## CHANGE HISTORY

<table>
<thead>
<tr>
<th>DATE</th>
<th>NOTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>02/17/15</td>
<td>Initial v1.00 version of the APIRI User Manual.</td>
</tr>
<tr>
<td>10/27/15</td>
<td>Version 01.01-01.02 Updates in content and formatting for IEEE Std 1063-2001.</td>
</tr>
<tr>
<td>11/10/15</td>
<td>Version 1.03 Adds details on changes from API specification in APIRI implementation</td>
</tr>
</tbody>
</table>
NOTICE

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

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

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

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

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

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

1 INTRODUCTION .............................................................................................................................. 5
  1.1 Purpose .................................................................................................................................. 5
  1.2 Scope .................................................................................................................................. 5
  1.3 Document Organization ...................................................................................................... 5

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

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

4 BUILDING THE APIRI LIBRARIES ............................................................................................. 8
  4.1 Unix / Macintosh .................................................................................................................. 8

5 CREATING APPLICATION SOFTWARE ....................................................................................... 8
  5.1 Sample Applications ............................................................................................................ 9
  5.2 Registering Applications in FPUI ...................................................................................... 12
  5.3 Supporting Legacy Applications ........................................................................................ 12
  5.4 Packaging Applications ..................................................................................................... 12

6 APIRI USER INTERFACE COMPONENTS .................................................................................. 12
  6.1 ATC APIRI Front Panel Manager ...................................................................................... 13
  6.2 Front Panel Manager Interface Guide ............................................................................... 14
  6.3 ATC APIRI Configuration Information Manager ................................................................ 14
  6.4 Setting Date/Time ............................................................................................................. 15
  6.5 Setting Ethernet 1 Parameters .......................................................................................... 16
  6.6 Setting Ethernet 2 Parameters .......................................................................................... 18
  6.7 Programs and Services Startup Behavior ......................................................................... 19
  6.8 Linux Information ............................................................................................................... 22
  6.9 ATC API Version Information ............................................................................................ 22
  6.10 Host EEPROM Information ............................................................................................... 23

7 APIRI PROGRAMMATIC INTERFACE ......................................................................................... 27
  7.1 Add Transaction to fio_fiod_outputs_set ........................................................................... 27
  7.2 Add Frame Sent Callback ................................................................................................. 28
  7.3 Add Timeout to fio_fiod_frame_read ................................................................................. 32
  7.4 Add App_Handle to fpui_open_aux_switch/fpui_read_aux_switch .................................. 32
  7.5 Add File Handle Parameter to tod_request/cancel_tick/onchange_signal() ..................... 34
  7.6 Add fpui_panel_present function to detect physical front panel presence ....................... 37
  7.7 Remove fpui_get/set_led functions ................................................................................... 38
  7.8 Rearrangement of System Configuration Menu Items (ATC5401 Section 3.2.1) ............. 38
  7.9 Separate and Enhanced Ethernet Configuration Screens (ATC5401 Section 3.2.3) ...... 38
  7.10 Enhanced Linux Information Screen (ATC5401 Section 3.2.5) ..................................... 39
  7.11 Enhanced API Information Screen (ATC5401 Section 3.2.5) ........................................ 40
  7.12 Time Source Configuration Screen Addition ..................................................................... 41

8 DEFINITIONS AND ACRONYMS .................................................................................................. 41

9 REFERENCES .................................................................................................................................. 44
1 INTRODUCTION

1.1 Purpose

This software user manual (manual) describes how to use the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation (RI) software (APIRI Software). It has been developed for both the software developers that create programs for ATC units and end users that use ATC units to perform their operational tasks.

1.2 Scope

The APIRI Software is an open source software (OSS) implementation of the ATC 5401 Application Programming Interface Standard. It is designed to operate on transportation controllers that conform to the ATC 5201 Transportation Controller Standard (ATC units). The APIRI Software allows application programs to be written so that they may run on any ATC unit regardless of the manufacturer. It also allows multiple application programs to be compatible on a single ATC unit by sharing the fixed resources of the controller. This manual describes: a) how to acquire, build and deploy the APIRI Software; b) how to build and deploy application programs that use the APIRI Software; and c) how to use the windowing system and configuration utilities that are provided with the APIRI Software.

1.3 Document Organization

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

2 CONCEPT OF OPERATIONS

The APIRI Software is designed to work on transportation controllers that conform to the ATC 5201 Transportation Controller Standard. ATC 5201 defines a controller architecture where the computational components reside on a single (5” x 4”) printed circuit board (PCB), called the “Engine Board,” with standardized connectors and pinout. It is made up of a central processing unit (CPU), a Linux operating system (O/S) and device drivers, memory, external and internal interfaces, and other associated hardware necessary to create an embedded transportation computing platform.

The APIRI Software provides both user interface facilities and programmatic interfaces for ATC units that are not provided through device drivers or the standard Linux O/S. The user interface facilities of the APIRI Software include a windowing system that allows operational users to interact with concurrently operating application programs (which in turn have their own user interfaces) and system-wide configuration management utilities. The programmatic interfaces of the APIRI Software provide C language functions that allow software developers to create application programs that share the resources of the ATC unit including the front panel, field I/O interfaces and real-time clock. When used with the Linux O/S and device drivers of the Engine Board, the APIRI Software provides for a software environment that facilitates application program portability (runs on any ATC manufacturers equipment), compatibility (will run concurrently with other application programs), and interchangeability (assuming the programs perform the same function) on a single ATC unit. The application program portability provided is “source code portability” where there may be modest efforts on the part of the application program developer such as recompiling and linking source code for a particular processor family (see Figure 1).
Figure 1. Application program portability using the APIRI Software.

Figure 2 illustrates the layered architecture of the ATC software environment. 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 5201, 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 unit hardware. As shown in Figure 2, end users, software developers and application programs use the APIRI Software to interface to ATC units.

Figure 2. Layered ATC software environment.
3 GETTING THE APIRI SOFTWARE SOURCE CODE

The APIRI Software source code is hosted on GitHub at the following URL:

• https://github.com/apiriadmin/APIRI

The entire source tree can be downloaded at the following URL as a zip file:

• https://github.com/apiriadmin/APIRI/archive/master.zip

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

For more information see the following link for GitHub documentation:

• https://help.github.com/articles/set-up-git/

The APIRI source code is released under the GNU LESSER GENERAL PUBLIC LICENSE (LGPL) available here:

• https://github.com/apiriadmin/APIRI/blob/master/LICENSE

3.1 APIRI Software Source Code Organization

The APIRI source code is organized as follows:

• fio/src
• fpu/FrontPanelSystem
• tod/src

The fio/src folder contains the Field Input and Output source code files.

The fpu/FrontPanelSystem folder contains the Front Panel Manager source code files.

The tod/src folder contains the Time of Day source code files.

In addition, the software design document (SDD) of each major sub component is available at the root level of each folder above.

Each folder is organized as follows:

• fio/src
  • fioapi – Contains the FIO source and header definitions
  • fiodriver - Contains FIO Driver source files
  • fiotest – Contains FIO test source files
  • fpu/FrontPanelSystem
  • AuxSampleApplication – Contains a sample application that accesses the Aux Switch
  • Documentation – Contains the FPS SDD
  • FrontPanelDriver – Contains the FrontPanelDriver source files
  • FrontPanelManager – Contains the FrontPanelManager source files
  • MasterSelection – Contains the MasterSelection source files
  • SampleApplication – Contains a sample application that registers with the FPM
  • SystemConfiguration – Contains the SystemConfiguration source files
  • Fpuiexec – Supports launches legacy applications with FPU support
  • libfpui
• tod/src
  • src – Contains the TOD source files
4 BUILDING THE APIRI LIBRARIES

4.1 Unix / Macintosh

Prerequisites:

- gcc 4.X or newer

Each project includes its own makefile and all projects can be built from a single top level makefile.

