***A User Comment Draft of the Joint Committee on the ATC***
***For Distribution by AASHTO, ITE and NEMA***

# ATC API SRS v02.04

# Advanced Transportation Controller (ATC) Application Programming Interface (API) Software Requirements Specification (SRS)

**December 9, 2005**



*Published by*

**American Association of State Highway and Transportation Officials (AASHTO)**
444 North Capitol St., N.W., Suite 249
Washington, DC, 20001

**Institute of Transportation Engineers (ITE)**
1099 14th St. N.W., Suite 300 West
Washington, DC, 20005-3438

**National Electrical Manufacturers Association (NEMA)**
1300 N. 17th Street, Suite 1847
Rosslyn, Virginia 22209-3801

# NOTICE

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

# REVISION NOTES

| DATE | NOTE |
|---|---|
| 11/19/04 | Initial draft of the ATC API SRS capturing the user needs for the entire API Standard. |
| 03/9/05 | Added requirements for the Front Panel Manager to Section 3. Added description of ATC Engine Board to Section 2. Added index. Revised overall document. |
| 03/15/05 | Updated traceability matrix. |
| 06/30/05 | Updated per direction from API WG Meeting 03/31/05 and modifications to the ATC Controller Standard made April – June 2005. |
| 10/31/05 | Updated per direction from the API WG Teleconference 07/29/05. |
| 12/09/05 | Updated per direction from the ATC JC Teleconference 12/09/05. |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

# CONTENTS

# 1 INTRODUCTION

This section provides an introduction for this document. It includes sections on "Purpose"; "Scope"; "Definitions, Acronyms, and Abbreviations"; "References"; and "Overview".

## 1.1 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 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 Software Requirements Specification (SRS) establishes the requirements for an interface for application programs designed to operate on ATC controller units. It has been prepared by the ATC API Working Group (WG), a technical subcommittee of the ATC JC. It establishes a common understanding of the user needs and requirements of this software for:

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

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

c) The public who benefits in the software applications that run on ATC equipment and directly or indirectly pays for these products.

## 1.2 Scope

The ATC Controller Standard defines a controller that can grow with technology. It is made up of a central processing unit (CPU), an operating system (OS), memory, external and internal interfaces, and other associated hardware necessary to create an embedded transportation computing platform. The goal of the requirements described in this SRS is to produce software that, when combined with the ATC OS, forms a universal interface for application programs. This interface will allow application programs to be written so that they may run on any ATC controller unit regardless of the manufacturer. It will also create a software environment that will allow multiple

applications to be interoperable on a single controller unit by sharing the fixed resources of the controller. This interface will be known as the ATC Application Programming Interface (API). The functions and operations of the API will be specified in the API Standard.

## 1.3 Definitions, Acronyms and Abbreviations

| TERM | DEFINITION |
|---|---|
| 170 | A traffic control device that meets one of the California Department of Transportation (Caltrans) standards for Model 170 traffic control devices. |
| 2070 | A traffic control device that meets one of the California Department of Transportation (Caltrans) standards for Model 2070 traffic control devices or one of the standards for the ATC 2070 traffic control devices. |
| AASHTO | American Association of State Highway and Transportation Officials |
| API | Application Programming Interface |
| ATC | Advanced Transportation Controller |
| ASCII | American Standard Code for Information Interchange. A standard character-coding scheme used by most computers to display letters, digits and special characters. See ANSI-X3.4-1986(R 1997). |
| 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 operating system. |
| CPU | Central Processing Unit |
| Device Driver | A software routine that links a peripheral device to the operating system. It acts like a translator between a device and the applications that use it. |
| DRAM | Dynamic Random Access Memory |
| EEPROM | Electrically erasable, programmable, read-only memory. EEPROMs differ from DRAMs in that the memory is saved even if electrical power is lost. |
| FAT32 | Microsoft Windows 32-bit file system format (File Allocation Table 32) |
| FHWA | Federal Highway Administration |
| I/O | Input/Output |
| IEC | International Engineering Consortium |

| | |
|---|---|
| IEEE | Institute of Electrical and Electronics Engineers |
| ISO | International Organization for Standardization |
| ITE | Institute of Transportation Engineers |
| ITS | Intelligent Transportation Systems |
| JC | Joint Committee |
| LED | Light Emitting Diode |
| Mb/s | Mega/Million Bits per Second |
| ms | Microsecond. One thousandth ($10^{-3}$) of a second. |
| NEMA | National Electrical Manufacturers Association |
| NEMA Controller | A traffic control device that meets one of the NEMA standards for traffic control devices. |
| Operational User | A technician or transportation engineer who uses the controller operationally to perform its application functions. |
| OS | operating system |
| RAM | Random Access Memory |
| RTC | Real-Time Clock |
| SDD | Software Design Document |
| SDO | Standards Development Organization |
| SRAM | Static Random Access Memory |
| SRS | Software Requirements Specification |
| SW | Software |
| TTL | Transistor Transistor Logic |
| TUI | Text User Interface |
| WG | Working Group |
| USDOT | United States Department of Transportation |
| User Developer | The “user developer” is a software developer that designs and develops programs for controllers. |

## 1.5 References

The documents referenced by this SRS are listed below.

“ATC Controller Standard Revision 5.1d”. ATC JC, 5 October, 2005. Available from the Institute of Transportation Engineers.[1]

[1] Note that this designation will be updated based on the final ballot version of the ATC controller unit standard.

"ATC Standard for the Type 2070 Controller v02.03", ATC JC, 12 March 2004. Available from the Institute of Transportation Engineers.

"GNU Coding Standards", 1 January 2005. Available from Free Software Foundation, Inc.

"IEEE Recommended Practice for Software Design Descriptions", IEEE Std 1016-1998. Available from the Institute of Electrical and Electronics Engineers.

"IEEE Recommended Practice for Software Requirements Specifications", IEEE Std 830-1998. Available from the Institute of Electrical and Electronics Engineers.

"Intelligent Transportation System (ITS) Standard Specification for Roadside Cabinets", ATC JC, March 26, 2003. Available from the Institute of Transportation Engineers.

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

## 1.6 Overview

This SRS is made up of three sections, appendixes and an index. Section 1, "Introduction", provides an overview of the entire SRS. Section 2, "Overall Description", provides background information and the user needs for the requirements defined in the subsequent section. Section 3, "Specific Requirements", defines the requirements for the ATC API.

# 2 OVERALL DESCRIPTION

This section provides the description of user needs for the ATC API. It includes sections on “Product Perspective”; “Product Functions”; “User Characteristics”; “Constraints”; “Assumptions and Dependencies”; and “Apportioning of Requirements”. User needs within a section are identified by numbers in square brackets (Ex. “Stated user need … [1].”).

## 2.1 Product Perspective

The ATC API is one of four standards efforts under the direction of the ATC JC. It is the intent of the ATC JC to create a set of standards that will fulfill the computing needs for transportation applications of today and also provide an architecture that can grow to help meet the transportation needs of the future. Other standards within the ATC program include the ATC/2070 Standard, the ATC Controller Standard, and the Intelligent Transportation Systems (ITS) Cabinet.

Historically, application software written by one manufacturer does not run on equipment from another manufacturer. This is not just on equipment as different as 170 and NEMA controllers, but even between NEMA controllers from different manufacturers. While there have been software portability successes within the 170 and 2070 controller markets, the hardware specifications of these architectures do not lend themselves to adopt the latest hardware or software technologies nor will they create the market that will be necessary to support the broad range of applications required in the future. See Figure 1.

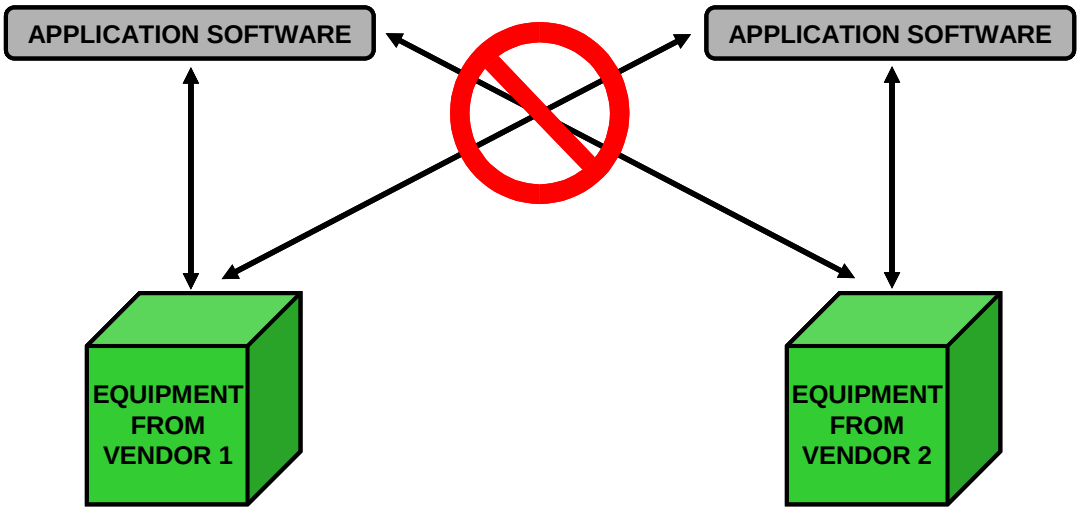


**Figure 1. Non-portable software traditionally used in transportation industry.**

When combined with the ATC operating system, the API forms an open architecture software (SW) platform that acts as a universal interface between application programs and the ATC controller units [1]. Open architecture, as used, means that the function specifications and associated source code will be available to everyone. This includes the ATC operating system and the API SW. The controller unit includes the central processing unit (CPU), the operating system (OS), memory, external and internal interfaces, and all other associated hardware that is defined by the ATC Controller Standard. It is said that the API "abstracts" the application software from the ATC hardware allowing applications to be written that can be made to operate on any ATC (regardless of manufacturer). In older controller architectures, source code would require considerable modification and, in some cases, to be completely rewritten to run on a different vendor's platform. The API, when used with the ATC OS, facilitates portability by requiring only modest efforts on the part of the developer such as recompiling and linking source code for a particular processor [2]. See Figure 2.

