Nicola Tavares: Welcome to the ITS Standards Training

Ken Leonard: ITS Standards can make your life easier. Your procurements will go more smoothly and you'll encourage competition, but only if you know how to write them into your specifications and test them. This module is one in a series that covers practical applications for acquiring and testing standards-based ITS systems.

I am Ken Leonard, director of the ITS Joint Program Office for USDOT and I want to welcome you to our newly redesigned ITS standards training program of which this module is a part. We are pleased to be working with our partner, the Institute of Transportation Engineers, to deliver this new approach to training that combines web based modules with instructor interaction to bring the latest in ITS learning to busy professionals like yourself.

This combined approach allows interested professionals to schedule training at your convenience, without the need to travel. After you complete this training, we hope that you will tell colleagues and customers about the latest ITS standards and encourage them to take advantage of the archived version of the webinars.

ITS Standards training is one of the first offerings of our updated Professional Capacity Training Program. Through the PCB program we prepare professionals to adopt proven and emerging ITS technologies that will make surface transportation safer, smarter and greener which improves livability for us all. You can find information on additional modules and training programs on our web site www.pcb.its.dot.gov

Please help us make even more improvements to our training modules through the evaluation process. We look forward to hearing your comments. Thank you again for participating and we hope you find this module helpful.

Nicola Tavares: Throughout the presentation this "Activity" slide will appear indicating there is a multiple choice pop quiz following this slide. The presentation lecture will pause at each quiz section to allow you to use your computer mouse to select your answer. There is only one correct answer. Selecting the submit button will record your answer and the Clear button will remove your answer if you wish to select another answer. You will receive instant feedback on your answer choice.
Please help us make even more improvements to our training modules by completing the post-course feedback form.

This module is A208: Using the ATC 5401 Application Programming Interface Standard to Leverage ITS Infrastructures.

Your instructor, Ralph Boaz is present of Pillar Inc. and is a transportation, engineering, and marketing consultant. He was formerly Vice President of Econolite Control Products and Transportation Section Manager for Ball Systems Engineering. He has been a project manager and consultant to the ATC and NTCIP standards programs.

Ralph Boaz: Hello. It's my pleasure to be your instructor today. Thank you for participating in the PCB program. We will be bringing a lot of things that you've learned throughout the program together in this module. Also, your student supplement has some material that we covered in modules A207a and A207b and it also has the quiz questions that we will be covering in this presentation, so you can refer to that. We have a broad audience here. We have engineering staff, TMC/operations staff, system and software developers, manufacturers and people from private and public sector that are users of this standard. We will be getting a bit technical in places and I just ask that those that are stretched a bit in those areas to be patient while we try to reach everyone who's taking this module.

Ralph Boaz: This is a list of recommended prerequisites to this module. We won't be going through these now. You can take a look at that at your leisure. I'd just like to say something about these last two modules, A207a and A207b. A207a is very basic, about transportation field equipment, but provides the background for why we've developed the ATC 5201 and ATC 5401 standards. So A207a is important for background information, understanding what's out there in the field today and then we go into the hardware aspects in A207b and today we'll be going into the application programming interface.

Here's an abbreviated graphic of the order. It doesn't actually even have all of the prerequisites we recommended, but you can just see that there's order to this.

Ralph Boaz: These are our learning objectives today. We'll identify the features of the ATC 5401 API standard. We'll describe the architecture. We'll describe how the standard works with other ITS standards. That will be a really important part of the course, again, putting together a lot of things that you've learned in the PCB program. And then we'll look at specifying API's software for system and equipment procurements. Our first objective here will be a bit longer than the others. The second one will be a bit more technical. Again, we'll bring you all through that. Identify the features of the ATC 5401 standard. We will talk about sharing the resources of the controller and field cabinet system. We will talk about the structure of the standard and we will talk about the features of the API software. Here is a list-- it's not a complete list--but it's a list of applications that have been discussed and identified four uses with ATC controller units. There is often course traffic signal control/management, transit and light rail priority, emergency management, lane use, red light enforcement, speed monitoring/enforcement, access control, advanced traveler information systems, data collection systems and data distribution, and connected vehicle applications such as safety and eco driving. Now the more traditional roles for traffic control equipment is listed in the first two traffic signal control management and transit/light rail priority. However, with what we are going to be talking about today, we will be able to see how these other applications, how the ATC can support some of these other applications and what the ATC program provides.

Ralph Boaz: if you recall, from our previous modules, the transportation field cabinet system or TFCS, is composed of the following elements. There's inputs, those are the things that come in from the field. We have the controller, which is the brain of the operation. We have the outputs, we have monitoring of the cabinet. We have the power supply. That supply is power to the elements in the cabinet and then there is an internal communications bus. So if you look at the field equipment, in the street there are field sensors and we have areas on the streets that we call zones that are used to detect the presence of vehicles or sometimes there are concerns for pedestrians and bicycles and the detection technologies are getting very, very sophisticated. Then there is also what we call the field displays, and most commonly these are the signal heads out on the street, but they could be all kinds of other devices that we have talked about in those other applications. So what happens is, we look at the basic operation. So from the field sensors, those come into the TFCS as inputs to the system. Then those inputs are sent to the controller. The controller is the brains, it's the computer, the field hardened computer that resides in the TFCS. It in the typical traffic program, the inputs all the zones out on the street are mapped to turning movements for traffic, and the controller has that knowledge of that mapping, which inputs apply to which turning movement. So then the controller can decide what's the safe operation to give service to a vehicle. The controller then sends this information to the outputs. The outputs are essentially switches, we call them load switches, that are controlled with something like 24 volts, to control them, but they actually allow the power in most commonly 120 volts going out to the signal heads. So the controller makes the decisions, then decides how to turn the signal heads in a safe manner. Just in case something happens, we have something called the monitor or monitoring element and that make sure that you never get a conflicting arrangement on the signal heads, so that you cause an accident. The monitoring does lots of things that it does, besides monitoring conflicting colors, and making sure there's no shorts and such, in terms of the times, allotted times. And then in some cases, depending on the TFCS standard that's being used, it may send the information back to the transportation controller. Now it's important to note that these inputs and the controller, the operation of the controller, and the outputs and the monitoring, if we want to be able to run multiple applications simultaneously on this device, we need to be able to share those elements of the TFCS amongst the applications. Until the ATC program, A TFCS was typically the port for a single application. Notice that power is not discussed in here, because power is just distributed in whatever fashion is current to the technology, and the API has nothing to do with that. Internal communications are shares depending on the technology being used in the cabinet. But these are the main things: inputs, the controller, the outputs and the monitoring. These are the things we need to shed to achieve our goals. So here is another illustration of what we're trying to accomplish. So at the top here, you see a common front panel of a controller and then in the center here, is our TFCS. You see an intersection control. It's been running an intersection control program and then we see it's also running a ramp metering program and then it's actually running a connected vehicle program where it's sending the phase and timing information to the vehicles as they approach the intersection. Then all those other applications that we talked about