Each makefile accepts the standard "all" and "clean" targets.

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

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

- source export-env.sh

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

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

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

Where,

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

For example, to build the APIRI project the following commands can be run from the root path:

- source export-env
- make

5 CREATING APPLICATION SOFTWARE

After building the ATC APIRI libraries applications can be built to leverage the FPU, FIO and TOD APIs.

The following sample applications highlight the basic use of the FPU, FIO and the TOD APIs as well as how to package and deploy a built application.
5.1 Sample Applications

The ATC APIRI includes several sample applications that highlight how to consume the APIRI APIs.

These sample applications are available in the source at the following location:

- **Full Sample Application**
  - /apps/aptcapitest.c

- **FIO Sample Application**
  - /fio/src/fiotest

- **FPU Sample Application**
  - /fpu/FrontPanelSystem/SampleApplication
  - /fpu/FrontPanelSystem/AuxSampleApplication

- **TOD Sample Application**
  - /tod/src/todtest

Each sample application can be built individually by running `make` in the sample application folder.

5.1.1 Full Sample Application

The Full Sample Application aims to highlight a fully integrated basic sample application that leverages the FPU system, the FIO API and the TOD API.

The Full Sample Application supports the following:

- registers itself with the FPU system,
- indicates date/time from the tod_get function
- indicates the state of a toggling output and a corresponding input based on the lowest numbered output point which can be reserved (332 FIO device).

The Full Sample application also supports starting multiple instances, each registers as a separate item with the front panel manager. This can allow ATC manufactures to test up to the full set of 16 applications running at once.

5.1.2 FIO Sample Application

The FIO Sample Application highlights how an application can interact with the physical inputs and outputs of the device through the ATC API software interface.

The FIO Sample Application first registers a 332 FIO module on SP5.

```c
/* Register fio device "FIO332" */
if ( 0 > ( dev_handle = fio_fiod_register( fio_handle, FIO_PORT_SP5, FIO332 ) ) ) {
    printf("Failed to fio_fiod_register(FIO_PORT_SP3, FIOTF1),err(%d),errno(%s)\n",
        dev_handle, strerror( errno ));
    return( -1 );
}
printf( "dev_handle(%d) = fio_fiod_register() successful\n", dev_handle );
```

The application then reserves a set of outputs that it will have exclusive control over.

```c
/* Reserve test set of outputs */
```
FIO_BITS_CLEAR( set_bits, sizeof( set_bits ) );
FIO_BIT_SET( set_bits, TEST_BIT_1 );
if ( 0 > ( err = fio_fiod_outputs_reservation_set( fio_handle, dev_handle, set_bits, sizeof(set_bits) ) ) ) {
    return( -1 );
}
printf( "errno(%s) = fio_fiod_outputs_reservation_set() successful\n", strerror( errno ) );

FIO_BIT_SET( set_bits, TEST_BIT_2 );
if ( 0 > ( err = fio_fiod_outputs_reservation_set( fio_handle, dev_handle, set_bits, sizeof(set_bits) ) ) ) {
    return( -1 );
}
printf( "errno(%s) = fio_fiod_outputs_reservation_set() successful\n", strerror( errno ) );

After the outputs are registered, the sample application then sets and gets the output state in a loop.

5.1.3 FPU Sample Application

The FPU Sample Application highlights how an application can register with and use the front panel (fpu) system.

The first step is to open the FPU device.

fprintf( stderr, "%s: opening device\n", argv[0] );
sprintf( buf, "Sample %d Application", id );
if( (fd = fpui_open(O_RDWR|O_SYNC, buf)) < 0 ) {
    fprintf( stderr, "%s: Fopen failed (%s)\n", argv[0], strerror( errno ) );
    exit( 99 );
}

The buffer passed to fpui_open will be used as the application name in the Front Panel Manager.

After the fpu device is opened, it can be used to read and write characters to the front panel using the fpui_write_* and fpui_read_* API set.

fprintf(stderr, "%s: fd=%d\n", argv[0], fd);
fpui_write(fd, "\f", 1);
sprintf( buf, "This is Sample Application %d\n\n", id );
fpui_write_string(fd, buf);
for( i = 0; i < id; i++ ) {
    fpui_write_char( fd, '\n' );
}
sprintf( buf, "     here is another line\n\n" );
fpui_write_string(fd, buf);
for( i = 0; i < 5 - id; i++ ) {
    fpui_write_char( fd, '\n' );
5.1.4 FPU Aux Sample Application

The FPU Aux Sample Application highlights how an application can read the aux switch state in a blocking and non-blocking way.

To open the aux switch in a blocking manner the device should be opened with the O_RDONLY flag as in the following example:

```c
fprintf( stderr, "%s: opening device (blocking)\n", argv[0] );
if( (fd = open( "/dev/aux", O_RDONLY )) < 0 ) {
    perror( argv[0] );
    exit( 99 );
}
for( ;; ) {
    read( fd, buf, sizeof( buf ) );
    printf("AUX Switch: state is %s\n", (buf[0])?"ON":"OFF");
}
close( fd );
```

To open the aux switch in a non-blocking manner the device should be opened with the O_NONBLOCK flag as in the following example:

```c
fprintf( stderr, "%s: opening device (nonblocking)\n", argv[0] );
if( (fd = open( "/dev/aux", O_RDONLY | O_NONBLOCK )) < 0 ) {
    perror( argv[0] );
    exit( 99 );
}
for( ;; ) {
    read( fd, buf, sizeof( buf ) );
    printf("AUX Switch: state is %s\n", (buf[0])?"ON":"OFF");
}
close( fd );
```

5.1.5 TOD Sample Application

The TOD Sample Application highlights how an application can read and set time and date information
using the TOD API set.

The TOD sample application gets and sets the time source using the TOD API as follows:

```c
fprintf( stderr, "getting tod source\n" );
int todsrc = tod_get_timesrc();
fprintf( stderr, "the current tod source is %d\n", todsrc);

fprintf( stderr, "setting tod source to linesync\n" );
int result = tod_set_timesrc(TOD_TIMESRC_LINESYNC);
int todsrc = tod_get_timesrc();
fprintf( stderr, "the current tod source is %d\n", todsrc);
```

5.2    Registering Applications in FPUI

Any application that registers itself through the FPU registration API (i.e. `fpui_open`) will be displayed in the list of available application on the Front Panel Manager.

The ATC APIRI Front Panel manager supports listing any application software that is compliant with the ATC API v2.17 or later specification.

5.3 Supporting Legacy Applications

The `fpuiexec` executable can be used redirect the IO stream of another executable to the FPU driver.

The source code for the `fpuiexec` program can be found in the following folder:

- `/fpu/FrontPanelSystem/fpuiexec/

This program allows shell scripts and programs not specifically compiled to use the `libfpui.so` library to run under the ATC Front Panel Manager environment.

The syntax used to start a program with `fpuiexec` is:

- `fpuiexec <registration name> <program>`

5.4 Packaging Applications

Application software should be installed to the Linux file system under the path `"/opt/vendor_name"`, for example Intelight’s `MaxTime` is installed at `"/opt/intelight/MaxTime"`

Applications installed in this path will be listed under the System Services menu of the front panel manager.

