Module 32
A315b, Part 1 of 2: Understanding Requirements for Actuated Traffic Signal Controllers (ASC) Based on NTCIP 1202 v03 Standard

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 website, 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.

Ken Vaughn: Hi, this is Ken Vaughn. We're here today to talk a little bit about some of the ITS training courses that are provided through the Professional Capacity Building system by the USDOT. The complete list of courses is provided at the bottom website you see there, www.pcb.its.dot.gov. This course is focused on Module A315b, Part 1. The Specifying Requirements for Actuated Traffic Signal Controllers Based on NTCIP 1202 v03 Standard. This is part one of a two-part course, and it's been updated last in June of 2020. I'll be your instructor today. My name is Ken Vaughn, President of Trevilon LLC. I have quite a few years' experience with NTCIP, traffic signal controllers, ITS standards in general, including with testing, architecture, and standards development, and deployment.

The Learning Objectives we have today are identifying NTCIP 1202 v03 standard requirements then we'll talk a little about the purpose and benefits of the Requirements Traceability Matrix or what we call the RTM, and we'll consider how you prepare a Project-Level RTM and what needs to go in there. And we will conclude our session today with preparing an ASC Specification or at least discussing steps to prepare an ASC Specification. And then, as I mentioned, this is part one of a two-part course, and there will be additional topics in that second part.

That brings us to our first Learning Objective, which is identifying the NTCIP 1202 v03 Standard Requirements. Our key points within this learning objective are to understand the scope of NTCIP 1202 v03, identifying and understanding what a requirement really is, and then reviewing the outline of the standard and then considering the format that the requirements are presented in. Let’s take a look at the scope of this particular standard. In the top middle of this page you see our actuated signal controller, and then it has connections to three different pieces of equipment with red lines. Those red lines symbolize the scope of this particular standard. All three of those lines are addressed by this standard.

The first one, in the upper left, is the Traffic Management Center that connects to the ASC for managing and controlling that ASC. Likewise, I might want to manage and control my ASC from a maintenance or technician’s laptop out in the field. That link is also part of the standard. The third aspect of the coverage of the standard is new to v03, and it’s the connection between the ASC and a Connected Vehicle Roadside Unit. Connected vehicles are a new extension of the architecture and this new version of the standard includes the finding—how a signal controller connects to an RSU so that we can enable connected vehicle communications.

What’s not covered by this standard but generally are covered by other standards in the industry are how the ASC interfaces with the Signal Controller Cabinet itself and all of the components of that cabinet. And then how that Signal Cabinet connects to the actual roadside equipment you might see, such as your signal heads both for vehicles and pedestrians and detectors, once again both for pedestrians and vehicles.

Also not within the scope of the 1203 standard—but certainly directly related considering how the whole system works—is what the RSU is connected to. That RSU controller is connected to a short range wireless radio. That radio then communicates wirelessly to a range of different local road users, including generalized vehicles, specialized vehicles that might have special permissions, as well as pedestrians and other vulnerable road users. But discussion of this course will focus on those lengths that are focused with the ASC, mainly the ones with a Traffic Management Center, maintenance laptop, and the Connected Vehicle Roadside Unit or the RSU.

There have been quite a few changes within the v03 standard. One of the biggest changes is the fact that we’ve added systems engineering content. So whereas the v02 standard and v01 standards both contained the data definitions and some of the core technical details of how to implement NTCIP, this standard actually provides a lot of the backup material. We now have user needs that explain why you might want to use different features, and then those user needs are traced to requirements. Requirements giving you very precise definitions of what is provided, what is the intent in order to meet those user needs? And then the dialogs are also defined. So no longer do we just provide the individual data elements, but we define exactly how they’re supposed to be used. This all is designed to increase the interoperability of devices. So when you buy a device from one manufacturer, it’s more likely to work off the shelf with a device from another manufacturer.

We’ve also added support for some new user needs. I mentioned before in the previous slide, we’ve added the connection to the RSU, thereby enabling a connected vehicle environment. We’ve also added a couple of other features. We’ve added the ability to manage operational performance by monitoring at a very detailed level the operations of the controller, and then have also provided a way to support exception reporting. So rather than always having to pull your device, you can now configure your device to report back when special conditions occur.

There's also been a few other minor improvements but those are the main three big improvements, the user needs. We've also enhanced some capabilities for existing user needs by tweaking and adding some additional options. Also have done some general maintenance of things that have been identified with the standard. A complete list of all changes is available in Annex D of the document and that's true with all of the NTCIP standard series. Annex D is where you'll find the list of improvements from version to version.

Now that we understand a little bit about the scope and what's new in NTCIP 1202 v03, let's consider what are these requirements that we've added? How are they different than what we've discussed before? So let's consider what a requirement really is. INCOSE defines a requirement as "a statement that identifies a system, product, or process characteristic or constraint, which is unambiguous, clear, unique, consistent, standalone (not grouped), and verifiable, and is deemed necessary for stakeholder acceptability." So the key to understanding this perhaps not fully clear definition is the fact that you can group these three terms: system, product, and process as a single unit that describe the type of characteristic or constraint. So it's a system characteristic constraint or a product characteristic constraint or a process characteristic or constraint. And it's describing something that's unambiguous, clear, unique, standalone, etc., etc.