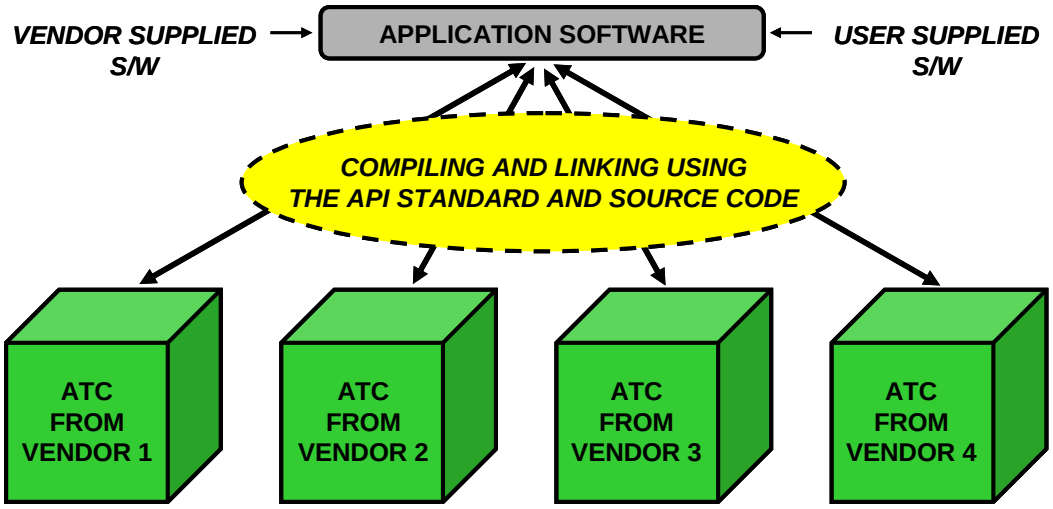


**Figure 2. Application portability using the API Standard.**

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. The Engine Board contains the following items:

a) CPU;
b) Linux OS and Device Drivers;
c) Non-Volatile (Flash) Memory;
d) Dynamic and Static RAM (DRAM and SRAM);

e) Real-Time Clock (RTC);
f) Two Ethernet ports;
g) One universal serial bus (USB) port that is used for a Portable Memory Device; and
h) Eight serial ports (some are designated for special interfaces and others general purpose).

The Engine Board plugs into a "Host Module" which 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. Figure 3 shows how the Engine Board can be used to create ATC controllers that work within different families of traffic controller equipment. This concept will also allow more powerful Engine Boards to be deployed in the future without changing the overall controller and cabinet architecture.

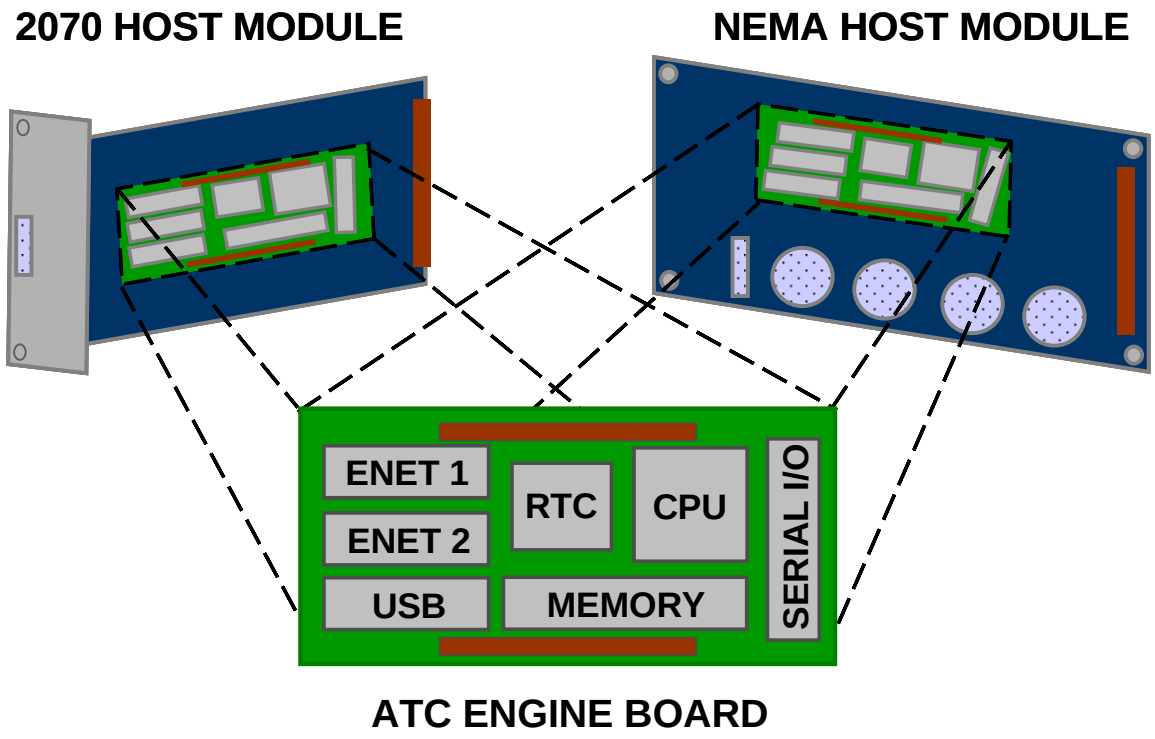


**Figure 3. ATC Engine Board being used to support different families of controllers.**

The ATC Controller Standard specifies a minimum level of real-time processing capability for the Engine Board. It also specifies the minimum physical and communication requirements for the Host Module. Some of the communication ports of

the Engine Board are general-purpose, intended to be used via the OS as needed by application programs. Other ports are specific to traffic controller devices. These ports are managed by the API and are intended to be shared across concurrently running application programs. Figure 4 shows the Engine Board communication ports and their intended use. As shown, the API must provide management functions for the Front Panel and the Field I/O Devices [3]. Using the API, future advances in processing power can be applied to Engine Boards, installed into existing ATC controllers, and still operate the application programs of the transportation system (recompiling may be required as shown in Figure 2).

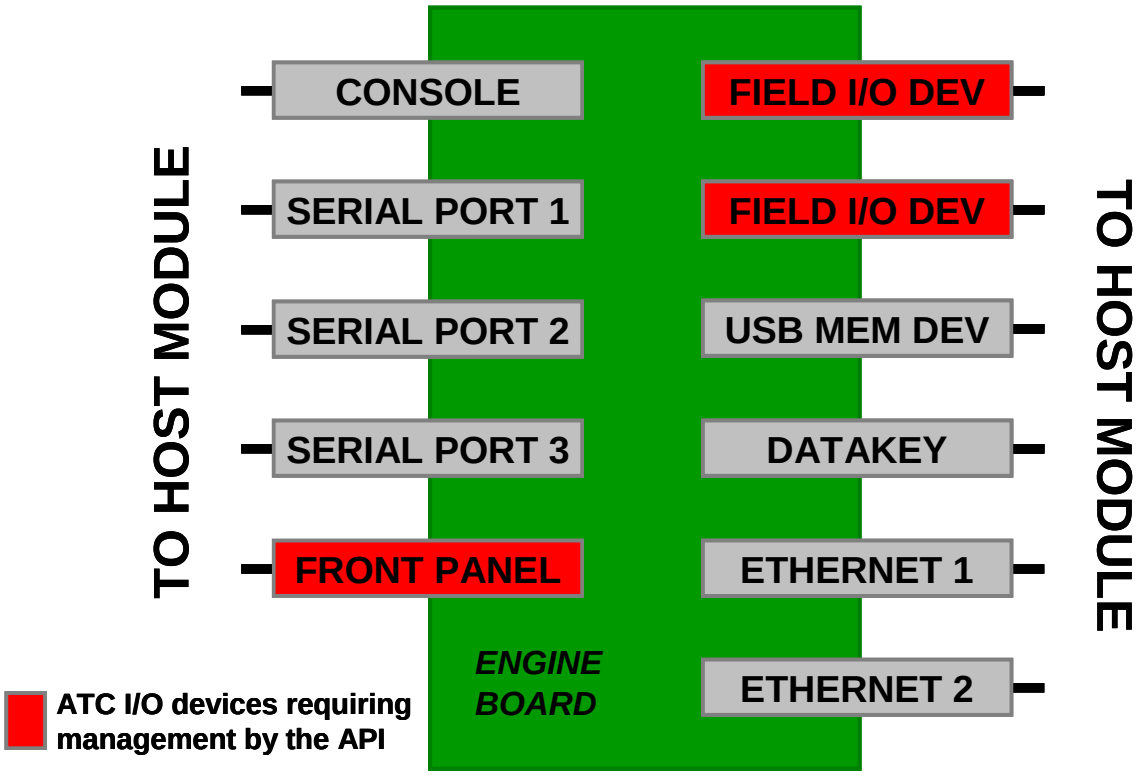


**Figure 4. ATC Engine Board I/O supported by the API software.**

Figure 5 illustrates the organization and layered architecture of the ATC software. The "Linux OS 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 OS to operate on the ATC hardware. "API Software" refers to the software to be developed in response to this SRS. As shown in Figure 5, both users and application

programs will use the API to interface to ATC controller units [4]. The API Standard specifies the API Software by describing its behavior and identifying how users and application programs interface to it [5]