By packaging and deploying an application in this way the user can then control the startup behavior of multiple applications using the System Services menu described in the subsequent sections.

6    APIRI USER INTERFACE COMPONENTS

The Front Panel Manager application enables multiple applications to run at once through the standard ATC API interfaces.
The Front Panel Manager main interface can be seen below in Figure 1.

**Figure 1 Front Panel Manager main window**

The ATC Configuration Information manager allows system wide access to core settings and service information.

The ATC Configuration Information manager main window can be seen in Figure 2 below.

```
ATC CONFIGURATION INFORMATION
SELECT ITEM [0-F]
0 System Time 1 Ethernet 1 Config
2 Ethernet 2 Config 3 System Services
4 Linux Info 5 API Info
6 Host EEPROM Info 7
8 9
[UP/DN ARROW] [APPLY-ENT] [QUIT-*-NEXT]
```

**Figure 2 ATC Configuration Information main window**

6.1 ATC APIRI Front Panel Manager

The Front Panel Manager provides a user interface which allows programs running concurrently on a controller to share the controller resources (e.g. front panel screen, keypad, etc.).

The Front Panel Manager interface is shown in Figure 3 with two installed applications, the DAT test program and the MaxTime signal control software.

```
FRONT PANEL MANAGER
SELECT WINDOW [0-F] SET DEFAULT *[0-F]
0*MaxTime 1 RampMeter
2
3
4
5
6
7
8
9
[UP/DN ARROW] [CONFIG INFO- NEXT]
```

**Figure 3 Front Panel Manager main window showing two installed applications**

The Front Panel Manager can be thought of as a “Start” menu for the controller and provides a way for a user to switch between different running programs.

The Front Panel Manager is always running in the background and is started automatically when the
controller is powered on.

The Front Panel Manager window can be brought into focus with the "** * <ESC>" key sequence from any application screen.

### 6.2 Front Panel Manager Interface Guide

The main Front Panel Manager window lists all the client application programs which have registered with the Front Panel Manager through the front panel API.

Each application can be brought into focus from the main Front Panel Manager screen by selecting the numbered menu key associated with the program.

The Front Panel Manager window can always be brought back into focus with the "** * <ESC>" key sequence from any application screen.

The Front Panel Manager also allows a given application to be set as the default application that is started and brought into focus when the controller is powered on automatically.

To set the default application use the "** [0-F] ENTER" key sequence from the Front Panel Manager window, where [0-F] is the menu number of the application. Figure 5 shows the Front Panel Manager interface where the MaxTime firmware is selected by default.

The default application will be brought into focus upon startup automatically. A user can always switch back to the Front Panel Manager main window using the "** * <ESC>" key sequence from any application screen.

![Figure 4 Front Panel Manager main window with a default application set](image)

The default application is indicated with a leading "**" in the Front Panel Manager application list.

### 6.3 ATC APIRI Configuration Information Manager

The ATC Configuration Information manager provides access to system settings and service information and can be easily accessed from any application that is running on the controller. The ATC Configuration Information Manager main window can be seen in Figure 6.
The ATC Configuration Information manager enables a user to configure and view the following settings and service information:

- 0 - System Time
- 1 - Ethernet 1 Configuration
- 2 - Ethernet 2 Configuration
- 3 - System Services
- 4 - Linux Info
- 5 - API Info
- 6 - Host EEPROM Info

To select an ATC Configuration Information item, press the “[0-F]” number of the menu item in the list.

To return to the previous screen within the ATC Configuration Information manager press “<ESC>”.

The ATC Configuration Information manager can be brought into focus with the “** <NEXT>” key sequence from any application screen.

6.4 Setting Date/Time

The current System Date and Time can be viewed and changed in the “0 – System Time” window. The System Time window can be seen below in Figure 7.
The current Date, Time, Time Zone Offset and Daylight Savings Time setting and status can be viewed and changed in the System Time window.

To set the date and time use the left and right arrow keys to select the desired field.

To change the selected value use the + and - keys or the keypad to enter a new value.

Select "<Enter>" to save any changes.

Select "** <Next>" to return to the ATC Setting Information manager main screen and save any changes.

### 6.5 Setting Ethernet 1 Parameters

The current settings of the Ethernet 1 adapter can be viewed and changed in the "1 – Ethernet 1 Configuration" window. The Ethernet 1 Configuration window can be seen below in Figure 8.
Figure 7 Ethernet 1 configuration settings window

The current Port Mode, IP address, Subnet mask, Default Gateway, DNS Server and Host name of Ethernet Adapter 1 can be viewed and changed in Ethernet 1 Configuration window.

The port mode for Ethernet Adapter 1 can be set to: Static, DHCP, or Disabled.

If the port mode if set to DHCP or disabled the port settings cannot be changed.

To change a given setting value, use the left and right arrow keys to navigate through the available fields. The currently selected value will be indicated with the cursor focus.

To change the selected value use the + and - keys or the keypad to enter a new value.

If the Ethernet 1 port mode is set to static the following network settings can be configured:

- IP Address
- Subnet Mask
- Default Gateway
- DNS Server

The Ethernet hostname can be set if the Ethernet port mode is set to static or DHCP.

- Host Name

Note: The hostname is shared by both Ethernet 1 and Ethernet 2.
Select "<Enter>" to save any changes.

Select "* * <Next>" to return to the ATC Setting Information manager main screen.

In addition to programming the various network settings, the current number of good (GD) and bad (BD) packets sent and received on the interface are displayed.

6.6 Setting Ethernet 2 Parameters

The current settings of the Ethernet 2 adapter can be viewed and changed in the “2 – Ethernet 2 Configuration” window. The Ethernet 2 Configuration window can be seen below in Figure 9.

![Ethernet 2 Configuration Window](image)

**Figure 8 Ethernet 2 Settings configuration window**

The current Port Mode, IP address, Subnet mask, Default Gateway, DNS Server and Host name of Ethernet Adapter 2 can be viewed and changed in Ethernet 2 Configuration window.

The port mode for Ethernet Adapter 2 can be set to: Static, DHCP, or Disabled.

If the port mode if set to DHCP or Disabled, the port settings cannot be changed.
To change a given setting value, use the left and right arrow keys to navigate through the available fields. The currently selected value will be indicated with the cursor focus.

To change the selected value use the + and - keys or the keypad to enter a new value.

If the Ethernet 2 port mode is set to static the following network settings can be configured:
- IP Address
- Subnet Mask
- Default Gateway
- DNS Server

The Ethernet hostname can be set if the Ethernet port mode is set to static or DHCP.
- Host Name

Note: The hostname is shared by both Ethernet 1 and Ethernet 2.

Select "<Enter>" to save any changes.

Select "** * <Next>" to return to the ATC Setting Information manager main screen.

In addition to programming the various network settings, the current number of good (GD) and bad (BD) packets sent and received on the interface are displayed.

** 6.7 Programs and Services Startup Behavior**

The current startup behavior of each system service can be viewed and changed in the "3 – System Services" window. The System Services window can be seen below in Figure 10.
### SYSTEM SERVICES CONFIGURATION

<table>
<thead>
<tr>
<th>SERVICE</th>
<th>STATUS</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01logging</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>13portmap</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>20urandom</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>25eeprom</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>25ramdisk</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]

<table>
<thead>
<tr>
<th>SERVICE</th>
<th>STATUS</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>25ramdisk</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>40network</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>45GPS</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>49ntp</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>50dropbear</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]

<table>
<thead>
<tr>
<th>SERVICE</th>
<th>STATUS</th>
<th>CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>50dropbear</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>67usb</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>70vsftpd</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>74mtdmd5</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>75crond</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]
System services are controlled through initialization scripts which are used to start/stop/restart Linux operating system service programs and daemons.

