***A Project Document of the***

***ATC Application Programming Interface Working Group***

ATC APIRI TEST PLAN v01.04

Test Plan for the

Advanced Transportation Controller (ATC) Application Programming Interface Reference Implementation (APIRI)

**July 14, 2016**

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

**For use by:** Siva Narla, Chief Engineer and ITS Standards Manager

Institute of Transportation Engineers

George Chen and Douglas Tarico, Co-Chairs

ATC API Working Group

Ralph W. Boaz, Project Manager and Systems Engineer

ATC API Reference Implementation Project

Members of the ATC API Working Group

Consulting Team for the ATC API RI Project

**Prepared by:** James Kinnard, Test Engineer

Adaptive Solutions, Inc.

Copyright 2015-2016 AASHTO/ITE/NEMA. All rights reserved.

**CHANGE HISTORY**

|  |  |
| --- | --- |
| **DATE** | **NOTE** |
| 10/7/15 | Initial Draft Test Plan and TDS v01.00 |
| 11/8/15 | Test Plan and TDS v01.01 |
| 12/1/15 | Test Plan and TDS v01.02 |
| 2/22/16 | Test Plan and TDS v01.03 (TRR) |
| 7/14/16 | Test Plan and TDS v01.04 (TRR2) |
|  |  |

**NOTICE**

**Joint NEMA, AASHTO and ITE Copyright and**

**Intelligent Transportation Systems (ITS) 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](#_Toc456255112)

**[1.1](#_Toc456255113)** [Purpose 5](#_Toc456255113)

**[1.2](#_Toc456255114)** [Background 5](#_Toc456255114)

**[1.3](#_Toc456255115)** [Scope 6](#_Toc456255115)

**[1.4](#_Toc456255116)** [Definitions, Acronyms and Abbreviations 7](#_Toc456255116)

**[1.5](#_Toc456255117)** [References 10](#_Toc456255117)

[2 TEST ITEMS 11](#_Toc456255118)

[3 FEATURES TO BE TESTED 11](#_Toc456255119)

[4 FEATURES NOT TO BE TESTED 12](#_Toc456255120)

[5 APPROACH 13](#_Toc456255121)

[6 ITEM PASS/FAIL CRITERIA 16](#_Toc456255122)

[7 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS 16](#_Toc456255123)

[8 TEST DELIVERABLES 16](#_Toc456255124)

[9 TESTING TASKS 16](#_Toc456255125)

[10 ENVIRONMENTAL NEEDS 17](#_Toc456255126)

[11 RESPONSIBILITIES 17](#_Toc456255127)

[12 STAFFING AND TRAINING NEEDS 17](#_Toc456255128)

[13 SCHEDULE 18](#_Toc456255129)

[14 RISKS AND CONTINGENCIES 18](#_Toc456255130)

[15 APPROVALS 18](#_Toc456255131)

[16 APPENDICES 19](#_Toc456255132)

**[16.1](#_Toc456255133)** [FPUI Library Requirements to Validation Description Matrix 19](#_Toc456255133)

**[16.2](#_Toc456255134)** [FIO Library Requirements to Validation Description Matrix 82](#_Toc456255134)

**[16.3](#_Toc456255135)** [TOD Library Requirements to Validation Description Matrix 126](#_Toc456255135)

**[16.4](#_Toc456255136)** [APIRI Test Design Specifications 127](#_Toc456255136)

# INTRODUCTION

This Test Plan is for the software referred to as the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation (APIRI). It has been developed as part of the “Reference Implementation of ATC 5401 Application Programming Interface (API) Standard Version 2” project funded by the USDOT Contract Number DTFH61-11-D-00052, Work Order T-13003 (referred to as the APIRI project).

## Purpose

The Advanced Transportation Controller (ATC) Standards are intended to provide an open architecture hardware and software platform that can support a wide variety of Intelligent Transportation Systems (ITS) applications including traffic management, safety, security and other applications. The ATC standards are being developed and maintained under the direction of the ATC Joint Committee (JC) which is made up of representatives from the American Association of State Highway and Transportation Officials (AASHTO), the Institute of Transportation Engineers (ITE) and the National Electrical Manufacturers Association (NEMA).

This document defines a test plan to test the Reference Implementation of the Advanced Transportation Controller (ATC) Application Programming Interface (API) Standard. It establishes a common understanding of the testing activities, identifies the detailed testing tasks, defines the testing environment and describes the testing tools required for:

a) The local, state and federal transportation agencies who specify ATC equipment;

b) The software developers, consultants and manufacturers who develop application programs for ATC equipment; and

c) The public who benefit in the application programs that run on ATC equipment and directly or indirectly pays for these products.

## Background

The ATC Controller Standard specifies a controller architecture where the computational components reside on a printed circuit board (PCB), called the “Engine Board,” with standardized connectors and pinout. It includes a central processing unit (CPU), a Linux operating system (O/S), memory, external and internal interfaces and other associated hardware necessary to create an embedded transportation computing platform. The Engine Board plugs into a “Host Module” that supplies power and physical connection to the I/O devices of the controller. While the interface to the Engine Board is completely specified, the Host Module may be of various shapes and sizes to accommodate controllers of various designs. Only minimum levels of performance are specified in the ATC Controller Standard. The CPU of an Engine Board may come from any manufacturer allowing future products to have higher performance processors and still be compliant to the standard.

The ATC Application Programming Interface (API) Standard defines a software interface which resides on the Engine Board. This interface allows application programs to be written so that they may run on any ATC controller unit regardless of the manufacturer. It also defines a software environment that allows multiple application programs to be interoperable on a single controller unit by sharing the fixed resources of the controller. Software developed in compliance with the API Standard is known as the API Software.

Using the ATC Controller and API Standards together enables future advances in processing power to be applied to deployed ATC controllers while retaining the ability to operate the software applications of the existing transportation system. The API Standard provides for application software portability at the source code level. The application software source code may need to be recompiled to operate on different Engine Boards. This provides design freedom for the Engine Board manufacturers and allows Engine Boards to evolve and incorporate new technologies over time.

Figure 1 illustrates the organization and layered architecture of the ATC software. The “Linux O/S and Device Drivers” reflects a specification of the Linux operating system defined in the ATC Board Support Package (BSP) (see ATC Controller Standard, Section 2.2.5, Annex A and Annex B). 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 the specification of the device drivers necessary for the Linux O/S to operate on the ATC hardware. “API” refers to the software to be tested under this test plan. As shown in Figure 1, both users and application programs use the API to interface to ATC controller units.



Figure . Layered organization of ATC software

## Scope

This document defines a test plan for the APIRI Software. It provides an agreed upon level of confidence in the software under test. It identifies the items being tested, the features to be tested, the overall approach of the testing activities, the testing tasks to be performed, the qualifications of the personnel required for each task, and the risks associated with this plan. Since this plan is not specific to a particular project, it does not include a particular test schedule.

## Definitions, Acronyms and Abbreviations

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

## References

Institute of Electrical and Electronics Engineers, IEEE Std 829-1998, IEEE Standard for Software Test Documentation. IEEE, 1998.

[http://standards.ieee.org/index.html](http://standards.ieee.org/index.html%20)

Institute of Transportation Engineers, API Reference Implementation Project Open Source Software (OSS) Concept Paper. Institute of Transportation Engineers, 12 June 2014.

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

Institute of Transportation Engineers, API Validation Suite APIVSXML Specification v02.00. ATC Joint Committee, 31 December 2010.

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

Institute of Transportation Engineers, Advanced Transportation Controller (ATC) Application Programming Interface (API) Validation Suite (APIVS) Concept of Operations (ConOps) v02.04. ATC Joint Committee, 20 November 2014.

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

Institute of Transportation Engineers, Advanced Transportation Controller (ATC) Application Programming Interface (API) Validation Suite (APIVS) Software Requirements Specification (SRS) v02.03. ATC Joint Committee, 20 November 2014.

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

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

Institute of Transportation Engineers, ATC APIRI PMP v01.03 Project Management Plan (PMP) for the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation Project. ATC Joint Committee, 5 November 2014.

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

Institute of Transportation Engineers, Intelligent Transportation System (ITS) Standard Specification for Roadside Cabinets v01.02.17b. ATC Joint Committee, 16 November 2006.

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

Institute of Transportation Engineers, User Comment Draft ATC 5201 Advanced Transportation Controller (ATC) Standard Version 06.10. ATC Joint Committee, 30 July 2012.

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

International Organization for Standardization, ISO/IEC 9899:2011 Programming Language C. ISO, 8 December 2011.

National Electrical Manufacturers Association, NEMA Standards Publication TS 2-2003 v02.06 Traffic Controller Assemblies with NTCIP Requirements. NEMA, 2003.

# TEST ITEMS

The test items covered by this test plan are the API Front Panel User Interface (FPUI) library, the API Field I/O (FIO) library and the API Time of Day (TOD) library as defined by requirements and function specifications in Sections 3 and 4, respectively, of the API 5401 Standard. Software adhering to these requirements and function specifications is to be operational on an ATC controller unit.

# FEATURES TO BE TESTED

This specific features and combination of features to be tested under this test plan are defined by the test design specifications (TDS), test case specifications (TCS) and test procedure specifications (TPS) listed in Table 1.

**Table 1. APIRI Test Specifics**

|  |  |  |
| --- | --- | --- |
| **Test ID** | **Document Name** | **Brief Description** |
| APIRI.TDS.2001 | APIRI Test Design Specification 1 | Test All APIRI FPUI Required Features |
| APIRI.TDS.3001 | APIRI Test Design Specification 2 | Test All APIRI FIO Required Features |
| APIRI.TDS.4001 | APIRI Test Design Specification 3 | Test All APIRI TOD Required Features |
| APIRI.TCS.2010 | APIRI Test Case Specification 1 | FPUI Text UI Virtual Displays |
| APIRI.TCS.2020 | APIRI Test Case Specification 2 | FPUI Front Panel Manager |
| APIRI.TCS.2030 | APIRI Test Case Specification 3 | FPUI Character Set and Screen Attributes |
| APIRI.TCS.2040 | APIRI Test Case Specification 4 | FPUI Reading and Writing Data |
| APIRI.TCS.2050 | APIRI Test Case Specification 5 | FPUI Special Characters |
| APIRI.TCS.2070 | APIRI Test Case Specification 6 | FPUI Key Mapping |
| APIRI.TCS.2080 | APIRI Test Case Specification 7 | FPUI Asynchronous Notification and Focus |
| APIRI.TCS.2090 | APIRI Test Case Specification 8 | FPUI Raw Data Handling |
| APIRI.TCS.2100 | APIRI Test Case Specification 9 | API Version Information (All Libraries) |
| APIRI.TCS.3010 | APIRI Test Case Specification 10 | General FIO Operations |
| APIRI.TCS.3020 | APIRI Test Case Specification 11 | FIO Inputs and Outputs |
| APIRI.TCS.3030 | APIRI Test Case Specification 12 | FIO Channel Mapping |
| APIRI.TCS.3040 | APIRI Test Case Specification 13 | FIO Filtered Inputs and Transition Buffering |
| APIRI.TCS.3050 | APIRI Test Case Specification 14 | FIO Frame Frequency |
| APIRI.TCS.3060 | APIRI Test Case Specification 15 | FIO Failed State and Fault Monitoring |
| APIRI.TCS.3070 | APIRI Test Case Specification 16 | FIO Watchdog Outputs |
| APIRI.TCS.3080 | APIRI Test Case Specification 17 | FIO Device Status |
| APIRI.TCS.3090 | APIRI Test Case Specification 18 | FIO Health Monitor |
| APIRI.TCS.3100 | APIRI Test Case Specification 19 | FIO CMU Configuration |
| APIRI.TCS.3110 | APIRI Test Case Specification 20 | FIO Module Status |
| APIRI.TCS.3120 | APIRI Test Case Specification 21 | FIO Asynchronous Notification |
| APIRI.TCS.3130 | APIRI Test Case Specification 22 | FIO Dark Channel Mapping |
| APIRI.TCS.4010 | APIRI Test Case Specification 23 | TOD Time Handling Functions |
| APIRI.TCS.6010 | APIRI Test Case Specification 24 | FPM and ATC Configuration Menu |
| APIRI.TCS.6020 | APIRI Test Case Specification 25 | System Configuration Utilities |
| APIRI.TCS.6030 | APIRI Test Case Specification 26 | Intrinsic API Requirements |
| APIRI.TCS.6040 | APIRI Test Case Specification 27 | FIO Serial Ports and Status Counters |
| APIRI.TCS.7010 | APIRI Test Case Specification 28 | FPUI Display Presence and Size |
| APIRI.TCS.7020 | APIRI Test Case Specification 29 | FPUI Bell Activation and App Termination |
| APIRI.TCS.7030 | APIRI Test Case Specification 30 | Test FPUI Display Graphics |
| APIRI.TCS.7040 | APIRI Test Case Specification 31 | FPUI Display Focus |
| APIRI.TCS.7050 | APIRI Test Case Specification 32 | System Configuration Menu Display |
| APIRI.TPS.1001 | APIRI Test Procedure Specification 1 | Auto-Execute Selected APIRI Script(s) |
| APIRI.TPS.6010 | APIRI Test Procedure Specification 2 | FPM and ATC Configuration Menu |
| APIRI.TPS.6020 | APIRI Test Procedure Specification 3 | System Configuration Utilities |
| APIRI.TPS.6030 | APIRI Test Procedure Specification 4 | Intrinsic API Requirements |
| APIRI.TPS.6040 | APIRI Test Procedure Specification 5 | FIO Serial Ports and Status Counters |
| APIRI.TPS.7010 | APIRI Test Procedure Specification 6 | FPUI Display Presence and Size |
| APIRI.TPS.7020 | APIRI Test Procedure Specification 7 | FPUI Bell Activation and App Termination |
| APIRI.TPS.7030 | APIRI Test Procedure Specification 8 | FPUI Display Graphics |
| APIRI.TPS.7040 | APIRI Test Procedure Specification 9 | FPUI Display Focus |
| APIRI.TPS.7050 | APIRI Test Procedure Specification 10 | System Configuration Menu Display |

Appendices 16.1, 16.2 and 16.3 contain Requirements to Validation Description Matricies for the FPUI, FIO and TOD libraries respectively. Listed in these matrices are the requirements and associated functions from the API Standard and the test identifiers from Table 1 above which represent tests for those requirements and functions.

# FEATURES NOT TO BE TESTED

As standards development and maintenance are continuous processes, there may be some features added to the API Standard after the development of this test plan. Any features added to versions of the API Standard that are not listed in the reference section of this document may not be tested as part of this test plan.

# APPROACH

An additional component of the ATC 5401 API Reference Implementation Project has been the development of specific test software for testing the various API library functions and other features. This software, referred to as the API Validation Suite (APIVS), consists of an executable application program – the Validation Suite Engine (VSE) - which is run on behalf of test cases which individually test specific elements of the API libraries and which collectively test most features of the APIRI software.

These test cases, identified in Table 1 and in Appendix 16, contain script files written in XML format which are parsed and interpreted by the VSE, invoking the API software tests, and validating results. The detailed structure of all of these testing components can be seen below in Figure 2.

