

2.3.2.3.2.12 FIO_TS1_VM_STATE Constants
The constants that are valid for FIO_TS1_VM_STATE are:

- #FIO_TS1_VM_ON
- #FIO_TS1_VM_OFF
2.3.2.3.2.13 FIO size Constants
The constants that are valid for FIO sizes are:
- #FIO_INPUT_POINTS_BYTES
- #FIO_OUTPUT_POINTS_BYTES
- #FIO_CHANNEL_BYTES

2.3.2.3.3 TOD Constants
This section describes constants that are specific to the TOD.

2.3.2.3.4 TOD_TIMESRC_ENUM Constants
The constants that are valid for TOD_TIMESRC_ENUM are:
- #TOD_TIMESRC_LINESYNC
- #TOD_TIMESRC_RTCSQWR
- #TOD_TIMESRC_CRYSTAL
- #TOD_TIMESRCEXTERNAL1
- #TOD_TIMESRCEXTERNAL2
2.3.3 APIVSXML Elements

This section describes all elements and corresponding legal attributes that are defined as part of the APIVSXML.

2.3.3.1.1 Block set up
At the start of a block, the attribute "setUp" may be used to specify a set up subroutine that may be utilized to perform initialization (C++ constructor like processing) for the block. Common sequences of ATC API functions may be processed in a set up subroutine, such as, registering for services and initialization of variables.

2.3.3.1.2 Block tear down
At the end of a block, the attribute "tearDown" may be used to specify a tear down subroutine that may be utilized to perform clean up (C++ destructor like processing) for the block. Tear down subroutines are commonly used for deregistering of services.
2.3.3.2 APIVSXML Elements

This section describes all elements of the APIVSXML.

2.3.3.3 APIVSXML Structure Elements

This section defines APIVSXML elements that define the structure of a validation test suite.

2.3.3.3.1 Element: <APIVSXML>

The top most, and root, element of an APIVSXML document is <APIVSXML>.

2.3.3.3.2 Element: <TestSuite>

A <TestSuite> defines all of the validation <TestCase> elements that encompass the validation test suite. A <TestSuite> element defines a block. Unless <Abort status="FAIL"/> has been called, the test suite exits with a “PASS” status.

Valid attributes for the <TestSuite> element are:

- **name**: REQUIRED – This attribute is a string that is referenced when the VSE is invoked to specify a validation test suite to be executed.
- **setUp**: optional – This attribute is a string that references a set up routine (defined with the <SetUp> element). This routine is called at the start of the test suite.
- **tearDown**: optional – This attribute is a string that references a tear down routine (defined with the <TearDown> element). This routine is called at the end of the test suite.
- **description**: optional – This attribute is used to add descriptive text to a test suite.

The <TestSuite> element then may contain an unbounded set of <TestCaseRef> elements. These <TestCaseRef> elements are executed sequentially until either an <Abort> element is encountered or all of the test cases have been executed.

Example:

```
<TestSuite name="Validation Test Suite #1"
           setUp="TestSuiteSetUp"
           tearDown="TestSuiteTearDown"
           description="Validation Test Suite #1" >

  <TestCaseRef ref="Test #1" />
  <TestCaseRef ref="Test #2" />
  <TestCaseRef ref="Test #N" />

</TestSuite>
```
2.3.3.3.3 Element: <TestCaseRef>

A <TestCaseRef> element references a <TestCase> element that actually tests and validates an ATC API feature under test (validation).

Valid attributes for the <TestCaseRef> element are:

- **ref**: REQUIRED – This attribute is a string that references a named <TestCase> element. The value of the ref attribute must match the value of a name attribute for one of the named <TestCase> elements in the document. <TestCase> must be defined before it is referenced.

Example:

```
<TestCaseRef ref="Test #1" />
<TestCaseRef ref="Test #2" />
<TestCaseRef ref="Test #N" />
```

2.3.3.3.4 Element: <TestCase>

A <TestCase> defines the sequence of steps that are executed by the VSE to perform validation of an ATC API feature. A <TestCase> element defines a block.