These scripts reside in the file-system at "/etc/init.d" and are listed in the System Services window, where each may be enabled or disabled for automatic start at boot time.

The current startup status of the installed system services is indicated under STATUS and is read only.

To change the startup behavior of a given system service use the left/right arrow keys to select the desired service.

To scroll through the available services page by page use the up and down keys.

The XX number before each service indicates the service startup order which cannot be changed.

Once the service is selected, use the + and - keys to change the behavior from Enabled to Disabled.

The services show in Figure 10 are an example of the services running on an Intelight ATC controller—the following is a brief description of each service:

- 01logging – Linux logging service
- 13portmap – Linux portmap service
- 20urandom – Linux pseudo-random generator
- 25eeprom – Linux EEPROM support
- 25ramdisk – Linux SRAM filesystem
- 40network – Linux network support
- 45gps – Linux GPS support
- 49ntp – Linux ntp support
- 50dropbear – Linux ssh/scp support
- 67usb – Linux usb driver support
- 70vsftpd – Linux ftp support
- 74mtmd5 – Linux filesystem validation service—compares a running
- 75crond – Linux scheduling service
- 81dat – Intelight’s device acceptance test application
- 91rampmeter – Intelight’s MaxTime RampMeter local controller firmware.
• 90maxtime – Intelight’s MaxTime local controller firmware.
• 80fpu – The front panel manager interface. If disabled the Front Panel Manager and the System Configuration window will not be available. To re-enable it you must connect to the controller over SSH or serial and manually edit the /etc/init.d/x80fpu

Select “* * <Next>” to return to the ATC Setting Information manager main screen.

**Note:** Setting a system service to disabled does not stop the service automatically. A reboot is required for the service to be disabled.

### 6.8 Linux Information

The system Linux Information can be viewed in the "4 – Linux Information" window. The Linux Information window can be seen in Figure 11.

![Linux Information settings window](image)

*Figure 10 Linux Information settings window*

The Linux Information screen indicates the following information about the installed Linux OS and kernel.

- Linux Release
- Linux Kernel Version/Configuration
- Machine Hardware Type
- Total System Memory
- System Uptime
- Load Average (1 min, 5 min, 15 min CPU demand)

All information in the Linux Information window is read from the system and is read only.

Select “* * <Next>” to return to the ATC Setting Information manager main screen.

### 6.9 ATC API Version Information

The system ATC API Version Information can be viewed in the “5 – API Inform” window. The API Information window can be seen in Figure 12.
The API Information screen indicates the following information about the installed ATC API:

- FPUI API Version
- FPUI API LKM Version
- FIO API Version
- FIO API LKM Version
- TOD API Version

All information in the API Information window is read from the system and is read only.

Select “**<Next>” to return to the ATC Setting Information manager main screen.

### 6.10 Host EEPROM Information

The information stored on the 2070 1C Host EEPROM module that describes the system hardware can be viewed in the "6 – Host EEPROM" window. The Host EEPROM Information window can be seen in Figure 13.
HOST EEPROM INFORMATION
Mod 1 Version: 20120401020200
Mod 1 Type: 2
Mod 2 Location: 4
Mod 2 Make: 36
Mod 2 Model: 2070-2A
Mod 2 Version: 20120401020200

[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]

HOST EEPROM INFORMATION
Mod 2 Type: 2
Mod 3 Location: 3
Mod 3 Make: 36
Mod 3 Model: 2070-3D
Mod 3 Version: 20120401020200
Mod 3 Type: 2

[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]

HOST EEPROM INFORMATION
Mod 4 Location: 5
Mod 4 Make: 36
Mod 4 Model: 2070-4B
Mod 4 Version: 20120401020200
Mod 4 Type: 2
Display #Char Lines: 16

[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]
HOST EEPROM INFORMATION
Display #Char Columns: 40
Display #Graphic Rows-1 (ymax): 127
Display #Graphic Cols-1 (xmax): 239
#Ethernets: 2
Enet 1 Type: 3
Enet 1 IP Address: 169.254.1.100
[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]
HOST EEPROM INFORMATION
Enet 1 Switch/Router MAC Address: 64:55:63:0:10:83
Enet 1 Subnet Mask: 255.255.0.0
Enet 1 Default Gateway: 0.0.0.0
Enet 1 Engine Board Interface: 2
Enet 2 Type: 3
[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]
HOST EEPROM INFORMATION
Enet 2 IP Address: 10.5.5.195
Enet 2 Switch/Router MAC Address: 64:55:63:0:90:83
Enet 2 Subnet Mask: 255.255.255.0
Enet 2 Default Gateway: 0.0.0.0
Enet 2 Engine Board Interface: 2
[UP/DN ARROW] [APPLY-ENT] [QUIT-**NEXT]
Figure 12 EEPROM Information setting window

The EEPROM Information window lists information on the installed and configured hardware modules that are part of the controller.

The information about each hardware device installed in Modules 1-4 is listed.

In addition, information about the available Ethernet and Serial devices is listed, including the currently configured settings.

All information in the Host EEPROM window is read only and cannot be modified.
The information stored on the EEPROM is specified in the ATC 6 specification. Please refer to this specification for additional detail.

Select “** <Next>**” to return to the ATC Setting Information manager main screen.

7  APIRI PROGRAMMATIC INTERFACE
The full ATC API specification and interface is described in the following approved and adopted specification.

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

This section is used to document any changes from the currently approved ATC API spec and the current API RI implementation.

It is envisioned that ultimately the changes documented below will be merged back into the broader ATC specification and included in any future ATC API specification.

7.1  Add Transaction to fio_fiod_outputs_set
The current fio_fiod_outputs_set function creates a race condition for apps that require simultaneous processing of updates to multiple IO modules.

The APIRI implementation has addressed this issue by adding two additional functions:
• fio_fiod_begin_outputs_set(FIO_APP_HANDLE app_handle)
• fio_fiod_commit_outputs_set FIO_APP_HANDLE app_handle)

These two functions add a transaction like functionality to the fio_fiod_outputs_set mechanism to allow an application to update multiple IO modules before the outputs will be written.

The following calling pattern would be utilized by an application that would like to update multiple IO modules:
• fio_fiod_begin_outputs_set(my_app_handle)
• fio_fiod_outputs_set(my_app_handle, device_1, …)
• fio_fiod_outputs_set(my_app_handle, device_2, …)
• fio_fiod_outputs_set(my_app_handle, device_3, …)
• fio_fiod_commit_outputs_set(my_app_handle)

The FIOMAN will not use any outputs set after the fio_fiod_begin_outputs_set call until the fio_fiod_commit_outputs_set is called by the application. If an application does not require a transaction on outputs set, and the fio_fiod_begin_outputs_set is not called, the fio_fiod_outputs_set will take effect immediately as currently specified.

The following describes the detail of the two new proposed functions:

NAME
fio_fiod_begin_outputs_set – Begins a FIOD output points transaction.

SYNOPSIS
#include <fio.h>
int fio_fiod_begin_outputs_set(FIO_APP_HANDLE app_handle)
DESCRIPTION
This function is used to start a FIO output points transaction for a given application. The data in any subsequent fio_fiod_outputs_set call will be buffered by the FIO manager and not sent to any devices until fio_fiod_commit_outputs_set is called.

app_handle is a FIO_APP_HANDLE returned by a previously successful fio_register(3fio) call.