The division of the ATC software into layers helps to insure consistent behavior of the software environment between ATC architectures and also provides a migration path to new ATCs in the future. The relationship between the Hardware Layer and ATC BSP Layer is maintained, for the most part, by the Linux operating system community of users. Linux source code licenses are free to the public and there are strong market incentives for Linux users to maintain the Linux standard and insure consistent functionality of the Linux commands for the operating system. The relationship between the ATC BSP Layer and the API Software Layer is maintained by the transportation community. Functions in the API Software Layer will be written so that they will only access the ATC unit through the functions in the ATC BSP Layer [6]. The API Software Layer functions will be written in the C programming language and be available in the public domain for free use by transportation industry software developers [7]. If programs written for the Application Layer only reference the ATC unit through the functions specified in the API Software Layer and ATC BSP Layer, they will be able to operate on any ATC provided the source code is recompiled for the target ATC's processor.

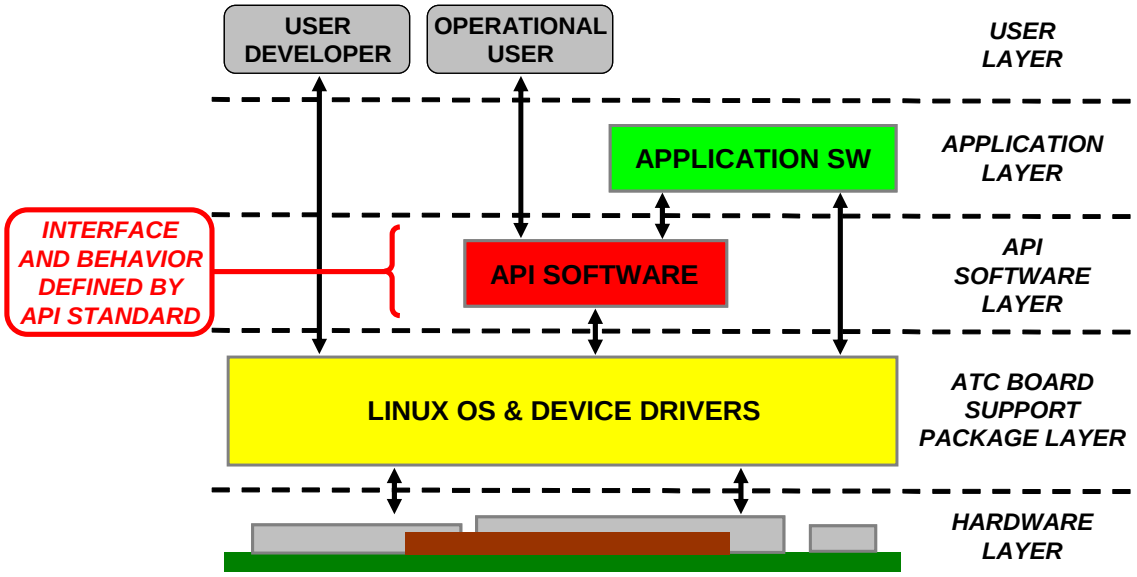


**Figure 5. ATC software layered organization.**

### 2.1.1 System Interfaces

There are no system interfaces addressed by this SRS. A system, as intended here, is a collection of related hardware and software products whose behavior satisfies an operational need. By this definition, a controller unit and its operational software form a system. The API is a component of this system but does not directly address any system level interfaces.

### 2.1.2 User Interfaces

The ATC Controller Standard defines a Front Panel interface for operational users to interact with the application programs that are the primary functions of the controller unit within the transportation system. Examples of these programs are intersection control, field management stations, ramp metering, radiation detection, etc. Historically, a controller unit would support only one application program at a time and application writers could assume they had unimpeded and constant access to a controller unit's Front Panel. It is intended that ATC controller units will support the use of multiple application programs running concurrently. This will require the API to support a user interface that will provide operational users the ability to select from a set of active application programs on the ATC controller unit and to interact with the programs individually [1]. The ATC Controller Standard describes the physical characteristics and the character set to be supported [2].

The API software must also provide C functions that will allow application software to access and control the Front Panel Interface programmatically [3]. User developers will use these functions in the source code of their application programs. A Console Interface is provided to allow User Developers to load programs and maintain both application software and the Linux environment itself. This interface is defined in the ATC Controller Standard and is outside the scope of the API as shown in Figure 5.

### 2.1.3 Hardware Interfaces

The API software manages the resources of the ATC controller unit which are intended to be shared between concurrently operating application programs and whose functionality is not inherently supported in Linux (or by the device drivers created for the ATC). These managed resources include the Front Panel (see Section 2.1.2 User Interfaces) and the Field I/O Devices [1]. The API does not communicate directly to these ATC resources but through the Linux operating system and associated device drivers provided with the ATC controller unit [2]. The hardware requirements for these interfaces are described in the ATC Controller Standard.

### 2.1.4 Software Interfaces

There are no software interfaces required other than the Linux OS defined in the ATC Controller Standard.

### 2.1.5 Communications Interfaces

The ATC Controller Standard states that ATC controller units will have three general purpose EIA 485 serial ports and two Ethernet ports used for communications purposes. The ATC controller unit will provide access to these communications interfaces via the Linux OS and Device Drivers [1].

### 2.1.6 Memory Constraints

The API must operate effectively within the memory constraints defined for ATC controller units in the ATC Controller Standard [1].

### 2.1.7 Operations

See Section 2.1.2 User Interfaces.

### 2.1.8 Site Adaptation Requirements

There are no site adaptation requirements.

## 2.2 Product Functions

The API Software Layer and ATC BSP Layer discussed in the Section 2.1 are further divided into five functional areas as shown in Figure 6. The ATC Controller and API Standards specify the requirements for these areas providing both a programmatic interface for application software and a user interface for operational use of the application software.

- The “Linux Kernel” provides general OS 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.

- "Linux Device Drivers" refers to low level software that is freely available in the Linux community for use with common hardware components operating in a standard fashion.

- "ATC Device Drivers" refers to low level software not included in standard Linux distributions that is necessary for ATC-specific devices to operate in a Linux OS environment.

- "API Managers" refers to the functions that are intended to manage the operation of the ATC hardware interfaces listed in Section 2.1.3.

- "API Utilities" refers to the functions invoked by ATC users that will be used to configure the manager functions.

The arrows in Figure 6 show the logical interfaces of the functional areas. Operational users will need to interface with applications through the API to perform the primary tasks assigned to the controller unit [1]. Utility functions will be developed, as necessary, to configure the API Managers for proper operation [2]. Application Software will interface to the Linux Kernel for those functions common to operating systems. Application Software will interface to the API Manager Functions and the Linux Kernel for access to the ATC-specific devices. The API Manager Functions and the API User Utilities, will access the ATC controller hardware through the Linux Kernel [3]. The Linux Kernel and Linux Device Drivers are defined in the ATC Controller Standard, Annex A, as a profile (selected subset) of commonly available Linux OS software [4]. The specifications for the ATC Device Drivers are defined in the ATC Controller Standard, Annex B [5]. The ATC Device Driver software is to be developed by manufacturers as appropriate for the architecture of their ATC hardware [6]. The software for the API Manager Functions and the API User Utilities will be developed in response to this SRS and specified in the API Standard [7].