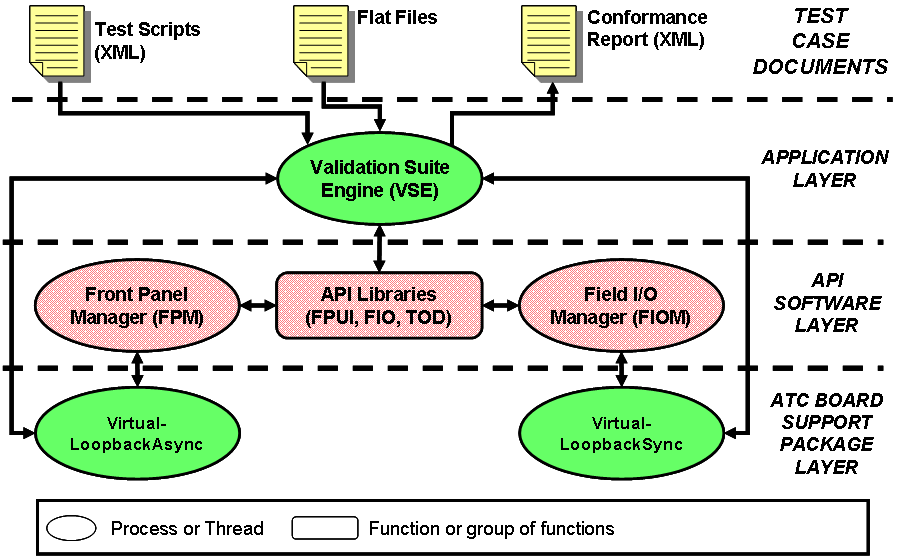


Figure . APIVS Software Architecture

In order to automate this testing, a capability needs to exist which brings the outputs of the functioning API software back to the inputs so that proper operation can be verified. This “loopback” capability could be done with some limitations using physical loopback cables connected to the various external serial inputs and outputs of the ATC unit. The design of the APIVS software incorporates a virtual loopback capability where the loopback occurs internal to the ATC unit. This includes a set of asynchronous and synchronous virtual loopback drivers for the Linux kernel which replace like drivers on the ATC Engine Board for the purposes of testing. This allows the testing to be more comprehensive and facilitate the testing for the test personnel. The layered software environment for the APIVS software is similar to the layered organization of the ATC software (see Figure 2).



Figure . Layered organization of APIVS software

The extensive combinations of functions that are possible using the API Software make exhaustive testing of the API libraries impractical. However, a set of practical tests can and should still be performed as follows:

1. Unit Testing. This testing will focus on completeness of the API Software with respect to the API Standard function call specifications. It will validate the presence and arguments for each API function.
2. Integration Testing. It is impractical to validate the operation of the API Software functions without the use of other API Software functions. Consequently, integrated testing involving groups of related API Software functions is required. Each API function shall be a part of at least one integrated test. The focus of integration testing will be on correctness of the API Software.
3. System Testing. This testing will focus on loading and stressing the API Software. This will include multiple test application programs running concurrently on the ATC controller unit. The focus of this testing will be on robustness of the API Software.

The API FPUI library functions are listed in groups below that may be used to form integration tests.

* General Functions – fpui\_apiver, fpui\_open, fpui\_close, fpui\_get\_window\_size, fpui\_get\_focus, fpui\_clear, fpui\_refresh and fpui\_set\_emergency.
* Attribute Functions – fpui\_set\_window\_attr, fpui\_get\_window\_attr, fpui\_set\_character\_blink, fpui\_get\_character\_blink, fpui\_set\_backlight, fpui\_get\_backlight, fpui\_set\_backlight\_timeout, fpui\_set\_cursor\_blink, fpui\_get\_cursor\_blink, fpui\_set\_reverse\_video, fpui\_get\_reverse\_video, fpui\_set\_underline, fpui\_get\_underline, fpui\_set\_auto\_wrap, fpui\_get\_auto\_wrap, fpui\_set\_auto\_repeat, fpui\_get\_auto\_repeat, fpui\_set\_cursor, fpui\_get\_cursor, fpui\_set\_auto\_scroll, fpui\_get\_auto\_scroll and fpui\_reset\_all\_attributes.
* Read Functions – fpui\_poll, fpui\_read, fpui\_read\_char and fpui\_read\_string.
* Write Functions – fpui\_write, fpui\_write\_char, fpui\_write\_string, fpui\_write\_at, fpui\_write\_char\_at and fpui\_write\_string\_at.
* Cursor Functions – fpui\_get\_cursor\_pos, fpui\_set\_cursor\_pos, fpui\_home, fpui\_set\_tab and fpui\_clear\_tab.
* Special Character Functions – fpui\_compose\_special\_char and fpui\_display\_special\_char.
* LED Functions – fpui\_set\_led and fpui\_get\_led.
* Aux Switch Functions – fpui\_open\_aux\_switch, fpui\_close\_aux\_switch and fpui\_read\_aux\_switch.
* Key Mapping Functions – fpui\_set\_keymap, fpui\_get\_keymap, fpui\_del\_keymap and fpui\_reset\_keymap.

The API FIO library functions are listed in groups below that may be used to form integration tests.

* General Functions – fio\_register, fio\_deregister, fio\_fiod\_register, fio\_fiod\_deregister, fio\_fiod\_enable, fio\_fiod\_disable, fio\_query\_fiod, fio\_fiod\_status\_get, fio\_fiod\_status\_reset and fio\_apiver.
* Input Configuration Functions – fio\_fiod\_inputs\_get, fio\_fiod\_inputs\_filter\_set and fio\_fiod\_inputs\_filter\_get.
* Output Configuration Functions – fio\_fiod\_outputs\_set, fio\_fiod\_outputs\_get, fio\_fiod\_outputs\_reservation\_set and fio\_fiod\_outputs\_reservation\_get.
* Frame Functions – fio\_fiod\_frame\_schedule\_set, fio\_fiod\_frame\_schedule\_get, fio\_fiod\_frame\_size, fio\_fiod\_frame\_read, fio\_fiod\_frame\_notify\_register, fio\_fiod\_frame\_notify\_deregister and fio\_query\_frame\_notify\_status.
* Transition Buffer Functions – fio\_fiod\_inputs\_trans\_set, fio\_fiod\_inputs\_trans\_get and fio\_fiod\_inputs\_trans\_read.
* Watchdog/Health Monitor Functions – fio\_fiod\_wd\_register, fio\_fiod\_wd\_deregister, fio\_fiod\_wd\_reservation\_set, fio\_fiod\_wd\_reservation\_get, fio\_fiod\_wd\_heartbeat, fio\_hm\_register, fio\_hm\_deregister, fio\_hm\_heartbeat and fio\_hm\_fault\_reset.
* Fault/Volt Monitor Functions – fio\_fiod\_ts\_fault\_monitor\_set, fio\_fiod\_ts\_fault\_monitor\_get, fio\_fiod\_ts1\_volt\_monitor\_set and fio\_fiod\_ts1\_volt\_monitor\_get.
* CMU/MMU/Channel Functions – fio\_fiod\_cmu\_fault\_set, fio\_fiod\_cmu\_fault\_get, fio\_fiod\_cmu\_dark\_channel\_set, fio\_fiod\_cmu\_dark\_channel\_get, fio\_fiod\_mmu\_flash\_bit\_set, fio\_fiod\_mmu\_flash\_bit\_get, fio\_fiod\_channel\_reservation\_set, fio\_fiod\_channel\_reservation\_get, fio\_fiod\_channel\_map\_set, fio\_fiod\_channel\_map\_count and fio\_fiod\_channel\_map\_get.

The API TOD library functions are listed in groups below that may be used to form integration tests.

* Set/Get Time Functions – tod\_set and tod\_get.
* Daylight Saving Time (DST) Functions – tod\_set\_dst\_state, tod\_get\_dst\_state, tod\_get\_dst\_info, and tod\_set\_dst\_info.
* Time Source and Signaling Functions – tod\_get\_timesrc, tod\_set\_timesrc, tod\_get\_timesrc\_freq, tod\_request\_tick\_signal, tod\_cancel\_tick\_signal, tod\_request\_onchange\_signal, and tod\_cancel\_onchange\_signal.

# ITEM PASS/FAIL CRITERIA

The test items as specified in Section 2 will be tested according to the TDSs, TCSs and TPSs identified in Section 3. If any of the tests for a test item fail, the test item is considered to have failed. If all of the tests for a test item pass, the test item is considered to have passed.

# SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

Suspension criteria and resumption requirements are subject to the organization exercising this test plan and its associated TDSs, TCSs and TPSs.

# TEST DELIVERABLES

It is recommended that agencies adopt this test plan as part of their acceptance test of ATC controller units. Test deliverables are identified in specific TDS, TCS or TPSs. It is requested that users of this test plan provide feedback to the ATC Program Manager and the API WG. Any suggested changes or enhancements to this test plan and the API Standard are welcomed. A Test Summary Report as defined by IEEE Std 829-1998 should be completed and sent to the chairpersons of the API Working Group via the ATC Program Manager of ITE. Unless expressly required by a TDS, TCS or TPS, other test deliverables including Test Item Transmittal Reports, Test Logs and Test Incident Reports, and/or test equipment are subject to specification by the organization exercising this test plan (see outlines for these documents in IEEE Std 829-1998).

# TESTING TASKS

The tasks necessary to prepare for and to perform testing are as follows:

1. Port APIRI and APIVS software packages to ATC Controller platform;

2. Gather and prepare test equipment and test software;

3. Perform testing per TDS, TCS and TPS; and

4. Deliver Test Summary Report to ATC Program Manager (optional)

Task #1 requires the skills of a software engineer. For more information regarding obtaining the source code and building the APIRI and APIVS software components, please refer to the APIRI User Manual and APIVS User Manual references in Section 1.3.

Tasks #2 and #3 may be performed by a technician but some tests may require understanding of traffic operation. Task #4 will likely require some formatting and editing of any automated reports.

# ENVIRONMENTAL NEEDS

The test environment shown in Figure 4 is to be used to execute this test plan. It consists of an ATC Controller, a USB flash drive and a personal computer (PC). This environment allows the Validation Suite software to run on the Engine Board and to test the APIRI Software. Both the PC and the controller are required to have an available USB port. The ATC is required to have a minimum 8x40 character LCD display and associated keyboard.

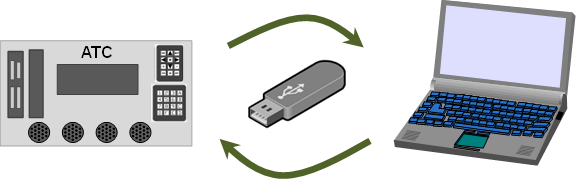


Figure . API Test Environment

The PC interface is necessary to load test software, configuration files and test scripts onto the USB flash drive prior to testing and to extract and examine conformance reports from the drive after testing. It is possible that the PC may also serve directly in the operation of some tests and both the controller and PC may need to have an available Ethernet connection for the purpose of making a console connection. Details of the operation of the test environment and tests can be found in the individual TDS, TCS and TPSs listed in Table 1.

Since Engine Boards may have been implemented using a variety of processors, software identified in this test plan will need to be compiled in a manner appropriate for the implementation of the Engine Board in the ATC Controller.

# RESPONSIBILITIES

It is the responsibility of the API WG to maintain this test plan and the TDSs, TCSs and TPSs identified in Section 3. It is the responsibility of the organization exercising this test plan to provide the test environment and test personnel to execute the software identified in the TDSs, TCSs and TPSs for the controller under test. It is requested that any organization exercising this test plan to notify the API WG via the ATC Program Manger of ITE of any corrections or suggested modifications to the API Standard, this test plan, TDSs, TCSs, TPSs and test software found during testing.

# STAFFING AND TRAINING NEEDS

Staffing and training needs are subject to the organization exercising this test plan.

# SCHEDULE

There is no fixed schedule for this test plan. It is designed to be applied whenever an organization determines it is necessary.

# RISKS AND CONTINGENCIES

As stated in Section 5, exhaustive testing of the APIRI Software is impractical. This means that there may be combinations of features that are not tested by this test plan. If anomalies are found in an implementation of API software that this test plan did not cover, the API WG should be contacted through the ATC Program Manager so that an appropriate test can be added to the next version of this test plan.

If this test plan is not employed, there is a high potential for ATC program risks including:

* API library vendors may make unsubstantiated claims of conformance
* There will be less confidence in the deployment of API Standard
* There will be lost time and money to the industry
* There will be slower deployment
* Some of the goals of the ATC program will not be achieved

# APPROVALS

This test plan has been developed by the ATC API WG to facilitate the deployment of the API Standard. The official approvals required are still to be determined.

# APPENDICES

## FPUI Library Requirements to Validation Description Matrix

The following table shows the relationship between the requirements and associated FPUI library functions of the API Standard and the test identifiers which represent the test documents for those requirements and functions.