Valid attributes for the <TestCase> element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a <TestCaseRef> element.
- **setUp**: optional – This attribute is a string that references a set up routine (defined with the <SetUp> element). This routine is called at the start of the test case.
- **tearDown**: optional – This attribute is a string that references a tear down routine (defined with the <TearDown> element). This routine is called at the end of the test case.
- **description**: optional – This attribute is used to add descriptive text to a test case. This description is output into the log file, by the VSE, at the appropriate time.

The <TestCase> element then may contain an unbounded set of <Statement> elements. These <Statement> elements are executed sequentially until either an <Abort> element is encountered or all of the statements have been executed. The default exit status of a <TestCase> is “PASS”, unless otherwise specifically stated in an <Abort> call.

Example:

```
<TestCase name="Test #1"
    setUp="TestCase1SetUp"
    tearDown="TestCase1TearDown"
    description="Test #1">
    <Call ref="FIORegisterDeregisterSub"
         description="Call subroutine to register and then deregister with FIO" />
</TestCase>
```

2.3.3.3.5 Element: <Statement>

A <Statement> element is substituted with any of the APIVSXML elements listed in sections below.
2.3.3.4 APIVSXML Statement Primitive Elements

This section describes APIVSXML elements that define the primitive statement elements of the APIVSXML programming language.

2.3.3.4.1 Element: <Call>

The <Call> element is used to make a call to a <Subroutine> element.

Valid attributes for the <Call> element are:

- **ref**: REQUIRED – This attribute is a string that references a named <Subroutine> element. The value of the ref attribute must match the value of the name attribute for one of the named <Subroutine> elements in the document.

- **setUp**: optional – This attribute is a string that references a set up routine (defined with the <SetUp> element). This routine is called at the start of the subroutine.

- **tearDown**: optional – This attribute is a string that references a tear down routine (defined with the <TearDown> element). This routine is called at the end of the subroutine.

- **description**: optional – This attribute is used to add descriptive text to a call. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Call ref="FIORegisterDeregisterSub"
     description="Call subroutine to register and then deregister with FIO"/>
```
2.3.3.4.2 Element: <Subroutine>

The <Subroutine> element is used to create a logical collection of <Statement> elements. A <Subroutine> element defines a block.

Valid attributes for the <Subroutine> element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a <Call> element.
- **description**: optional – This attribute is used to add descriptive text to a subroutine. This description is output into the log file, by the VSE, at the appropriate time.

The <Subroutine> element then may contain an unbounded set of <Statement> elements. These <Statement> elements are executed sequentially until either an <Abort> element is encountered or all of the statements have been executed.

Example:

```xml
<Subroutine name="FIORegisterDeregisterSub"
            setUp="FIORegDeregSetUp"
            tearDown="FIORegDeregTearDown"
            description="Subroutine to process FIO register/deregister validation">

  <Function name="fio_register"
            return="$FIO_HANDLE"
            errno="$errno" />

  <Function name="fio_deregister"
            return="$returnInt"
            errno="$errno"
            p1="$FIO_HANDLE" />

</Subroutine>
```
2.3.3.4.3 Element: <Function>

The <Function> element is used to execute an ATC API function.

Valid attributes for the <Function> element are:

- **funcName**: REQUIRED – This attribute is a string that specifies the ATC API function to call.
- **description**: optional – This attribute is used to add descriptive text to a function call. This description is output into the log file, by the VSE, at the appropriate time.
- **return**: optional – This attribute specifies where the function's return value should be stored. If this attribute is omitted, the return value is ignored. If this attribute is specified, the variable referenced by the string value must be of the same type as the return value of the function. If a mismatch is detected, the VSE will output an error message and terminate.
- **errno**: optional – This attribute specifies where the errno value should be stored. If this attribute is omitted, the errno value is ignored. If this attribute is specified, the variable referenced by the string value must be of type integer. If a mismatch is detected, the VSE will output an error message and terminate.
- **p1 – p6**: optional – These attributes specify the parameters to be passed to the ATC API function call. Parameters are passed in the order specified by the p#. Each p# may be referenced only once. If a p# is referenced more than once or is absent, the VSE will output an error message and terminate. The variable or constant referenced by the string value must be of the same type as the ATC API function parameter. If a mismatch is detected, the VSE will output an error message and terminate.

Example:

```xml
<Function name="fio_register"
         return="$FIO_HANDLE"
         errno="$errno"/>

<Function name="fio_deregister"
         return="$returnInt"
         errno="$errno"
         p1="$FIO_HANDLE"/>
```
2.3.3.4.4 Element: <If>

The <If> element is used to perform validation of returned information (return code, errno and output variables) from a <Function> or a series of <Function> elements.

Valid attributes for the <If> element are:

- **expression**: REQUIRED – This attribute is a boolean expression of information that is to be validated. If the boolean expression evaluates to true, the <Then> element is executed, otherwise the <Else> element is executed. The <Then> and <Else> elements are optional. If they are not present, processing continues with the next <Statement>.
- **description**: optional – This attribute is used to add descriptive text to a <If> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<If expression="$FIO_HANDLE EQ #NULL" >
  <Then>
    <Print>FIO_HANDLE is NULL ... Error</Print>
    <Abort status="FAIL"/>
  </Then>
</If>

<If expression="$errno NE %0" >
  <Then>
    <Print>Errno is not 0 ... Error</Print>
    <Abort status="FAIL"/>
  </Then>
</If>

<If expression="!VDCompare(@cleared_screen) EQ #FALSE">
  <Then>
    <Print>Display is not cleared, contents of display</Print>
    <Dump force="#TRUE" dump="VD" file="@cleared_display_fail"/>
    <Abort status="FAIL"/>
  </Then>
</If>

<If expression="!CMDMSGCompare(@message_received,$frame) EQ #FALSE">
  <Then>
    <Print>Message not received</Print>
  </Then>
</If>
```

<Then>
  <Print >The correct message was not received</Print>
  <Dump force="#TRUE" dump="CMDMSG" file="@message_received_fail" />
  <Abort status ="FAIL"/>
</Then>
</If>

2.3.3.4.4.1 Element: <Then>

The <Then> element is used to perform “true” processing, following a <If> element evaluation.

Valid attributes for the <Then> element are:

- **description**: optional – This attribute is used to add descriptive text to a <Then> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

<If expression="$errno NE %0" >
  <Then>
    <Print >FIO_HANDLE is NULL … Error</Print>
    <Abort status ="FAIL"/>
  </Then>
</If>
2.3.3.4.4.2 Element: <Else>

The <Else> element is used to perform “false” processing, following a <If> element evaluation. An <Else> element defines a block.

Valid attributes for the <Else> element are:

- **description**: optional – This attribute is used to add descriptive text to a <Else> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<If expression="$FIO_HANDLE NE %0" >
    <Else description="FIO_HANDLE is valid" />
</If>
```

2.3.3.4.5 Element: <While>

The <While> element is used to perform a loop of <Statement> elements.

Valid attributes for the <While> element are:

- **expression**: REQUIRED – This attribute is a boolean expression of information that is to be validated. If the boolean expression evaluates to true, the <Statement> elements in the loop are executed. If the boolean expression evaluates to false, the <Statement> following the loop is executed.

- **description**: optional – This attribute is used to add descriptive text to a <While> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Define type="int" var="$iii" />
<SetUp name="setUpiii" >
    <Set var="$iii" value="%10" />
</SetUp>
<While expression="$iii NE %0" >
    <Set var="$iii" operation="subtract" value="%1" />
</While>
<Print >Loop is complete</Print>
```
2.3.3.4.6 Element: `<FPUIInput>`

The `<FPUIInput>` element is used to place input data on the FPUI Loopback Port, to be processed by FPUI read functions.