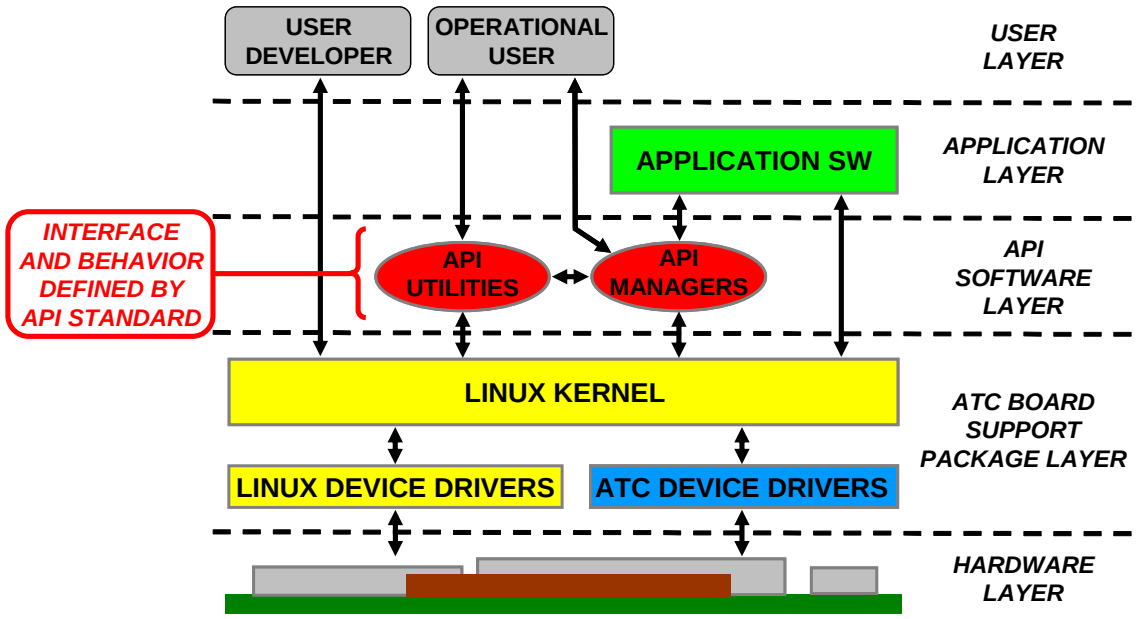


**Figure 6. ATC API functional areas.**

Functions in the area Linux Kernel include those which are called from within software programs and those that are invoked by the user via the console interface. The Linux Kernel is not considered a part of the API software but the high-level user needs are described here to identify assumptions made in the API Standard.

a) The Linux Kernel will provide software signaling services including creating/deleting, scheduling, and sending/receiving software events [8]. These services must include single occurrence, multiple occurrence, and signal by date [9].

b) The Linux Kernel will provide process management services which allow processes to be spawned from other processes, process priority management, waiting on a process, process sleep, process termination, retrieving process information, and exiting a process [10].

c) The Linux Kernel will provide shared memory capabilities between processes including allocation, access control and deletion [11].

d) The Linux Kernel will provide semaphore and mutex synchronization services including initialization/termination, acquisition/release, increment/decrement, and waiting [12].

e) The Linux Kernel will provide serial service functions including opening/closing a port, reading/writing, setting/getting properties, status checking, and signal notification [13].

f) The Linux Kernel will provide pipe service functions including opening/closing a pipe, reading/writing, setting/getting properties, status checking, and signal notification [14].

g) The Linux Kernel will provide system time functions including getting/setting the system time [15].

h) The Linux Kernel will provide capabilities to set/get system global variables [16].

i) The Linux Kernel will provide network configuration capabilities with TCP/IP socket communication services including open, close, read, and write [17].

j) The Linux Kernel will provide basic shell user interface and scripting mechanism via a console port [18].

The functions in Linux Device Drivers support the Linux OS functions used in the Linux Kernel that operate on common hardware components. The Linux Device Drivers are specified in the ATC Controller Standard, Annex A.

Functions in the area ATC Device Drivers are driver programs (software) that are necessary to interface to the ATC-specific hardware devices. A custom driver may have to be developed if a suitable driver is not available within the Linux open source community. These drivers are not considered a part of the API software but the high-level user needs are described here to identify assumptions made in the API Standard.

a) Datakey Interface Driver. This device driver must support all of the allowable key sizes as specified in the ATC Controller Standard [19]. The device driver must allow multiple applications to access the data key as a file system [20].

b) Front Panel Interface Driver. This device driver provides a serial interface to the controller's Front Panel as described in the ATC Controller Standard [21]. The device driver will be used by the API Front Panel Manager exclusively.

c) Serial I/O Interface Drivers. These device drivers provide synchronous and asynchronous serial ports as described in the ATC Controller

Standard [22]. These drivers are used by the API to manage the Field I/O devices.

d) FLASH Interface Driver. This device driver provides a flash file system interface as described in the ATC Controller Standard [23]. The device driver must allow multiple applications to access the file system concurrently [24].

e) SRAM Interface Driver. This device driver provides a file system interface as described in the ATC Controller Standard [25].

f) USB Interface Driver. This device driver provides FAT File I/O services for the USB Portable Memory Device [26].

g) Real-time Clock Interface Driver. This device driver provides access to the battery-backed real-time clock on the ATC Engine Board [27].

h) Miscellaneous Parallel I/O Drivers. These device drivers provide access to specific I/O pins such as POWERDOWN, CPU_RESET, CPU_ACTIVE, and DKEY_PRESENT as described in the ATC Controller Standard [28]. The CPU_ACTIVE device driver controls the CPU ACTIVE LED Indicator. Access to this device is managed by the API (see Section 3.3.1.2).

i) Serial Peripheral Interface Drivers (SPI). These device drivers provide an interface to the EEPROM and also work with the Datakey as described in the ATC Controller Standard [29].

j) Ethernet Interface Driver. This driver provides an interface to the Ethernet ports of the Engine Board as described by the ATC Controller Standard [30].

Functions in the area API Managers provide the interface to and management of the shared resources of the ATC controller not sufficiently covered by functions inherent in Linux or provided for in any custom driver. The software is to be developed in response to this SRS.

a) Front Panel Manager. This set of functions provides a user interface which allows programs running concurrently on an ATC controller to share the controller's Front Panel through the dedicated Front Panel Port of the ATC Engine Board [31].

b) Field I/O Manager. This set of functions gives concurrently running programs the capability to communicate with field devices connected to an

ATC controller through the dedicated Field I/O ports of the ATC Engine Board [32].

Functions in the area API User Utilities provide the operational user tools in which to configure and maintain the API software installed on an ATC. While most of these functions will be based on the detailed design of the managers, there is a need for operational users to have access to the API software version and configuration information [33].

## 2.3 User Characteristics

There are two types of users of the API. The “operational user” is a technician or transportation engineer that uses the controller operationally to perform its application functions. If they are engineers, they typically have civil or mechanical engineering backgrounds. The “user developer” is a software developer that designs and develops programs for controllers. These individuals typically have software, computer, or electrical engineering backgrounds.

## 2.4 Constraints

The API must operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard [1]. Any software developed as part of this effort will be written in the C programming language as described by “ISO/IEC 9899:1999” commonly referred to as the C99 Standard [2]. All software developed or referenced as part of the API must be attainable via the public domain [3]. The operational look and feel of user interfaces developed for the API should be consistent with each other [4]. If API functions have a similar operation to existing Linux functions, they should have a similar name and argument style to those functions [5].

## 2.5 Assumptions and Dependencies

It is assumed that the API will operate on an ATC controller unit as defined by the ATC Controller Standard. By definition, the requirements in this SRS are to be testable. To test the functionality of each function drawn from the Linux OS and included in the Linux Kernel, Linux Device Drivers and ATC Device Drivers is beyond the scope of this effort. It is assumed that proper adherence to the ATC Specification for these functional areas will be assured by certification from the manufacturer.

## 2.6 Apportioning of Requirements

Due to funding issues, development of the API software may be performed in stages. If this is the case, the API Management Functions and any interfacing API Utilities will be developed together to insure a cohesive design.

# 3 SPECIFIC REQUIREMENTS

This section specifies the requirements for the ATC API. It includes sections on “Linux Kernel and Linux Device Driver Requirements”; “ATC Device Driver Requirements”; “API Manager Requirements”; “API User Utility Requirements”; “Performance Requirements”; “Design Constraints”; “Software System Attributes”; and “Other Requirements”. Requirements within a section are identified by numbers in square brackets (Ex. “Stated requirement … [1].”).

## 3.1 Linux Kernel and Linux Device Driver Requirements

The requirements in this section have been moved to the ATC Controller Standard, Annex A.

## 3.2 ATC Device Driver Requirements

The requirements in this section have been moved to the ATC Controller Standard, Annex B.

## 3.3 API Manager Requirements

### 3.3.1 Front Panel Manager Requirements

The API shall provide a text-based user interface capability to allow programs running concurrently on an ATC controller unit to share the controller’s Front Panel display [1]. 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 [2]. The display size of the windows shall be based on the character size of the controller’s Front Panel display (if one exists) [3]. The display size of the windows shall have a minimum of 4x40 characters and a maximum of 24 x 80 characters [4]. If no physical display exists, the API shall operate as if it has a display with a size of 8 x 40 characters [5]. Only one window shall be exposed at a time on the Front Panel display [6]. When a window is exposed, the API shall display the character representation of the window on the Front Panel display (if one exists) [7]. The application associated with the window displayed shall receive the characters input from the Front Panel input device (Ex. keyboard or keypad) [8]. When an application window is exposed, it is said to have “focus”. The window which has focus when the controller is powered up is called the “default window.” The API shall support the display character set as defined in the ATC Controller Standard, Section 7.1.4 [9]. Screen attributes described by the ATC

Controller Standard, Section 7.1.4, shall be maintained for each window independently [10]. Each window shall have separate input and output buffers unique from other windows [11].

### 3.3.1.1 Front Panel Manager Window Requirements

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 [1]. Application names associated with each window shall be listed [2]. The application names shall be limited to 16 characters [3]. If there is no application associated with a window, the window number shall be listed with a blank application name [4]. The default Front Panel Manager Window size shall be 8 x 40 characters (rows x columns) with the format as shown in Figure 7 [5].