RETURN VALUES
Upon successful completion, 0 is returned. On error, -1 is returned with errno set appropriately.

ERRORS
Error codes returned in errno:
EINVAL The app_handle argument is invalid or the app already has a pending transaction.

NOTES
An application is not required to call fio_fiod_begin_outputs_set to set a FIOD outputs. If this function is never called all calls to fio_fiod_outputs_set will update the outputs without requiring fio_fiod_commit_outputs_set to be called.
The transaction started by fio_fiod_begin_outputs_set will not timeout.

NAME
fio_fiod_commit_outputs_set – Commits a FIOD output points transaction.

SYNOPSIS
#include <fio.h>
int fio_fiod_commit_outputs_set(FIO_APP_HANDLE app_handle)

DESCRIPTION
This function is used to commit a FIO output points transaction for a given application.
app_handle is a FIO_APP_HANDLE returned by a previously successful fio_register(3fio) call.
When an application calls fio_fiod_commit_outputs_set any pending FIOD outputs will be processed by the FIO manager and sent to the corresponding devices.

RETURN VALUES
Upon successful completion, 0 is returned. On error, -1 is returned with errno set appropriately.

ERRORS
Error codes returned in errno:
EINVAL The app_handle argument is invalid or the app does not have a pending transaction.

NOTES
An application is not required to call fio_fiod_commit_outputs_set to set a FIOD outputs if fio_fiod_begin_outputs_set was never called. If this function is never called all calls to fio_fiod_outputs_set will update the outputs without requiring fio_fiod_commit_outputs_set to be called.

7.2 Add Frame Sent Callback
The current API standard provides a mechanism for an application to be notified when a response frame is received, but it does not provide a notification for an application when an output frame is sent.

Without this mechanism it is difficult for an application to synchronize FIOD output updates with the underlying FIO manager frame writes.

The APIRI implementation addresses this issue by adding three additional functions and 1 new signal:
- fio_fiod_frame_send_notify_deregister
- fio_fiod_frame_send_notify_register
- fio_query_frame_send_notify_status
The following describes the detail of the two new functions:

**NAME**

- **fio_fiod_frame_send_notify_deregister** – Deregister a notification request for when a frame is sent.

**SYNOPSIS**

```c
#include <fio.h>

int fio_fiod_frame_send_notify_deregister( FIO_APP_HANDLE app_handle,
                                         FIO_DEV_HANDLE dev_handle,
                                         unsigned int tx_frame )
```

**DESCRIPTION**

This function is used to deregister a notification request for when a command frame is sent.

- **app_handle** is a FIO_APP_HANDLE returned by a previously successful `fio_register(3fio)` call.
- **dev_handle** is a FIO_DEV_HANDLE returned by a previously successful `fio_fiod_register(3fio)` call. **tx_frame** is a valid frame number / type for which notification has been registered, using the `fio_query_frame_send_notify_status(3fio)` call. Valid frame numbers/types are in the range 128 - 255.

Notification is generated to the application program utilizing the FIO_SIGIO_SENT (SIGRTMIN + 5) real-time signal. The application program using this service must establish a FIO_SIGIO_SENT handler, prior to calling this function. The default handling for a FIO_SIGIO_SENT real-time signal is to terminate the process. When the indicated rx_frame is received or declared in error by the FIO API, the FIO API will generate a FIO_SIGIO_SENT real-time signal to the waiting application program. The application program must then perform a `fio_query_frame_send_notify_status(3fio)` call to discover why a FIO_SIGIO_SENT real-time signal was generated. See `fio_query_frame_send_notify_status(3fio)` for further details.

An application program may register notifications for multiple response frames.

**FIO_SIGIO_SENT** is defined as:

```c
#define FIO_SIGIO_SENT (SIGRTMIN + 5)
```

**RETURN VALUES**

Upon successful completion 0 is returned. On error, -1 is returned with `errno` set appropriately.

**ERRORS**

Error codes returned in `errno`:

- **EINVAL** One or more arguments are invalid.
- **ENOMEM** There is not enough memory available for this operation.
- **EACCESS** The Frame Notification Service is not currently enabled.

**NAME**

- **fio_fiod_frame_send_notify_register** – Register a notification request for when a frame is sent.
SYNOPSIS

```
#include <fio.h>

int fio_fiod_frame_send_notify_register( FIO_APP_HANDLE app_handle,
                                          FIO_DEV_HANDLE dev_handle,
                                          unsigned int tx_frame,
                                          FIO_NOTIFY notify)
```

DESCRIPTION

This function is used to register a notification request for when a command frame is acknowledged (response frame is received by the FIO API) or when an error occurs.

`app_handle` is a `FIO_APP_HANDLE` returned by a previously successful `fio_register(3fio)` call. `dev_handle` is a `FIO_DEV_HANDLE` returned by a previously successful `fio_fiod_register(3fio)` call. `tx_frame` is a valid frame number/type for which notification is to be registered. Valid frame numbers/types are in the range 128 - 255. `notify` is an indication as to the frequency of notification; `FIO_NOTIFY_ONCE` and `FIO_NOTIFY_ALWAYS` may be specified to set notification for one occurrence or for all occurrences, respectively.

Notification is generated to the application program utilizing the `FIO_SIGIO_SENT` (SIGRTMIN + 5) real-time signal. The application program using this service must establish a `FIO_SIGIO_SENT` handler, prior to calling this function. The default handling for a `FIO_SIGIO_SENT` real-time signal is to terminate the process. When the indicated `tx_frame` is received or declared in error by the FIO API, the FIO API will generate a `FIO_SIGIO_SENT` real-time signal to the waiting application program. The application program must then perform a `fio_query_frame_send_notify_status(3fio)` call to discover why a `FIO_SIGIO` real-time signal was generated. See `fio_query_frame_send_notify_status(3fio)` for further details.

An application program may register notifications for multiple response frames.

`FIO_NOTIFY` is defined as:

```
enum fio_notify
{
    FIO_NOTIFY_ONCE,
    FIO_NOTIFY_ALWAYS
};
typedef enum fio_notify FIO_NOTIFY;
```

`FIO_SIGIO_SENT` is defined as:

```
#define FIO_SIGIO_SENT (SIGRTMIN + 5)
```

RETURN VALUES

Upon successful completion 0 is returned. On error, -1 is returned with `errno` set appropriately.

ERRORS

Error codes returned in `errno`:

- EINVAL: One or more arguments are invalid.
- ENOMEM: There is not enough memory available for this operation.
RESTRICTIONS
    None

NAME
    fio_query_frame_send_notify_status – Discover why a sent frame notification occurred

SYNOPSIS
    #include <fio.h>

    int fio_query_frame_send_notify_status( FIO_APP_HANDLE app_handle,
                                         FIO_SEND_NOTIFY_INFO *notify_info )

DESCRIPTION
    This function is used to discover why a notification, via a FIO_SIGIO real-time signal, was sent to
    the application program by the FIO API.

    app_handle is a FIO_APP_HANDLE returned by a previously successful fio_register(3fio) call.
    notify_info is a pointer to a FIO_SEND_NOTIFY_INFO structure, which will be filled with
    response frame notification information upon successful completion.