| **Req ID** | **Req Description** | **ATC API Function** | **APIRI SDD Design Narrative** | **Test Cases** | **Test Procedures** |
| --- | --- | --- | --- | --- | --- |
| APIR3.1.1[1] | The API shall provide a text-based user interface capability to allow application programs running concurrently on an ATC controller unit to share the controller’s Front Panel display. | fpui\_open(3fpui) | The fpui\_open library call and the system described in Section 3.1 provide for up to 16 concurrent application programs to share access to the ATC unit’s front panel display. The FrontPanelDriver general interface (Section 3.3.1) allows multiple application connections to the system. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1[2] | The API shall provide up to 16 virtual display screens (referred to as “windows”) that can be used by application programs as their user interface display. | fpui\_open(3fpui) | The system described in Section 3.1 includes a VirtualTerminal handler module (Section 3.4.2) with storage for up to 16 application virtual displays. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1[3] | The display size of the windows shall be equal to the physical display size (lines x characters) of the controller’s Front Panel display (if one exists). | fpui\_open(3fpui) | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which attempts to obtain the dimensions of the physical display on startup or on detection of a display reconnection. The VirtualTerminal handler module (Section 3.4.2) creates virtual displays sized according to the dimensions of the physical display if attached, or 8x40 if not. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1[4] | The display size of the windows shall have a minimum size of 4 lines x 40 characters and a maximum size of 24 lines x 80 characters. | fpui\_open(3fpui) | The application virtual display storage of the VirtualTerminal handler module (Section 3.4.2) is sized to represent a maximum display size of 24 rows of 80 columns. | APIRI.TCS.7010 | APIRI.TPS.7010 |
| APIR3.1.1[5] | If no physical display exists, the API shall operate as if it has a display with a size of 8 lines x 40 characters. | fpui\_open(3fpui) | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which attempts to obtain the dimensions of the physical display on startup or on detection of a display reconnection. The VirtualTerminal handler module (Section 3.4.2) creates virtual displays sized according to the dimensions of the physical display if attached, or 8x40 if not. | APIRI.TCS.7010 | APIRI.TPS.7010 |
| APIR3.1.1[6] | Only one window shall be displayed at a time on the Front Panel display. | fpui\_open(3fpui) | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which keeps track of which virtual display shall have focus, and thus be the only window displayed to the physical display. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1[7] | When a window is displayed, the API shall display the character representation of the window on the Front Panel display (if one exists). | fpui\_open(3fpui) | The system described in Section 3.1 includes a VirtualTerminal handler (Section 3.4.2) which maintains the character representation for each window (virtual display). When a window is selected for display (focused) the virtual display is also copied to the physical display. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1[8] | The application program associated with the window displayed shall receive the characters input from the Front Panel input device (Ex. keyboard or keypad). | fpui\_open(3fpui) | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which routes the characters received on the Front Panel input device to the application whose window has focus via the Routing process (Section 3.4.1) and the FrontPanelDriver interface (Section 3.3.1) held open by the application. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1[9] | The API shall support the display character set as defined in the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | fpui\_open(3fpui) | The system described in Section 3.1 includes a VirtualTerminal handler (Section 3.4.2) whose display character set matches that of the latest ATC 5201 Standard. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1[10] | Screen attributes described by the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4), shall be maintained for each window independently. | fpui\_open(3fpui) | The system described in Section 3.1 includes a VirtualTerminal handler (Section 3.4.2) which maintains a separate set of screen attributes relating to each virtual terminal. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1[11] | Each window shall have separate input and output buffers unique from other windows. | fpui\_open(3fpui) | The system described in Section 3.1 maintains separate and unique input and output buffers by use of the FrontPanelDriver general interface (Section 3.3.1) allowing up to 16 separate application connections. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1[12] | The screen attributes of the Front Panel Manager Window shall be set to the values of the controller unit at power up as described in ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | fpui\_open(3fpui) | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) to maintain the Front Panel Manager Window and whose virtual terminal attribute defaults are set to the controller unit power up defaults as defined in the ATC 5201 Standard. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1[13] | The API shall provide an additional window referred to as the "ATC Configuration Window" for use by system configuration utility programs. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a system configuration window and support system configuration utility programs. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.1.1[14] | The ATC Configuration Window shall not be assignable to application programs. | fpui\_open\_config\_window(3fpui) | The FrontPanelDriver design element provides a reserved System Configuration Manager Interface (Section 3.3.4) for use by the SystemConfiguration manager process and separate from the General Application Interface (Section 3.3.1) used by application programs. | Requirement APIR3.1.1[14] was deemed unnecessary and removed from the API Standard by vote of the API Working Group at the Houston meeting in March, 2016. | N/A |
| APIR3.1.1.1[1] | The API shall provide a window selection screen called the Front Panel Manager Window from which Operational Users may select a window to have focus. | fpui\_open(3fpui) | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) to maintain the Front Panel Manager Window. | APIRI.TCS.2020  APIRI.TCS.6010 | APIRI.TPS.1001  APIRI.TPS.6010 |
| APIR3.1.1.1[2] | Application names associated with each window shall be listed. | fpui\_open(3fpui) | The MasterSelection manager process (Section 3.4.6) displays the list of registered application names on the Front Panel Manager Window. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[3] | The application names shall be limited to 16 characters. | fpui\_open(3fpui) | The implementation of the fpui\_open() library call (Section 3.4.8) parses the registered application name parameter to ensure it is valid including being of valid length. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[4] | If there is no application program associated with a window, the window number shall be listed with a blank application name. | fpui\_open(3fpui) | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which maintains a list of all registered applications, and displays blank application name fields for windows which have no registered application. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[5] | The default Front Panel Manager Window size shall be 8 lines x 40 characters with the format as shown in Figure 7. | fpui\_open(3fpui) | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the Front Panel Manager window in the required default size and format. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[6] | If the Operational User has not set the default window, the Front Panel Manager Window shall be the default window. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the Front Panel Manager window if a default application window has not been set. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[7] | The default window shall be settable by the Operational User from the Front Panel Manager Window by pressing {\*,[0-F],<ENT>}. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which handles the key sequence to set the corresponding application window as the default. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[8] | The Operational User shall be capable of setting the default window to the Front Panel Manager Window by pressing {\*,<ENT>} from the Front Panel Manager Window. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which handles the key sequence to set the Front Panel Manager window as the default. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[9] | The default window shall be designated by a star “\*” character next to the window number. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the list of registered application names and indicates the default application with the “\*” character, if set. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[10] | The Operational User shall be able to put the Front Panel Manager Window in focus by pressing {\*\*,<ESC>} from the keypad on the controller’s Front Panel regardless of the application program in operation. | N/A | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which listens for the special key sequence and sets focus to the MasterSelection manager process (Section 3.4.6) which displays and maintains the Front Panel Manager window. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[11] | The Operational User shall be able to enter {\*\*} by pressing an asterisk (\*) twice within a 1.0 second time period. | N/A | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which listens for any special key sequence and times special key sequence intervals. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[12] | If the {\*\*} sequence is not completed within the 1.0 second time period or if the {\*\*} sequence is not followed by <ESC> character within a 1.0 second time period, then the characters shall be interpreted as individual “\*” characters. | N/A | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which listens for any special key sequence and times special key intervals to determine when to enact special functions and when to pass characters transparently. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[13] | The Operational User shall have the capability to put a window in focus that is assigned to an application program by pressing {[0-F]} from the Front Panel Manager Window. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which switches focus to the corresponding application window on detection of a key press in the required range. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[14] | The only possible window selections for focus from the Front Panel Manager Window shall be itself, the ATC Configuration Window, or a window assigned to an application program. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which is capable of switching focus to the system configuration window or a selected application window. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[15] | If the Front Panel Manager Window is the default window, no asterisk shall be displayed next to any application name in the Front Panel Manager Window. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the list of registered applications with no “\*” character if no application window is set as the default. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[16] | The Operational User shall be able to put the ATC Configuration Window in focus by pressing {<NEXT>} in the Front Panel Manager Window. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which responds to the {<NEXT>} key by switching focus to the System Configuration window. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.1.1.1[17] | The top two lines and bottom line of the Front Panel Manager Window shall be fixed as shown in Figure 7. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the Front Panel Manager window with the required top two lines and bottom line. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.1.1.1[18] | The number of lines between the second line and bottom lines used for displaying window names shall vary according to the size of the ATC display. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the Front Panel Manager window with the appropriate number of lines according to the current size of the ATC display determined at startup and on notice of a display reconnection. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.1.1.1[19] | The Operational User shall be able to scroll up and down the names of the windows in the Front Panel Manager Window one line at a time using the up and down arrow keys of the controller keypad. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the Front Panel Manager window and adjusts the displayed list of application window names in response to up and down arrow key presses. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[20] | The Operational User shall have the capability to put a window in focus that is assigned to an application program by pressing {\*\*,[0-F]} from the keypad on the controller’s Front Panel regardless of the application program in operation. | N/A | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which listens for the special key sequences and sets focus to the application window corresponding to the key press value. | APIRI.TCS.2020 | APIRI.TPS.1001 |
| APIR3.1.1.1[21] | If the Operational User attempts to put a window in focus that that does not have an application window assigned either by pressing {[0-F]} while the Front Panel Manager Window is in focus or {\*\*,[0-F]} while any other window is in focus, the API shall activate the bell of the controller unit (see Section 7.1.4 of the ATC Controller Standard) (ATC 5201 has this in Section 6.1.4) and the {[0-F]} or {\*\*,[0-F]} shall be ignored. | N/A | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which listens for the special key sequences and sets focus to the application window corresponding to the key press value. If no corresponding application is registered for the key press value, tested by a search for the instantiated virtual window, the bel character is sent to the front panel device. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.1.1.1[22] | If the user presses any key that is undefined in the context of the Front Panel Manager Window, the API shall activate the bell of the controller unit and the key shall be ignored. | N/A | The system described in Section 3.1 includes a MasterSelection manager process (Section 3.4.6) which displays the Front Panel Manager window and receives key press input via the ViewportControl process while in focus. In response to key values which do not correspond to defined functions of the Front Panel Manager window, the bel character is sent to the front panel device. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.1.1.2[1] | The API shall provide a function to return the dimensions of a window in terms of number of lines and number of columns. | fpui\_get\_window\_size(3fpui) | The implementation of the fpui\_get\_window\_size() library call (Section 3.4.8) obtains the window dimensions by sending a “query panel type” sequence via the associated virtual window, the response to which reflects the physical front panel type and therefore dimensions. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[2] | The API shall provide a function to open a window and register a name for display on the Front Panel Manager Window. | fpui\_open(3fpui) | The implementation of the fpui\_open() library call (Section 3.4.8) allocates a virtual display via the VirtualTerminal handler (Section 3.4.2) and allows an application to register a name for display on the Front Panel Manager Window via the MasterSelection manager process (Section 3.4.6). | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[3] | An application program shall be able to open multiple windows providing the windows resources are available. | fpui\_open(3fpui) | The system described in Section 3.1 includes a VirtualTerminal handler (Section 3.4.2) that allows an application to register for one or more windows, following multiple calls to the fpui\_open() library function (Section 3.4.8). | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[4] | The API shall provide the ability for an application program to reserve exclusive access to the Aux Switch (see ATC Controller Standard, Section 7.1.4) (ATC 5201 has this in Section 6.1.4). | fpui\_open\_aux\_switch(3fpui), | The system described in Section 3.1 includes the AuxSwitch interface to the FrontPanelDriver (Section 3.3.5) which allows an application to reserve the aux switch for exclusive access via the fpui\_open\_aux\_switch() library function (Section 3.4.8). | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1.2[5] | An application program that has reserved exclusive access to the AUX Switch shall maintain exclusive access to the switch even if the application program has no window in focus. | N/A. | The AuxSwitch interface to the FrontPanelDriver (Section 3.3.5) makes use of a separate file context not associated with a virtual display, which may be used by an application independently from any virtual display association file context. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1.2[6] | The API shall provide a function to close a window and release the resource for other application programs. | fpui\_close(3fpui) | The fpui\_close() library call (Section 3.4.8) causes the VirtualTerminal handler (Section 3.4.2) to release the associated virtual display resources and the FrontPanelDriver (Section 3.3) to free the associated file descriptor slot for other application use. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[7] | The API shall provide a function or set of functions to set the attributes of a Front Panel display as described in the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | fpui\_set\_\* | The set of library functions named fpui\_set\_\*() (Section 3.4.8) allows the display attributes to be set via the functions of the VirtualTerminal handler (Section 3.4.2) which maintains a separate set of screen attributes relating to each virtual terminal, and reflects those attributes on the physical display when a virtual display has focus. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1.2[8] | The API shall provide a function or set of functions to return the attributes of a Front Panel display as described in the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | fpui\_get\_\* | The set of library functions named fpui\_get\_\*() (Section 3.4.8) returns the requested display attributes of a virtual window via the functions of the VirtualTerminal handler (Section 3.4.2) which maintains a separate set of screen attributes relating to each virtual terminal. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1.2[9] | The API shall provide a function that is used to determine if there is data in the input buffer of a window. | fpui\_poll(3fpui) | The implementation of the fpui\_poll() library function (Section 3.4.8) makes use of Linux operating system calls acting on the file descriptor parameter to obtain the input buffer status from the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[10] | The API shall provide a function to read a queued character or key code from the input buffer of a window. | fpui\_read\_char | The implementation of the fpui\_read\_char() library function (Section 3.4.8) makes use of the Linux operating system call to return a single character from the input buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[11] | The API shall provide a function to write a character to the current cursor position of a window. | fpui\_write\_char | The implementation of the fpui\_write\_char() library function (Section 3.4.8) makes use of the Linux operating system call to write a single character to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[12] | The API shall provide a function to write a character to a window at a position defined by column and line number. | fpui\_write\_char\_at | The implementation of the fpui\_write\_char\_at() library function (Section 3.4.8) makes use of the Linux operating system call to write the escape sequence for cursor position followed by the single character parameter to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[13] | The API shall provide a function to write a string to a window at the current cursor position. | fpui\_write\_string | The implementation of the fpui\_write\_string() library function (Section 3.4.8) makes use of the Linux operating system call to write the string parameter to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[14] | The API shall provide a function to write a string to a window at a starting position defined by column number and line number. | fpui\_write\_string\_at | The implementation of the fpui\_write\_string\_at() library function (Section 3.4.8) makes use of the Linux operating system call to write the escape sequence for cursor position followed by the string parameter to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[15] | The API shall provide a function to write a buffer of characters to a window at the current cursor position. | fpui\_write | The implementation of the fpui\_write() library function (Section 3.4.8) makes use of the Linux operating system call to write the character buffer parameter to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[16] | The API shall provide a function to write a buffer of characters to a window at a starting position defined by column number and line number. | fpui\_write\_at | The implementation of the fpui\_write\_at() library function (Section 3.4.8) makes use of the Linux operating system call to write the escape sequence for cursor position followed by the character buffer parameter to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[17] | The API shall provide a function to set the cursor position of a window defined by column and line number. | fpui\_set\_cursor\_pos | The implementation of the fpui\_set\_cursor\_pos() library function (Section 3.4.8) makes use of the Linux operating system call to write the escape sequence for cursor position to the output buffer of the FrontPanelDriver device interface (Section 3.3). | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[18] | The API shall provide a function to return the cursor position of the window defined by column and line number. | fpui\_get\_cursor\_pos | The implementation of the fpui\_get\_cursor\_pos() library function (Section 3.4.8) makes use of the Linux operating system calls to write the escape sequence for query cursor position followed by reading and interpreting the response escape sequence. | APIRI.TCS.2040 | APIRI.TPS.1001 |
| APIR3.1.1.2[19] | If a window was registered with access to the Aux Switch, the API shall provide a function to return its status. | fpui\_read\_aux\_switch | The implementation of the fpui\_read\_aux\_switch() library function (Section 3.4.8) makes use of the Linux operating system call to perform a read from the FrontPanelDriver AuxSwitch Interface (Section 3.3.5). | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1.2[20] | The API shall provide a function to compose special characters as described by the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | fpui\_compose\_special\_char | The implementation of the fpui\_compose\_special\_char() library function (Section 3.4.8) makes use of the Linux operating system call to write the escape sequence with the special character definition buffer to the FrontPanelDriver device interface (Section 3.3) which causes the VirtualTerminal handler (Section 3.4.2) to store the special character definition in the corresponding virtual display and reflect such definitions to the physical display when the virtual display has focus. | APIRI.TCS.2050 | APIRI.TPS.1001 |
| APIR3.1.1.2[21] | The API shall support the display of a composed character in the same manner as any other valid character. | fpui\_display\_special\_char | The implementation of the fpui\_display\_special\_char() library function (Section 3.4.8) makes use of the Linux operating system call to write the “display special character” escape sequence along with the special character index number to the FrontPanelDriver interface (Section 3.3) which causes the VirtualTerminal handler (Section 3.4.2) to update the corresponding virtual display character position with an encoded value indicating the special character index number. When the corresponding virtual display gains focus, the physical display is updated to show the indexed special character as previously defined. | APIRI.TCS.2050 | APIRI.TPS.1001 |
| APIR3.1.1.2[22] | The API shall provide a function to clear a window that operates on a window whether it is in or out of focus. | fpui\_clear | The VirtualTerminal handler (Section 3.4.2) includes functionality to handle the “clear screen” escape sequence and perform the operation on the virtual display regardless of focus. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[23] | The API shall provide a function to refresh a window that operates on a window whether it is in or out of focus. | fpui\_refresh | The VirtualTerminal handler (Section 3.4.2) includes functionality to refresh the physical display window from the virtual display both as required for changes to the virtual display and on request from the fpui\_refresh() library call (Section 3.4.8) while the virtual display has focus. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[24] | The bell of the controller’s Front Panel shall be activated only if a bell character, ^G (hex value 07), is sent from an application program which has a window that has focus. | N/A | The VirtualTerminal handler (Section 3.4.2) includes functionality to only forward a bell character, received from an application, to the physical display when the application virtual display has focus. | APIRI.TCS.7020 | APIRI.TPS.7020 |
| APIR3.1.1.2[25] | If a bell character is sent from an application program that does not have a window that has focus, the bell character shall be ignored by the API. | N/A | The VirtualTerminal handler (Section 3.4.2) includes functionality to ignore a bell character, received from an application, when the application virtual display does not have focus. | APIRI.TCS.7020 | APIRI.TPS.7020 |
| APIR3.1.1.2[26] | The API shall allow application programs to illuminate or extinguish the backlight of the ATC controller’s display if the command is received through a window that is in focus. | fpui\_set\_backlight(3fpui),  fpui\_get\_backlight(3fpui),  fpui\_set\_backlight\_timeout(3fpui) | The VirtualTerminal handler (Section 3.4.2) includes functionality to operate the front panel backlight according to application requests via the fpui\_set\_backlight() library call (Section 3.4.8) when the associated virtual display has focus. | APIRI.TCS.2010 | APIRI.TPS.1001 |
| APIR3.1.1.2[27] | Display configuration and inquiry command codes (escape sequences) specified in the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4), shall be supported as separate functions in the API. | fpui\_get\_attributes(3fpui),  fpui\_set\_character\_blink(3fpui),  fpui\_get\_character\_blink(3fpui),  fpui\_set\_backlight(3fpui),  fpui\_get\_backlight(3fpui),  fpui\_set\_cursor\_blink(3fpui),  fpui\_get\_cursor\_blink(3fpui),  fpui\_set\_reverse\_video(3fpui),  fpui\_get\_reverse\_video(3fpui),  fpui\_set\_underline(3fpui),  fpui\_get\_underline(3fpui),  fpui\_set\_auto\_wrap(3fpui),  fpui\_get\_auto\_wrap(3fpui),  fpui\_set\_auto\_repeat(3fpui),  fpui\_get\_auto\_repeat(3fpui),  fpui\_set\_cursor(3fpui),  fpui\_get\_cursor(3fpui),  fpui\_set\_auto\_scroll(3fpui),  fpui\_get\_auto\_scroll(3fpui),  fpui\_reset\_all(3fpui) | The VirtualTerminal handler (Section 3.4.2) includes functionality to store and maintain the full set of window attributes associated with an application virtual display, and to modify and return state of the virtual display attributes according to application settings via the fpui\_set\_\*() and fpui\_get\_\*() library calls (Section 3.4.8). The attributes are reflected in the attributes of the physical front panel when the virtual display has focus. | APIRI.TCS.2030 | APIRI.TPS.1001 |
| APIR3.1.1.2[28] | Application programs shall be able to interpret all ATC controller keys as individual key codes. | fpui\_set\_keymap(3fpui),  fpui\_get\_keymap(3fpui),  fpui\_del\_keymap(3fpui),  fpui\_keymap(3fpui) | The VirtualTerminal handler (Section 3.4.2) includes functionality to translate front panel key press escape sequences, each to a single character key code according to the default mapping of Table 1 of the ATC 5401 API Standard. The keycode mappings may be customized or disabled by application programs through use of the fpui\_\*\_keymap() library calls (Section 3.4.8). | APIRI.TCS.2070 | APIRI.TPS.1001 |
| APIR3.1.1.2[29] | The escape sequences representing keys that do not have standard ASCII character codes on an ATC controller shall be mapped to specific character codes in the API as shown in Table 1. | fpui\_set\_keymap(3fpui),  fpui\_get\_keymap(3fpui),  fpui\_del\_keymap(3fpui),  fpui\_keymap(3fpui) | The VirtualTerminal handler (Section 3.4.2) includes functionality to translate front panel key press escape sequences, each to a single character key code according to the default mapping of Table 1 of the ATC 5401 API Standard. The keycode mappings may be customized or disabled by application programs through use of the fpui\_\*\_keymap() library calls (Section 3.4.8). | APIRI.TCS.2070 | APIRI.TPS.1001 |
| APIR3.1.1.2[30] | The ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4), describes a graphics interface to the Front Panel’s display. The API shall support the operation of the graphics commands on a window only if that window is in focus. | fpui\_open(3fpui) | The VirtualTerminal handler (Section 3.4.2) forwards application escape sequences related to graphics mode programming to the physical front panel display when the application virtual display has focus, and otherwise ignores such escape sequences. | APIRI.TCS.7030 | APIRI.TPS.7030 |
| APIR3.1.1.2[31] | If application programs use graphics on a window, the API shall not redisplay these graphics when a window is refreshed or goes out/in focus. | fpui\_open(3fpui) | The VirtualTerminal handler (Section 3.4.2) forwards application escape sequences related to graphics mode programming to the physical front panel display when the application virtual display has focus, and otherwise ignores such escape sequences. | APIRI.TCS.7030 | APIRI.TPS.7030 |
| APIR3.1.1.2[32] | The API shall provide an asynchronous notification to alert programs when their associated windows go in and out of focus. | The Linux SIGWINCH signal is provided. Application level handler functions can assess changes. | The ViewportControl process (Section 3.4.3) which keeps track of which virtual display has focus and enacts focus change, also sends notification, via each open application FrontPanelDriver interface (Section 3.3), in the form of the Linux SIGWINCH signal. | APIRI.TCS.2080 | APIRI.TPS.1001 |
| APIR3.1.1.2[33] | The API shall provide a function which application programs may use to determine if their window is in focus. | fpui\_get\_focus(3fpui) | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which keeps track of which virtual display shall have focus and allows a given application to query its state via the fpui\_get\_focus() library call (Section 3.4.8). | APIRI.TCS.2080 | APIRI.TPS.1001 |
| APIR3.1.1.2[34] | The API shall provide a method to allow application programs to indicate that a window desires focus from the Operational User. | fpui\_set\_emergency(3fpui) | The fpui\_set\_emergency() library call implementation (Section 3.4.8) allows an application to indicate on the physical front panel that it requires focus be switched to its virtual display, or to cancel such a request. | APIRI.TCS.7040 | APIRI.TPS.7040 |