But when we talk about a characteristic or constraint, what do we really mean within our context? Within NTCIP, we're mainly talking about functional requirements, design requirements, and traceability requirements. There are many other types of requirements that we could go into, but these are the three areas that we focus on within NTCIP. We're not concerned about, say, the physical requirements of the unit or the electrical requirements, whether it operates on American-style power or European-style power or something else. We're concerned about functional design and traceability within the scope of NTCIP standards.

So now that we understand that we have these three different types of requirements, how do those requirements get relayed in the body of the standard? We've traced this map—we've mapped this out, color coded-wise, within the outline of the body of the standard. So the gray text indicates other tech that's not actually a requirement but is relevant to the discussion. The green text, in the case of the functional requirements, these define what we are trying to do. The design requirements, then, are in blue and identify how we achieve these functional capabilities. And then finally the orange is what ties it all together, and the orange ties the—for example, the user needs to functional requirements, as well as the functional requirements into the design requirements.

So if we look at the outline now, you see the first two sections are in gray. They are other text. They're not actually requirements, but they're very important. General section provides normative references, definitions, things like this, and Section 2 provides a concept of operations. That Concept of Operations includes your user needs formally defined, i.e., it lifts out specific features that users may want to achieve with their signal controller, and then those are all later traced to functional requirements. So it's a guide of how you select your features.

Section 3 provides those functional requirements as well as a Protocol Requirements List, or what we call the PRL, that ties the user needs defined in Section 2 to the functional requirements defined in Section 3. Sections 4, 5, 6, and 7 then provide the design details.

Section 4 is the dialogs about how the two components talk to each other and the sequence of the messages that are exchanged.

Section 5, 6, and 7 define different types of objects. Section 5 are the standard base objects used to control the signal controller. Section 6 defines what we call block objects. These are used for data upload and download when you're trying to exchange large portions of your database. It just provides more efficient operations. And then, finally, Section 7 deals with objects that we have coordinated with SAE on in order to support the connected vehicle environment. So a lot of those definitions, rather than reinventing the wheel, we imported from SAE. Those are contained in Section 7.

Annex A, then, is the Requirements Traceability Matrix, also known as the RTM. This RTM traces from the functional requirements to the design elements. So whereas the PRL traced from the user needs to functional requirements, the RTM traces from the functional requirements to the design elements being the objects and the dialogs. Annex B is an Object Tree. That's informational, describing how all of the different objects or data definitions fit together.

So Annex C describes the test procedures. Normally this is true for all NTCIP 1200 series documents. However, within NTCIP 1202 v03, this is a placeholder. We don't currently have test procedures defined for NTCIP 1202. There is a project that's looking at developing some test procedures, but they're not yet developed, so this is a future placeholder for test procedures in a future version of NTCIP 1202.

Annex D describes documents—is a documentation of revisions in this standard, and then Annex E describes other user requests that were not actually included in this version of the standard, so features that users may have requested. We explain why they weren't included. Sometimes it may be for a future version. In other cases, maybe we decided that they weren't really needed due to other ways of doing things.

Then Annex F, G, H, and I are, once again, requirement sections. Annex F provides some base function requirements. They are very generic and apply to all of the data exchanges. Annex G provides some generic dialogs describing how generic SNMP interfaces work. Annex H is a set of functional requirements that are generalized, that really almost belong in NTCIP 1201 or 1103. They may be moved there at some future date, but, in the meantime, we have formal definitions within this document.

There are also some dialogs contained in that section. And then Annex I includes some more details about communication ports and the protocols related to those. So that's the full outline of the standard. As you see, the functional and design requirements are scattered about. The key is understanding those traceability tables. They allow you to quickly identify where the requirements are located. You can trace it through the document very easily that way.

So how does this all work? We mentioned two different traceability tables, and if we use that same color-coding we had on the previous slides, you see that on the left, we have user needs. Those user needs are traced to functional requirements using the PRL, the Protocol Requirements List. That's the traceability that ties those two together. Those functional requirements are then fully described when you trace through the Requirements Traceability Matrix to the dialogs and the objects.

So let's take a little bit closer look at that with some examples. The PRL organizes functional requirements by user need. So it takes you down to a logical path of how must users would kind of see the different features that I want for my traffic signal controller. One of the challenges we have is, when you actually go to implement a user need, and detailing what you actually want with this particular need, you often start realizing that the same requirement is needed by different user needs. So if I want to manage my timing pattern scheduler or log user access, I have a need to be able to configure the time, because if I want to schedule something, I need a clock, and if I want to log something I need to timestamp that, which means I also need a clock.

So we have these common sets of requirements that relate to different user needs. So people sometimes ask, "Why don't we just define our requirements directly underneath the user need?" Well, if we did that, we'd have to redefine the same requirement over and over again. So instead we break those out into different sections, and then we provide this traceability table that shows how the needs trace to requirements, then, likewise, if you need to, you can identify requirements and trace back to the needs that are traced to them. So that avoids us from having to write duplicate requirements. We know what the requirements are. We write it once. It increases maintainability of the standard.

So we talked about the user needs being arranged logically through the way a user would use them, while the requirements, they come much more muddled, so we've organized these separately. We have architectural requirements where we talk about basic communications, logging data, exception reporting, and accessing the device. And then we have data exchange and operational environment requirements. There we talk about the basic configuration of your controller, managing signal operations, detector management, connected vehicle management, and backwards compatibility.

We then have Section 3.6 that provides supplemental requirements that are non-communications-related, or are at least non-data exchange-related. So here we get into things like response times, condition-based transmission start times, signal phase, and timing performance, things like this. So a lot of performance characteristics and other things that aren't directly related to requiring a data exchange itself, but other related requirements.