Valid attributes for the `<FPUIInput>` element are:

- **file**: REQUIRED – This attribute is a string that references a human-readable flat file that contains the data to be placed on the FPUI Loopback Port, that will be subsequently processed by a FPUI read call.
- **description**: optional – This attribute is used to add descriptive text. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<FPUIInput file="@FPUI_Input_Keystrokes" description="Input keystrokes from the front panel" />
```

2.3.3.4.7 Element: `<FIOResponse>`

The `<FIOResponse>` element is used to pre-load SDLC response message data for a given response frame. The FIO emulator will send this response frame when the correlated command message is received for the ATC FIOM.

Valid attributes for the `<FIOResponse>` element are:

- **file**: REQUIRED – This attribute is a string that references a human-readable flat file that contains the data to be pre-loaded for a given response frame.
- **frame**: REQUIRED – This attribute indicates the response frame that this input applies to.
- **description**: optional – This attribute is used to add descriptive text. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<FIOResponse input="@FIO_Response_Message" frame="%183" description="SDLC Response Message" />
```

2.3.3.4.8 Element: `<Aux>`

The `<Aux>` element is used to simulate the flipping of the Auxiliary Switch. The VT100 emulator will send the appropriate <ESC> sequences to the ATC, via the FP Loopback Cable, when the switch is flipped.

Valid attributes for the `<Aux>` element are:

- **flip**: REQUIRED – This attribute specifies the state of the Auxiliary Switch.
- **description**: optional – This attribute is used to add descriptive text. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Aux flip="#ON" />
```
2.3.3.4.9 Element: <Signal>

The <Signal> element is used set up signal processing by a <SignalHandler> element.

Valid attributes for the <Signal> element are:

- **signal**: REQUIRED – This attribute indicates the signal number that should be handled.
- **action**: REQUIRED – This attribute is used to enable or disable handling of the indicated signal.
- **ref**: optional – This attribute is a string that references a named <SignalHandler> element. The value of the ref attribute must match the value of the name attribute for one of the named <SignalHandler> elements in the APIVSXML document. The ref attribute is required if the action element is “enabled”. A name must be defined before it is referenced.
- **description**: optional – This attribute is used to add descriptive text to a call. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Signal signal="#SIGFIO" action="enable" ref="SigHandlerFIO" description="Handle the SIGFIO signal" />
<Signal signal="#SIGFIO" action="disable"/>
```

2.3.3.4.10 Element: <SignalHandler>

The <SignalHandler> element is used to define a signal handler. What signals are processed by this handler is defined by using the <Signal> statement.

Valid attributes for the <SignalHandler> element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a <Signal> element.
- **description**: optional – This attribute is used to add descriptive text to a signal handler. This description is output into the log file, by the VSE, at the appropriate time.

The <SignalHandler> element then may contain an unbounded set of <Statement> elements. These <Statement> elements are executed sequentially until either an <Abort> element is encountered or all of the statements have been executed.

Example:

```xml
<SignalHandler name="SigHandlerFIO" description="Signal Handler for handling SIGFIOs">
  <Print>fio notify has occurred</Print>
  <Set var="$sig_occurred" value="#TRUE"/>
</SignalHandler>
```
In order to capture information from a `<SignalHandler>` into the VSE foreground environment `<Set>` a variable in the `<SignalHandler>`. This variable can then be tested in the VSE foreground and take appropriate action.

### 2.3.3.5 APIVSXML Variable Elements

This section describes APIVSXML elements that are used to define and manipulate variables.

#### 2.3.3.5.1 Element: `<Define>`

The `<Define>` element is used to define a variable to be used during processing. A variable must be defined before it is referenced. When a variable is referenced, it must be preceded by a ‘$’, to indicate a reference to a defined variable.

Valid attributes for the `<Define>` element are:

- **var**: REQUIRED – This attribute indicates the variable to initialize. The variable must first the created using `<Define>` and the type of variable must match the context in which the variable is initialized or an error will occur and the VSE will terminate.

- **type**: REQUIRED – This attribute indicates the type of variable to create. When a variable is referenced, the type of variable must match the context in which the variable is referenced or an error will occur and the VSE will terminate.

- **size**: optional – This attribute is used to define the number of elements to be contained in the variable.