| APIR3.1.1.2[35] | This method shall cause the Front Panel backlight to flash and the window name in the Front Panel Manager Window to blink. | fpui\_set\_emergency(3fpui) | The fpui\_set\_emergency() library call implementation (Section 3.4.8) makes a ioctl() call via the FrontPanelDriver interface (Section 3.3) which causes the ViewportControl process (Section 3.4.3) to flash the backlight of the physical front panel display, and the MasterSelection process (Section 3.4.6) to blink the name of the associated application on the Front Panel Manager window, if focused. | APIRI.TCS.7040 | APIRI.TPS.7040 |
| APIR3.1.1.2[36] | The window name blinking shall cease once the indicated window receives focus. | fpui\_set\_emergency(3fpui) | The MasterSelection process (Section 3.4.6) ceases to blink the name of an application once it is selected for focus. | APIRI.TCS.7040 | APIRI.TPS.7040 |
| APIR3.1.1.2[37] | The backlight flashing shall cease when all windows requesting focus have been given focus. | fpui\_set\_emergency(3fpui) | The ViewportControl process (Section 3.4.3) which keeps track of which virtual display has focus and enacts focus change, ceases to flash the backlight once the virtual window of the last application requesting focus, gains focus. | APIRI.TCS.7040 | APIRI.TPS.7040 |
| APIR3.1.1.2[38] | The API shall provide a mechanism to allow application programs to detect the presence or absence of a Front Panel. | fpui\_get\_window\_size(3fpui) | The system described in Section 3.1 includes a ViewportControl process (Section 3.4.3) which attempts to obtain the dimensions of the physical display on startup or on detection of a display reconnection. If no physical display is detected, the dimensions are set to 8 rows x 40 columns. | APIRI.TCS.7010 | APIRI.TPS.7010 |
| APIR3.1.1.2[39] | The API shall recognize the presence or absence of a Front Panel in 5 seconds. | N/A | The ViewportControl process (Section 3.4.3) repeatedly polls the front panel device by sending a “get cursor position” escape sequence at a 4 second interval, the response to which, or lack thereof, allows the recognition of the presence or absence of a physical front panel. | APIRI.TCS.7010 | APIRI.TPS.7010 |
| APIR3.1.1.2[40] | The API shall provide an asynchronous notification to alert application programs of a change in the presence or absence of a Front Panel. | The Linux SIGWINCH signal is provided. Application level handler functions can assess changes. | The ViewportControl process (Section 3.4.3) sends notification of a change in the presence or absence of a physical front panel, via each open application FrontPanelDriver interface (Section 3.3), in the form of the Linux SIGWINCH signal. | APIRI.TCS.7010 | APIRI.TPS.7010 |
| APIR3.1.1.2[41] | The API shall provide an asynchronous notification to alert all application programs when their associated windows change size. | The Linux SIGWINCH signal is provided. Application level handler functions can assess changes. | The ViewportControl process (Section 3.4.3) sends notification of a change in the presence or absence of a physical front panel, via each open application FrontPanelDriver interface (Section 3.3), in the form of the Linux SIGWINCH signal. | APIRI.TCS.7010 | APIRI.TPS.7010 |
| APIR3.1.1.2[42] | The API shall provide a function to allow application programs to reset the display as described in the ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | fpui\_reset(3fpui) | Recommend removing requirement. | No Test  API Function Removed |  |
| APIR3.1.1.2[43] | The API shall provide a function to illuminate or extinguish the CPU ACTIVE LED described in the ATC Controller Standard, Section 7 and Section B.2. | fpui\_set\_led(3fpui) | Recommend removing requirement. | No Test  API Function Removed |  |
| APIR3.1.1.2[44] | The function shall only operate for application programs with a window in focus. | fpui\_set\_led(3fpui) | Recommend removing requirement. | No Test  API Function Removed |  |
| APIR3.1.1.2[45] | The API shall provide a function to send raw output data to the display. | fpui\_write(3fpui),  fpui\_open(3fpui) | The FrontPanelDriver Direct Interface (Section 3.3.2) is available by setting the “O\_DIRECT” bit of the flags parameter to the fpui\_open() library function call (Section 3.4.8). | APIRI.TCS.2090 | APIRI.TPS.1001 |
| APIR3.1.1.2[46] | If the application window is in focus, the data shall be sent to the display port without interpretation or buffering by the API. | fpui\_write(3fpui),  fpui\_open(3fpui) | The VirtualTerminal handler (Section 3.4.2) implements the direct mode as described in Direct Pass Through (Section 3.4.5). | APIRI.TCS.2090 | APIRI.TPS.1001 |
| APIR3.1.1.2[47] | If the application window is not in focus, the API shall discard the data. | fpui\_write(3fpui),  fpui\_open(3fpui) | The VirtualTerminal handler (Section 3.4.2) implements the direct mode as described in Direct Pass Through (Section 3.4.5). | APIRI.TCS.2090 | APIRI.TPS.1001 |
| APIR3.1.1.2[48] | The API shall provide a function to read raw input data from the display (this does not include the Aux Switch which is handled separately; see Item “c”). | fpui\_read(3fpui),  fpui\_open(3fpui) | The FrontPanelDriver direct interface (Section 3.3.2) implements raw (unmapped) reads by returning data from the raw portion of data packets returned by the VirtualTerminal handler (Section 3.4.2). | APIRI.TCS.2090 | APIRI.TPS.1001 |
| APIR3.1.1.2[49] | This function shall return raw data from the input buffer without the key code interpretation described in item “y”. | fpui\_read(3fpui),  fpui\_open(3fpui) | The FrontPanelDriver direct interface (Section 3.3.2) implements raw (unmapped) reads by returning data from the raw portion of data packets returned by the VirtualTerminal handler (Section 3.4.2). | APIRI.TCS.2090 | APIRI.TPS.1001 |
| APIR3.2[1] | The API shall provide a method to determine the version number(s) of the API. | fpui\_apiver(3fpui),  fio\_apiver(3fio),  tod\_apiver(); | Each API library shall have a function to return the version information as specified in the ATC 5401 API Standard. | APIRI.TCS.2100 | APIRI.TPS.1001 |
| APIR3.2.1[1] | The API shall provide Operational User the ability to view or configure system-wide parameters of the ATC controller unit using the ATC Configuration Window. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[2] | The display size of the ATC Configuration Window shall be consistent with the windows of the Front Panel Manager (see Section 3.1.1). | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[3] | The API shall provide a menu ("Configuration Menu") of up to 16 items ("configuration items") which can be selected by the Operational User from the keypad of the Front Panel display. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[4] | When the associated key of a configuration item is pressed, a configuration utility program ("configuration utility") shall be executed to view or configure the parameters associated with the configuration item. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[5] | The configuration utility shall use the ATC Configuration Window as its user interface display and receive the characters input from the Front Panel input device. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[6] | Only one configuration utility shall be operational at a time. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[7] | The configuration items shall be limited to 16 characters. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[8] | If there is no configuration item associated to a key, the configuration item shall be listed as a blank. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[9] | There shall be five configuration items defined in the API standard as follows: "System Time," "Ethernet," "System Services," "Linux/API Info" and "Host EEPROM Info". | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[10] | The Configuration Menu shall be extensible by software developers with up to twelve additional configuration items that are not defined in the API Standard. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.7050 | APIRI.TPS.7050 |
| APIR3.2.1[11] | The Configuration Menu shall have the format shown in Figure 9. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[12] | The Configuration Menu shall be displayed in the ATC Configuration Window when there is no configuration utility in operation. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[13] | When the API displays the Configuration Menu, the screen attributes of the ATC Configuration Window shall be set to the values of the controller unit at power up as described in ATC Controller Standard, Section 7.1.4 (ATC 5201 has this in Section 6.1.4). | N/A. | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[14] | The Operational User shall be able to put the ATC Configuration Window in focus from any application program or the Front Panel Manager Window by pressing {\*\*,<NEXT>} (\*\* is as defined in Section 3.1.1.1 Item "e") from the keypad on the controller’s Front Panel. | N/A | The ViewportControl process (Section 3.4.3) enacts the focus change to the SystemConfiguration manager process (Section 3.4.7) on detection of the {\*\*,<NEXT>} key sequence. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[15] | The Operational User shall have the capability to select a configuration utility by pressing the corresponding key [0-F] from the Configuration Menu. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[16] | The top two lines and bottom line of the Configuration Menu shall be fixed as shown in Figure 9. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[17] | The number of lines between the second line and bottom line used for displaying the configuration items shall vary according to the size of the ATC Front Panel display. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[18] | The Operational User shall be able to scroll the Configuration Menu up and down one line at a time to view the configuration items using the up and down arrow keys of the controller keypad. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[19] | The Operational User shall be able to put the Front Panel Manager Window in focus by pressing {<NEXT>} in the ATC Configuration Menu. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[20] | The only possible selections from the ATC Configuration Menu shall be itself, the Front Panel Manager Window (see Section 3.1.1.1), a window selected by pressing {\*\*,[0-F]} or a configuration item. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[21] | The API shall provide a mechanism for Operational Users to modify configuration fields as identified in the requirements of the configuration utilities (see Section 3.2.1.1 through Section 3.2.1.5). | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[22] | The user shall be able to move the cursor from field to field using the left and right arrow keys from the keypad of the controller's Front Panel. | N/A. | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[23] | If a right arrow is pressed, the cursor shall jump to the next configurable field to the right or, if there is no configurable field to the right, it will jump to the first configurable field of the next downward line that contains a configurable field. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[24] | If there are no configurable fields to the right or downward, then the right arrow key shall have no effect. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[25] | If a left arrow is pressed, the cursor shall jump to the next configurable field to the left or, if there is no configurable field to the left, it will jump to the last configurable field of the next upward line that contains a configurable field. | N/A. | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[26] | If there are no configurable fields to the left or upward, then the left arrow key shall have no effect. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[27] | The user shall be able to apply the values of fields modified by the user by pressing the <ENT> key. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[28] | When the <ENT> key is pressed, the configuration utility shall remain in focus. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[29] | When the cursor is over the value of a configurable field, the user shall then be able to scroll continuously through all of the possible values of the field using the + and – keys. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[30] | When the user finds the desired value, the user shall be able set the value by pressing the <YES> key, by pushing arrow (left, right, up or down) keys, or by pushing the <ENT> key. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[31] | If the user presses the <NO> key while scrolling through values of a field, the field shall revert to the value prior to pushing the + and – key. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[32] | In addition to the method described in (ii), when the cursor is over the value of a configurable field that is made up of one or more digits 0-9, the user shall be able to enter the value using the keys 0-9 from the keypad of the controller's Front Panel display. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[33] | Numeric values as entered shall be displayed right justified within the field. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[34] | If while entering digits, the user enters more digits than the field will hold, the left most digit shall be removed from the field. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[35] | The user shall be able to set the value by pressing the <YES> key by pushing arrow keys (left, right, up or down), or by pushing the <ENT> key. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[36] | If the user presses the <NO> key while modifying a field, the field shall revert to the value prior to pushing a numerical key. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[37] | If the user edits a field and enters a value that is out of range, the API shall activate the bell of the controller unit (see Section 7.1.4 (ATC 5201 has this in Section 6.1.4) of the ATC Controller Standard) and the field shall revert to the value prior to editing. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[38] | If the user enters a keyboard character that cannot be interpreted under the context of the ATC Configuration Window or active configuration utility, the API shall activate the bell of the controller unit and the invalid key shall be ignored. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[39] | The Operational User shall quit a configuration utility by pressing {\*\*,<NEXT>}. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[40] | If the configuration utility terminates in 2 seconds or less, the API shall display the Configuration Menu | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[41] | If a configuration utility does not terminate within the 2 second period, the user shall be provided an option to terminate the configuration utility as shown in Figure 10. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[42] | If the user presses <YES>, the API shall terminate the configuration utility using the Linux SIGTERM signal then display the Configuration Menu. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[43] | If the user presses <NO>, the configuration utility shall be redisplayed. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[44] | When a configuration utility terminates, changes made to configurable fields that were not applied using the <ENT> key prior to the termination shall be discarded. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.6010 | APIRI.TPS.6010 |
| APIR3.2.1[45] | The API shall provide a configuration file called "ATCConfigurationMenu.txt" to be used by the API to form the Configuration Menu. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.7050 | APIRI.TPS.7050 |
| APIR3.2.1[46] | The format of the file shall be comma delimited with each line representing a configuration item for the Configuration Menu as follows:  configitemname, executablepathname  configitemname, executablepathname  configitemname, executablepathname  etc.  where "configitemname" represents the text displayed in the Configuration Menu and "executablepathname" is the pathname to the executable file of the appropriate configuration utility. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.7050 | APIRI.TPS.7050 |
| APIR3.2.1[47] | The lines shall be processed by the API in order until the end of file is reached or the Configuration Menu is full. | N/A | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.7050 | APIRI.TPS.7050 |
| APIR3.2.1[48] | The API shall provide a function to allow a configuration utility to open the ATC Configuration Window and reserve the resource. | fpui\_open\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.7050 | APIRI.TPS.7050 |
| APIR3.2.1[49] | The API shall provide a function to close the ATC Configuration Window and release the resource. | fpui\_close\_config\_window(3fpui) | The system described in Section 3.1 includes a SystemConfiguration manager process (Section 3.4.7) to maintain a single system configuration window from which all required ATC API configuration utilities may be activated. The look and feel and interaction model of the system configuration window adheres to the requirements listed in ATC 5401 API Standard as APIR3.2.1 [2] through APIR3.2.1 [49]. | APIRI.TCS.7050 | APIRI.TPS.7050 |
| APIR3.2.2[1] | The API shall provide a method for the Operational User to view and set the system time through the System Time Utility. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[2] | The System Time Configuration Utility shall have the format as shown in Figure 12. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[3] | The top line and bottom line of the System Time Configuration Utility shall be fixed as shown in Figure 12. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[4] | The number of lines between the top line and bottom line used for displaying the Ethernet configuration information shall vary according to the size of the ATC Front Panel display. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[5] | The Operational User shall be able to scroll the System Time Configuration Utility up and down one line at a time using the up and down arrow keys of the controller unit's keypad. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[6] | The Operational User shall be able to modify the following fields on Line 7 of Figure 12 using the method described in Section 3.2.1 Item "h":  i) MM/DD/YYYY is the date where MM is the two digit month with values ranging 01-12, DD is the two digit day with values ranging 01-31, and YYYY is the two digit year with values ranging 0000-9999.  ii) hh:mm:ss is the time based on a 24 hour clock (00:00:00 is 12:00 midnight) where hh is the two digit hour with values ranging 00-23, mm is the two digit minute with values ranging 00-59 and ss is the two digit second with values ranging 00-59.  iii) shh:mm is the time zone represented as an offset from Coordinated Universal Time (UTC) where s is the sign with values + or -, hh is the two digit number of hours with values 00-12, and mm is the two digit number of minutes with values ranging 00-59.  iv) enadis is a field with values "Enable" or "Disabl" indicating whether Daylight Savings Time (DST) is enabled or disabled for the location of the controller unit. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[7] | Line 3 of Figure 12 shall represent the current date and time values of the controller unit and shall be updated once a second by the System Time Configuration Utility. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[8] | The fields of Line 3 shall be as follows:  i) All of the fields described in Item "d" of this section.  ii) status is a field with values "Active" or "Inactv" indicating whether or not the controller unit is currently applying DST. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.2[9] | The default values of all fields shall be those values at the time the Operational User invokes the System Time Configuration Utility | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Time configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[1] | The API shall provide a method for the Operational User to view and configure the Ethernet parameters of the ATC controller unit through the Ethernet Configuration Utility. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[2] | The Ethernet Configuration Utility shall have the format as shown in Figure 13. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[3] | The top line and bottom line of the Ethernet Configuration Utility shall be fixed as shown in Figure 13. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[4] | The number of lines between the top line and bottom line used for displaying the Ethernet configuration information shall vary according to the size of the ATC Front Panel display. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[5] | The Operational User shall be able to scroll the Ethernet Configuration Utility up and down one line at a time using the up and down arrow keys of the controller unit's keypad. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[6] | The Operational User shall be able to modify the following fields using the method described in Section 3.2.1 Item "h" :  i) yon is a field that can be "Yes" or "No" indicating whether a port is enabled.  ii) ### is a numerical field 0-255 used to set the Ethernet addresses (IPv4 addresses) associated with the port or network.  iii) hostname is an alphanumeric field up to 255 characters long indicating the Host Name of the controller unit. If hostname extends beyond the edge of the display, it will wrap to the next line. Each character of the field may be letters a-z, letters A-Z, the digits 0-9, the period (.) and the hyphen (-). In addition to the valid alphanumeric characters, the user may enter an asterisk (\*) to clear the existing character and all characters to the right of the cursor position. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[7] | The field identified as numofpackets shall be an unsigned integer value up to 10 digits long indicating the number of packets Sent and Received, Good and Bad over the Ethernet port as shown in Figure 13. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[8] | numofpackets shall be updated by the Ethernet Configuration Utility one time per second. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[9] | The Operational User shall not be able to modify these fields directly. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.3[10] | The default values of all fields shall be those values at the time the Operational User invokes the Ethernet Configuration Utility. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Ethernet Settings configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[1] | The API shall provide a method for the Operational User to enable and disable services through the System Services Configuration Utility | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[2] | The Systems Services Configuration Utility shall have the format as shown in Figure 14 | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[3] | The top line and bottom line of the Systems Services Configuration Utility shall be fixed as shown in Figure 14 | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[4] | The number of lines between the top line and bottom line used for displaying the system services information shall vary according to the size of the ATC Front Panel display | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[5] | The Operational User shall be able to scroll the Systems Services Configuration Utility up and down one line at a time using the up and down arrow keys of the controller unit's keypad | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[6] | The Systems Services Configuration Utility shall display all of the services available on the controller with up to 50 services listed | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[7] | The services listed in the Systems Services Configuration Utility shall be limited to 22 characters | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[8] | The current status of the services listed shall be displayed in the "STATUS" column as shown in Figure 14 | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[9] | The field enaordis shall be "Enabled" or "Disabled" based on the status of the service | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[10] | The Operational User shall not be able to modify this field directly. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[11] | The Operational User shall be able to modify the enaordis field in the "CHANGE" column as shown in Figure 14 using the method described in Section 3.2.1 Item "h". | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.4[12] | The possible values shall be "Enabled" or "Disabled". | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the System Services configuration screen. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[1] | The API shall provide a method for the Operational User to view Linux system and API library information through the Linux/API Information Utility as shown in Figure 15. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[2] | The top line and bottom line of the Linux/API Information Utility shall be fixed as shown in Figure 15. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[3] | The number of lines between the top line and bottom line used for displaying the system services information shall vary according to the size of the ATC Front Panel display. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[4] | The Operational User shall be able to scroll the Linux/API Information Utility up and down one line at a time using the up and down arrow keys of the controller unit's keypad. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[5] | The Linux/API Information Utility shall display the Linux system information available from the Linux "uname()" function (see also ATC Controller Standard, Section 2.2.5) as follows: "Kernel Name," "Network Node Name," "Kernel Release," "Machine Hardware Name," "Processor Type," "Hardware Platform," and "Operating System" as shown in Figure 15 | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[6] | The text field shall represent the character strings returned from the uname() function. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[7] | The Linux/API Information Utility shall display the version information for API library and drivers including the manufacturer's name, the manufacturer's version number of the software, and the version of the API Standard to which the software is conformant. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[8] | The API library and driver lines shall be repeated for each API library and driver installed on the controller unit | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[9] | The API shall provide function(s) which return the version information of the API software in use on the controller unit. | fpui\_apiver(3fpui), fio\_apiver(3fio),  tod\_apiver(3tod) | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.5[10] | If the value of a field is too long to fit on the remainder of a line, the Linux/API Information Utility shall provide additional lines to accommodate the field in a wrapping fashion and the remaining Linux/API information shall start on a new line as shown in Figure 16. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Linux Information and API Information configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[1] | The API shall provide a method for the Operational User to view the contents of the ATC Host Module EEPROM information (as defined by ATC Controller Standard, Annex B) through the Host EEPROM Information Utility as shown in Figures 17 and 18. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[2] | The field values shown are generally consistent with the "Default Configuration" as described in the ATC Controller Standard, Annex B. Actual field values will vary. The "User Data" field shall not be included in the Host EEPROM Information Utility. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[3] | The top line and bottom line of the Host EEPROM Information Utility shall be fixed as shown in Figures 17 and 18. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[4] | The number of lines between the top line and bottom line used for displaying the system services information shall vary according to the size of the ATC Front Panel display. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[5] | The Operational User shall be able to scroll the Host EEPROM Information Utility up and down one line at a time using the up and down arrow keys of the controller unit's keypad. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[6] | The values the "Ethernet Switch/Router Mac Addresses," "Host Board Serial Ports Used" and the "Agency Reserved" fields shall be displayed as hexadecimal pairs as represented by HH in Figure 18. | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.2.6[7] | If the value of a field is too long to fit on the remainder of a line, the Host EEPROM Information Utility shall provide additional lines to accommodate the field in a wrapping fashion and the remaining Linux/API information shall start on a new line (similar operation to that of Linux/API Information Utility in Figure 16). | N/A | The SystemConfiguration manager process (Section 3.4.7) provides functionality to implement the Host EEPROM Content configuration screens. | APIRI.TCS.6020 | APIRI.TPS.6020 |
| APIR3.4[1] | The API shall operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard. | N/A | The reference implementation is designed and shall be compiled for, tested and validated on actual ATC Hardware with a BSP compliant with the latest ATC 5201 Standard. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| APIR3.4[2] | The API function calls shall be specified using the C programming language as described by “ISO/IEC 9899:1999” commonly referred to as the C99 Standard. | Section 4 describes the API accordingly. | The reference implementation shall include API header files with all function prototypes conforming to the C99 standard. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| APIR3.5.2[1] | The operational look and feel of user interfaces developed for the API shall have consistent window titling conventions, scrolling methods, menu styles and selection methods. | N/A | The FPUI API windows shall be implemented as prescribed in the ATC 5401 API Standard. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| APIR3.5.2[2] | If API functions have a similar operation to existing Linux functions, they shall have a similar name and argument style to those functions to the extent possible without causing compilation issues. | See  fpui\_close(3fpui) & close(2),  fpui\_open(3fpui) & open(2),  fpui\_read(3fpui) & read(2),  fpui\_write(3fpui) & write(2) | Function naming and argument style of API function prototypes is dictated by the definitions in the ATC 5401 API Standard. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| APIR3.5.2[3] | The API function names shall be lower case. | Section 4 defines the API functions in lower case. | The API library symbols shall resolve the lower case names accordingly. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| APIR3.5.2[4] | API functions shall use the Linux “errno” error notification mechanism if an error indication is expected for a function. | Section 4 API functions are defined accordingly. | The reference implementation shall ensure that all API function calls comply with the errno error notification mechanism. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| APIR3.5.2[5] | The API shall be loadable as an ELF (Executable and Linking Format) library | N/A | All executable elements of the reference design shall be compatible to ELF file format. | APIRI.TCS.6030 | APIRI.TPS.6030 |
| **This requirement should be added to ATC 5401. Likely location is Section 3.6. The likely identifier to be APIR3.6[1].** | The API software shall only reference operating system commands and features that are available in the Linux environment defined in the ATC Board Support Package (see ATC Controller Standard, Section 2.2.5, Annex A and Annex B). | N/A | The reference implementation shall be compatible with a Linux kernel and runtime system with a BSP as defined in the ATC 5201 Standard. | APIRI.TCS.6030 | APIRI.TPS.6030 |