Annex H, then, deals with more generalized requirements that apply to many different devices. So we talk about general configuration, generic status monitoring, data retrieval, control, and performance. That's all how you go from a user need to a requirement, but once you've identified a requirement, how do you get to the design details? Well, as we mentioned, that's through the RTM, the Requirements Traceability Matrix. Each functional requirement involving a data exchange traces to at least—well, exactly one dialog—and at least one and typically more objects.

So if we see the diagram at the bottom here, we have a couple of examples of functional requirements in green. They're connected by the RTM on both sides in this case, in this figure. To the left, we have them connecting to a dialog. And, in this case and many cases, we use what are called these generic dialogs. This particular one is a generic configure table row so typically when we're interfacing with tables we have a very small set of dialogs that we just keep on reusing because the process is the same. On the right side, you see the specific objects we're dealing with for this particular requirement. And actually we have two different requirements here, and we discover they're traced to exactly the same two objects, which shows you, once again, we can have one or multiple requirements all tracing to the same object. Likewise, the same object can trace to multiple objects. So it's a many-to-many relationship on that side.

Let's take a look at how that works. We have—the dialogs are defined in Annex H.2, which is where generic dialogs are, which is the one we just referenced. There's also some more customized dialogs specifically for signal controllers in Section 4. The data elements or the individual objects that we talked about, are, as we mentioned, defined in Section 5, 6, and 7, as well as in other standards. So the traceability table not only traces to elements contained within the standard, but also elements contained in other standards with specific cause references. And those other standards include NTCIP 1201, NTCIP 1103, as well as some other RFCs developed by the Internet Engineering Passports.

Now that we understand kind of the base level of how everything connects together, let's take a look at how we actually write and put these functional requirements into the standard. We try to use the same basic construct for all of our statements so that we make them as unambiguous as possible. It comes with a structure where we have a localization, actor or action target, followed by constraint where the localization of constraint are both optional fields in that statement. The actor is obviously the person or thing that has a responsibility, that does the action. The action is simply what is to happen. And then the target identifies what is to receive or the object of the action. The two terms you may be less familiar with: localization identifies the circumstances under which a requirement implies and the constraint identifies how to measure success or failure.

Let's take a look at this in practice. We go back to the example we had in our previous slide, though we'll figure—configure vehicle phase minimum green time—and we look at the requirement as stated in the standard for that requirement. It says, "Upon request from a management station." That's in green, that's localization. Under what conditions does this requirement apply? The ASC, then, is the actor. When there's a request from a management station, the ASC, the actor, shall do what? It shall store information. So "it shall store" is the action, and then the information, the target, is the minimum amount of time the green indication is to be displayed for a phase.

So those are the three core components. The ASC is the actor. "Shall store" is the action, and the minimum of time the green indication should be displayed for a phase is target. Finally, we have the constraint. The constraint is that this will be recorded in seconds between 0 and 255 seconds. So a very precise definition conforming to our structure.

When we trace that functional requirement to an object, we have a similar sort of statement defined in that object definition. Once again, this is taken directly from the standard. So we're now in a section of the text that is designed to be computer-readable. So we can understand it, but the reason it's in this strange format is because we also need to be able to send it to a computer and have the computer understand it. So all of the spaces and line breaks and all of that are significant within this environment.

So the phaseMinimumGreen is the name of the object. It starts out with just an object-type. The syntax, the way it's presented, is an integer from 0 to 255. The access indicates that you can either read this information, retrieve it, or you can set it for a right operation. The status largely is kind of an archaic reference, but is required in the computer format. We defined our mandatory and optional capabilities elsewhere in the RTM and the PRL, as we discussed, but this field is required by the structure.

The key aspect here, though, is the description, which includes the definition, phaseMinimumGreen parameter in seconds, and here we see that this definition actually is derived from a NEMA TS 2 definition. We then have—down at the bottom, we see that this is units as recorded in seconds, and the references from TS 2 with specific clause references. The other fields there are the object identifier of 1.3.6, etc., and the phaseEntry at the very bottom. Those are how the computer sees this name of the—how it exchanges that name. So rather than saying "phaseMinimumGreen," it identifies it through a long computer code as you see there.

That's what the object looks like. Now what does the dialog look like? So this is one sample dialog. It is the one we referenced before, H.2.7. It starts out with a statement, "standardized dialog from management station to configure a table row shall be as follows:" (a) as a precondition, "the management station shall be aware of which row in the table is to be configured," and (b) "for the specified row, the management station shall SET all objects (to their desired values)" and all objects "referenced by the specific dialog that references this generic dialog, except for the index object(s)." Kind of a long statement. What exactly does that mean?

Well, if we break it down, we see that Precondition A: a "management station shall be aware of which row in the table is to be configured." Well, we're looking for the index of that row. So typical controller, we might have eight phases. We might be currently interested in Phase 2. We want to configure the table row for Phase 2, so in our example we look at the bottom right. We see our traceability. Our requirement traces to phaseNumber. That's the row index of the table. In this case, we want—our example is for Phase 2. And you see that Phase 2 in the bold diagram down to the lower left as the phaseMinimumGreen.2. That .2 represents the index of the row we're dealing with. We go to Step B. We see that we're going to—that's all objects referenced by the specific dialog, and we have two objects here: a phaseNumber and a phaseMinimumGreen. So all objects referenced by the dialog that reference its generic dialog, except for the index objects.