previously, they all could be running on this device, on this TFCS also on the controller, but I'm not suggesting all of those are running all at the same time. The point here is that this is more than just a traffic controller. It's a field hardened computer for our industry.

Ralph Boaz: So let's take a poll, do our activity. Which of the following elements of the TFCS is not shared by API software? Your answer choices are a) outputs, b) inputs, c) controller, or d) power. Please make your selection. If you selected power, you were correct. API software has nothing to do with the distribution of power within the cabinet. The fourth element that was shared is monitoring and if we want to go through the other answers, if you answered outputs that was incorrect, because the API software allows applications to share the outputs. It also allows application programs to share the inputs. It also allows application programs to share the controller and its resource and as I said previously, the fourth element shared is monitoring.

Ralph Boaz: this is the structure of the ATC 5401 standard. There is a section of introduction that has things like the purpose the scope, definitions, references and overview, those kinds of things that are in an introduction. There is an overall description so we have a product perspective. This overall description covers things that we've often talked about as being in the concept of operations the con ops. There's user characteristics, constraints. Essentially we identify user needs in this section. Then we have the requirements section and these requirements are organized by what we call API manager requirements and the API utility requirements, and then there's various other kinds of requirements listed in there and then we have the application programming interface. So these are C language function calls that are documented in the same fashion as Linux man pages. And for those of you who don't know, I'm speaking to the software people in the course right now, but they will understand that. So this detailed interface tells the user developers how to interface to the API software to achieve the goals of the program. Then there are appendices and in appendices there's traceability matrix, showing the connection of the user needs, the requirements to the functions. There are some code examples and there are other things that are helpful to the developers, and then there's an index. Now let's clarify some terms because this can be confusing. ATC 5401 standard specifies software. So when we say the API standard, the API standard itself specifies software and usually I will try to say API software, but sometimes we just say API for short so when we say API, we are usually talking about the software that is specified by the standard. Now API software operates on ATC controller units, allowing the application programs to share the resources of the controller and of the TFCS. So there's really two kinds of users of the API software. There's operational users and those are technicians and transportation engineers that use the applications, programs that run on the controllers, and then there's user developers and those are the software people that design and develop the application programs that run on the equipment. So there's two types of interfaces provided by API software. There's user interfaces and those are the interfaces defined for operational users so that multiple

programs can run concurrently on the ATC controller unit. So these are user interfaces defined by API software. And then there's the software interfaces and these are the C language function calls that allow user developers to create the programs that share the resources of the TFCS. So you have user developers, you have operational users and user interfaces. You have user developers and software interfaces. If you were to look at the ATC 5401 standard and you're an operational user, these are the selections that would be of most interest to you, because you would understand what the software is trying to do for you, and that's explained in plain English in sections one, two and three. The index can help you find things throughout the standard.

Ralph Boaz: There are four feature areas or main features for API software. There's front panel management. There's field input and output management, or field I/O management. There's access to the real time clock on the controller, like many applications have in our industry, are all about time, so we need access and management of the real time clock. And then there's a thing that we call API utilities and we will go through these features as we go along here. So front panel management: there's a front panel manager window and that's the user interface part that will allow you to have access to the application programs running on the controller. There's a C function interface, as we said, for the user developer to be able to interface with this front panel management system. There are application programs that are able to use dedicated windows provided by the API. So these dedicated windows are similar to a very scaled down version of a Windows operating system in the sense that there are different windows that pop up for the different programs. So operational users interact with the window or application program that we say is in focus. So whichever one, if you're a traffic signal guy and you're working with the traffic signal program, you would select that program and that would come up with all its menus etc., whatever the application interface looks like. It would pop up and that would be on the front panel, and that's what the user would see. IF you were some emergency management guy and there was a program running on this and then you might select a different program and it would pop up on the screen. All those other programs are still running. It's just what the user interface displays. We say that window's in focus. There's a default window and that's the window that has focus when the control unit is powered up. The API was designed to work with screen sizes up to 24 lines by 80 characters. Other common configurations are 24 and 16 and eight line displays. Then there's a screen and keypad defined by the ATC 5201 standard, and that's a minimum screen and keypad definition and capability. It is allowed in the standard for conforming controllers to exceed those minimum standards. This is the minimum keypad defined in the ATC 5401 standard and these are commonly seen on 2070 controllers, so that's where we thought that we should say this interface, this screen keypad should have at least what's there.