    FIO_SEND_NOTIFY_INFO is defined as:

    struct fio_send_notify_info
    {
        unsigned int  tx_frame; /* Response Frame # */
        unsigned int  seq_number; /* Sequence Number of frame */
        unsigned int  count; /* # of bytes in frame */
        FIO_DEV_HANDLE fiod; /* FIOD of response frame */
    };
    typedef struct fio_send_notify_info FIO_SEND_NOTIFY_INFO;

    FIO_FRAME_STATUS is defined as:

    enum fio_frame_status
    {
        FIO_FRAME_ERROR,
        FIO_FRAME_RECEIVED
    };
    typedef enum fio_frame_status FIO_FRAME_STATUS;

    tx_frame will be set to the frame number/type that was received or in error, and is being notified.
    Valid frame numbers/types are in the range of 128 – 255. Status will be set to an indication as to
    why the notification occurred. Valid values are: FIO_FRAME_ERROR and
    FIO_FRAME_RECEIVED, indicating an error occurred in the transmission of the frame or a valid
    response frame was received, respectively. seq_number is the sequence number given to the
    response frame that caused the notification.

RETURN VALUES
    Upon successful completion, 0 is returned. On error, -1 is returned with errno set appropriately.

ERRORS
    Error codes returned in errno:
EINVAL One or more arguments are invalid.
ENOMEM There is not enough memory available for this operation.
EACCESS The Frame Notification Service is not currently enabled.

7.3 Add Timeout to fio_fiod_frame_read
The APIRI implementation allows an application to specify a maximum time to wait for the next received frame of the indicated type, by adding a timeout parameter.

- fio_fiod_frame_read

The timeout parameter is required by all callers of fio_fiod_frame_read. If a new response frame is not received within the timeout an ETIMEDOUT error code will be returned.

The following highlights the updated function signature:

```c
int fio_fiod_frame_read( FIO_APP_HANDLE app_handle,
                        FIO_DEV_HANDLE dev_handle,
                        unsigned int rx_frame,
                        unsigned int *seq_number,
                        unsigned char *buf,
                        unsigned int count,
                        unsigned int timeout )
```

- The timeout parameter is the timeout used to determine when the read should fail. The value is in milliseconds.

7.4 Add App_Handle to fpui_open_aux_switch/fpui_read_aux_switch
The current API standard does not require an application handle to be specified when opening or reading the aux switch making it impossible to enforce the exclusive nature of the aux switch.

The APIRI implementation has addressed this issue by updating three functions to add an app_handle parameter.

- fpui_open_aux_switch
- fpui_read_aux_switch
- fpui_close_aux_switch

The following describes the updated function signatures:

NAME
fpui_open_aux_switch – Read from the Aux Switch interface

SYNOPSIS
```
#include <fpui.h>

fpui_aux_handle fpui_open_aux_switch()
```

DESCRIPTION
The fpui_open_aux_switch(3fpui) library call is used to reserve exclusive access to the Aux Switch. One, and only one, process may hold the reservation at a time.
fpui_open_aux_switch(3fpui) returns the new descriptor, or -1 if an error occurred with errno set appropriately.

ERRORS
   ENOMEM     Insufficient memory was available.
   EACCES     The request access to the underlying device or object is not allowed. This could occur if the Aux Switch is currently opened by another process.
   EFAULT     A reference to an inaccessible memory area was attempted.
   EINVAL     Request not valid.

NAME
   fpui_read_aux_switch – Read from the Aux Switch interface

SYNOPSIS
   #include <fpui.h>

   int fpui_read_aux_switch( fpui_aux_handle aux_handle )

DESCRIPTION
   The fpui_read_aux_switch(3fpui) library call will return TRUE if the Aux Switch is on and FALSE if the SWITCH is off.

   aux_handle is a fpui_aux_handle returned by a previously successful fpui_open_aux_switch(3fpui) call.

RETURN VALUE
   fpui_read_aux_switch(3fpui) will return 1 if the Aux Switch is on, 0 if the Aux Switch is off, or -1 if an error occurred with errno set appropriately.

ERRORS
   EBADF      The underlying object or device used to access the Aux Switch is invalid.
   EINTR      The fpui_close_aux_switch(3fpui) call was interrupted by a signal.
   EIO        An I/O error occurred.

NOTES
   None

RESTRICTIONS
   None

SEE ALSO
   fpui_open_aux_switch(3fpui), fpui_close_aux_switch(3fpui)

NAME
   fpui_close_aux_switch – Close the Aux Switch interface

SYNOPSIS
   #include <fpui.h>

   int fpui_close_aux_switch( fpui_aux_handle aux_handle )
DESCRIPTION
The `fpui_close_aux_switch(3fpui)` releases exclusive access and closes the Aux Switch interface. Any resources allocated when opened are returned to the system for reuse.

`aux_handle` is a `fpui_aux_handle` returned by a previously successful `fpu_open_aux_switch(3fpui)` call.

RETURN VALUE
On success, 0 (zero) is returned. On error, -1 is returned with `errno` set appropriately.

ERRORS
- **EBADF** The underlying object or device used to access the Aux Switch is invalid.
- **EINTR** The `fpui_close_aux_switch(3fpui)` call was interrupted by a signal.
- **EIO** An I/O error occurred.

NOTES
None

RESTRICTIONS
None

SEE ALSO
- `fpu_open_aux_switch(3fpui)`
- `fpui_read_aux_switch(3fpui)`

7.5 Add File Handle Parameter to tod_request/cancel_tick/onchange_signal()

In order to maintain a persistent file descriptor to the ATC time-of-day driver in support of the signaling functions, necessary for signal delivery to the calling process, it is recommended that the `tod_request_tick_signal()` and `tod_request_onchange_signal()` return a context variable which may be passed by the `tod_cancel_tick_signal()` and `tod_cancel_onchange_signal()` functions.

The following describes the updated function signatures:

NAME
`tod_request_onchange_signal` – Request local time changed signals

SYNOPSIS
```
#include <tod.h>

int tod_request_onchange_signal(int signalnum)
```

DESCRIPTION
The `tod_request_onchange_signal(3tod)` library call requests that the signal `signalnum` be sent to the calling process whenever the local time is changed by any source other than the time tick source. This includes a signal being sent when local time changes due to a daylight saving time adjustment.

RETURN VALUES
On success, a file descriptor to the underlying time-of-day device is returned. On error, -1 is returned with `errno` set appropriately.
ERRORS

EINVAL  signalnum is not a valid signal.
Any other errors shall be due to a system call error in the library implementation, in which case
the values of errno shall correspond to the standard Linux system call error codes

NOTES

If a signal has already been requested by the calling process when this library function is called,
the previous signal being sent will be replaced with the signal specified by signalnum.

RESTRICTIONS

None

SEE ALSO

tod_cancel_onchange_signal(3tod)

NAME
tod_cancel_onchange_signal – cancels local time changed signals

SYNOPSIS

#include <tod.h>

int tod_cancel_onchange_signal(int fd)

DESCRIPTION

The tod_cancel_onchange_signal(3tod) library call cancels any local time change signal from
being sent to the calling process.
fd is a file descriptor returned by a previous tod_request_onchange_signal(3tod) call.

RETURN VALUES

On success, 0 is returned. On error, -1 is returned with errno is set appropriately.

ERRORS

EINVAL  if the fd parameter is invalid.
Any other errors shall be due to a system call error in the library implementation, in which case
the values of errno shall correspond to the standard Linux system call error codes.

NOTES

If the process that has requested this signal dies, the equivalent to this call will be performed
automatically to release any necessary resources.