And remember, we just said that the phaseNumber is an index object. So we're not going to set phaseNumber. That's our reference, our index point. So now we're only left with one object, phaseMinimumGreen, so that's our reference to object. So the command in our diagram is that we set phaseMinimumGreen.2. And what value do we set that to? Well, we set it to the desired value. In this case, we're going to say it's four seconds. So that's the meaning of that dialog and how that works.

Well, that concludes Learning Objective One. And that brings us to our first question. Which of the following is missing from NTCIP 1202 v03? Your answer choices are: (a) user needs; (b) functional requirements; (c) test procedures; or (d) all of the above. So go ahead and make your choices, and we'll then take a look at responses.

Well, the correct answer is test procedures. We mentioned that, while the NTCIP 1202 v03 includes an Annex C, which will eventually contain test procedures, it's currently empty and left for future work. User needs is incorrect, v02—or, sorry, v02 did not have user needs. That would have been correct, but this course is on NTCIP 1202 v03, and we've added user needs as a part of this. Version 03 also added the functional requirements, so those are both now contained in the standard. And, of course, if (a) and (b) are not true, then "all of the above" would also not be true.

Well, that brings us to Learning Objective Two: Explaining the Purpose and Benefits of the RTM. The two key points in this learning objective are understanding the interoperability and interchangeability, and then, once we understand what those terms mean, we'll start looking at how we obtain interoperability and interchangeability.

So what do these terms precisely mean? Interoperability, as defined by ISO/IEC, is the "degree to which two or more systems, products or components"—so you kind of see that coupling, the three-part coupling there—"systems, products, or components can exchange information and use the information that has been exchanged." So clearly ISO/IEC worked with INCOCE. They liked this three-part of systems, products, or components. It probably makes incidents easier to read if we only talk about one of those at a time, so the degree to which two or more components can exchange their information and use the information that has been exchanged.

So pretty straightforward, but there's two key elements of this feature. One is the exchange of information; the second is being able to use the information once it's been exchanged. So how does NTCIP accomplish this? Well, that exchanging of information is that dialog. It is the definition of how we exchange that information, and we do that through the dialogs coupled with the specific protocols that we use and the definition of those protocols.

Using the information that has been exchanged, that gets into making sure that we have fully and unambiguously defined every data element so that people accessing that data through the protocols are able to know how to use that information and know what it means. So that's different use cases, but those different use cases can all reference the same data elements. Our goal is to standardize that data element. So between these two components, we've fully defined how we can achieve interoperability for the features we've defined.

Well, that brings us to interchangeability. Here we see that interchangeability is the ability of one product, process, or service—once again, a triple of ideas complicating the sentence—"ability of one product, process, or service to be used in place of another to fulfill the same requirements." So we're just switching out one for the other, but a key aspect here is we talk about "the same requirements." Well, what requirements are we dealing with? Well, once again, obviously we're talking about NTCIP, so we're not talking about every possible requirement. We're talking about interchangeability within the scope of NTCIP. And that primarily is with functional requirements that we deal with and also with performance requirements.

Within NTCIP, we're not generally interested in electrical, environmental, structural, and other types of requirements. We don't care about the color of the controller, for example. But there are other things that you may care about when you're actually exchanging equipment. Is the controller going to fit into my cabinet? Does it accept the same power that I have at that cabinet? Those are not part of NTCIP, but might be very important for you when you actually go to interchange your equipment.

So now we look at the rules within data exchanges. Interoperability entails two system components exchanging information. NTCIP defines two specific roles in every information exchange. A manager, which is responsible for making a request and also receiving the responses and notifications, and an agent, who is responsible for responding to requests as well as generating unsolicited notifications. So typically the example is that your manager is your traffic management system that you see here on the left, and your agent is your traffic signal controller over on the right. That is a very typical arrangement of the manager to the agent. But it's not always that simple.

In the case of an RSU, it would work essentially the same way. The manager would be the traffic management system, whereas the agent would be the RSU. And, of course, we have the traffic management system also acting as a manager to the ASC acting as an agent. But, as we talked about before, that signal controller can also talk directly to the RSU. We had that within our scope picture of the standard. So in that environment, which one is the manager and which one is the agent? Because, remember, when they're talking to central they're both agents. But now, in order to talk to each other, one of them has to be a manager. The other one has to be an agent. And the fact is: the standard, NTCIP 1202, doesn't define which is which. It is up to you to decide for your project which one you want to operate as a manager, which one you want to operate as an agent. That decision does impact the requirements for your device. It does impact the capabilities of that device, how it's going to interoperate. It has to be one or the other for that particular link. That means it has—whichever the manager is, you're adding functionality to that box. So the question then becomes: which of those two boxes do you want to add that functionality to? NTCIP supports either design. It's up to you on your project.

So that brings us to NTCIP interchangeability. And if we combine the two definitions here, that we actually want NTCIP interchangeability for the purposes of interoperability. We combine those two statements in green and in blue, and we see that we have a degree to which one product can be used in the place of another to exchange information and use the information that has been exchanged. So that's what we really mean when we talk about NTCIP interchangeability and interoperability. Now it's very important to note here: we use the "degree to which" nomenclature at the start of this definition. That's because we're still not precise. You can have different levels of NTCIP interchangeability, but what you need on your project is to know whether it will work or not. So what information needs to be exchanged on your project? That's what you need to decide, because within NTCIP, many of the elements are optional, and that creates different levels of interchangeability. So for NTCIP as a whole, products have a degree of interchangeability, but on your project, you need to determine: is it interchangeable or not?