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | F | R | O | N | T | | P | A | N | E | L | | M | A | N | A | G | E | R | | | | | | | | | | | |
| 2 | S | E | L | E | C | T | | W | I | N | D | O | W | : | | 0 | - | F | | | | S | E | T | | D | E | F | A | U | L | T | : | | * | , | 0 | - | F | |
| 3 | 0 | * | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *0* | | | 1 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *1* | | |
| 4 | 2 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *2* | | | 3 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *3* | | |
| 5 | 4 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *4* | | | 5 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *5* | | |
| 6 | 6 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *6* | | | 7 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *7* | | |
| 7 | 8 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *8* | | | 9 | | *a* | *p* | *p* | *l* | *i* | *c* | *a* | *t* | *i* | *o* | *n* | *n* | *a* | *m* | *e* | *9* | | |
| 8 | [ | U | P | / | D | N | | A | R | R | O | W | ] | | | | | | | | | [ | C | O | N | F | I | G | | I | N | F | O | - | | N | E | X | T | ] |

**Shaded areas ☐ are row and column headings. They are not a part of the actual display.**
***applicationname#* refers to the application name associated with the window.**
**If there is no task assigned to a window, it will be blank.**
*** indicates the default window displayed when the controller unit is started. It user settable.**

**Figure 7. ATC Front Panel Manager Window.**

Key sequences used in the requirements below use the following conventions:
- Curled brackets ({}) are used to delimit a key sequence.
- Square brackets ([]) signify a key from a range or set such as [0-F] or [A, B, C].
- A range of [0-F] signifies the hexadecimal characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
- Angle brackets (<>) are used to identify a special character such as <ESC> or <NEXT>.
- A comma (,) used in a key sequence is not entered but is used to delimit keys.

a) If the operational user has not set the default window, the Front Panel Manager Window shall be the default window [6].

b) The default window shall be settable by the operational user from the Front Panel Manager Window by pressing {*,[0-F]} [7].

c) The operational user shall be capable of setting the default window to the Front Panel Manager by pressing {*,*} from the Front Panel Manager [8].

d) The default window shall be designated by a star “*” character next to the window number [9].

e) 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 in operation [10].

f) The operational user shall be able to enter {**} by pressing an asterisk (*) twice within a 0.5 second time period [11].

g) 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 [12].

h) The only possible window selections for focus from the Front Panel Manager Window shall be itself, the ATC Configuration Window (see Section 3.4), or a window assigned to an application [13].

i) If the Front Panel Manager Window is the default window, no asterisk shall be displayed next to any task name in the Front Panel Manager Window [14].

j) The operational user shall be able to put the ATC Configuration Window in focus by pressing {<NEXT>} in the Front Panel Manager Window [15].

k) The top two lines and bottom line of the Front Panel Manager Window shall be fixed as shown in Figure 7 [16].

l) 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 [17].

m) 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 [18].

#### 3.3.1.2 Front Panel Manager Software Interface Requirements

The API provides a software interface enabling application programs to manipulate windows as described below.

a) The API shall provide a function to return the dimensions of a window in terms of number of lines and number columns [1].

b) The API shall provide a function to open a window and register a name for display on the Front Panel Manager Window [2]. An application shall be able to open multiple windows providing the resource is available [3]. The API shall provide the ability for an application to reserve exclusive access to the Aux Switch (see ATC Controller Standard, Section 7.1.4) [4]. This application shall maintain exclusive access to the Aux Switch even if the application has no window in focus [5].

c) The API shall provide a function to close a window and release the resource for other application programs [6].

d) 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 [7].

e) 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 [8].

f) The API shall provide a function that is used to determine if there is data in the input buffer of a window [9].

g) The API shall provide a function to read a queued character from the input buffer of a window [10].

h) The API shall provide a function to write a character to the current cursor position of a window [11].

i) The API shall provide a function to write a character to a window at a position defined by column and line number [12].

j) The API shall provide a function to write a string to a window at the current cursor position [13].

k) The API shall provide a function to write a string to a window at a starting position defined by column number and line number [14].

l) The API shall provide a function to write a buffer of characters to a window at the current cursor position [15].

m) 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 [16].

n) The API shall provide a function to set the cursor position of a window defined by column and line number [17].

o) The API shall provide a function to return the cursor position of the window defined by column and line number [18].

p) If a window was registered with access to the Aux Switch, the API shall provide a function to return its status [19].

q) The API shall provide a function to compose special characters as described by the ATC Controller Standard, Section 7.1.4 [20].

r) The API shall support the display of a composed character in the same manner as any other valid character [21].

s) The API shall provide a function to clear a window that will operate on a window whether it is in or out of focus [22].

t) The API shall provide a function to refresh a window that will operate on a window whether it is in or out of focus [23].

u) The bell of the controller's Front Panel shall be activated only if a bell character, ^G (hex value 07), is sent to a window that has focus [24]. Otherwise, the bell character shall be ignored by the API [25].

v) 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 [26].

w) Display configuration and inquiry command codes (escape sequences) specified in the ATC Controller Standard, Section 7.1.4, shall be supported as separate functions in the API [27].

x) Application programs shall be able to interpret all 2070 controller keys as individual key codes [28]. The escape sequences representing keys that do not have standard ASCII character codes on a 2070 shall be mapped to specific character codes in the API as shown in Table 1 [29].

y) The ATC Controller Standard, Section 7.1.4, 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. If applications use graphics on a window, the API is not responsible for redisplaying these graphics when a window is refreshed or goes out/in to focus [30]. Note: Application programs may store graphics commands used on a window that is out of focus and execute them once the window has gone into focus.

z) The API shall provide a mechanism to alert programs when their associated windows are in and out of focus [31].

aa) The API shall provide a function which applications may use to determine if their window is in focus [32].

bb) The API shall provide a function to set the default window [33].

cc) The API shall provide a mechanism to allow application programs to detect the presence or absence of a front panel [34]. The API shall recognize the presence or absence of the front panel in 5 seconds [35]. The API shall provide a mechanism to alert application programs of a change in the presence or absence of the front panel [36].

dd) The API shall provide a mechanism to alert application programs of a change in window size [37].

ee) 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 [38].

ff) The API shall provide a function to illuminate or extinguish the CPU ACTIVE LED described in the ATC Controller Standard, Section 7 [39]. The function shall only operate for application programs with a window in focus [40].

| Character Pressed on ATC Keypad | Escape Sequence Received By API | Code Returned by API Function to Application Program |
|---|---|---|
| Up Arrow | 0x1B 0x5B 0x41 | 0x80 |
| Down Arrow | 0x1B 0x5B 0x42 | 0x81 |
| Right Arrow | 0x1B 0x5B 0x43 | 0x82 |
| Left Arrow | 0x1B 0x5B 0x44 | 0x83 |
| Next | 0x1B 0x4F 0x50 | 0x84 |
| Yes | 0x1B 0x4F 0x51 | 0x85 |
| No | 0x1B 0x4F 0x52 | 0x86 |
| Aux On | 0x1B 0x4F 0x54 | 0x87 |
| Aux Off | 0x1B 0x4F 0x55 | 0x88 |

**Table 1. ATC controller escape sequences mapped to API character codes.**

### 3.3.2 Field I/O Manager Requirements

The API provides a software interface enabling concurrently running application programs to communicate with field devices as described in the ATC Controller Standard, Section 8. API requirements for this interface are described below.

a) The API shall have exclusive access to the serial communications ports of the ATC Engine Board that are designated for Field I/O devices [1].

b) The API shall provide functions which allow application programs to share access to the I/O points of various transportation field devices including:
   i) the 2070 controller modules 2070-2A, 2070-2N and 2070-8 as described by the ATC 2070 Standard (see Section 1.5 of this document);
   ii) the Serial Interface Unit (SIU) as described in the ITS Cabinet Standard (see Section 1.5 of this document); and
   iii) the Bus Interface Unit (BIU) and Malfunction Management Unit (MMU) as described in the NEMA TS-2 Standard (see Section 1.5 of this document) [2].

c) The API shall support communication to multiple Field I/O devices provided the devices on the same serial port have compatible communication attributes [3].

d) The API shall provide functions which allow application programs to open and close “virtual ports” for supported Field I/O device [4].

e) The API shall poll Field I/O devices to check to the state of their input and output points [5]. The API shall provide functions which allow application programs to set the polling rate of a Field I/O device [6]. If multiple applications set the polling rate, the highest rate shall be used [7]. The maximum polling rate shall be 10 milliseconds [8].

f) The API shall provide functions to allow application programs to reserve or register output points of a Field I/O device for “write access” [9]. The API shall restrict write access to the output points to a single application program [10]. The API shall provide functions to allow application programs to set the state (write operation) of their registered output points [11]. The API shall provide an error code signifying the status of the write operation [12].

g) The API shall provide functions which allow multiple application programs to get the state (read operation) of the input and output points of each Field I/O device in operation [13]. This shall include both filtered and non-filtered inputs [14]. The API shall provide an error code signifying the status of the read operation [15].

h) The API shall maintain message frequencies according to the applicable standard for proper communication with the Field I/O devices in operation [16]

i) The API shall provide functions which allow application programs to check the status of a Field I/O device [17]. The API shall check the status of a Field I/O device on a once per second basis [18].

j) The API shall provide applications access to the Field I/O device transition buffers on a .5 second basis [19].

## 3.4 API User Utility Requirements

The API shall provide a function call *apiver()* that returns the API version number as a string [1]. The API shall provide additional utility functions to configure and maintain the API managers and software. Since these functions will based on the design of the managers, the detailed requirements for these functions cannot be determined in this SRS [2].