## FIO Library Requirements to Validation Description Matrix

The following table shows the relationship between the requirements and associated FIO library functions of the API Standard and the test identifiers which represent the test documents for those requirements and functions.

| **Req ID** | **Req Description** | **ATC API Function** | **APIRI SDD Design Narrative** | **Test Cases** | **Test Procedures** |
| --- | --- | --- | --- | --- | --- |
| APIR3.1.2[1] | The API shall assume it has exclusive access to the serial communications ports of the ATC Engine Board that are designated for Field I/O Devices. | fio\_register(3fio) | The FIOMSG entity (Section 4.4.4) of the FIO API system design described in Section 4.1 uses the kernel-level interface of the ATC SPxS serial driver to obtain serial access to Field I/O Devices. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[2] | The supported Field I/O serial communications ports shall be SP3, SP5 and SP8. | fio\_fiod\_register(3fio),  FIO\_PORT parameter | The fio\_fiod\_register() API library function (Section 4.4.2) allows the FIO\_PORT parameter to be specified when registering a field i/o device. The registered fiod is represented in the FIOMAN entity module (Section 4.4.3) by a data structure with a field to record the FIO\_PORT parameter. The FIOMSG entity (Section 4.4.4) of the design described in Section 4.1 uses the kernel-level interface of the ATC SPxS serial driver to communicate with the respective serial port when the field i/o device is enabled. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[3] | The supported communication modes on those ports shall be 153.6 Kbps and 614.4 Kbps SDLC. | fio\_fiod\_register(3fio) | The FIOMSG entity (Section 4.4.4) configures the ATC SPxS serial driver for the appropriate port speed at open time by using the kernel-level interface, and based on the type of first Field I/O Device registered per port. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[4] | The API shall not open any serial communications port or initiate communications to any Field I/O Device unless explicitly commanded to do so by an application program. | fio\_fiod\_enable(3fio) | The fio\_fiod\_enable() API function call (Section 4.4.2) must be made by the application in order to initiate communication to a field i/o device on its respective serial port. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[5] | The API shall support all cabinet architectures and associated Field I/O Device types as listed in the ATC Controller Standard Section 8. | fio\_register(3fio) | The FIOMAN (Section 4.4.3) and FIOMSG (Section 4.4.4) entities of the system described in Section 4.1 support functionality and message frame definitions according to the Field I/O Device types listed in the latest ATC Standard. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[6] | The API shall support the Field I/O Device types shown in Table 2. | fio\_fiod\_register(3fio),  FIO\_DEVICE\_TYPE parameter | The full set of field i/o module types enumerated by the FIO\_DEVICE\_TYPE parameter and listed in Table 2 of the ATC 5401 API Standard is supported and represented in the design described in Section 4.1. The FIOMAN entity (Section 4.4.3) includes rules to ensure that appropriate frame types are scheduled for each Field I/O Device type. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[7] | The API shall assume that BIU and MMU Field I/O Devices operate at 153.6 Kbps and all other Field I/O Device types operate at 614.4 Kbps. | fio\_fiod\_register(3fio) | The FIOMSG entity (Section 4.4.4) of the design described in Section 4.1 opens the appropriate serial port for each Field I/O Device type registered by applications. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[8] | The API shall support communication to multiple Field I/O Devices on a single communications port provided the Field I/O Devices have compatible physical communication attributes. | fio\_fiod\_register(3fio) | The FIOMAN (Section 4.4.3) and FIOMSG (Section 4.4.4) entities of the design described in Section 4.1 allow registration of multiple field i/o device types and check for compatibility with already-registered devices on each serial port, rejecting incompatible combinations. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[9] | The API shall support a maximum of one Field I/O Device of each type per communications port except in the case of BIUs and SIUs. | fio\_fiod\_register(3fio),  FIO\_DEVICE\_TYPE parameter | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 performs a check for already-registered devices in response to a device registration request for a new device, and enforces this restriction for each serial port. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[10] | The API shall support up to 8 Detector BIU and 8 Terminal & Facilities BIU Field I/O Devices per communications port. | fio\_fiod\_register(3fio),  FIO\_DEVICE\_TYPE parameter | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 allows the registration of each of the devices enumerated in Table 2 of the ATC 5401 API Standard, including FIODR1-FIODR8 and FIOTF1-FIOTF8, for each serial port. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[11] | The API shall support up to 5 Input SIU, 2 14-Pack Output SIU and 4 6-Pack Output SIU Field I/O Devices per communications port. | fio\_fiod\_register(3fio),  FIO\_DEVICE\_TYPE parameter | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 allows the registration of each of the devices enumerated in Table 2 of the ATC 5401 API Standard, including FIOINSIU1-FIOINSIU5, FIOOUT14SIU1-FIOOUT14SIU2, FIOOUT6SIU1-FIOOUT6SIU4, for each serial port. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[12] | The API shall only support valid Output SIU combinations as defined in the ITS Cabinet Standard, Section 4.7. | fio\_fiod\_register(3fio),  FIO\_DEVICE\_TYPE parameter | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 performs a check for already-registered devices in response to a device registration request for a new device, and enforces this restriction for each serial port. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[13] | The API shall identify specific Field I/O Devices using the API Field I/O Device Names in Table 2. | FIO\_DEVICE\_TYPE is an enum meeting this requirement,  fio\_fiod\_register(3fio) | The full set of field i/o module types enumerated by the FIO\_DEVICE\_TYPE parameter and listed in Table 2 of the ATC 5401 API Standard is supported and represented in the design described in Section 4.1 | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[14] | The API shall provide a method for application programs to register and deregister with the API for access to the API Field I/O services. | fio\_register(3fio), fio\_deregister(3fio) | The system described in Section 4.1 supports the fio\_register and fio\_deregister calls via the FIOAPI library (Section 4.4.2) in turn making respective calls to the ioctl interface of the FIOMAN entity (Section 4.4.3). | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[15] | The process of application program registration shall not cause the API to perform any communications with the Field I/O Device. | fio\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 contains application registration functionality which does not affect any serial communications to Field I/O devices. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[16] | When an application program deregisters for access to Field I/O services, the API shall deregister (as defined in Item “e”) all Field I/O devices registered by that application program. | fio\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes application deregistration functionality to perform device deregistration for all devices previously registered by that application. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[17] | The API shall provide a method to allow application programs to register and deregister for access to specific Field I/O Devices by specifying the communications port, device type, and where applicable, the Field I/O Device number. | fio\_fiod\_register(3fio), fio\_fiod\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality allowing applications to register and deregister for access to specified Field I/O devices via the FIOAPI library calls fio\_fiod\_register() and fio\_fiod\_deregister() (Section 4.4.2). | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[18] | Once a device has been registered on a communications port, the API shall permit the registration of additional compatible Field I/O Devices on the same communications port and prohibit the registration of incompatible Field I/O Devices on the same communications port. | fio\_fiod\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 performs a check for already-registered devices in response to a device registration request for a new device, and enforces this restriction for each serial port. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[19] | The Field I/O Device registration process shall not cause the API to perform any device communications. | fio\_fiod\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 contains application registration functionality which does not affect any serial communications to Field I/O devices. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[20] | When an application program deregisters for access to a Field I/O Device, the API shall disable (as defined in Item “g”) the Field I/O Device, relinquish all output points for that device and set all application program settable states to their default values. | fio\_fiod\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes application deregistration functionality to perform device deregistration for all devices previously registered by that application. Output points reserved by an application for a device it is deregistering are marked unreserved and the values set to zero. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[21] | The API shall provide a method for application programs to query for the presence of a Field I/O Device using the communications port, device type, and where applicable, the Field I/O Device number. | fio\_query\_fiod(3fio) | The FIOAPI library call fio\_query\_fiod() (Section 4.4.2) returns information about Field I/O device presence, obtained via a call to the ioctl interface of the fiodriver and corresponding functionality in the FIOMAN entity (Section 4.4.3). | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[22] | If the API does not have the communications port open at the time of the query and it is necessary for the API to open the communications port to determine the Field I/O Device, the API shall close the communications port after the query is completed. | fio\_query\_fiod(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to open and close each communications port as required to attempt to communicate with each Field I/O device. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[23] | If the API has the communications port open at the time of the query and the communications attributes for the Field I/O Device used in the query are not compatible with the current settings on the communications port, the API shall assume that the Field I/O Device is not present. | fio\_query\_fiod(3fio) | The FIOMAN entity (Section 4.4.3) provides functionality to indicate that a Field I/O Device is not present if the check for compatible devices on the same serial port fails when processing the fio\_query\_fiod() FIOAPI library call (Section 4.4.2). | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[24] | If the API has the communications port open at the time of the query and API is already successfully completing scheduled communications to the Field I/O Device, the API shall indicate that the Field I/O Device is present without sending any additional frames to the device. | fio\_query\_fiod(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to test if a device on any open communications port is successfully communicating, and to indicate that a Field I/O Device is present when processing the fio\_query\_fiod() FIOAPI library call (Section 4.4.2), without sending any additional frames. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[25] | The API shall provide a method which allows an application program to enable and disable communications to a Field I/O Device for which the application program has registered. | fio\_fiod\_enable(3fio), fio\_fiod\_disable(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to begin and end communications to a Field I/O device according to fio\_fiod\_enable() and fio\_fiod\_disable() function calls from the FIOAPI library (Section 4.4.2). | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[26] | When the communications enable method is called, the API shall initiate scheduled communications between the API and the specified Field I/O Device if not already active. | fio\_fiod\_enable(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to begin or continue communicating with a registered Field I/O device on receiving the fio\_fiod\_enable() call for that device via the FIOAPI library (Section 4.4.2). | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[27] | When the disable communications method is called, the API shall cease scheduled communications between the API and the specified Field I/O Device if the device is no longer enabled by any application program. | fio\_fiod\_disable(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to cease communicating with a registered Field I/O device on receiving the fio\_fiod\_disable() call for that device via the FIOAPI library (Section 4.4.2), unless that device is still enabled by another application. | APIRI.TCS.3010 | APIRI.TPS.1001 |
| APIR3.1.2[28] | When a Field I/O Device is disabled, any output points which have been reserved by that application program shall be set to Off. | fio\_fiod\_disable(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains data structures to store per application output reservations, and removes those reservations relating to outputs of a Field I/O device being disabled by an application via the fio\_fiod\_diable() call of the FIOAPI library (Section 4.4.2). | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[29] | The API shall provide a method for application programs to read the states of the input and output points on registered Field I/O Devices, including both filtered and non-filtered states for the input points (depending on which input frames are scheduled). | fio\_fiod\_inputs\_get(3fio),  fio\_fiod\_outputs\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 causes the FIOMSG entity (Section 4.4.4) to periodically send certain default frames to each enabled Field I/O device in order to set output points and read both filtered and unfiltered input points as required by applications. The FIOMAN entity (Section 4.4.3) returns the latest known state of the inputs or outputs in response to a fio\_fiod\_inputs\_get() or fio\_fiod\_outputs\_get() call made by an application via the FIOAPI library (Section 4.4.2). | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[30] | If multiple application programs have registered for the same Field I/O Device, the API shall provide shared read access to the input and output point states for all application programs which have registered that device. | fio\_fiod\_inputs\_get(3fio), fio\_fiod\_outputs\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 supports multiple application registrations and shared registration of Field I/O devices, allowing read access to all input and output states for each application, by maintaining both system and application views of the latest such data as received from and sent to the physical device. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[31] | When the state of an output point is read, the API shall return the current state of that output point within the API. | fio\_fiod\_outputs\_get(3fio) | The implementation of the fio\_fiod\_outputs\_get() FIOAPI library function (Section 4.4.2) by the FIOMAN entity (Section 4.4.3) returns the last-sent state of the output points, stored in its data structures, for the Field I/O device requested. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[32] | The API shall provide a method for application programs to reserve/relinquish exclusive “write access” to individual output points of a Field I/O Device. | fio\_fiod\_outputs\_reservation\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to maintain a record of output point reservations and perform adequate checks during processing of output reservation requests to ensure exclusive ownership of each output point by a single application. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[33] | If an application program reserves a point that has already been reserved by that application program, it shall not be considered an error. | fio\_fiod\_outputs\_reservation\_set(3fio) | The processing of the fio\_fiod\_outputs\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[34] | If an application program relinquishes a point that is already in the relinquished state for that application program, it shall not be considered an error. | fio\_fiod\_outputs\_reservation\_set(3fio) | The processing of the fio\_fiod\_outputs\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[35] | If a point in a group of points cannot be reserved, the reservation attempt shall fail for all of them. | fio\_fiod\_outputs\_reservation\_set(3fio) | The processing of the fio\_fiod\_outputs\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[36] | The API shall allow only one application program to reserve write access to any individual output point. | fio\_fiod\_outputs\_reservation\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to maintain a record of output point reservations and perform adequate checks during processing of output reservation requests to ensure exclusive ownership of each output point by a single application. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[37] | The API shall allow multiple application programs to reserve different output points on a single Field I/O Device. | fio\_fiod\_outputs\_reservation\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to maintain a record of output point reservations and perform adequate checks during processing of output reservation requests to ensure exclusive ownership of each output point by a single application. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[38] | Exclusive reservation of an output point for write access by one application program shall not preclude other application programs from reading the state of the output point. | fio\_fiod\_outputs\_get(3fio) | The processing of the fio\_fiod\_outputs\_get() request by the FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 does not consider output point reservation settings when returning the requested view of output states. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[39] | The API shall provide error codes so that the application program can determine if the reservation action was successful or if there was a conflict with another application program. | fio\_fiod\_outputs\_reservation\_set(3fio) | The processing of the fio\_fiod\_outputs\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[40] | The API shall make output point reservations on a “first come first served basis.” | fio\_fiod\_outputs\_reservation\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 processes the output point reservation requests in the order received. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[41] | An application program shall be able to set the state of an output point if it has registered the associated Field I/O Device and reserved exclusive write access to the output point. | fio\_fiod\_outputs\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 contains functionality to perform checks to ensure that the application calling fio\_fiod\_outputs\_set() via the FIOAPI library (Section 4.4.2) has registered the Field I/O device and to only allow changes to output points reserved by that application. | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[42] | To set the state of an output point and control dimming, the API shall use separate arrays for control of the Load Switch + and Load Switch – (see Section 3.3.1.4.1.5 of the TS 2 Standard). | fio\_fiod\_outputs\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains data structures for output states which include both “Load Switch Plus” and “Load Switch Minus” values per output point. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[43] | The API shall provide a method for application programs to query the reservation status of output points on registered Field I/O Devices. | fio\_fiod\_outputs\_reservation\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to return the latest state of output reservations on handling the fio\_fiod\_outputs\_reservation\_get() function call from the FIOAPI library (Section 4.4.2). | APIRI.TCS.3020 | APIRI.TPS.1001 |