That brings us to the project-level NTCIP interchangeability. This is where we say it's the ability of one product to be used in the place of another, to exchange and use the information. Which information? It's the information required for a specific project. This means I have to identify what information I need on my specific project to determine whether or not a particular device will be project-level NTCIP interchangeable. So it's that project-level PRL that will identify the options you have for your project. We discussed how to do that as a part of the A315a module. And then the RTM traces those user needs and functional requirements to a detailed design. So once you identified that project-level PRL, you know you'll get something that's interoperable and, at that point, interchangeable.

Well, that actually raises two other terms that we need to consider, as well. Informants—conformance is "adherence of and implementation to the requirements of one or more specific standards or technical specifications." Now don't get confused by the word "technical specification" here. This really relates to a standard. Technical specification within the terms of ISO is a particular type of standard document, although, within ISO, you have true international standards. They're called international standards, and they get a number. You also have technical specifications that are kind of like preliminary standards. They're for experimental use. They plan to become international standards at some point, but right now they're still kind of being tested out. So "technical specification" within ISO terminology is a particular type of preliminary standard. So when you read the concept of "conformance," you're really talking about: it relates to a standard or similar standard document.

By comparison, we have the term "compliance." And compliance is simply "doing what has been asked or ordered as required by rule or law." That's per IEEE. It's not limited to standards, but can include things like project specifications, which you may think is being very technical, but in ISO terms they're not technical specifications. They're project specifications. So you're compliant to a project specification. You're conformant to a standard.

So pulling all of this together, we have various types of PRLs. The standard PRL that is the PRL that is contained in the NTCIP 1202 document. It provides what I call a baseline menu. It gives you various options you can pull out and select for your particular project. It also defines which items are absolutely mandatory if you want to claim conformance to the standard itself, right? So that's conformance. Your project PRL is looking at that baseline menu and actually making the selections. Which of these optional features do I want on my project? And then that defines what constitutes compliance for your particular project.

Finally, there is a product PRL. This is where a manufacturer might go through that same baseline menu and identify which features their product actually supports, and then that defines the capabilities of that project. The nice thing about this is, at the end of the day, you can take a project PRL, compare it to a product PRL, and see whether or not that product will meet the requirements for your project. So you can very quickly identify what capabilities are there and what differences exist, the goal being: as long as that product meets or exceeds your expectations in your project, then it meets the requirements.

So this is what that PRL looks like. We've gone through here and made some selections from the menu column. The support column. And, once again, equipment may meet or exceed that level of conformance, so it could very easily support both of these options. A product could very easily support both those options at the bottom and still meet your project requirements.

So the PRL defines the what, and the PRL defines kind of the baseline, the foundation for interchangeability. But we've not defined the details of how it works yet, so you don't yet have interchangeability. But it provides the platform. It says what I need to do at a minimum. It's the functional and performance requirements as well as the user needs.

The RTM is that second traceability table, and that's where we get into how we achieve interoperability. It's the design details for each element. So the PRL provides the foundation for interchangeability. The RTM extends that definition to ensure interoperability once the devices have been interchanged.

So there's another distinction between these two terms, of course, though, and that is products that comply with the same project specification are interoperable if they fulfill opposite roles. So one's a manager, and one is an agent. They're interoperable. If they fulfill the same role, if they're both agents, those devices are interchangeable. Likewise, I can have two different interchangeable managers TMC-wise, or in my maintenance laptop. Those are interchangeable. They fulfill the same roles. So the project specification—we actually go out with your procurement. You need to include your project PRL that is the filled-out standardized PRL from the standard and any supplemental PRL.

So if you have user needs and requirements beyond those defined in the standard PRL, you need to make sure you include those, as well. And then likewise you need a project RTM. Now we say it's a project RTM. In fact, to the extent that you're constraining yourself to just the standard off-the-shelf components, then you can use just the standard RTM.

There's only one trace that occurs for every data exchange to the RTM, therefore you don't need any customization. But you may actually want to include a supplemental RTM, both to handle any supplemental requirements or user needs you've defined, as well as details you may need to specify for your particular project. And we'll get into those later. And then, of course, there's additional materials that we'll discuss at the end of the session today.

But that brings us to the end of Learning Objective Number Two, and it brings us to our second question of the day. What does a project PRL identify?: (a) the functional requirements for a project; (b) the objects to be supported for a project; (c) the testing requirements for a project; or (d) all of the above. So go ahead and make your choices. What does a project PRL identify?

All right. We'll take a look at the answers here. And we discover that the correct answer is: (a) a project PRL identifies which functional requirements are required for a device within a project. The objects to be supported for a project are identified in the RTM. That's the other traceability table. The testing requirements for a project are not a part of the PRL. They're not a part of the RTM. There is a Test Traceability Table that will be included in Annex C. That's not currently included within the v03 of NTCIP 1202. And, of course, if two of the statements are wrong, they can't all be true, so item (d) is incorrect, as well.

That brings us to Learning Objective Three: Preparing a Project-Level RTM. Now, I gave you a hint at the end of the last learning objective that, if you're doing everything in a standardized fashion, you don't need any specific project-level RTM. You just use a standardized one. But we'll go through here a little bit further to understand the RTM trace requirements within the

RTM and then supplementing the RTM if needed, referencing the RTM, and understanding the RTM benefits to stakeholders.