Ralph Boaz: We talked about a front panel manager window. We have to have a way to move from one program to another to allow the different users or different application

programs to be used and that's what this screen does. So this screen is in focus, users can select the user interface they want for their application by simply entering the number for that application. Now I want to make a point here, is the applications that are listed in this example are for example purposes only and they're not included in the API software. So everything that has a number next to it there is not a part of the API software, but is the application. Now one feature that's nice is, once these are assigned, for instance, if you were in a ramp metering program, you have that display up, you don't have to go back to the front panel manager to get to the other programs. You can just hit star, star, and the number to go to these other application programs. If you did want to come back to the front panel manager window, hitting star, star, escape brings this front panel manager window into focus no matter what application program you're in. There's up to 16 windows that can be assigned to application programs within the API software. That does not limit the number of application problems, but just that we had to put a limit on how much resources we were going to use in the API software. So, for instance, if we had a 24 line display here, you would see all the rest of the numbers. In this case, what happens is the top two lines and the bottom line stay fixed and if you use up arrow and down arrow, it will scroll down through the other application programs. This is just an example, so if someone would have hit star, zero, this ramp metering program would come up and it would have the interface look and feel of whoever provided the software. So this kind of thing really helps when you have technicians and maintenance people that are used to what a particular program looks like, how you enter data, etc. etc. So the application programs look and feel just as if they were the only thing on the machine. Here's just an example of intersection control program. And here's an example of emergency operations program. Again, this is just invented. This is not a product, but I'm hoping that part of this module will be to inspire you and the people you know to be writing applications for ATC controller equipment.

Ralph Boaz: I talked about there being a C function interface, so this is for the software people. There's general functions to open, close, refresh, clear windows. There's window attribute functions. There’s reading a function. You have to get input from the keypad. There's writing functions to write things onto the screen and you have to go to the standard to go into more details and we won't go into more details here either. Let's take another poll. Let's do another activity here. Which of the following statements is true regarding the operation of the front panel manager window? The default window is always position one on the front panel manager window? B) an intersection control window comes standard with API software, c) pressing star, star, escape causes the front panel manager window to be put in focus, d) the front panel manager is designed for an eight line display only. Please make your selection. Let's review our answers. If you said c) you are correct. The star, star, escape causes the front panel manager window to be put in focus. It works regardless of what window is in focus at the time. If you said a) this is incorrect. The default window is user selectable and assignable by pressing star zero through F and enter while you're in the front panel manager window. And actually, I think

the question was, the default window is always in position one. The default window isn't always in any position. It's user selectable. If you said b) this was incorrect. The intersection control is an application and that's not contained in the specifications for the ATC 5401 standard. If you said d) that the front panel manager is designed for an eight line display only, that's incorrect. It's set up for eight, 16 and 24 lines.

Ralph Boaz: Now the second major feature we will look at here, after talking about the front panel manager system, is the field input and output management. Now there's no user interface for this. It's a C function interface only. What it does is makes it easy for application programs to be written so that they can work in all of the major field architectures that are deployed today. That's the Caltrans family of 33X or maybe 3XX type cabinets, NEMA TS 1, TS 2, Type 1 and 2 cabinets and also ITS cabinets. The way that works is, using the API functions, application programs are able to query the API to determine what types of devices are used in the particular TFCS and then they know, because of that, that the software can make adjustments to talk to those types of devices. The way this works is the application programs register to access a field input/output device. Now it doesn't hurt to read something, so application programs have read access to all the inputs and output field devices. Although different applications may only be interested in some of them, it's conceivable that some sort of monitoring tool or system function might want to look at all of them, maybe some tool used by a technician, etc. However, application programs must reserve exclusive writing access to the output points of a field I/O device and this makes sense because you only want a single application to be able to write to, for instance, the traffic signals displays. Otherwise, you could potentially have a conflict and unsafe conditions, etc. So the point here is, again to review, application programs can read all inputs and outputs, but application programs must reserve, it's a register and reserve process, to be able to write to the output points of the TFCS. So let's go through a little illustration of that. This is a graphic representation of an ITS cabinet and there's the parts of this. The SIU components there have to do with the communications. It's a 0 interface unit and the AMU, the auxiliary management unit, that is part of the monitoring function which is covered by CMU, or cabinet monitoring unit. So we have the controller, we have the input cage. That's where all the input devices are located. At the bottom we have the output cage and we also have the power being distributed within the cabinet through the power distribution assembly. So let's say our controller is running these five applications, a ramp meter program, a signal program, emergency management, maybe some sort of data distribution program that has to do with maybe sending traffic information to the next controller downstream etc. Or as we talked about, maybe there's some sort of system checker here. These are just examples. So what happens is, for instance, if these applications are going to write to the output devices, this is a blow up that I just put up here on the screen. It's a blow up of the output cage and these boxes with the three LEDs on them; those represent the load switches that go to the traffic signals displays. So we might have a ramp meter program that boots up and it requests access to two of the load switches. Then the emergency management

system maybe activates a sign or a gate or something and it controls another load switch. Then maybe all the rest of the load switches in the cabinet are used by the signal program. So what happens is, as these programs are invoked, as the computer is turned on and these programs are loaded, they do their register and reserve on these load switches. Note that there is no communication that's necessary between these programs. The API handles this and manages it for all the applications running on the controller.