RESTRICTIONS

None

SEE ALSO

tod_request_onchange_signal(3tod)

NAME
tod_request_tick_signal – Request a signal on each TOD tick

SYNOPSIS
#include <tod.h>

int tod_request_tick_signal(int signalnum)

DESCRIPTION
The **tod_request_tick_signal** library call requests that the signal `signalnum` be sent to the calling process at each tick of the time of day clock. The frequency of the time of day clock can be determined by calling **tod_get_timesrc_freq**.

RETURN VALUES
On success, a file descriptor to the underlying time-of-day device is returned. On error, -1 is returned with `errno` is set appropriately.

ERRORS
- **EINVAL** `signalnum` is not a valid signal.
  - Any other errors shall be due to a system call error in the library implementation, in which case the values of `errno` shall correspond to the standard Linux system call error codes.

NOTES
If a signal has already been requested by the calling process when this library function is called, the previous signal being sent will be replaced with the signal specified by `signalnum`.

RESTRICTIONS
None

SEE ALSO
- **tod_cancel_tick_signal**
- **tod_get_timesrc_freq**

NAME
tod_cancel_tick_signal – Cancel signal request for TOD ticks

SYNOPSIS
#include <tod.h>

int tod_cancel_tick_signal(int fd)

DESCRIPTION
The **tod_cancel_tick_signal** library call cancels any time of day clock tick signal from being sent to the calling process. `fd` is a file descriptor returned by a previous **tod_request_tick_signal** call.

RETURN VALUES
On success, 0 is returned. On error, -1 is returned with `errno` is set appropriately.

ERRORS
- **EINVAL** if the `fd` parameter is invalid.
  - Any errors shall be due to a system call error in the library implementation, in which case the values of `errno` shall correspond to the standard Linux system call error codes.

NOTES
If the process that has requested this signal dies, the equivalent to this call will be performed automatically to release any necessary resources.
RESTRICTIONS
None

SEE ALSO
tod_request_tick_signal(3tod)

7.6 Add fpui_panel_present function to detect physical front panel presence

The APIRI implementation added a new function to allow an application program to determine if a physical front panel device is present. In conjunction with this change, the fpui_get_window_size function shall always return the dimensions of the virtual display, even when a physical front panel device is not present. The following describes the new fpui_panel_present function signature:

NAME
fpui_panel_present – Return if front panel display is present

SYNOPSIS
#include <fpui.h>

int fpui_panel_present( fpui_handle hdl )

DESCRIPTION
The fpui_panel_present(3fpui) library call is used to detect the presence or absence of a physical Front Panel Display.

RETURN VALUE
fpui_panel_present(3fpui) returns 1 if a front panel display is present, 0 if no front panel display is connected, or -1 if an error occurred with errno set appropriately.

ERRORS
EBADF hdl is not a valid descriptor.
EFAULT A reference to an inaccessible memory area was attempted.
EINVAL Request not valid.

NAME
fpui_get_window_size – Get the current window size

SYNOPSIS
#include <fpui.h>

int fpui_get_window_size( fpui_handle hdl, int * row, int * column )

DESCRIPTION
The fpui_get_window_size(3fpui) library call will return the current size of the associated virtual display to the application. Parameters: hdl is the descriptor returned by fpui_open(3fpui), row a reference of an integer where the number of rows will be stored. column a reference of an integer where the number of columns will be stored.

RETURN VALUE
fpui_get_window_size(3fpui) will return 0 (zero) upon success and -1 upon error with errno set appropriately.
ERRORS
EBADF hdl is not a valid descriptor.
EFAULT A reference to an inaccessible memory area was attempted.
EINVAL Request not valid.

NOTES
This function always returns the dimensions of the associated virtual display regardless of the presence or otherwise of the actual Front Panel Display. The Front Panel hardware does not support this operation directly. The inquiry is serviced by the window which maintains this information.

RESTRICTIONS
None

SEE ALSO
None

7.7 Remove fpui_get/set_led functions
The APIRI implementation has removed the fpui_get_led and fpui_set_led functions as they restrict access to the front panel LED to the application whose virtual window currently has focus, and require any application accessing the front panel LED to register a FPUI virtual window.

7.8 Rearrangement of System Configuration Menu Items (ATC5401 Section 3.2.1)

<table>
<thead>
<tr>
<th>ATC CONFIGURATION INFORMATION</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT ITEM [0-F]</td>
<td></td>
</tr>
<tr>
<td>0 System Time</td>
<td>1 Ethernet Port 1</td>
</tr>
<tr>
<td>2 Ethernet port 2</td>
<td>3 System Services</td>
</tr>
<tr>
<td>4 Linux Info</td>
<td>5 API Info</td>
</tr>
<tr>
<td>6 Host EEPROM</td>
<td>7 configitem7</td>
</tr>
<tr>
<td>8 configitem8</td>
<td>9 configitem9</td>
</tr>
<tr>
<td>[UP/DN ARROW]</td>
<td>[FRONT PANEL- NEXT]</td>
</tr>
</tbody>
</table>

The proposed split of the Ethernet Configuration into port 1 and port 2 separate configuration screens also requires separate menu item entries.

The proposed split of the Linux/API Information into separate screens also requires separate menu item entries.

7.9 Separate and Enhanced Ethernet Configuration Screens (ATC5401 Section 3.2.3)

<table>
<thead>
<tr>
<th>ETHERNET CONFIGURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETHERNET PORT 1 (#.#.#.#)</td>
</tr>
<tr>
<td>Port Mode: disabled/static/dhcp</td>
</tr>
<tr>
<td>IP Address: #.#.#.#.#.#.#.</td>
</tr>
<tr>
<td>Subnet Mask: #.#.#.#.#.#.</td>
</tr>
<tr>
<td>Default Gateway: #.#.#.#.#.#.</td>
</tr>
<tr>
<td>DNS Server: #.#.#.#.#.#.</td>
</tr>
</tbody>
</table>
The APIRI implementation has separated the Ethernet port 1 and port 2 configuration into separate screens to avoid excessive scrolling to access Ethernet port 2 configuration.

The APIRI implementation has added the display of the Ethernet port MAC address alongside the port number heading line 2.

In the APIRI implementation, the “Port Enabled” field has been changed to “Port Mode” and the option changed to a 3-way option comprising the choices “disabled”, “static” and “dhcp”.

The subsequent fields are rearranged in order from that of the ATC5401 Standard layout, to provide a more conventional order of related fields with the basic settings visible with less or no scrolling.

7.10 Enhanced Linux Information Screen (ATC5401 Section 3.2.5)
The APIRI implementation has separated the Linux Information screen from the combined Linux/API Information screen of the ATC5401 Standard in order to access related information in a more concise manner and with less or no scrolling necessary.

In addition it is proposed to display more relevant fields from the options returned by the “uname” utility function: the Linux release (actually kernel version number, e.g. “2.6.32”); the Linux version (actually the kernel build number and date of build); the machine hardware type (e.g. "ppc"). The “Network Node Hostname” field is dropped due to being a duplicate of the “Host Name” field of the Ethernet Configuration Screens.

The “Processor Type” and “Hardware Platform” fields are dropped due to only being available from the “uname” utility function as “unknown” on non-x86 architecture machines. The “Operating System” field is dropped due to obviousness, i.e. always returns “GNU/Linux” on Linux platforms.

The APIRI implementation has added the following Linux information fields, available from the “sysinfo” and “df” utility functions, in order to provide additional diagnostic data: “Memory Total and Free”, the total and free usable main memory size; “Filesystem Total and Free”, the total and free space on the root filesystem; “Load Average”, the 1, 5 and 15 minute processor load averages; “Uptime”, the time since last boot. These fields shall be refreshed with current data once every minute while the screen is displayed.