So the PRL needs to be tailored for each project by selecting options. Okay, remember this is the PRL, the first traceability table that goes from user needs to requirements. We've defined how to do that already in module A315a. And then the next module, A315b Part 2, will discuss some other special considerations for how you actually fill that out and other things you have to consider. The RTM, the second traceability table, provides exactly one design for each functional requirement that includes a data exchange. So there's no options to select in the RTM. There's no need to duplicate this in a procurement and, in fact, the copyright statements related to the RTM really prevent what changes even could be made if you want to maintain within that agreement. So if you add this to your project specifications rather than just referencing it, you end up massively increasing the size of your specification while really only confusing the vendors because they're going to be looking at this saying, "Well, I think this is the same thing that's in the standard." But if you included it here, now I have to look at every single page and make sure you didn't change anything inappropriately. So really the goal here should be to simply reference the standard. That makes life easier on everyone. And all implementations must support the design and the standard to claim support for the associated requirements.

So what does this RTM look like? Well, you'll see we use shading. Shaded rows indicate that they're headings, not actual functional requirements themselves, but they're section headers. So when we get down to an actual functional requirement, you'll often see that there's multiple levels of shading above it showing different levels of section headings. But once we get down to a specific functional requirement, you'll see that the first three columns, or the first two columns, identify the section number where that requirement is defined and the name of that section heading, so for that specific functional requirement.

So the functional requirement, in this case, is: configure vehicle phase minimum green time, same example we've used before. You see the clause number in the document over there on the left, and we see also the dialog ID indicates which dialog this traces to. When you get to empty rows and that functional ID and functional requirement and dialog ID area, that's indicating that we're actually going to start tracing from the functional requirement to more detailed elements and the objects area. So, in this case, we have one functional requirement. Remember we said every functional requirement traces to exactly one dialog. That dialog, in this case, is H.2.7, and then it can trace to one or more objects.

Typically it's more than one object, but here we see the three objects this time: phaseTable, phaseNumber, and phaseMinimumGreen. The phaseTable actually is the table as a whole. It's just kind of a generic reference. So it's not really an object per se, but we've listed it out in the table to give you context. So how does this really work? The last three columns, then—so same as before. The object ID is the section number where you can find that object defined in the document. The name of the specific object and then any additional specifications—in this case, we don't have anything in the additional specifications column. We do have an example coming up to show how that's used.

But first, let's take a look at how this relates to a previous example we've discussed before. That H.2.7 dialog is the same exact dialog that we've presented before. It's basically just a set of the object that we're dealing with to a particular value. Our traces over the object area, the phaseTable, points to the table as a whole. It gives you a context of what we're dealing with. The phaseNumber, as we talked about, is the index, so that's just pointing me to a specific row in that table. And then, finally, the phaseMinimumGreen is pointing to a column. And that column is one column of that phase table. So by knowing the row and the column, I know of a specific cell in the table, and that cell contains the value that I want to set things to or retrieve data from or whatever. So that's how everything works. Everything traces very nicely together to form a complete solution.

Now we also talked about—there's an additional specifications column. There are some cases where a particular object is not an indivisible component, but is, in fact, like a bitmap or some other arrangement where it actually contains multiple sub-pieces of information. So there's cases where we have functional requirements. In this case, where we want to enable or disable a phase, and that functional operation only addresses one bit out of the phaseOptions object. So that "additional specifications" is where we provide that supplemental information that we're dealing with a specific bit—Bit 0 in this case—of the larger phaseOptions object.

So, so far we've talked about the same dialog over and over again. That's a very simple dialog. I want to set information. Very straightforward. But dialogs can be complex. Dialogs—I might need to set the device in a particular state before I conduct my actions, and then after I finish up all of my downloading of information, I might need to reset the state of the controller and make sure everything works. Be prepared to get error responses and other things. So the dialogs can be deep interactions with a device or they might be very simple interactions. And there are cases—the most common cases are simple interactions, but the more customized cases are why we have a dedicated Section 4, to identify how more complex operations work.

Now, in all of these cases, when we trace to a dialog in the standard, we're tracing to a standardized dialog. That sounds circular, but what we're getting at is: these are the ways that we intend the operation to work and that we will end up developing test procedures to ensure that they work. But SNMP, the base protocol that we use to exchange all information, is a very flexible sort of protocol. It allows a manager to pick and choose what data they want to retrieve or store in the device whenever they want to. The base logic of SNMP allows a manager to combine these data elements largely in whatever combination they wish to.

We define standardized dialogs that trace very nicely to our standardized set of requirements so that we can test against those requirements in a logical fashion that supports all deployments. But when you get to actual operations, your live operation Traffic Management Center might frequently poll your device. And rather than abiding it precisely with the standardized dialogs, they might optimize their operations so they use fewer communications bandwidth than would otherwise be required.

And by doing that, they're changing the way that information is exchanged. In theory, they're fully conformant. They're fully compliant with all of the SNMP rules. They're conformant with the standards. But they're using a different set of dialogs, and you can't be one hundred percent certain this is going to work unless you test against those dialogs. And that's what this really drives down to is, if they're going to vary from the standardized dialogs, you might want to start considering testing to the specific dialogs that they use in their system so that before you actually purchase a device you know it will work in your system.

So project dialogs allow an implementation to support specialized dialogs for better efficiency, better operations of the system as a whole. This is particularly true when central systems need to frequently monitor information from multiple functional requirements. The standardized dialogs often, within the standard, are not the most efficient way to exchange that data. NTCIP provides multiple ways to improve this efficiency. You can have flexibility of the SNMP request. You can change the order of which objects I have. I can combine requests together. All sorts of things.