### 3.4.1 ATC Configuration Window Requirements

The API shall provide a window called the ATC Configuration Window [1]. Operational users shall be able to view ATC configuration information on this window provided by

the Linux *uname()* function (ATC Controller Standard, Section 2.2.5), the API's *apiver()* function, and the ATC Host Module EEPROM information (ATC Controller Standard, Annex B) [2]. The default ATC Configuration Window size shall be 8 x 40 characters with the format as show in Figure 8 [3].

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | A | T | C | | C | O | N | F | I | G | U | R | A | T | I | O | N | | I | N | F | O | R | M | A | T | I | O | N | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | [ | U | P | / | D | N | | A | R | R | O | W | ] | | | | | | | | [ | F | R | O | N | T | | P | A | N | E | L | - | | N | E | X | T | ] |

**Shaded areas ▢ are row and column headings. They are not a part of the actual display.**

**Figure 8. The ATC Configuration Window.**

a) The top two lines and bottom line of the ATC Configuration Window shall be fixed as shown in Figure 8 [4].

b) The number of lines between the second line and bottom lines used for displaying the ATC configuration information shall vary according to the size of the ATC display [5].

c) The operational user shall be able to scroll the ATC Configuration Window to view the configuration information using the up and down arrow keys of the controller keypad [6].

d) The operational user shall be able to put the Front Panel Manager in focus by pressing {<NEXT>} in the ATC Configuration Window [7].

## 3.5 Performance Requirements

Any performance requirements are identified in Sections 3.1 through 3.4.

## 3.6 Design Constraints

The API software shall operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard [1]. Any software developed as part of this effort shall be written in the C programming language as described by "ISO/IEC 9899:1999" commonly referred to as the C99 Standard [2]. Software produced in response to this SRS shall be documented using IEEE Std 1016-1998 and GNU Coding Standards, 1

January 2005 [3]. The API software shall only reference operating system commands and features that are available in the Linux environment defined by the ATC Controller Standard, Annex A and B [4].

## 3.7 Software Systems Attributes

### 3.7.1 Portability

The API software shall be portable such that only compilation and linking of the API source code should be necessary to operate the API on any controller that meets the ATC Controller Standard [1].

### 3.7.2 Consistency

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 [1]. If API functions have a similar operation to existing Linux functions, they shall have a similar name and argument style to those functions [2].

## 3.8 Other Requirements

All software developed for the API shall be attainable in the public domain [1]. Software developed in response to these requirements shall constitute the API software [2]. The API Standard shall specify the API software by describing its behavior and identifying how users and application programs interface to it [3].

# APPENDIXES

# Appendix A: Needs-To-Requirements Matrix

The following table shows the relationship between user needs and the requirements for the ATC Application Programming Interface. User needs and requirements have been given identifiers using the form ***dip[n]***. Where: ***d*** is the initials of the document where the need/requirement is stated; ***i*** is either "N" for user need or "R" for requirement; ***p*** refers to the paragraph where the need/requirement is found; and ***n*** is the number of the item within the paragraph. An identifier "SRSN2.1[3]" would refer to the third user need stated in section 2.1 of the Software Requirements Specification.

The "Validation" column provides guidance on the primary method of validating the API to the corresponding requirement. The possible techniques are as follows:

- Analysis – Indicates a study, investigation or calculation is necessary to validate the requirement.
- Certification – Indicates compliance or satisfaction of a requirement is guaranteed or certified by the product provider.
- Observation – Indicates the primary validation is performed by observing or inspecting a device or display.
- Test Program – Indicates the primary validation is performed by a test program or test suite.