Ralph Boaz: So there's quite a bit that goes into this. This again is for our software people. There are general functions for registering and deregistering, enable, disabling the field I/O points, etc. There's input configuration functions, there's output configuration functions. There's frame functions and that has to do with the communications within the cabinet. If you're one of the later TFCS architectures that use a serial interface, you need to be able to work within that communications structure and there's a transition buffer. This is getting into a very detailed operation of a traffic controller. There are those functions. There's watchdog and health monitor functions. We talked about the monitoring system and we need our application to be able to work within that, and again, there's also the fault and volt monitor functions, and then the CMU/MMU channel functions. Those are the channels that go out to the devices themselves. Again, this is all for the software people and you can refer to the standard to get more details. So we've talked about the front panel, we've talked about the field inputs and outputs and now we'll just talk about the real time clock. Linux does not allow application level programs to access or to set the real time clock, manipulate the real time clock and we need to be able to do that to make our applications portable, so the API provides an interface to do that. There's set and get time functions, there's daylight saving time functions and there's time source and signaling functions in case there's maybe a power outage and you move from the street power to, for instance, a backup system. Let's take a poll. Which of the following is true in regards to the field I/O management of the API software? Is it a) all application programs may have read access to all of the input points, b) all application programs may have write access to all of the input points, c) all applications programs may have write access to all of the output points or d) application programs must communicate with each other to avoid writing to the same output points? Please make your selection? Which of the following is true in regards to field I/O management of the API software? If you said a) you were correct. All application programs may have read access to all of the input points, and there’s no restriction on that. If you said b) this was incorrect. No application program has write access to the input points. C) this was incorrect because application programs must reserve exclusive write access to modify an output point. It's the register and reserve operation that needs to take place for an application to be able to write to an output point. And then if you said d) this was incorrect. The API software provides the reservation and reserve capability so that application programs don't need to communicate with each other to use the TFCS resources.

Ralph Boaz: Finally we're getting to this last feature area. We're going to talk about the API utilities. We have a window, and if you recall from the front panel manager window, if you hit the next key, it would go to the ATC configuration window. This is something similar to a control panel on a Windows-type PC, a place where there's system-wide things set. So there's standard utilities that do come with API software. There's ability to set the system time. There's the setting up of the Ethernet ports, selecting, enabling and disabling systems services. There's getting the information about the Linux operation system that's running on the controller or the API software information. Then there's a part of the ATC controller called the EEPROM, where there's various information stored about the controller and this utility allows the user operator to take a look at that information. Now I want to say, these utilities are the standard ones that come with the API software. It's extensible so additional utilities can be added later. Now these utility pieces of software differ from application programs. Application programs are the things that are running constantly on the controller. It may or may not be assigned to a window, but those are operating all the time. Utilities are things that are selected from the configuration window and they typically pop up and the user makes a selection to change some system wide parameter and then they exit. So that's kind of the difference between the utilities and the application programs. And to use all this capability, there's of course a C function interface for the software developers. So this is the default look of the ATC configuration window and again, you select which utility by selecting a zero through F on the key. And again, there's also a blank here, because it's extensible. Other utilities can be loaded. Now remember, you get here by hitting next from the front panel, but you can also get here by pushing-- I'm sorry, I can't remember what the function was for that, but we'll just move on here. This is what the system time utility looks like. There's the current date/time, time zone and whether daylight savings time is enabled or active. That's the daylight savings status. Then there's a method for changing that where the arrow keys are used and the keypad is used what you want it to change to. And then at the point you want to apply it, you hit the enter key, or you can exit by hitting star, star, next. Here's the Ethernet configuration utility. There's all the typical things that you would need to set up for an Ethernet interface you can see on your PC. So there's quite a bit of other parameters that you can scroll down to. Here is shown the packets that were good that were sent, the packets that were bad were sent, packets received that were good, the packets that were bad. These are the packets of information that are coming across the Ethernet media. Your IP address and subnet mask, etc., things you'd recognize. This is a systems services utility and we've just invented here two services. One is a Bluetooth service and another one is a signal phase and timing service. This would be something that's connected with the connected vehicle program in the vehicle to infrastructure integration portion of that. So these services are again just examples, but this shows you how something that applies to the entire controller could be used as a service that the user may or may not want to enable. He would do it from this menu. So there's a C function interface to use that kind of capability, to use the configuration window, there's a reserve and release of that resource. There's time of day functions. And the way that these different utility programs are loaded into the configuration windows is very simple

for developers. You simply have a configuration, you have a file and on the left side of the file, you have the configuration item name and what that is what do you want to appear in the configuration utility. Then on the right, there's the path name; that's where your program is on the controller and so when someone selects the configuration item name on the config utility window, then that executable will be activated. We have another activity here. Which of the following is not a standard utility provided in API software? A) Ethernet configuration, b)Bluetooth configuration, c)Linux/API information or d) system time. Please make your selection. Which of the following is not a standard utility provided in API software? If you said Bluetooth configuration, you were correct. There's no utility for Bluetooth communications currently in API software, but a utility could be created and added for Bluetooth by a third party. So if you said Ethernet configuration or Linux and API information, or system time, you were incorrect, because those are all provided within the API software. The Linux and API information is a standard utility that you can get about the ATC operating system and then the build, the software information about the API libraries.

Ralph Boaz: Summarizing our learning objective, we talked about identifying the features of the ATC 5401 API standard. We talked about sharing the resources of the controller and the TFCS. We talked about the structure of the standard and then we looked at the features of the API software. Now we'll go into learning objective two. We'll describe the ATC 5401 architecture. We'll talk about key elements of the architecture. We'll describe how the ATC 5401 standard works with the ATC 5201 standard, and that area will get a bit more technical there. Then we'll talk about ATC software portability, compatibility and interchangeability. These things are some of the primary goals of the ATC program and would apply to everyone. Key elements of the ATC 5401 architecture is that it's based on the engine board concept that was specified in the ATC 5201 standard, which we spoke about in module A207b. It uses the Linux operating system that was defined there, which is open source, multi-process and multi-application. It allows multiple application programs to operate concurrently on the ATC controller units, as you saw in all the previous learning objectives, all those different screens. It showed these programs running at the same time. It shares the resources of the front panel, field I/O elements and of the TFCS and the real time clock. And then it provides source code portability, compatibility and interchangeability. Here's a graphic illustration of the ATC engine board. You see there's the Ethernet interfaces, there's a USB port. There's a real time clock. You see there's memory and the mini serial interfaces and there's a CPU chip running the Linux operating system. And those brown things at the top and bottom represent the connectors that are used to connect the engine port to the host module of whatever controller it's operating. So this is an actual picture of one company's ATC engine board. Just to give you a little bit of a real feel, we'll step through this a little bit. Here's our CPU chip, again it's running a Linux operating system. There are three kinds of memory. There's DRAM. That's the typical computer memory you might want on a PC. Then there's SRAM and that's fast access memory that retains data as long as power is