- **description**: optional – This attribute is used to add descriptive text to a `<Define>` element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Define var="$ii" type="int" />
<Define var="$FIO_HANDLE" type="FIO_APP_HANDLE" />
<Define var="$timeval" type="struct timeval" />
<Define var="$ls_plus" type="unsigned char []" size="#FIO_INPUT_POINTS_BYTES" />
```
### 2.3.3.5.2 Element: `<Set>`

The `<Set>` element is used to initialize a variable. The content of a variable is undefined until it is initialized. A variable must be defined before it is referenced. When a variable is referenced, it must be preceded by a `$`, to indicate a reference to a defined variable.

Valid attributes for the `<Set>` element are:

- **var**: REQUIRED – This attribute indicates the variable to initialize. The variable must first be created using `<Define>` and the type of variable must match the context in which the variable is initialized or an error will occur and the VSE will terminate.
- **value**: optional – This attribute specifies the value to set the variable to. The type of the value must be consistent with the type of the variable. Either value or file must be specified, but not both. Not all variable types support the `value` attribute.
- **file**: optional – This attribute specifies a human-readable file from which the data set in variable will be input from. Either value or file must be specified, but not both. Not all variable types support the `file` attribute.
- **operation**: optional – This attribute specifies the type of operation to be performed by the set. If absent, the variable is simply set to the indicated value or file. Not all variable types support all operations.
- **member**: optional – This attribute is utilized when the variable being initialized is a structure. In this case, each structure member must be initialized separately.
- **description**: optional – This attribute is used to add descriptive text to a `<Initialize>` element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Set var="$ii" value="%10" />
<Set var="$timeval" member="tv_sec" value="%100" />
<Set var="$ls_plus" file="@ls_plus_set" />
```
2.3.3.6 APIVSXML Flow Control Elements

This section describes APIVSXML elements that are used to control the flow of execution.

2.3.3.6.1 Element: <Sleep>

The <Sleep> element is used to delay processing for the indicated number of seconds.

Valid attributes for the <Sleep> element are:

- **time**: REQUIRED – This attribute indicates the number of seconds to sleep the execution of the APIVSXML program.
- **description**: optional – This attribute is used to add descriptive text to a <Sleep> element. This description is output into the log file, by the VSE, at the appropriate time.

Example:

```xml
<Sleep time="%10"/>
```

2.3.3.6.2 Element: <Abort>

The <Abort> element is used to terminate processing of a <TestSuite>. The current <TestSuite> will be marked in the output with the **status** that is specified.

Valid attributes for the <Abort> element are:

- **description**: optional – This attribute is used to add descriptive text to a <Abort> element. This description is output into the log file, by the VSE, at the appropriate time.
- **status**: required – This attribute is used to print a special tag to the output log file to indicate PASS or FAIL, if desired.

Example:

```xml
<Abort status="FAIL"/>
```
2.3.3.7 APIVSXML Supporting Block Elements

This section describes APIVSXML elements that are used for APIVSXML block processing. Several APIVSXML statements are used to enclose a block of `<Statement>` elements. These statements allow for the set up and tear down of information and processing associated with a block. This processing is the equivalent functionality to that of C++ constructors and destructors.

The APIVSXML `<Statement>` elements that define a block are: `<TestSuite>`, `<TestCase>`, and `<Call>`.

2.3.3.7.1 Element: `<SetUp>`

The `<SetUp>` element is used to define a set up processing sequence of statements.

Valid attributes for the `<SetUp>` element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a `setUp` attribute.
- **description**: optional – This attribute is used to add descriptive text to a `setUp` sequence. This description is output into the log file, by the VSE, at the appropriate time.

The `<SetUp>` element then may contain an unbounded set of `<Statement>` elements. These `<Statement>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the statements have been executed.

Example:

```
<SetUp name="setUpii">
    <Initialize var="$ii" value="%10" />
</SetUp>
```

2.3.3.7.2 Element: `<TearDown>`

The `<TearDown>` element is used to define a tear down processing sequence of statements.

Valid attributes for the `<TearDown>` element are:

- **name**: REQUIRED – This attribute is a string that is referenced by a `tearDown` attribute.
- **description**: optional – This attribute is used to add descriptive text to a `tearDown` sequence. This description is output into the log file, by the VSE, at the appropriate time.

The `<TearDown>` element then may contain an unbounded set of `<Statement>` elements. These `<Statement>` elements are executed sequentially until either an `<Abort>` element is encountered or all of the statements have been executed.