| APIR3.1.2[44] | The API shall provide a method for application programs to map/unmap reserved output points to reserved channels and colors on a registered FIOMMU or FIOCMU device. | fio\_fiod\_channel\_map\_set(3fio)  Provides both map and unmap functionality.  fio\_fiod\_channel\_map\_count(3fio), fio\_fiod\_channel\_map\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to maintain data structures and perform checks relating to the mapping of reserved output points to the respective colors of reserved channels. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[45] | The API shall use this mapping to set the contents of FIOMMU Frame 0 and FIOCMU Frames 61 and 67. | fio\_fiod\_channel\_map\_set(3fio), fio\_fiod\_outputs\_set(3fio) | The FIOMSG entity (Section 4.4.4) of the design described in Section 4.1 performs a callback function immediately prior to sending the type 0 MMU frame and type 61 and 67 CMU frames wherein the frame data is populated from the latest output states translated with reference to the mapped channel data structures maintained by the FIOMAN entity (Section 4.4.3). | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[46] | Any channel and color not mapped to an output point shall be set to Off. | fio\_fiod\_channel\_map\_set(3fio) |  | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[47] | The API shall provide a method for application programs to reserve/relinquish exclusive control of individual monitored channels on the FIOMMU or FIOCMU device. | fio\_fiod\_channel\_reservation\_set(3fio)  Provides both reserve and relinquish functionality.  fio\_fiod\_channel\_reservation\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to maintain a record of channel reservations and perform adequate checks during processing of channel reservation requests to ensure exclusive ownership of each monitored channel of each FIOMMU or FIOCMU device by a single application. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[48] | If an application program reserves a channel that has already been reserved by that application program, it shall not be considered an error. | fio\_fiod\_channel\_reservation\_set(3fio) | The processing of the fio\_fiod\_channel\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[49] | If an application program relinquishes a channel that is already in the relinquished state for that application program, it shall not be considered an error. | fio\_fiod\_channel\_reservation\_set(3fio) | The processing of the fio\_fiod\_channel\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[50] | If a channel in a group of channels cannot be reserved, the reservation attempt shall fail for all of them. | fio\_fiod\_channel\_reservation\_set(3fio) | The processing of the fio\_fiod\_channel\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3)returns the appropriate error code for this case. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[51] | The API shall allow multiple applications to reserve different channels on a single FIOMMU or FIOCMU device. | fio\_fiod\_channel\_reservation\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to maintain a record of channel reservations and perform adequate checks during processing of channel reservation requests to ensure exclusive ownership of each monitored channel of each FIOMMU or FIOCMU device by a single application. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[52] | The API shall provide error codes so that the application program can determine if the reservation action was successful or if there was a conflict with another application. | fio\_fiod\_channel\_reservation\_set(3fio) | The processing of the fio\_fiod\_channel\_reservation\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code in each case. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[53] | The API shall make channel reservations on a “first come first served basis.” | fio\_fiod\_channel\_reservation\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 processes the channel reservation requests in the order received. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[54] | The API shall provide a method for applications to query the reservation status of channels on registered FIOMMU or FIOCMU devices. | fio\_fiod\_channel\_reservation\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to return the latest state of channel reservations on handling the fio\_fiod\_channel\_reservation\_get() function call from the FIOAPI library (Section 4.4.2). | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[55] | Relinquishing a reserved output point or channel shall clear the associated assignments. | fio\_fiod\_outputs\_reservation\_set(3fio), fio\_fiod\_channel\_reservation\_set(3fio),  fio\_fiod\_channel\_map\_set(3fio),  fio\_fiod\_channel\_map\_count(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains the channel mapping data structures and removes any mappings for channel reservations or associated output reservations being relinquished. | APIRI.TCS.3030 | APIRI.TPS.1001 |
| APIR3.1.2[56] | The API shall provide functions which allow application programs to set and get the leading and trailing edge filter values on a per input basis for all Field I/O Devices that support configurable filtered inputs. | fio\_fiod\_inputs\_filter\_set(3fio),  fio\_fiod\_inputs\_filter\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains data structures to record the latest leading and trailing edge input filter values for both a system and application view. On handling a call to fio\_fiod\_inputs\_filter\_set() (Section 4.4.2) and after appropriate checks, new filter values are set and a type 51 frame is scheduled via FIOMSG entity (Section 4.4.4) to update the physical Field I/O device. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[57] | If multiple application programs set the filter values of an input, the shortest filter values shall be used. | fio\_fiod\_inputs\_filter\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1, on handling a call to fio\_fiod\_inputs\_filter\_set() (Section 4.4.2) and after comparison of the requested input filter values to the latest values, new filter values are set if necessary. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[58] | The API shall provide a return code containing the status and the value used for the set filter operation. | fio\_fiod\_inputs\_filter\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1, on handling a call to fio\_fiod\_inputs\_filter\_set() (Section 4.4.2) returns the latest leading edge and trailing edge input filter values from its data structures on success, or appropriate error code otherwise. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[59] | The default leading and trailing edge filter values shall be 5 consecutive samples. | #define constant FIO\_FILTER\_DEFAULT 5,  fio\_fiod\_inputs\_filter\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 ensures that the default leading and trailing edge input filter values are set by scheduling an appropriate frame type 51 at start of communications with the physical Field I/O device. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[60] | The API shall have the ability to collect and buffer the transition buffer information for each registered Field I/O Device used for input. | fio\_fiod\_inputs\_trans\_set(3fio), fio\_fiod\_inputs\_trans\_read(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and associated per-application data buffering to maintain separate 1024-entry FIFOs of input point transitions for each registered Field I/O input device, and to schedule appropriate frames via the FIOMSG entity (4.4.4) to gather input transition data from each capable device. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[61] | When the API reads the transition buffer of a Field I/O Device, it shall read the entire transition buffer. | fio\_fiod\_inputs\_trans\_read(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to configure input transitioning and polling for input transition buffer via appropriate frames sent and received by the FIOMSG entity (Section 4.4.4). The callback function on receipt of frame type 182 contains functionality to reschedule further transition buffer request frames as necessary to read the entire buffer from the physical Field I/O input device. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[62] | The API shall buffer the transition data on a per application program basis with the capability of storing 1024 transition entries in a FIFO fashion. | fio\_fiod\_inputs\_trans\_read(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and associated per-application data buffering to maintain separate 1024-entry FIFOs of input point transitions for each registered Field I/O input device, and to schedule appropriate frames via the FIOMSG entity (Section 4.4.4) to gather input transition data from each capable device. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[63] | The API shall provide a function which allows application programs to enable or disable transition monitoring of selected input points. | fio\_fiod\_inputs\_trans\_set(3fio),  fio\_fiod\_inputs\_trans\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1, on handling a fio\_fiod\_inputs\_trans\_set() call via the FIOAPI library (Section 4.4.2), causes the appropriate frame type 51 to be sent via the FIOMSG entity (Section 4.4.4) to configure transition reporting as requested. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[64] | By default, transition monitoring for all input points shall be disabled. | fio\_fiod\_inputs\_trans\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 only configures transition reporting for input points following a successful fio\_fiod\_inputs\_trans\_set() call via the FIOAPI library (Section 4.4.2). | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[65] | If an application program enables an input point for transition monitoring and that input point is already in the enabled state, it shall not be considered an error. | fio\_fiod\_inputs\_trans\_set(3fio) | The processing of the fio\_fiod\_inputs\_trans\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[66] | If an application program disables an input point for transition monitoring and that input point is already in the disabled state, it shall not be considered an error. | fio\_fiod\_inputs\_trans\_set(3fio) | The processing of the fio\_fiod\_inputs\_trans\_set() function call by the FIOMAN entity (Section 4.4.3) returns the appropriate error code for this case. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[67] | The API shall provide functions that allow application programs to access the API transition buffer information asynchronously (i.e. read the transition entries from the API buffer independent of any Field I/O Device communications). | fio\_fiod\_inputs\_trans\_read(3fio) | By maintaining separate FIFO queues for each application and for each input device, the FIOMAN entity (Section 4.4.3) returns transition entries asynchronous to the serial communications with the physical Field I/O devices. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[68] | When an application program reads a transition entry from an API transition buffer, that transition entry shall be cleared for that application program only, without affecting the API transition buffers for other application programs. | fio\_fiod\_inputs\_trans\_read(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application FIFO queues of input transition entries, such that calls to fio\_fiod\_inputs\_trans\_read() for one application do not affect the FIFO queues maintained for other registered applications. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[69] | If the transition buffer in the Field I/O Device overruns before information can be copied to the API transition buffer information, the API shall indicate that a device overrun condition has occurred in the transition buffer for that Field I/O Device. | fio\_fiod\_inputs\_trans\_read(3fio)    The return value is success (count of entries), FIOD Overrun or FIO API overrun. | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains a per-application status variable associated with each FIFO queue of input transition entries such that the FIO\_TRANS\_STATUS return value can be set appropriately. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[70] | If the transition buffer of the API overruns before the information is retrieved by the application program, the API shall indicate that an API overrun condition has occurred. | fio\_fiod\_inputs\_trans\_read(3fio)    The return value is success (count of entries), FIOD Overrun or FIO API overrun. | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains a per-application status variable associated with each FIFO queue of input transition entries such that the FIO\_TRANS\_STATUS return value can be set appropriately. | APIRI.TCS.3040 | APIRI.TPS.1001 |
| APIR3.1.2[71] | The ATC Controller Standard, Section 8, specifies the frames for communication with Field I/O Devices for Model 332 Cabinets, NEMA TS 1 and TS 2 Type 2 Cabinets and ITS Cabinets. The API shall support a subset of these frames at the scheduled frame frequencies as shown in Table 3. | fio\_fiod\_frame\_schedule\_set(3fio), fio\_set\_local\_time\_offset(3fio), | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes support for handling all frame types specified in Table 3 of the ATC 5401 API Standard, and for scheduling default frames at the specified frequencies for registered devices. | APIRI.TCS.3050 | APIRI.TPS.1001 |
| APIR3.1.2[72] | The NEMA TS 2 Standard, Section 3.3, specifies the frames for communication with Field I/O Devices for NEMA TS 2 Type 1 Cabinets. The API shall support a subset of these frames at the scheduled frame frequencies as shown in Table 4. | fio\_fiod\_frame\_schedule\_set(3fio), fio\_set\_local\_time\_offset(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes support for handling all frame types specified in Table 4 of the ATC 5401 API Standard, and for scheduling default frames at the specified frequencies for registered devices. | APIRI.TCS.3050 | APIRI.TPS.1001 |
| APIR3.1.2[73] | The timing for the command/response cycle of the frames shall be defined by the “Handshaking” algorithm in Section 3.3.1.5.3 of the NEMA TS 2 Standard. | fio\_fiod\_frame\_schedule\_set(3fio) | The FIOMSG entity (Section 4.4.3) of the design described in Section 4.1 includes a table of command/response frame timings and “dead-time” calculation functionality based on the NEMA TS2 Standard requirements. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[74] | The API shall provide a method for application programs to set/get the scheduled frame frequencies for a registered Field I/O Device. | fio\_fiod\_frame\_schedule\_set(3fio), fio\_fiod\_frame\_schedule\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to modify the frame schedule frequencies used by the FIOMSG entity (Section 4.4.4) according to values requested on receiving a fio\_fiod\_frame\_schedule\_set() call via the FIOAPI library (Section 4.4.2). | APIRI.TCS.3050 | APIRI.TPS.1001 |
| APIR3.1.2[75] | The frame frequency used by the API shall be the highest frequency requested by all application programs registered for that Field I/O Device. | fio\_fiod\_frame\_schedule\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application tables of frame schedule frequencies, such that on receipt of a fio\_fiod\_frame\_schedule\_set() call via the FIOAPI library (Section 4.4.2), a comparison is made between all registered applications’ values to select the highest frequency for each frame schedule. | APIRI.TCS.3050 | APIRI.TPS.1001 |
| APIR3.1.2[76] | The API shall provide a method to send a frame from either Table 3 or Table 4 one time (non-scheduled). | fio\_fiod\_frame\_schedule\_set(3fio) frequency of FIO\_HZ\_ONCE | The FIOMSG entity (Section 4.4.3) of the design described in Section 4.1 is responsible for maintaining the frame schedules and ensures that individual frames from Table 3 or Table 4 of the ATC 5401 API Standard, added to the schedule with a frequency value of FIO\_HZ\_ONCE are removed after being sent once. | APIRI.TCS.3050 | APIRI.TPS.1001 |
| APIR3.1.2[77] | The API shall provide a method to set/get the Failed State Action of a FIOCMU Field I/O Device. | fio\_fiod\_cmu\_fault\_set(3fio),  fio\_fiod\_cmu\_fault\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application and per-device system copies of the FIO\_CMU\_FSA structure, which are referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant CMU frames. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[78] | The Failed State Action shall be settable to None (LFSA=0, NFSA=0), Latched (LFSA=1, NFSA=0), or Non Latched (LFSA=0, NFSA=1). | fio\_fiod\_cmu\_fault\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application and per-device system copies of the FIO\_CMU\_FSA structure, which are referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant CMU frames. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[79] | The default Failed State Action shall be None. | fio\_fiod\_cmu\_fault\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application and per-device system copies of the FIO\_CMU\_FSA structure, which are referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant CMU frames. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[80] | If any application program sets the state to Latched, the API shall set the Failed State Action to Latched. | fio\_fiod\_cmu\_fault\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application and per-device system copies of the FIO\_CMU\_FSA structure, which are referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant CMU frames. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[81] | If no application program has set the Failed State Action to Latched, then if any application program sets the state to Non Latched, the API shall set the Failed State Action to Non Latched. | fio\_fiod\_cmu\_fault\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application and per-device system copies of the FIO\_CMU\_FSA structure, which are referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant CMU frames. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[82] | If all application programs have a state of None, then the API shall set the Failed State Action to None. | fio\_fiod\_cmu\_fault\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 maintains per-application and per-device system copies of the FIO\_CMU\_FSA structure, which are referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant CMU frames. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[83] | The API shall provide a method to set/get the state of the Fault Monitor output point of FIOTS1 and FIOTS2 Field I/O Devices. | fio\_fiod\_ts\_fault\_monitor\_set(3fio), fio\_fiod\_ts\_fault\_monitor\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Fault Monitor output point according to application requests via the fio\_fiod\_ts\_fault\_monitor\_set() and fio\_fiod\_ts\_fault\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The fault state data is referenced by the FIOMSG entity prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[84] | The API shall retain ownership of the Fault Monitor output point and not allow application programs to reserve this output point. | fio\_fiod\_ts\_fault\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Fault Monitor output point according to application requests via the fio\_fiod\_ts\_fault\_monitor\_set() and fio\_fiod\_ts\_fault\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The fault state data is referenced by the FIOMSG entity prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[85] | If any application program sets the Fault Monitor state to Off, the API shall turn Off the Fault Monitor output point on that device. | fio\_fiod\_ts\_fault\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Fault Monitor output point according to application requests via the fio\_fiod\_ts\_fault\_monitor\_set() and fio\_fiod\_ts\_fault\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The fault state data is referenced by the FIOMSG entity prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[86] | If all application programs have a Fault Monitor state of On for a FIOTS1 or FIOTS2 Device, then the API shall turn On the Fault Monitor output point on that device. | fio\_fiod\_ts\_fault\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Fault Monitor output point according to application requests via the fio\_fiod\_ts\_fault\_monitor\_set() and fio\_fiod\_ts\_fault\_monitor\_get() calls of the FIOAPI library (4.4.2). The fault state data is referenced by the FIOMSG entity prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[87] | The default state of the Fault Monitor output point shall be On. | fio\_fiod\_ts\_fault\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Fault Monitor output point according to application requests via the fio\_fiod\_ts\_fault\_monitor\_set() and fio\_fiod\_ts\_fault\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The fault state data is referenced by the FIOMSG entity prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[88] | The API shall provide a method to set/get the state of the Voltage Monitor output point of a FIOTS1 Field I/O Device. | fio\_fiod\_ts1\_volt\_monitor\_set(3fio), fio\_fiod\_ts1\_volt\_monitor\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Voltage Monitor output point according to application requests via the fio\_fiod\_ts1\_volt\_monitor\_set() and fio\_fiod\_ts1\_volt\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The voltage monitor state data is referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[89] | The API shall retain ownership of the Voltage Monitor output point and not allow application programs to reserve this output point. | fio\_fiod\_ts1\_volt\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Voltage Monitor output point according to application requests via the fio\_fiod\_ts1\_volt\_monitor\_set() and fio\_fiod\_ts1\_volt\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The voltage monitor state data is referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[90] | If any application program sets the Voltage Monitor state to Off, the API shall turn Off the Voltage Monitor output point on that device. | fio\_fiod\_ts1\_volt\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Voltage Monitor output point according to application requests via the fio\_fiod\_ts1\_volt\_monitor\_set() and fio\_fiod\_ts1\_volt\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The voltage monitor state data is referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[91] | If all application programs have a Voltage Monitor state of On for a FIOTS1 Device, then the API shall turn On the Voltage Monitor output point on that device. | fio\_fiod\_ts1\_volt\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Voltage Monitor output point according to application requests via the fio\_fiod\_ts1\_volt\_monitor\_set() and fio\_fiod\_ts1\_volt\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The voltage monitor state data is referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[92] | The default state of the Voltage Monitor output point shall be On. | fio\_fiod\_ts1\_volt\_monitor\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes data storage and comparison functionality to control the state of the Voltage Monitor output point according to application requests via the fio\_fiod\_ts1\_volt\_monitor\_set() and fio\_fiod\_ts1\_volt\_monitor\_get() calls of the FIOAPI library (Section 4.4.2). The voltage monitor state data is referenced by the FIOMSG entity (Section 4.4.4) prior to sending the relevant frames to the physical Field I/O device. | APIRI.TCS.3060 | APIRI.TPS.1001 |