supplied. Think of cache memory on a PC. Then there's flash memory and that's memory that retains data without power. Think of memory cards, USB flash drives and solid straight drives. So then there's the real time clock. There's a little chip there for that. There's Ethernet ports. There's serial I/O built into the CPU, but there's enough ports on this engine port that there's additional ones to be controlled, so the serial I/O chip there. Then there's this USB chip and the ATC controller units use a USB interface for plugging in memory devices. It's not to drive printers or do other USB things, but it's used basically for file transfers.

Ralph Boaz: This is the layered architecture when you put the standards together. At the bottom of this picture, you see the engine board, and we have it laid down on the side here, and we call that the hardware layer. Then built onto that hardware layer is the Linux operating system and device drivers, and we call that the ATC board support package layer. Those things are actually brought to you by the manufacturers, typically. Then we have the API software layer and that API software interfaces through the Linux operating system to control the things that need to be shared on the controller unit. The next layer up is the application layer and that's where application software will use API functions and then in some cases, the direct Linux/UNIX calls to actually operate on the controller. And depending on which screen you're looking at, the operational user will be interacting with the API software, or it will be a pass through to the application programs. So this is kind of the structure of how the things work on the ATC controller unit itself. If you look to the left here, you see that the bottom layers are hardware layer and ATC board support package layer. I mentioned that came from the manufacturers. That hardware and operating system is defined by the ATC 5201 standard. The API software is defined by the ATC 5401 standard. By the way, the API software that comes on the engine board will probably come from the manufacturer also, but we'll talk about that more in a minute. Then the applications may come from multiple sources and that's the idea. We're really opening the TFCS to be able to serve a lot of different kinds of applications. So we saw this picture previously in A207b. This is focusing in on the interfaces on the engine board. There's a console port and that's for user developers to access the operating system. Then there's general purpose serial ports for doing things that application programs might want to do. There's a front panel port that connect s to the screen and keypad. There's the field I/O interfaces and that connects to the equipment in the TFCS. There's that USB memory device interface. There's something called a data key. That's a port for non-volatile memory. These data keys, which are seen on some controllers, are an optional feature of the ATC controller standard. Then we have the Ethernet ports, which are used of course for putting this into some sort of connected system. So we talked about managing the different features of the controller. So when we are running API software on this engine board, the API software then writes on top of Linux and controls the front panel. It controls the field I/O devices and it controls the real time clock. I just put that in that little box there within this diagram. If you'll notice, there's these general purpose

serial ports. Those aren't controlled by the API software at all, because they're reserved for use by the application programs directly.

Ralph Boaz: So let's talk about portability, compatibility and interchangeability. First we'll talk about portability. That's the ease with which application software and data can be transferred from one application program platform to another, and that's the ability to move with minimal changes application software between computers. That's portability. And this is a graphic illustration of that. Now it's important to note that portability is not like what one might think, where you have three PCs in your office, they're from different companies and you can buy one box of let's say Word or Office and you can put them on your different platforms using the same DVD. It's not that kind of portability, because in our case, these engine boards may be made with different CPUs. Even though they have the same operating system, they may have different CPUs and board support packages. In this case, we call it source code portability where the application software is compiled and linked for whichever engine board it's going to be resident on. So it's important for application writers to use the API functions wherever they're applicable, right? Because if you went down and used the Linux functions, then the API software wouldn't be able to manage those applications for you. So again, this is source code portability. And you might think of it as like a Macintosh and a PC, where you can get Word or your protection software for either of these. You might buy them from the same company, but they actually have a different object code to run on those and so the company provides different versions of that, and that would be the same case here, where you would talk to your provider and say, "I have brand X's engine board here and brand Y's engine board here and would you give me your application program for both of those." That's done by the provider, by compiling and linking using the application source code and the API software libraries. Let's talk about compatibility. That's where two or more systems or components perform their required functions while sharing the same environment, and that these components, they don't need to communicate. To be compatible, they don't need to communicate with each other, so we're talking about residing in the same environment here. The API software allows that. Application softwares use the API functions so that those resources can be managed and consequently, these applications can run concurrently on the same environment on an engine board. Now let's talk about interchangeability. That's where an application might have the same functional and physical characteristics, so as to be equivalent in performance and durability. Now we say that's subjective, because vendors may argue about their performance and/or features being superior to others. And then we also call interchangeability the ability to exchange devices of the same type without alteration of the device or adjoining items. In this case, we could say programs, ability to exchange the programs of the same type without altering the program or adjoining items. Now we say again that this is subjective. It's really in the eye of the beholder of whether something is interchangeable, because for instance, a good example is the case of internet browsers. Some people may prefer Firefox. Others may prefer Internet Explorer. Others may prefer Chrome. And if you look