Example:

```
<TearDown name="tearDownSequence">
    <Function name="fio_deregister" />
</TearDown>
```
2.3.3.8 APIVSXML Logging Elements

This section describes APIVSXML elements that are used to support logging and the validation process.

2.3.3.8.1 Element: <Print>

The <Print> element is used to print information to the log file. <Print> is used to print a string of text to the output file.

Valid attributes for the <Print> element are:

- description: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.
- level: optional – This attribute allows for the control of at what run level a print occurs. When level is not present, summary level is assumed.

Example:

<Print>This text is output to the log file</Print>

<Print  description="This is the description"  level="always"  >*** FAILURE ***</Print>

2.3.3.8.2 Element: <Dump>

The <Dump> element is used to capture information to a human-readable file. <Dump> only occurs during a “capture” mode run of the VSE, unless the attribute force is specified.

Valid attributes for the <Dump> element are:

- description: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.
- level: optional – This attribute allows for the control of at what run level a print occurs. When level is not present, summary level is assumed.
- file: REQUIRED – This attribute specifies the name of the file for the <Dump>.
- dump: REQUIRED – This attribute specifies the type of <Dump> to perform.
- sequence: optional – For VD dumps, an optional sequence number may be appended to the file name. This allows <Dump> to occur in a loop.
- frame: optional – For CMDMSG dumps, a frame must be specified which indicates with command message frame to dump.
- force: optional – This attribute may be used to force the <Dump> to occur, regardless of the “capture” mode.

Example:

<Dump file="@VD_dump" dump="VD" />

<Dump description="This is the description" force="#TRUE" file="@CMDMSG_42" frame="%42" />
### 2.3.3.8.3 Element: `<Load>`

The `<Load>` element is used to load human-readable file information into the VSE. This is mostly used for testing the VSE itself.

Valid attributes for the `<Load>` element are:

- **description**: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.
- **file**: REQUIRED – This attribute specifies the name of the file for the `<Dump>`.
- **load**: REQUIRED – This attribute specifies the type of `<Load>` to perform.
- **frame**: optional – For CMDMSG loads, a frame must be specified which indicates with command message frame to load.

Example:

```xml
<Load file="@VD_dump" load="VD" />
<Load description="This is the description" file="@CMDMSG_42" frame="%42" />
```

### 2.3.3.8.4 Element: `<Format>`

The `<Format>` element is used print variables.

Valid attributes for the `<Format>` element are:

- **description**: optional – This attribute is used to add descriptive text to a print sequence. This description is output into the log file, by the VSE, at the start of the print.
- **level**: optional – This attribute allows for the control of at what run level a format occurs. When level is not present, summary level is assumed.
- **var**: optional – This attribute specifies a specific variable to be output. If omitted, all active variables are output.

Example:

```xml
<Format var="$ii" />
<Format description="Print all active variables" />
```
3. APIVSXML QUICK REFERENCE GUIDE

This section contains a quick APIVSXML reference guide for all APIVSXML elements and attributes; to have one place to look.

```xml
<TestSuite name="" setUp="" tearDown="" description="" />

<TestCaseRef ref="" />

<TestCase name="" setUp="" tearDown="" description="" />

<Call ref="" description="" setUp="" tearDown="" />

<Subroutine name="" description="" />

<Function name="" description="" return="" errno="" p1="" p2="" p3="" p4="" p5="" p6="" />

<If expression="" description="" />

<Then description="" />

<Else description="" />

<While expression="" description="" />

<FPUIInput file="" description="" />

<FIOResponse file="" frame="" description="" />

<Aux flip="" description="" />

<Signal signal="" action="" ref="" description="" />

<SignalHandler name="" description="" />

<Define var="" type="" description="" />

<Set var="" member="" description="" value="" file="" operation="" />

<Sleep time="" description="" />

<Abort status="" description="" />

<SetUp name="" description="" />

<TearDown name="" description="" />

<Print description="" level="" >STRING</Print>

<Dump dump="" file="" sequence="" frame="" description="" level="" />

<Load load="" file="" description="" frame="" />

<Format var="" description=""
```