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| SRSN2.1[1] | When combined with the ATC OS, the API forms an open architecture software (SW) platform that acts as a universal interface between application programs and the ATC controller units | SRSR3.6[1] | The API software shall operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard | Test Program. |
| | | SRSR3.6[4] | The API software shall only reference operating system commands and features that are available in the Linux environment defined by the ATC Controller Standard, Annex A and B | Analysis. |
| SRSN2.1[2] | The API, when used with the ATC OS, facilitates portability by requiring only modest efforts on the part of the developer such as recompiling and linking source code for a particular processor | SRSR3.6[2] | Any software developed as part of this effort shall be written in the C programming language as described by "ISO/IEC 9899:1999" commonly referred to as the C99 Standard | Observation. |
| | | SRSR3.6[3] | Software produced in response to this SRS shall be documented using IEEE Std 1016-1998 and GNU Coding Standards, 1 January 2005 | Observation. |
| | | SRSR3.7.1[1] | The API software shall be portable such that only compilation and linking of the API source code should be necessary to operate the API on any controller that meets the ATC Controller Standard | Analysis/Observation/Test Program. The API should be evaluated on a least two separate platforms from different vendors. |
| | | SRSR3.7.2[2] | If API functions have a similar operation to existing Linux functions, they shall have a similar name and argument style to those functions | Analysis. |
| SRSN2.1[3] | As shown, the API must provide management functions for the Front Panel and the Field I/O Devices | SRSR3.3.1[1] | The API shall provide a text-based user interface capability to allow programs running concurrently on an ATC controller unit to share the controller's Front Panel display | Observation. |
| | | SRSR3.3.2[1] | The API shall have exclusive access to the serial communications ports of the ATC Engine Board that are designated for Field I/O devices | Test Program. |
| SRSN2.1[4] | As shown in Figure 5, both users and application programs will use the API to interface to ATC controller units | SRSR3.3.1[1] | The API shall provide a text-based user interface capability to allow programs running concurrently on an ATC controller unit to share the controller's Front Panel display | Observation. |
| | | SRSR3.3.2[1] | The API shall have exclusive access to the serial communications ports of the ATC Engine Board that are designated for Field I/O devices | Test Program. |
| SRSN2.1[5] | The API Standard specifies the API Software by describing its behavior and identifying how users and application programs interface to it | SRSR3.8[3] | The API Standard shall specify the API software by describing its behavior and identifying how users and application programs interface to it | Observation |
| SRSN2.1[6] | Functions in the API Software Layer will be written so that they will only access the ATC unit through the functions in the ATC BSP Layer | SRSR3.6[4] | The API software shall only reference operating system commands and features that are available in the Linux environment defined by the ATC Controller Standard, Annex A and B | Analysis. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| SRSN2.1[7] | The API Software Layer functions will be written in the C programming language and be available in the public domain for free use by transportation industry software developers | SRSR3.6[2] | Any software developed as part of this effort shall be written in the C programming language as described by “ISO/IEC 9899:1999” commonly referred to as the C99 Standard | Observation. |
| SRSN2.1.2[1] | This will require the API to support a user interface that will provide operational users the ability to select from a set of active application programs on the ATC controller unit and to interact with the programs individually | SRSR3.3.1.1[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 | Observation. |
| | | SRSR3.3.1.1[2] | Application names associated with each window shall be listed | Observation. |
| | | SRSR3.3.1.1[3] | The application names shall be limited to 16 characters | Observation/Test Program. |
| | | SRSR3.3.1.1[4] | If there is no application associated with a window, the window number shall be listed with a blank application name | Observation. |
| | | SRSR3.3.1.1[5] | The default Front Panel Manager Window size shall be 8 x 40 characters (rows x columns) with the format as shown in Figure 7 | Observation. |
| | | SRSR3.3.1.1[6] | If the operational user has not set the default window, the Front Panel Manager Window shall be the default window | Observation. |
| | | SRSR3.3.1.1[7] | The default window shall be settable by the operational user from the Front Panel Manager Window by pressing {*,[0-F]} | Observation. |
| | | SRSR3.3.1.1[8] | The operational user shall be capable of setting the default window to the Front Panel Manager by pressing {*,*} from the Front Panel Manager | Observation. |
| | | SRSR3.3.1.1[9] | The default window shall be designated by a star “*” character next to the window number | Observation. |
| | | SRSR3.3.1.1[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 in operation | Observation. |
| | | SRSR3.3.1.1[11] | The operational user shall be able to enter {**} by pressing an asterisk (*) twice within a 0.5 second time period | Observation. |
| | | SRSR3.3.1.1[12] | 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 | Observation. |
| | | SRSR3.3.1.1[13] | The only possible window selections for focus from the Front Panel Manager Window shall be itself, the ATC Configuration Window (see Section 3.4), or a window assigned to an application | Observation. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| | | SRSR3.3.1.1[14] | If the Front Panel Manager Window is the default window, no asterisk shall be displayed next to any task name in the Front Panel Manager Window | Observation. |
| | | SRSR3.3.1.1[15] | The operational user shall be able to put the ATC Configuration Window in focus by pressing {<NEXT>} in the Front Panel Manager Window | Observation. |
| | | SRSR3.3.1.1[16] | The top two lines and bottom line of the Front Panel Manager Window shall be fixed as shown in Figure 7 | Observation. |
| | | SRSR3.3.1.1[17] | 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 | Observation. Multiple and varying displays should be evaluated. |
| | | SRSR3.3.1.1[18] | 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 | Observation. |
| SRSN2.1.2[2] | The ATC Controller Standard describes the physical characteristics and the character set to be supported | SRSR3.6[1] | The API software shall operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard | Test Program. |
| | | SRSR3.3.1.2[28] | Application programs shall be able to interpret all 2070 controller keys as individual key codes | Test Program. |
| | | SRSR3.3.1.2[29] | The escape sequences representing keys that do not have standard ASCII character codes on a 2070 shall be mapped to specific character codes in the API as shown in Table 1 | Test Program. |
| SRSN2.1.2[3] | The API software must also provide C functions that will allow application software to access and control the Front Panel Interface programmatically | SRSR3.3.1.2[1] | The API shall provide a function to return the dimensions of a window in terms of number of lines and number columns | Test Program. |
| | | SRSR3.3.1.2[2] | The API shall provide a function to open a window and register a name for display on the Front Panel Manager Window | Test Program. |
| | | SRSR3.3.1.2[3] | An application shall be able to open multiple windows providing the resource is available | Test Program. |
| | | SRSR3.3.1.2[4] | The API shall provide the ability for an application to reserve exclusive access to the Aux Switch (see ATC Controller Standard, Section 7.1.4) | Test Program. |
| | | SRSR3.3.1.2[5] | This application shall maintain exclusive access to the Aux Switch even if the application has no window in focus | Test Program. |
| | | SRSR3.3.1.2[6] | The API shall provide a function to close a window and release the resource for other application programs | Test Program. |
| | | SRSR3.3.1.2[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 | Test Program. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| | | SRSR3.3.1.2[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 | Test Program. |
| | | SRSR3.3.1.2[9] | The API shall provide a function that is used to determine if there is data in the input buffer of a window | Test Program. |
| | | SRSR3.3.1.2[10] | The API shall provide a function to read a queued character from the input buffer of a window | Test Program. |
| | | SRSR3.3.1.2[11] | The API shall provide a function to write a character to the current cursor position of a window | Test Program. |
| | | SRSR3.3.1.2[12] | The API shall provide a function to write a character to a window at a position defined by column and line number | Test Program. |
| | | SRSR3.3.1.2[13] | The API shall provide a function to write a string to a window at the current cursor position | Test Program. |
| | | SRSR3.3.1.2[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 | Test Program. |
| | | SRSR3.3.1.2[15] | The API shall provide a function to write a buffer of characters to a window at the current cursor position | Test Program. |
| | | SRSR3.3.1.2[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 | Test Program. |
| | | SRSR3.3.1.2[17] | The API shall provide a function to set the cursor position of a window defined by column and line number | Test Program. |
| | | SRSR3.3.1.2[18] | The API shall provide a function to return the cursor position of the window defined by column and line number | Test Program. |
| | | SRSR3.3.1.2[19] | If a window was registered with access to the Aux Switch, the API shall provide a function to return its status | Test Program. |
| | | SRSR3.3.1.2[20] | The API shall provide a function to compose special characters as described by the ATC Controller Standard, Section 7.1.4 | Test Program. |
| | | SRSR3.3.1.2[21] | The API shall support the display of a composed character in the same manner as any other valid character | Test Program. |
| | | SRSR3.3.1.2[22] | The API shall provide a function to clear a window that will operate on a window whether it is in or out of focus | Test Program. |
| | | SRSR3.3.1.2[23] | The API shall provide a function to refresh a window that will operate on a window whether it is in or out of focus | Test Program. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| | | SRSR3.3.1.2[24] | The bell of the controller's Front Panel shall be activated only if a bell character, ^G (hex value 07), is sent to a window that has focus | Test Program. |
| | | SRSR3.3.1.2[25] | Otherwise, the bell character shall be ignored by the API | Test Program. |
| | | SRSR3.3.1.2[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 | Test Program/Observation. |
| | | SRSR3.3.1.2[27] | Display configuration and inquiry command codes (escape sequences) specified in the ATC Controller Standard, Section 7.1.4, shall be supported as separate functions in the API | Test Program. |
| | | SRSR3.3.1.2[28] | Application programs shall be able to interpret all 2070 controller keys as individual key codes | Test Program. |
| | | SRSR3.3.1.2[29] | The escape sequences representing keys that do not have standard ASCII character codes on a 2070 shall be mapped to specific character codes in the API as shown in Table 1 | Test Program. |
| | | SRSR3.3.1.2[30] | The ATC Controller Standard, Section 7.1.4, 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. If applications use graphics on a window, the API is not responsible for redisplaying these graphics when a window is refreshed or goes out/in to focus | Test Program. |
| | | SRSR3.3.1.2[31] | The API shall provide a mechanism to alert programs when their associated windows are in and out of focus | Test Program. |
| | | SRSR3.3.1.2[32] | The API shall provide a function which applications may use to determine if their window is in focus | Test Program. |
| | | SRSR3.3.1.2[33] | The API shall provide a function to set the default window | Test Program/Observation. |
| | | SRSR3.3.1.2[34] | The API shall provide a mechanism to allow application programs to detect the presence or absence of a front panel | Test Program. |
| | | SRSR3.3.1.2[35] | The API shall recognize the presence or absence of the front panel in 5 seconds | Test Program. |
| | | SRSR3.3.1.2[36] | The API shall provide a mechanism to alert application programs of a change in the presence or absence of the front panel | Test Program. |
| | | SRSR3.3.1.2[37] | The API shall provide a mechanism to alert application programs of a change in window size | Test Program. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| | | SRSR3.3.1.2[38] | 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 | Test Program. |
| | | SRSR3.3.1.2[39] | The API shall provide a function to illuminate or extinguish the CPU ACTIVE LED described in the ATC Controller Standard, Section 7 | |
| | | SRSR3.3.1.2[40] | The function shall only operate for application programs with a window in focus | |
| SRSN2.1.3[1] | These managed resources include the Front Panel (see Section 2.1.2 User Interfaces) and the Field I/O Devices | SRSR3.3.1[1] | The API shall provide a text-based user interface capability to allow programs running concurrently on an ATC controller unit to share the controller's Front Panel display | Observation. |
| | | SRSR3.3.2[1] | The API shall have exclusive access to the serial communications ports of the ATC Engine Board that are designated for Field I/O devices | Test Program. |
| SRSN2.1.3[2] | The API does not communicate directly to these ATC resources but through the Linux operating system and associated device drivers provided with the ATC controller unit | SRSR3.6[4] | The API software shall only reference operating system commands and features that are available in the Linux environment defined by the ATC Controller Standard, Annex A and B | Analysis. |
| SRSN2.1.5[1] | The ATC controller unit will provide access to these communications interfaces via the Linux OS and Device Drivers | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A, and Annex B. | Out of Scope |
| SRSN2.1.6[1] | The API must operate effectively within the memory constraints defined for ATC controller units in the ATC Controller Standard | SRSR3.6[1] | The API software shall operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard | Test Program. |
| SRSN2.2[1] | Operational users will need to interface with applications through the API to perform the primary tasks assigned to the controller unit | SRSR3.3.1.1[1] through SRSR3.3.1.1[18] | Listed Previously | |
| SRSN2.2[2] | Utility functions will be developed, as necessary, to configure the API Managers for proper operation | SRSR3.4[1] | The API shall provide a function call apiver() that returns the API version number as a string | Test Program. |
| | | SRSR3.4[2] | The API shall provide additional utility functions to configure and maintain the API managers and software. Since these functions will based on the design of the managers, the detailed requirements for these functions cannot be determined in this SRS | Test Program. |