We'd also—we talked about block objects. I can create blocks of data to exchange them more efficiently. We also have another protocol called STMP that uses dynamic objects. This is a kind of a whole new level of efficiency capabilities, but also a lot more complex. And then, finally, we have exception reporting. So rather than actually polling for this information, I can actually configure my device to give me data at particular times. All of this is allowed by the standards, but yet aren't included in what would be produced as standardized dialogs or tested in a normal procedure. So if you want to do any of these advanced features to improve your operations, then it's strongly suggested that you consider having that—defining what those dialogs are and then testing for them in your test procedures.

In summary, referencing the RTM and your procurement specifications, the RTM is 140 pages. There's no need to copy that standardized RTM. All that you need to do is reference it from your procurement specifications. And, as we mentioned, the NTCIP copyright limitations limit what you can do to even change that document. It's a lot better to just reference the document. Makes it a lot easier on anyone who's reading your specification to know you mean the actual standardized RTM. But if you want to define your own specific dialogs, then you do that through an extension RTM. You supplement that standard RTM with a reference, and now you add your customizations as a separate supplement.

Why is this all important? Well, because there's a lot of benefits to having the RTM. The procuring agency, it simplifies their procurement specification. They just reference this 140-page document. It reduces the amount of work. That work has already been done. You don't need to reinvent the wheel there. It promotes a competitive marketplace because all the manufacturers are developing according to a standard set of rules. You don't have to do any custom work for every new client. And then, finally, it promotes interoperability because everyone has deployed this before. They're more likely to interoperate together once they get to your project.

For operations personnel, it also provides detailed design for the desired functionality. That design has been fully documented. So when they're trying to debug a system within their system, they can quickly go to the RTM specification, identify how a particular requirement is to be met in the design features, go to the exact page that they need to find within that standard, and identify the details. So it's all very nicely traced. It promotes a consistent user interface so that the users and the operators can get used to working with all of their equipment, regardless of who the manufacturer is, with one standard interface at the central system. It also simplifies field maintenance, because all of your devices are going to start tending to work the same way.

System developers also benefit. They promote common design for different manufacturers so they build an interface once and it uses—works with all the different components they get in. It simplifies systems integration, because they know what to expect. And it also, once again, promotes a common marketplace for more advanced systems. So it actually—they spend less time developing, reinventing the wheel of how you communicate with different manufacturers. Now they only have to do that once, and they can devote their extra time on building higher, more advanced logic within their system.

Manufacturers and vendors benefit because they—it promotes a common design for all central systems. It doesn't matter who they're going to integrate with. It enables a standard product for all of their clients, and simplifies procurement specifications, and, ideally, reduces disputes because the specifications fall into a standard format. We all understand what that format means.

Finally, conformance testers also benefit. It clearly identifies what must be supported and its design. Promotes development of a common set of test procedures. Developing test procedures are extremely expensive. Implementing them are even more expensive. And this allows us to develop test procedures once, make them really rock solid, and then even automate a lot of the processes to implement those test procedures.

That brings us to our third question of the day. What does the dialog column in the following table mean? So we have a table here that we showed before. The third column in is the dialog column showing H.2.7. H.2.7. What does that column mean? Answer (a) the dialog is the only way to exchange the objects; (b) the dialog defines operations that are prohibited; (c) the dialog provided a baseline reference for testing; or (d) all of the above. So go ahead and make your answer and then we will review the results.

Well, the correct answer is: (c) the dialog provides a baseline reference for testing. So it does provide a baseline to be used to develop test procedures. It is not the only way to exchange objects. We talked about SNMP providing flexibility in the way we exchange objects, and you may want to supplement your RTM with additional dialogs that correspond more closely to the way your central system will perform things. "The dialog defines operations that are prohibited" is also incorrect. It does not define any prohibitions. It only says this is one way in which you can actually exchange the data. And, of course, if two answers are wrong, "all of the above" would not be true either.

That brings us to our fourth Learning Objective: Preparing an ASC Specification. The key points here are identifying potential issues with a specification, going through an interface specification checklist, and then completing the specification package.

So there are various issues that can arise with poor specifications. So if you don't specify your user needs or you have inadequate specifications of functional requirements, you might end up getting a compliant system but it doesn't actually meet your user needs. It's compliant because it fulfills what you said it had to fulfill, but you weren't very good in specifying what you needed to fulfill, so it doesn't actually meet your needs. Another possibility is: inadequate specification of system dialogs can result in inconsistent behavior of the system.

Another possibility is that not clearly defining custom features can lead to inability to support customized user needs, similar to the first couple of options we listed. Inadequate specification of communications stack might mean you have a non-interoperable system. So, you know, if I need an Ethernet stack and, all of a sudden, I'm just given a control that only supports serial communications, that doesn't really meet my needs. If I have inadequate testing, I might have anomalies occur during operations that might cause problems, and that might be problematic to solve, because a vendor may have already been paid. And then, finally, copying someone else's specification might result in a system that doesn't meet my particular user needs because I didn't go through the process of identifying what those were.

So let's take a look at these and how we resolve them. NTCIP 1202 v03 simplifies the process for selecting needs and requirements. So this really shouldn't be a problem. Now that we have a PRL that's basically providing you a menu, you can just go down, select the features that you need on your project. So fill out the project PRL, and include that in your specification, make sure you include a reference to the RTM that defined how those requirements are actually met with design details. The one caveat there is to make sure that you don't over-specify, because the more requirements you add in, the more expensive the overall project will be.