at those, you can say, "Oh, they're all internet browsers, so they're interchangeable." In fact, I will switch between those in my own day to day operations here at my office. I'll switch depending on what's needed. So at a high level, they're interchangeable, but at a low level, you'll have camps saying, "No, no, this is superior," or "That one's superior," depending on their features. So interchangeability is, in a sense, in the eye of the beholder. Now being that the software may be interchanged, in other words, you may have a controller that your running company A's software in and you're going to do an upgrade on your system years later for instance, and you may want to buy company B's software and run it on the same device. Well, it's still considered interchangeable if you put in the new software from company B and set it up. That's still considered interchangeable. So this is an example. This is kind of a graphic example of interchangeability, where we have the compatible software here on the left two-thirds of the screen and we show that an application that's running, application 4 that's shown on the picture is interchanged with application 3 and that's because these applications are of the same time, we say they're interchangeable. Okay, let's do our activity. Which of the following parts of the ATC engine board is not managed by API software? Your answer choices are a) the front panel port, b) real time clock, c) general purpose serial I/O ports or d) field I/O ports. Please make your selection. Which of the following parts of the ATC engine board is not managed by API software? Let's review our answers. If you said c) you were correct. The use of the general purpose serial ports is controlled by Linux, and not the API software. So those are reserved out for application programs to use. If you said a) this was incorrect because API software provides a window in capability for the application programs. If you said b)real time clock, the API supplies a system time utility and clock functions to allow users and application programs to set the real time clock. And if you said d) field I/O ports, you were incorrect because API software manages the field I/O ports of the engine board and thus the field I/O devices, allows the applications to share the field I/O devices within the TFCS. Let's summarize our learning objective. We described the ATC 5401 architecture. We looked at key elements of the architecture. We described how the ATC 5401 standard and the AC 5201 standard work together. And then we discussed ATC software portability, compatibility and interchangeability.

Ralph Boaz: Now we'll go into learning objective three, and here we'll describe how the ATC 5401 standard works with other ITS standards. We'll talk about transportation field cabinet systems, the various ones. We'll talk about the ITS communication standards and then we'll talk about considering new ways to solve ITS issues. So the ATC 5401 standard works with all of the major TFCSs out there today. ATC 5201 standard we talked about in A207b, specifies controller unit that work with Model 332 cabinets or that family, NEMA TS 1, NEMA TS 2, and NEMA 2 type 2 cabinets and NEMA TS 2 type 1 cabinets and the ITS cabinets. The API software provides the internal cabinet interfaces suitable for all the major TFCSs. I should say API software provides the access to those devices within the TFCS. Because of this, the application software can be made portable, compatible and interchangeable to controllers designed to be used in any TFCS through

compilation of the API libraries, which we spoke about previously. So this whole idea here is that all of this is critical to upgrading the infrastructures in a cost-effective manner. In other words, you don't have to replace all your field equipment. You can either install ATCs or maybe you already have an ATC in there, and this API software will provide all the capabilities we've talked about in this module. So the ATC 5201 standard and 5401 standard provide the computational power and interfaces for ITS communications. This is very important here, is that most TFCSs that use 170 controller units will require replacement hardware in order to utilize NTCIP ASC communications. That's talked about in another module in the PCB program. So center to controller communication, the ASC, actuated signal control communication, the 170 units don't have the computational power to do that. The ATC standards provide the capability to support multiple applications using different or the same types of NTCIP communications simultaneously and we'll show you a picture of that in a moment. So not only can you run the ASC, but you can run multiple ASC programs, programs that are using multiple different kinds of ASC communications or various other standards. And the ATC standards provide the interface and computational power for applications such as adaptive control and vehicle to infrastructure integration. So these are just a couple of examples. Here on the left, we show an ATC that is running in a model 332 type of cabinet. And on the right, we have an ATC that is running in a NEMA TS 2 type of cabinet. Notice they look quite different and their internal architectures could be quite different. The appearance of the controller can be quite different, but they're all running key parts of these ATC standards. We'll go there right now. So here we show just the top portions of that picture that we just described, of the ATC controllers running in the TFCSs. These are defined by the TFCS standards and specifications that have been used for decades in our industry. So if we make these controllers and we use the ATC, based on the ATC 5201 standard, that's what specifies the engine board with all these capabilities and the Linux operating system, etc., and then based on the ATC 5401 standard, we have this software running on this engine board, on top of the Linux operating system, so that we can share the resources of the TFCS. Then we can run multiple applications doing all kinds of ITS applications, solving all kinds of different ITS issues. So this, just for example, here we show a signal control application running with ASC communications. We have a field master program running, using the FMS standard. Then we have a ramp metering program using RMC, or the ramp metering control communications. We might be running the signal phase and timing message service that provides the intersection information to vehicles. There's basically all of those other applications that we talked about at the very beginning of our module here. We talked about emergency management, lane use, advanced traveler information, data collection and distribution. All these different applications that we're talking about can now run concurrently on a single controller unit and share the resources of the transportation field cabinet system. So really let this soak in, because this really captures the picture of a lot of our PCB program right here.

Ralph Boaz: Now because of this capability, we can consider new ways to solve ITS issues. First of all, the open systems we've talked about promote innovation from more sources. The Linux operating system is available. You can download it from the web, from internet. A developer can see the definition of how to interface to the equipment through the API, etc. So it just promotes more innovation. It's opening up new markets for third party software, so we have new methods for solving ITS problems. For instance, software that's duplicated in the field devices may be able to run as a single application on the ATC controller. That may drop the cost of the field devices. You may have custom applications. You say, "You know, I don't really see a product out there that meets my needs and I have the money. I have some funding. I'd like to have a custom application that does this." So if you're going that route, of course good systems engineering practices are essential and we've covered a lot of that in this PCB program. Now I'm not saying that everything should be a design build project. That's not cost effective. I'm just saying, you have choices now. Let's have another activity. What must a user developer do to make an application portable and compatible on two different ATC controllers in different TFCS architectures? Choose the best answer. Is it a) write the application software so that it uses ATC 5401 function calls whenever they're applicable, b) compile and link the application source code with API libraries, c) compile and link the application source code for the ATC engine board used in each controller or d) all of the above? Please make your selection. Recall that we're asking for the best answer and so the best answer was all of the above. Writing the application software so it uses ATC 5401 function calls, that's essential for making the software portable and compatible, but there are others, so a) was not the correct answer or best answer. B) wasn't the best answer, even though compiling and linking the application source code is essential to creating software with the API libraries, making the software portable and compatible. That's essential, but there are other answers, so it wasn't the best. And then c) wasn't the best either, but this is true, it's an essential part of making the software portable and compatible, is to compile and link the application source code for the engine board that is going to be used in the controller. So let's summarize our learning objective three. We described how the 5401 standard works with other ITS standards. We talked about different transportation field cabinet systems. We talked about the ITS communications standards and we talked about considering new ways to solve ITS issues.