| APIR3.1.2[93] | The API shall provide a method which allows application programs to assign the output point used for the Watchdog output of any registered Field I/O Device. | fio\_fiod\_wd\_reservation\_set(3fio),  fio\_fiod\_wd\_reservation\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes per-device data storage and functionality to reserve and control the watchdog output point for each capable registered Field I/O device. | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[94] | The API shall restrict the ability to assign the Watchdog output point to the first application program to call the assignment method. | fio\_fiod\_wd\_reservation\_set(3fio),  fio\_fiod\_wd\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 enforces this condition during handling of the fio\_fiod\_wd\_reservation\_set() function call via the FIOAPI library (Section 4.4.2). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[95] | The API shall retain ownership of the Watchdog output point and not allow application programs to reserve that output point directly. | fio\_fiod\_outputs\_reservation\_set(3fio),  fio\_fiod\_wd\_reservation\_set(3fio),  fio\_fiod\_wd\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes per-device data storage and functionality to reserve and control the watchdog output point for each capable registered Field I/O device. | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[96] | The API shall provide a method for application programs to register for shared control of the Watchdog output point. | fio\_fiod\_wd\_register(3fio)’  fio\_fiod\_wd\_deregister | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to manage the control of the watchdog function for any reserved watchdog output point, including detecting the watchdog trigger condition and setting the state of the watchdog output point accordingly via the relevant scheduled frames of the FIOMSG entity (Section 4.4.4). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[97] | The API shall provide a method for Watchdog registered application programs to “request” that the API toggle the state of the Watchdog output point. | fio\_fiod\_wd\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to manage the control of the watchdog function for any reserved watchdog output point, including detecting the watchdog trigger condition and setting the state of the watchdog output point accordingly via the relevant scheduled frames of the FIOMSG entity (Section 4.4.4). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[98] | The API shall only toggle the Watchdog output point if all Watchdog registered application programs have made the toggle request (Watchdog Triggered Condition). | fio\_fiod\_wd\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to manage the control of the watchdog function for any reserved watchdog output point, including detecting the watchdog trigger condition and setting the state of the watchdog output point accordingly via the relevant scheduled frames of the FIOMSG entity (Section 4.4.4). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[99] | Upon a Watchdog Triggered Condition, the API shall toggle the state of the Watchdog output point within the API. | fio\_fiod\_wd\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to manage the control of the watchdog function for any reserved watchdog output point, including detecting the watchdog trigger condition and setting the state of the watchdog output point accordingly via the relevant scheduled frames of the FIOMSG entity (Section 4.4.4). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[100] | When the API updates the output states of the Field I/O Device (see Item “n”), the API shall clear all previous toggle requests and the Watchdog Triggered Condition so that a new Watchdog Triggered Condition can be generated. | fio\_fiod\_wd\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to manage the control of the watchdog function for any reserved watchdog output point, including detecting the watchdog trigger condition and setting the state of the watchdog output point accordingly via the relevant scheduled frames of the FIOMSG entity (Section 4.4.4). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[101] | The API shall not toggle the Watchdog output point more than once per update of the output states on the Field I/O Device. | fio\_fiod\_wd\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality to manage the control of the watchdog function for any reserved watchdog output point, including detecting the watchdog trigger condition and setting the state of the watchdog output point accordingly via the relevant scheduled frames of the FIOMSG entity (Section 4.4.4). | APIRI.TCS.3070 | APIRI.TPS.1001 |
| APIR3.1.2[102] | The API shall provide functions which allow application programs to obtain status information of a registered Field I/O Device. | fio\_fiod\_status\_get(3fio) | The FIOMAN entity (4.4.3) of the design described in Section 4.1 includes functionality and data structures required to maintain status information for each registered Field I/O device in the form of the FIO\_FIOD\_STATUS structure, including communication status and error counts on a per-frame and per-device basis. | APIRI.TCS.3080 | APIRI.TPS.1001 |
| APIR3.1.2[103] | All counters contained in the Field I/O Device status information shall be four byte unsigned values each with a maximum value of 4,294,967,295. | fio\_fiod\_status\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality and data structures required to maintain status information for each registered Field I/O device in the form of the FIO\_FIOD\_STATUS structure, including communication status and error counts on a per-frame and per-device basis. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[104] | The counters shall be frozen when they reach the maximum value to prevent rollover. | fio\_fiod\_status\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality and data structures required to maintain status information for each registered Field I/O device in the form of the FIO\_FIOD\_STATUS structure, including communication status and error counts on a per-frame and per-device basis. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[105] | The API shall provide the following communication status information for each registered Field I/O Device:  i) Communications Enabled/Disabled;  ii) Cumulative successful response count for all frames to this device;  iii) Cumulative error count for all frames to this device; and  iv) Command frames sent to this device with the following information for each frame type: current scheduled frequency, cumulative successful response count, cumulative error count, numbers of errors in the last 10 frames, a response frame sequence number, frame size in bytes and the raw data from the most recent response frame. | fio\_fiod\_status\_get(3fio)    Frame access supported by  fio\_fiod\_frame\_read(3fio).    Frame size and sequence number are supported by fio\_fiod\_frame\_size(3fio).    A sequence number will be provided on all returned frames for frame aging. | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality and data structures required to maintain status information for each registered Field I/O device in the form of the FIO\_FIOD\_STATUS structure, including communication status and error counts on a per-frame and per-device basis. | APIRI.TCS.3080 | APIRI.TPS.1001 |
| APIR3.1.2[106] | The response frame sequence number shall be a four byte unsigned value and rollover after the maximum value. | fio\_fiod\_frame\_size(3fio),  fio\_fiod\_frame\_read(3fio),  fio\_query\_frame\_notify\_status(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality and data structures required to maintain status information for each registered Field I/O device in the form of the FIO\_FIOD\_STATUS structure, including communication status and error counts on a per-frame and per-device basis. | APIRI.TCS.6040 | APIRI.TPS.6040 |
| APIR3.1.2[107] | The API shall provide a method for application programs to reset the communications status counters to 0 (zero) for a registered Field I/O Device. | fio\_fiod\_status\_reset(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality and data structures required to maintain status information for each registered Field I/O device in the form of the FIO\_FIOD\_STATUS structure, including communication status and error counts on a per-frame and per-device basis. | APIRI.TCS.3080 | APIRI.TPS.1001 |
| APIR3.1.2[108] | A response frame shall only be considered successful if it is fully received within the time period defined by the “Handshaking” algorithm in Section 3.3.1.5.3 of the NEMA TS 2 Standard. | fio\_fiod\_status\_get(3fio) | The FIOMSG entity (Section 4.4.3) of the design described in Section 4.1 includes a table of command/response frame timings and “dead-time” calculation functionality based on the NEMA TS2 Standard requirements. | APIRI.TCS.3080 | APIRI.TPS.1001 |
| APIR3.1.2[109] | The API shall provide an API Health Monitor Function which registered application programs use to indicate to the API that they are operational. | fio\_hm\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[110] | The API shall provide a method to set an API Health Monitor Timeout for each application program (each application program has its own unique API Health Monitor Timeout). | fio\_hm\_register(3fio),  fio\_hm\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[111] | This API Health Monitor Timeout shall indicate the maximum allowable time between calls to the API Health Monitor Function. | fio\_hm\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[112] | The API Health Monitor Timeout shall be specified in tenths of a second. | fio\_hm\_register(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[113] | If the API Health Monitor Timeout expires for an application, the API shall disable (as defined previously in Item “g”) all Field I/O Devices registered by that application program. | fio\_hm\_register(3fio),  fio\_hm\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[114] | The API shall provide a method for an application program to disable the API Health Monitor feature for itself. | fio\_hm\_register(3fio),  fio\_hm\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[115] | The API shall provide a method for an application program to reset an API Health Monitor fault condition and allow the API to resume Field I/O Device communications. | fio\_hm\_fault\_reset(3fio),  fio\_hm\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[116] | An application shall only be able to reset its own Health Monitor fault condition and not that of any other application program. | fio\_hm\_fault\_reset(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[117] | If an application program resets the API Health Monitor fault condition, then any devices that were disabled due to that condition shall be re-enabled. | fio\_hm\_fault\_reset(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[118] | If an application program attempts to enable a device (as defined in Item “g”) that has been disabled due to an API Health Monitor fault condition, then the enable operation shall return an error and the Field I/O Device remain disabled. | fio\_hm\_fault\_reset(3fio),  fio\_fiod\_enable(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[119] | A call to the API Health Monitor Function after a Health Monitor fault has occurred shall not reset the Health Monitor fault condition. | fio\_hm\_fault\_reset(3fio),  fio\_hm\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[120] | The API Health Monitor Function shall return whether an API Health Monitor fault condition exists. | fio\_hm\_heartbeat(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 includes functionality to manage the health monitor/application heartbeat mechanism, including maintenance of per-application status and timers. | APIRI.TCS.3090 | APIRI.TPS.1001 |
| APIR3.1.2[121] | The API shall provide a method for an application program to send the Get CMU Configuration frame to a registered FIOCMU device. | This requirement is satisfied by:  - Set frame notification for response frame 193 using fio\_fiod\_frame\_notify\_register(3fio)  - Send frame 65 one time using fio\_fiod\_frame\_schedule\_set(3fio)  - Received response frame notification: SIGIO and fio\_query\_frame\_notify\_status(3fio)  - Read response frame 193 using fio\_fiod\_frame\_read(3fio)  Applications can determine if the CMU Status has changed with fio\_fiod\_cmu\_config\_change\_count(3fio) |  | APIRI.TCS.3100 | APIRI.TPS.1001 |
| APIR3.1.2[122] | The API shall reset all Module Status bits using the Request Module Status frame when a FIO332, FIOTS1, FIOTS2 or SIU device is first Enabled (as defined in Item “g”). | fio\_fiod\_enable(3fio),  fio\_fiod\_frame\_schedule\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 and the callback function of the FIOMSG entity (Section 4.4.4) on receipt of frame type 177 implement the required startup sequence on first enable of modules of types FIO332, FIOTS1, FIOTS2. | APIRI.TCS.3110 | APIRI.TPS.1001 |
| APIR3.1.2[123] | Anytime a response to a Request Module Status frame has Module Status bits indicating hardware reset, comm loss, or watchdog reset, then the API shall clear those bits, reset the input point filter values (Item “k”) and reconfigure transition reporting (Item “l”). | fio\_fiod\_enable(3fio),  fio\_fiod\_frame\_schedule\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 and the callback function of the FIOMSG entity (Section 4.4.4) on receipt of frame type 177 provides functionality to handle any module status error bits and schedule the re-send of any required input configuration frames. | APIRI.TCS.3110 | APIRI.TPS.1001 |
| APIR3.1.2[124] | The API shall provide a method to notify an application program when a command frame is acknowledged (response frame received by the API) or when an error occurs. | fio\_fiod\_frame\_notify\_register(3fio)  Allows application to say I want to be notified when this frame is received.    fio\_fiod\_frame\_notify\_deregister(3fio) Allows application to dismiss the notification service.    fio\_query\_frame\_notify\_status(3fio)  For retrieving information on frame status after notify – what frame is being notified and why (received, error, other ). This function returns the FIO\_DEV\_HANDLE that represents the FIOD that responded. | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 and the callback functions of the FIOMSG entity (Section 4.4.4) provide per-application data structures and functionality to manage receive frame notification requests submitted via the fio\_fiod\_frame\_notify\_register() function of the FIOAPI library (Section 4.4.2). | APIRI.TCS.3120 | APIRI.TPS.1001 |
| APIR3.1.2[125] | The command frame shall be identified by the frame type and a registered Field I/O Device. | fio\_fiod\_frame\_notify\_register(3fio),  fio\_fiod\_frame\_notify\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 and the callback functions of the FIOMSG entity (Section 4.4.4) provide per-application data structures and functionality to manage receive frame notification requests submitted via the fio\_fiod\_frame\_notify\_register() function of the FIOAPI library (Section 4.4.2). | APIRI.TCS.3120 | APIRI.TPS.1001 |
| APIR3.1.2[126] | The response frame notification shall include the Field I/O Device, response frame type, response frame sequence number, response frame size in bytes and an indication as to why the notification occurred (response received or error detected). | fio\_query\_frame\_notify\_status(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 and the callback functions of the FIOMSG entity (Section 4.4.4) provide per-application data structures and functionality to manage receive frame notification requests submitted via the fio\_fiod\_frame\_notify\_register() function of the FIOAPI library (Section 4.4.2). | APIRI.TCS.3120 | APIRI.TPS.1001 |
| APIR3.1.2[127] | The notification shall be able to be set for a one time occurrence or continuous occurrence. | fio\_fiod\_frame\_notify\_register(3fio), fio\_fiod\_frame\_notify\_deregister(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 and the callback functions of the FIOMSG entity (Section 4.4.4) provide per-application data structures and functionality to manage receive frame notification requests submitted via the fio\_fiod\_frame\_notify\_register() function of the FIOAPI library (Section 4.4.2). | APIRI.TCS.3120 | APIRI.TPS.1001 |
| APIR3.1.2[128] | The API shall provide a method to set and get the Dark Channel Map selection for a registered FIOCMU device. | fio\_fiod\_cmu\_dark\_channel\_set(3fio),  fio\_fiod\_cmu\_dark\_channel\_get(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and per-application data to manage the Dark Channel Map settings to be sent via the relevant FIOMSG entity (Section 4.4.4) frame types for FIOCMU Field I/O devices. | APIRI.TCS.3130 | APIRI.TPS.1001 |
| APIR3.1.2[129] | If multiple application programs attempt to set the Dark Channel Map selection, the API shall use the most recent selection. | fio\_fiod\_cmu\_dark\_channel\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and per-application data to manage the Dark Channel Map settings to be sent via the relevant FIOMSG entity (Section 4.4.4) frame types for FIOCMU Field I/O devices. | APIRI.TCS.3130 | APIRI.TPS.1001 |
| APIR3.1.2[130] | The default value of the Dark Channel Map Select bits shall be 0 (Mask #1). | fio\_fiod\_cmu\_dark\_channel\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and per-application data to manage the Dark Channel Map settings to be sent via the relevant FIOMSG entity (Section 4.4.4) frame types for FIOCMU Field I/O devices. | APIRI.TCS.3130 | APIRI.TPS.1001 |
| APIR3.1.2[131] | The API shall provide a method to set and get the state of the Load Switch Flash bit of a registered FIOMMU device. | fio\_fiod\_mmu\_flash\_bit\_set(3fio),  fio\_fiod\_mmu\_flash\_bit\_get(3fio).    Bit 112 of MMU frame 0. Default to value 0. | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and per-application data to manage the Load Switch Flash bit setting to be sent via the relevant FIOMSG entity (Section 4.4.4) frame types for FIOMMU Field I/O devices. | APIRI.TCS.3130 | APIRI.TPS.1001 |
| APIR3.1.2[132] | If multiple application programs attempt to set the state of the Load Switch Flash bit, the API shall use the most recent state. | fio\_fiod\_mmu\_flash\_bit\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and per-application data to manage the Load Switch Flash bit setting to be sent via the relevant FIOMSG entity (Section 4.4.4) frame types for FIOMMU Field I/O devices. | APIRI.TCS.3130 | APIRI.TPS.1001 |
| APIR3.1.2[133] | The default value of the Load Switch Flash bit shall be 0. | fio\_fiod\_mmu\_flash\_bit\_set(3fio) | The FIOMAN entity (Section 4.4.3) of the design described in Section 4.1 provides functionality and per-application data to manage the Load Switch Flash bit setting to be sent via the relevant FIOMSG entity (Section 4.4.4) frame types for FIOMMU Field I/O devices. | APIRI.TCS.3130 | APIRI.TPS.1001 |
| APIR3.1.2[134] | When an application program exits or terminates for any reason, the API shall deregister the application program from the API (as defined in Item “d”). | fio\_deregister(3fio) | The fiodriver design described in Section 4.1 allows detection of normal and abnormal program termination for applications registered via the fio\_register() function of the FIOAPI library, and upon which detection a deregister sequence is enacted. | APIRI.TCS.7020 | APIRI.TPS.7020 |