| SRSN2.2[3] | The API Manager Functions and the API User Utilities, will access the ATC controller hardware through the Linux Kernel | SRSR3.6[4] | The API software shall only reference operating system commands and features that are available in the Linux environment defined by the ATC Controller Standard, Annex A and B | Analysis. |
| SRSN2.2[4] | The Linux Kernel and Linux Device Drivers are defined in the ATC Controller Standard, Annex A, as a profile (selected subset) of commonly available Linux OS software | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5 and Annex A. | Out of Scope |
| SRSN2.2[5] | The specifications for the ATC Device Drivers are defined in the ATC Controller Standard, Annex B | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5 and Annex B. | Out of Scope |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| SRSN2.2[6] | The ATC Device Driver software is to be developed by manufacturers as appropriate for the architecture of their ATC hardware | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[7] | The software for the API Manager Functions and the API User Utilities will be developed in response to this SRS and specified in the API Standard | SRSR3.8[2] | Software developed in response to these requirements shall constitute the API software | Analysis, Observation, and Test Program. |
| | | SRSR3.8[3] | The API Standard shall specify the API software by describing its behavior and identifying how users and application programs interface to it | Analysis. |
| SRSN2.2[8] | The Linux Kernel will provide software signaling services including creating/deleting, scheduling, and sending/receiving software events | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[9] | These services must include single occurrence, multiple occurrence, and signal by date | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[10] | The Linux Kernel will provide process management services which allow processes to be spawned from other processes, process priority management, waiting on a process, process sleep, process termination, retrieving process information, and exiting a process | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[11] | The Linux Kernel will provide shared memory capabilities between processes including allocation, access control and deletion | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[12] | The Linux Kernel will provide semaphore and mutex synchronization services including initialization/termination, acquisition/release, increment/decrement, and waiting | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[13] | The Linux Kernel will provide serial service functions including opening/closing a port, reading/writing, setting/getting properties, status checking, and signal notification | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[14] | The Linux Kernel will provide pipe service functions including opening/closing a pipe, reading/writing, setting/getting properties, status checking, and signal notification | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[15] | The API will provide system time functions including getting/setting the system time | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[16] | The Linux Kernel will provide capabilities to set/get system global variables | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[17] | The Linux Kernel will provide network configuration capabilities with TCP/IP socket communication services including open, close, read, and write | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| SRSN2.2[18] | The Linux Kernel will provide basic shell user interface and scripting mechanism via a console port | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[19] | Datakey Interface Driver. This device driver must support all of the allowable key sizes as specified in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[20] | The device driver must allow multiple applications to access the data key as a file system | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[21] | Front Panel Interface Driver. This device driver provides a serial interface to the controller's Front Panel as described in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[22] | Serial I/O Interface Drivers. These device drivers provide synchronous and asynchronous serial ports as described in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[23] | FLASH Interface Driver. This device driver provides a flash file system interface as described in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[24] | The device driver must allow multiple applications to access the file system concurrently | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[25] | SRAM Interface Driver. This device driver provides a file system interface as described in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[26] | USB Interface Driver. This device driver provides FAT File I/O services for the USB Portable Memory Device | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[27] | Real-time Clock Interface Driver. This device driver provides access to the battery-backed real-time clock on the ATC Engine Board | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[28] | Miscellaneous Parallel I/O Drivers. These device drivers provide access to specific I/O pins such as POWERDOWN, CPU_RESET, CPU_ACTIVE, and DKEY_PRESENT as described in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[29] | Serial Peripheral Interface Drivers (SPI). These device drivers provide an interface to the EEPROM and also work with the Datakey as described in the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |
| SRSN2.2[30] | Ethernet Interface Driver. This driver provides an interface to the Ethernet ports of the Engine Board as described by the ATC Controller Standard | Out of Scope | The requirements for this user need are detailed in the ATC Controller Standard Section 2.2.5, Annex A and Annex B. | Out of Scope |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| SRSN2.2[31] | Front Panel Manager. This set of functions provides a user interface which allows programs running concurrently on an ATC controller to share the controller's Front Panel through the dedicated Front Panel Port of the ATC Engine Board | SRSR3.3.1[1] | The API shall provide a text-based user interface capability to allow programs running concurrently on an ATC controller unit to share the controller's Front Panel display | Observation. |
| | | SRSR3.3.1[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 | Test Program. |
| | | SRSR3.3.1[3] | The display size of the windows shall be based on the character size of the controller's Front Panel display (if one exists) | Observation/Test Program. |
| | | SRSR3.3.1[4] | The display size of the windows shall have a minimum of 4x40 characters and a maximum of 24 x 80 characters | Test Program. |
| | | SRSR3.3.1[5] | If no physical display exists, the API shall operate as if it has a display with a size of 8 x 40 characters | Test Program. |
| | | SRSR3.3.1[6] | Only one window shall be exposed at a time on the Front Panel display | Observation. |
| | | SRSR3.3.1[7] | When a window is exposed, the API shall display the character representation of the window on the Front Panel display (if one exists) | Observation. |
| | | SRSR3.3.1[8] | The application associated with the window displayed shall receive the characters input from the Front Panel input device (Ex. keyboard or keypad) | Test Program. |
| | | SRSR3.3.1[9] | The API shall support the display character set as defined in the ATC Controller Standard, Section 7.1.4 | Test Program. |
| | | SRSR3.3.1[10] | Screen attributes described by the ATC Controller Standard, Section 7.1.4, shall be maintained for each window independently | Observation/Test Program. |
| | | SRSR3.3.1[11] | Each window shall have separate input and output buffers unique from other windows | Test Program. |
| | | SRSR3.3.1.2[1] through SRSR3.3.1.2[40] | Listed Previously | |
| SRSN2.2[32] | Field I/O Manager. This set of functions gives concurrently running programs the capability to communicate with field devices connected to an ATC controller through the dedicated Field I/O ports of the ATC Engine Board | SRSR3.3.2[1] | The API shall have exclusive access to the serial communications ports of the ATC Engine Board that are designated for Field I/O devices | Test Program. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| | | SRSR3.3.2[2] | The API shall provide functions which allow application programs to share access to the I/O points of various transportation field devices including: i) the 2070 controller modules 2070-2A, 2070-2N and 2070-8 as described by the ATC 2070 Standard (see Section 1.5 of this document); ii) the Serial Interface Unit (SIU) as described in the ITS Cabinet Standard (see Section 1.5 of this document); and iii) the Bus Interface Unit (BIU) and Malfunction Management Unit (MMU) as described in the NEMA TS-2 Standard (see Section 1.5 of this document) | Test Program. |
| | | SRSR3.3.2[3] | The API shall support communication to multiple Field I/O devices provided the devices on the same serial port have compatible communication attributes | Test Program. |
| | | SRSR3.3.2[4] | The API shall provide functions which allow application programs to open and close “virtual ports” for supported Field I/O device | Test Program. |
| | | SRSR3.3.2[5] | The API shall poll Field I/O devices to check to the state of their input and output points | Test Program. |
| | | SRSR3.3.2[6] | The API shall provide functions which allow application programs to set the polling rate of a Field I/O device | Test Program. |
| | | SRSR3.3.2[7] | If multiple applications set the polling rate, the highest rate shall be used | Test Program. |
| | | SRSR3.3.2[8] | The maximum polling rate shall be 10 milliseconds | Test Program. |
| | | SRSR3.3.2[9] | The API shall provide functions to allow application programs to reserve or register output points of a Field I/O device for “write access” | Test Program. |
| | | SRSR3.3.2[10] | The API shall restrict write access to the output points to a single application program | Test Program. |
| | | SRSR3.3.2[11] | The API shall provide functions to allow application programs to set the state (write operation) of their registered output points | Test Program. |
| | | SRSR3.3.2[12] | The API shall provide an error code signifying the status of the write operation | Test Program. |
| | | SRSR3.3.2[13] | The API shall provide functions which allow multiple application programs to get the state (read operation) of the input and output points of each Field I/O device in operation | Test Program. |
| | | SRSR3.3.2[14] | This shall include both filtered and non-filtered inputs | Test Program. |
| | | SRSR3.3.2[15] | The API shall provide an error code signifying the status of the read operation | Test Program. |
| | | SRSR3.3.2[16] | The API shall maintain message frequencies according to the applicable standard for proper communication with the Field I/O devices in operation | Test Program. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| | | SRSR3.3.2[17] | The API shall provide functions which allow application programs to check the status of a Field I/O device | Test Program. |
| | | SRSR3.3.2[18] | The API shall check the status of a Field I/O device on a once per second basis | Test Program. |
| | | SRSR3.3.2[19] | The API shall provide applications access to the Field I/O device transition buffers on a .5 second basis | Test Program. |
| SRSN2.2[33] | Functions in the area API User Utilities provide the operational user tools in which to configure and maintain the API software installed on an ATC. While most of these functions will be based on the detailed design of the managers, there is a need for operational users to have access to the API software version and configuration information | SRSR3.4[1] | The API shall provide a function call *apiver()* that returns the API version number as a string | Test Program. |
| | | SRSR3.4[2] | The API shall provide additional utility functions to configure and maintain the API managers and software. Since these functions will based on the design of the managers, the detailed requirements for these functions cannot be determined in this SRS | Test Program. |
| | | SRSR3.4.1[1] | The API shall provide a window called the ATC Configuration Window | Observation. |
| | | SRSR3.4.1[2] | Operational users shall be able to view ATC configuration information on this window provided by the Linux uname() function (ATC Controller Standard, Section 2.2.5), the API's apiver() function, and the ATC Host Module EEPROM information (ATC Controller Standard, Annex B) | Observation. |
| | | SRSR3.4.1[3] | The default ATC Configuration Window size shall be 8 x 40 characters with the format as show in Figure 8 | Observation. |
| | | SRSR3.4.1[4] | The top two lines and bottom line of the ATC Configuration Window shall be fixed as shown in Figure 8 | Observation. |
| | | SRSR3.4.1[5] | The number of lines between the second line and bottom lines used for displaying the ATC configuration information shall vary according to the size of the ATC display | Observation. |
| | | SRSR3.4.1[6] | The operational user shall be able to scroll the ATC Configuration Window to view the configuration information using the up and down arrow keys of the controller keypad | Observation. |
| | | SRSR3.4.1[7] | The operational user shall be able to put the Front Panel Manager in focus by pressing {<NEXT>} in the ATC Configuration Window | Observation. |
| SRSN2.4[1] | The API must operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard | SRSR3.6[1] | The API software shall operate on an ATC controller unit under the hardware limitations defined in the ATC Controller Standard | Test Program. |

| User Need ID | User Need Description | Requirement ID | Requirement Description | Validation Guidance |
|---|---|---|---|---|
| SRSN2.4[2] | Any software developed as part of this effort will be written in the C programming language as described by “ISO/IEC 9899:1999” commonly referred to as the C99 Standard | SRSR3.6[2] | Any software developed as part of this effort shall be written in the C programming language as described by “ISO/IEC 9899:1999” commonly referred to as the C99 Standard | Observation. |
| SRSN2.4[3] | All software developed or referenced as part of the API must be attainable via the public domain | SRSR3.6[4] | The API software shall only reference operating system commands and features that are available in the Linux environment defined by the ATC Controller Standard, Annex A and B | Analysis. |
| | | SRSR3.8[1] | All software developed for the API shall be attainable in the public domain.[1] | Observation. |
| SRSN2.4[4] | The operational look and feel of user interfaces developed for the API should be consistent with each other | SRSR3.7.2[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 | Observation. |
| SRSN2.4[5] | If API functions have a similar operation to existing Linux functions, they should have a similar name and argument style to those functions | SRSR3.7.2[2] | If API functions have a similar operation to existing Linux functions, they shall have a similar name and argument style to those functions | Analysis. |

# INDEX