Ralph Boaz: Now we'll go to learning objective four, specifying API software for system and equipment procurements. Here we'll discuss how the 5401 fits into the systems life cycle. We'll talk about the development of migration strategies and mitigating deployment issues. We'll talk about creating a specification based on the ATC 5401 standard. So if we're talking about a system, in the system's life cycle, the TFCSs, including the controllers, can be considered a subsystem of a center to field system. TFCSs may be in place for multiple center-to-field systems and generations of center to field systems. For any of you operational folks, this is old news. ATC controller units may extend the life of a center to field system. They may extend that life by, maybe there's some applications that

may be able to be provided that run actually in a field to augment the central system software. And deployment of the ATC 5201 and 5401 standards may occur at any stage of the life of a center to field system. So there are methods and ways to do that so that if you're looking at your transportation infrastructure from a system point of view, there are ways to integrate the ATC standards into that at any point. That was somewhat discussed in A207b. So we suggest that you develop a migration strategy, you have a written plan and policy for deploying ATC equipment as part of a strategic plan; that you write your procurement specifications and procurement processes and that those are integrated with the strategic plan. They go together. You need to consider some things that may influence your migration strategy. Will it be working with the existing TFCSs? In other words, if you're going to be migrating, are you going to use the existing ones or going to be moving to new ones, and which ones are you going to be moving to? This will affect what you say in gathering your software. Are new projects going to be paying for this? Are you going to add them as new projects come on? Do you have regularly scheduled maintenance? I know the agencies do. DO you want to upgrade your systems as part of that? It's possible to upgrade the ATC hardware while using an agency's current application software. In other words, with some of the vendors, you can buy a version of the current software you're using for the ATC units. You may be upgrading to new operational software. You need to consider that in your plan. You may want to consider compatibility with existing or planned system software, whether it's staying or whether you're getting something new. And then in changing over to the ATC equipment, there is going to be some training for your operational staff. Let's talk about mitigating deployment issues. Although the 5401 standard has been published, complete API software meeting the standard is not yet available. The USDOT has started a project to create a reference implementation of the API software-- we call it the APIRI-- for public use in 2015. This APIRI project includes development of a validation suite to test the API software on an ATC engine board. Here's this concept. So what will happen is, we'll have the API software run as an open source project that can be accessed via the internet. Then manufacturers and software vendors and consultants and agencies can all have access and actually help maintain that software. This is just analogous to the way Linux is maintained today. So you have this whole community that is making sure that their equipment runs using the software, and they are providing corrections, bug fixes etc. to the software. Then that way, manufacturers can include this reference implementation or base their API software on the reference implementation included in the products. So the benefits of this open-source approach for this API reference implementation, as I said, it is consistent with the Linux O/S operating system open source concept, and it promotes collaboration of developers, provides a forum for users to express ideas and concerns, promotes quick bug fixes and alternative solutions. It facilitates the introduction to new application developers where they can have access to this API software. It could be downloaded in universities where they are training about traffic and ITS and things like that. It lowers cost to the industry; there are no runtime licenses required. And it provides the best opportunity for consistent API software behavior. We have much better opportunity here for consistent behavior than having multiple vendors all trying to build

software to the standard itself. If there's any question, when we have this reference implementation, if there is any questions about how something should be done or how something should behave, then instead of pointing the fingers at each other, maybe they can say, "Oh, this is the way it should work as it shows in the reference implementation." Now that we have discussed the API reference implementation, how do we know that it works? Once it is on an engine board, how do we know that it works? We are also developing as part of that project, and API validation suite, which we call the APIVS. S APIVS software runs on a PC, and there is a test fixture used to host an ATC engine board with API software on it. Then the PC uses test scripts written in XML, which is for the computer guys, extensible markup language that drive the testing and the engine board and the AP eyes software which is residing now on the test fixture. There are crossover cables, we saw. Remember all those inputs and outputs on the engine board. We looked at all the connection points on the engine board and these crossover cables are used to connect the inputs to the outputs, then we can exercise the API software functionality. Again, this is to be made available through this open-source project. Here is an example. This is not an exact picture, it's still to be determined, but this is an example of where the various ports of the engine board of brought out as cable connections and the engine board would then plug into this test fixture. This next picture shows how the crossover cables would be used and then there is a connection from a PC over to the test fixture.