## TOD Library Requirements to Validation Description Matrix

The following table shows the relationship between the requirements and associated TOD library functions of the API Standard and the test identifiers which represent the test documents for those requirements and functions.

| **Req ID** | **Req Description** | **ATC API Function** | **APIRI SDD Design Narrative** | **Test Plans** | **Test Procedures** |
| --- | --- | --- | --- | --- | --- |
| APIR3.2.2[10] | The API shall provide a function(s) to set and get the system time including the date, time, time zone, and DST information. | tod\_set(3tod),  tod\_get(3tod),  tod\_set\_dst\_state(3tod),  tod\_get\_dst\_state(3tod),  tod\_get\_dst\_info(3tod),  tod\_set\_dst\_info(3tod),  tod\_get\_timesrc(3tod),  tod\_set\_timesrc(3tod),  tod\_get\_timesrc\_freq(3tod), tod\_request\_tick\_signal(3tod), tod\_cancel\_tick\_signal(3tod), tod\_request\_onchange\_signal(3tod),  tod\_cancel\_onchange\_signal(3tod). | The TOD API Library (Section 5.4) wraps all ATC API calls to underlying Linux system calls or ATC 5201 Standard BSP driver ioctl calls to enable TOD control. | APIRI.TCS.4010 | APIRI.TPS.4010 |
| APIR3.2.2[11] | The system time function(s) shall not require "root" permissions to operate. | tod\_set(3tod),  tod\_set\_dst\_state(3tod),  tod\_set\_dst\_info(3tod), | **Open Issue: This requirement should be debated as a security risk** | APIRI.TCS.4010 | APIRI.TPS.4010 |

## APIRI Test Design Specifications

### Test Design Specification 1 - Test All APIRI FPUI Features

#### Test Design Specification Identifier

The identifier for this Test Design Specification is APIRI.TDS.2001.

#### Features To Be Tested

This Test Design Specification will test all FPUI features of the API Reference Implementation (APIRI) which are subject to testing for validation (i.e., which have a numbered Test Case of the form APIRI.TCS.nnnn).

Specific features to be tested (referenced to the API Standard) and assigned Test Cases to be used can be found in Appendix 16.1.

#### Approach Refinements

All test cases will be tested using the general approach as defined in this test plan and as further refined in Test Procedure Specification APIRI.TPS.0001 unless otherwise noted in an individual test case specification.

#### Test Identification

All test documents to be used by this Test Design Specification can be found in Section 3, Table 1.

#### Feature Pass/Fail Criteria

This Test Design Specification will have considered to have passed if and only if every individual test case passes according to it’s own pass/fail criteria as well as any pass/fail criteria associated with the test procedure used to execute the test case.

### Test Design Specification 2 - Test All APIRI FIO Features

#### Test Design Specification Identifier

The identifier for this Test Design Specification is APIRI.TDS.3001.

#### Features To Be Tested

This Test Design Specification will test all FIO features of the API Reference Implementation (APIRI) which are subject to testing for validation (i.e., which have a numbered Test Case of the form APIRI.TCS.nnnn)..

Specific features to be tested (referenced to the API Standard) and assigned Test Cases to be used can be found in Appendix 16.2.

#### Approach Refinements

All test cases will be tested using the general approach as defined in this test plan and as further refined in Test Procedure Specification APIRI.TPS.0001 unless otherwise noted in an individual test case specification.

#### Test Identification

All test documents to be used by this Test Design Specification can be found in Section 3, Table 1.

#### Feature Pass/Fail Criteria

This Test Design Specification will have considered to have passed if and only if every individual test case passes according to it’s own pass/fail criteria as well as any pass/fail criteria associated with the test procedure used to execute the test case.

### Test Design Specification 3 - Test All APIRI TOD Features

#### Test Design Specification Identifier

The identifier for this Test Design Specification is APIRI.TDS.4001.

#### Features To Be Tested

This Test Design Specification will test all TOD features of the API Reference Implementation (APIRI) which are subject to testing for validation (i.e., which have a numbered Test Case of the form APIRI.TCS.nnnn)..

Specific features to be tested (referenced to the API Standard) and assigned Test Cases to be used can be found in Appendix 16.3.

#### Approach Refinements

All test cases will be tested using the general approach as defined in this test plan and as further refined in Test Procedure Specification APIRI.TPS.0001 unless otherwise noted in an individual test case specification.

#### Test Identification

All test documents to be used by this Test Design Specification can be found in Section 3, Table 1.

#### Feature Pass/Fail Criteria

This Test Design Specification will have considered to have passed if and only if every individual test case passes according to it’s own pass/fail criteria as well as any pass/fail criteria associated with the test procedure used to execute the test case.