### 7.11 Enhanced API Information Screen (ATC5401 Section 3.2.5)
The APIRI implementation has separated the API Information screen from the combined Linux/API Information screen of the ATC5401 Standard in order to access related information in a more concise manner and with less or no scrolling necessary.

The fields are renamed to be more relevant to the provisions of the API Reference Implementation.

7.12 Time Source Configuration Screen Addition

The APIRI implementation has added a new system configuration menu item and screen to allow configuration of the time source including additional settings relating to TOD_TIMESRC_EXTERNAL1 (GPS) for serial port connection, and TOD_TIMESRC_EXTERNAL2 (NTP) for NTP network peer address. The “timesrc” field is configurable from the enumeration “LINESYNC/RTCSQWR/CRYSTAL/GPS/NTP”.

8 DEFINITIONS AND ACRONYMS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AASHTO</td>
<td>American Association of State Highway and Transportation Officials</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>API Managers</td>
<td>API software that manages an ATC resource for use by concurrently running application programs.</td>
</tr>
<tr>
<td>API Utilities</td>
<td>API software not included in the API Managers that is used for configuration purposes.</td>
</tr>
<tr>
<td>APIRI</td>
<td>API Reference Implementation (software)</td>
</tr>
<tr>
<td>APIRI Project</td>
<td>Entire project managed by this PMP including software, hardware and documentation.</td>
</tr>
<tr>
<td>APIVS</td>
<td>API Validation Suite (software and fixture)</td>
</tr>
<tr>
<td>Application Program</td>
<td>Any program designed to perform a specific function directly for the user or, in some cases, for another application program. Examples of application programs include word processors, database programs, Web browsers and traffic control programs. Application programs use the services of a computer's O/S and other supporting programs such as an application programming interface.</td>
</tr>
<tr>
<td>API</td>
<td>Application Programmer Interface</td>
</tr>
<tr>
<td>ATC</td>
<td>Advanced Transportation Controller</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ATC Device Drivers</td>
<td>Low-level software not included in a typical Linux distribution that is necessary for ATC-specific devices to operate in a Linux O/S environment.</td>
</tr>
<tr>
<td>ATP</td>
<td>Authorization to Proceed</td>
</tr>
<tr>
<td>Board Support Package</td>
<td>Software usually provided by processor board manufacturers which provides a consistent software interface for the unique architecture of the board. In the case of the ATC, the Board Support Package also includes the O/S</td>
</tr>
<tr>
<td>BSP</td>
<td>See Board Support Package</td>
</tr>
<tr>
<td>ConOps</td>
<td>Concept of Operations</td>
</tr>
<tr>
<td>CO</td>
<td>Contracting Officer</td>
</tr>
<tr>
<td>COR</td>
<td>Contract Officer’s Representative</td>
</tr>
<tr>
<td>COTM</td>
<td>Contract Officer’s Task Manager</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit. A programmable logic device that performs the instruction, logic and mathematical processing in a computer.</td>
</tr>
<tr>
<td>Device Driver</td>
<td>A software routine that links a peripheral device to the operating system. It acts like a translator between a device and the application programs that use it.</td>
</tr>
<tr>
<td>FHWA</td>
<td>Federal Highway Administration</td>
</tr>
<tr>
<td>FIO</td>
<td>Field Input and Output</td>
</tr>
<tr>
<td>FIOMAN</td>
<td>Field I/O Manager</td>
</tr>
<tr>
<td>FIOMSG</td>
<td>Field I/O Message Scheduler</td>
</tr>
<tr>
<td>FPMW</td>
<td>Front Panel Manager Window</td>
</tr>
<tr>
<td>FPUI</td>
<td>Front Panel User Interface</td>
</tr>
<tr>
<td>H/W</td>
<td>Hardware</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITE</td>
<td>Institute of Transportation Engineers</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transportation Systems</td>
</tr>
<tr>
<td>JC</td>
<td>Joint Committee</td>
</tr>
<tr>
<td>JPO</td>
<td>Joint Program Office</td>
</tr>
<tr>
<td>Linux</td>
<td>Low-level software that is freely available in the Linux community for use with common hardware components operating in a standard fashion.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Linux Kernel</td>
<td>The Unix-like operating system kernel that was begun by Linus Torvalds in 1991. The Linux Kernel provides general O/S functionality. This includes functions for things typical in any computer system such as file I/O, serial I/O, interprocess communication and process scheduling. It also includes Linux utility functions necessary to run programs such as shell scripts and console commands. It is generally available as open source (free to the public). The Linux Kernel referenced in this document is defined in the ATC 5201 Standard, Appendix A and Appendix B.</td>
</tr>
<tr>
<td>N/A</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>Operational User</td>
<td>A technician or transportation engineer who uses the controller to perform its operational tasks.</td>
</tr>
<tr>
<td>O/S</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PMP</td>
<td>Project Management Plan</td>
</tr>
<tr>
<td>POP</td>
<td>Period of Performance</td>
</tr>
<tr>
<td>PRL</td>
<td>Protocol Requirements List</td>
</tr>
<tr>
<td>RI</td>
<td>Reference Implementation</td>
</tr>
<tr>
<td>RITA</td>
<td>Research and Innovative Technology Administration</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Clock</td>
</tr>
<tr>
<td>RTM</td>
<td>Requirements Traceability Matrix</td>
</tr>
<tr>
<td>SDD</td>
<td>Software Design Document or Software Design Descriptions</td>
</tr>
<tr>
<td>SDO</td>
<td>Standards Development Organization</td>
</tr>
<tr>
<td>SE</td>
<td>Systems Engineer</td>
</tr>
<tr>
<td>SEP</td>
<td>Systems Engineering Process</td>
</tr>
<tr>
<td>SEMP</td>
<td>Systems Engineering Management Plan</td>
</tr>
<tr>
<td>SOW</td>
<td>Statement of Work</td>
</tr>
<tr>
<td>SPDD</td>
<td>Serial Port Device Driver</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirements Specification</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TBD</td>
<td>To Be Determined</td>
</tr>
<tr>
<td>TOD</td>
<td>Time of Day</td>
</tr>
<tr>
<td>TOPR</td>
<td>Task Order Proposal Request</td>
</tr>
<tr>
<td>TX</td>
<td>Transmission</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>USDOT</td>
<td>United States Department of Transportation</td>
</tr>
<tr>
<td>User Developer</td>
<td>A software developer that designs and develops programs for controllers.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Walkthrough</td>
<td>A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content.</td>
</tr>
<tr>
<td>WBS</td>
<td>Work Breakdown Structure</td>
</tr>
<tr>
<td>WG</td>
<td>Working Group</td>
</tr>
</tbody>
</table>

9 REFERENCES


Institute of Transportation Engineers, *ATC 5401 Application Programming Interface (API) Standard for the Advanced Transportation Controller (ATC) v02*. ATC Joint Committee, 15 September 2013.  

Institute of Transportation Engineers, *ATC APIRI PMP v01.01 Project Management Plan (PMP) for the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation Project*. ATC Joint Committee, 3 January 2014.  

Institute of Transportation Engineers, *ATC APIRI SEMP v01.01 Systems Engineering Management Plan (SEMP) for the Advanced Transportation Controller (ATC) Application Programming Interface (API) Reference Implementation Project*. ATC Joint Committee, 3 January 2014.  