Ralph Boaz: Now, how to make sure you get what you want. We will talk about creating a specification based on the 5401 standard. The specifications for the API software is really a part of the ATC controller procurement. What you need to do is require the API software that conforms to the standard is available on your controller unit when the reference implementation is available. Many of the manufacturers may have started at one point in API software development but when the DOT said this was a good idea to develop as an industry, then those were put on hold, so there is no proven complete library at this time. So in your procurements, if you were procuring today, or when you do procure all, you want to make sure that it conforms to the standard and it should apply when that APIRI project is completed. So you want to establish a contractual commitment from the engine board suppliers to supply the software once the APIRI is available, and you also want to require that the tool chain and source code used to produce the Linux environment for the engine board, that's important so the manufacturers, so that the application developers know how to compile software for your ATC controllers. Now these may not come without a cost, something that could be negotiated, these last two points, as far as contracts are concerned, the tool chain, source code, and the commitment to supply the API software, that all needs to be negotiated during your procurement process. You should require a demonstration from manufacturers that each of the API interfaces are operational and multiple programs run concurrently. This is besides the application software that you may run. The vendor should have some sort of application program of their own that illustrates that these interfaces are operational and multiple programs, even

if they are just stops, are running continually, concurrently I should say. Then some agencies have software developers and hardware developers, and a few agencies may have those types of people on board, but most do not. So what you may have to do is get a certification from the manufacturer that the ATC being purchased past testing during the API validation suite. Of course, back is when it's available. So we have some additional guidance when procuring application software. Prior to purchase of that software, have the application providers demonstrate their software on your agency's ATC units. You want to establish a contractual commitment from the application providers to supply software compatible with the API once the API reference implementation is available. So you may want to buy an ATC application now, but what you want to do is build something into your contracts that says once the API software is available through the reference implementation, then I want that software to run on that also. It will probably take recompilation of some sort by your application vendor but you want to build that into your contracts. Again that's a negotiated thing. You want to purchase support service with your application software. You don't know what the future may hold so this is a good practice. Again that's negotiated. And then this is a suggestion here, that is that you separate your specifications for your hardware, the ATC 5201 and the ATC 5401 specifications of the ATC controller from that of your application programs. This way, you are really defining in your specification what the controller unit has on it, the software it has on it, a standard from the application programs which you may buy from the same vendor, or you may buy from other vendors. So you may end up being in one kind of procurement document, but it needs to be understood that really the software and the hardware are two different things in this kind of architecture. We have another activity here. We were lost here in our reverse condition. What is not a benefit of the API reference implementation? A) it guarantees but free software, b) it lowers development cost to the industry, c) it's the best opportunity for consistent API software behavior or d) promotes collaboration of developers across the transportation industry. Please make your selection. What is not the benefit of the API reference implementation? If you said a) guarantees but free software, you are correct. That is not the benefit of the reference implementation any nontrivial software there are bound to have unexpected behaviors or bug fixes required. It doesn't guarantee that it's perfect, but it guarantees that we have a method for dealing with any issues. So these other items in here are benefits of the API reference implementation. It does lower development costs to the industry. It's a joint effort of developers, costs are reduced for everyone. It's the best opportunity for consistent behavior. A common source defining the proper software behavior creates the most consistency. And d) if the APIRI promotes the collaboration of developers across the industry. Solutions and improvements are required to go back to the API. That's part of the license--back to the API group and then they are assessed and tested by a group of developers prior to being added to the APIRI. So summarizing our learning objective four, we discussed how ATC 5401 fits in the system's lifecycle. We said it can fit in just about anywhere in the life of the system. We talked about developing migration strategies and mitigating deployment issues, and we talked about creating a specification based on the

ATC 5401 standard. We gave you your suggestions on what to include in your specification.

Ralph Boaz: Let's summarize what we learned today. One, we learned that API software is grouped into the following feature areas: front panel management, field input/output management, real-time clock management and API utilities. We learned that the ATC 5201 and ATC 5401 defined a layered architecture that provides an application software portability, compatibility, and interchangeability. Third, we learned that the ATC 5401 standard defines API software which will work with all major TFCS, transportation field cabinet system architectures in use today. And four, we learned that the API open source reference implementation will promote a collaboration of developers across the transportation industry. Here we have a list of resources. There is more of that in the participants supplement. Now what we would like to do is go into some questions that have been submitted previously as this course has evolved. Someone had expressed a concern about applications competing for the CPU time or resources and that some of these can be kind of critical applications. Well, just to get in their ballpark, the processors that are being used in the ATC standards and being built by manufacturers today are up to 2500 times more powerful than are being used out in the field. So the point here is that you have much more power, but more importantly, there is a way during the integration or adding in of these applications, there is a way to prioritize the applications, and the red case then maybe some sort of conflict. But again, it's such a miniscule level compared to what we've been dealing with every day for the decades, that we don't see this as a big issue. Someone arts, does the API work with graphics as well as text? Yes it does. The API and the front panel will work with graphics as long as the manufacturers are providing a graphics LCD. It's up to the applications that are written to make sure that they display their graphics properly. It's just that the graphics part of it is not managed by the API software, whereas text is managed by the API software. Someone asked again, what is the difference between an application and a utility? An application runs continuously, a utility typically does something and then exits. Remember, you can think of the application utilities and the configuration manager window as the control panel of a PC. it was asked, how many applications can be run simultaneously using the API? The API was built to have 16 interfaces, but the Linux operating system, there's no limit except to how much speed on this processor and memory in the controller. There is no limitation on the number of applications that we can run under the API. There is a limitation of 16 when there's that can be used. The question was, it is said that the ATC promotes portable, compatible and interchangeable software. What about interoperable? Well really interoperability, the definition is, the ability of two or more devices to exchange information and the ability for those applications or devices to use the information that is being exchanged. So generally, interoperability takes place at the application level. For instance, one can talk about the interoperability of two applications using the NTCIP on the ATC controller. You could actually have NTCIP communications going not just from the TFCS to the central site, but there may be cases where you want NTCIP field communications

taking place between applications running on the same control unit, and that's made possible by the API software. But the communications itself, the interoperability part of it, is not handled by the software but by the applications. And then finally, someone asked, how long before the API reference implementation will be available? Officially the project is scheduled to end the third quarter of 2015, but we expect that there will be interim deliverables of software being tested by the manufacturers in 2014. And finally, keep an eye out for further PCB modules on the ATC equipment. Those are in the plans right now. Thank you all for participating, and this concludes this module.

End of 2013_12_06_13.42_Final_Recording_A208_.wmv