Then you also need to make sure you define any custom items you might need to include. Those customizations might include custom dialogs, which we talked about. That's probably the most typical case, particularly for signal controllers. But you might also have other custom extensions. Additional user needs, requirements, as well as associated design. See Modules A202 and A203, also the follow-on course to this will also give some more guidance, the A315b Part 2.

Then also the communications stack. We mentioned that this module is focused on the interoperability of the data, but in order to be fully interoperable, you need that entire communications stack. You don't achieve that if one device is Ethernet and another device is only serial. So you need to identify what standards you want throughout the stack in order to make your system work. When you define acceptance testing, clearly define it and then follow through with it. Without testing, requirements are of limited use. You can say that this is required, but if you never test for it, you have no proof that you're actually getting that.

With that in mind, testing does require time and needs to be in the budget and scheduled up front. The test procedures are not currently included in the v03 standard, but, as you'll find out in the next module, A315b Part 2, there is a project that's underway that is going to be developing test procedures for the standard. So you should be able to access those test procedures and have something available for your particular project.

So the next one is developing your own specifications. Don't just blindly copy someone else's. We have the PRL. Go through each of those options. Identify what your needs are for your project and then specify those requirements.

So those are our key checklists: using the project PRL, referencing the standardized RTM, and then including customized dialogs and extensions in your extended RTM, and then specify your communication stack, make sure you include testing/acceptance requirements, and don't just copy someone else's specifications.

But it doesn't end there. Your interface specification, which is what we've really just talked about, is only one portion of a larger specification package. That specification package needs to consider the hardware specifications, software specifications, environmental issues and other things. So what's important to note here is that all of these things fit together and there's an overlap. We need to make sure that, where these different specifications overlap, that we're consistent. So if we think about something like the cabinet door, and you want to be able to be notified when the cabinet door is opened, you have to have all three of these components working together to solve this problem. I need a hardware-based sensor that can detect a camera door being opened. I also need the software that will process that signal from that hardware sensor to make sure that that gets processed and handed to my communications environment, and then finally, I need an interface so that my field device can report back to central if a cabinet door is opened.

There's a tendency for agencies to view this as one document and say, "I have defined that I need this data in my interface, therefore you should know all of these different components are needed." And that makes sense. The problem is this becomes a very large document. This very large document gets received by a manufacturer who has a large organization and they divide up that large document saying, "Okay, you interface people, go do this. You hardware people, go do this. You software people go do this."

And what has happened on some projects before is it gets divided up just like that. The end product comes back and you don't have the hardware there to do what is required by the interface. And at the end of the project, when everyone spent their budget, and their schedule, and everything else, and everyone wants to get the project signed off, and you're missing one little feature like this, there's a real tendency for discussions to break down and for the manufacturer to say, "Well look, we gave you the interface you asked for. You didn't actually ask for the hardware. We've already delivered it. It's already on site. We just want our payment now." And what you want to do is avoid those complications up front by making sure your specification is properly documented, properly in sync across all lines. It is a hassle, but it's a practical issue that needs to be addressed. Best to address it up front and to get everyone right. You avoid problems down the road.

So that brings us to our Interface Specification Checklist. A component may need to support multiple interfaces. So if you're a signal controller and you're specifying the signal controller, you might need your signal controller to support an old version of NTCIP 1202 because that's what your current system operates with. Or maybe you need to support proprietary interface because your system's that old and you don't yet support NTCIP 1202. Of course, you also want it to support the current version because, when I upgrade my system, I want to be able to use the latest features. And I might also want it to support other NTCIP functionality, especially if I'm getting an advanced controller.

So, for example, I might have a traffic signal in an area on a hill where it gets icy. And when it gets icy I want to be able to monitor the icy conditions with a little weather station there, and I just want to do that in the same controller rather than buying a completely separate weather station. So I can do that all within one controller, but that means grabbing objects from NTCIP 1204, so I'm having to do all of this in one specification. We need to consider all of that within your specification package.

Likewise, I might need to support multiple communication stacks. I might have one interface for central. I might have another interface—maybe my central interface is Ethernet. A second Ethernet interface for my local RSU. And then a third serial interface for my maintenance laptop. So I need to consider all of my interfaces, make sure that gets into the specification. If I'm dealing on a central side, I might have to be dealing with multiple device types. I might have a single central system that's managing signal controllers, message signs, ramp meters, and other devices, and each one of those, I might have multiple versions. I might have some older NTCIP v02 controllers mixed in with my v03 controllers. I need to make sure I can handle all of those. And, likewise, multiple different communication stacks across my different devices.

The specification also needs to consider contractual requirements throughout the maintenance—the Vee cycle of operations. The systems engineering Vee cycle. So that includes the left-hand side. I need to consider contractual requirements on the system development, what support, what interactions will take place with my developers as they develop the system? What level of internal testing is required between the two branches of the Vee? What set of deployment and integration tests will be provided at the end on the right side of the Vee? And then, as we get up to the top of the Vee, on the right side, what sort of operational and maintenance support do I expect from the manufacturer? And then, finally, what's the overall project management that covers all of this? So you have to consider each step of your project that's symbolized through the Vee diagram, the systems engineering.

Finally, we need to remember, you need to complete all aspects of the PRL. In particular, don't forget the Additional Specifications column. There are many times blanks that are to be filled out over there. So the ASC shall support at least (blank)____ phases. Make sure you indicate: is this an eight-phase controller, a sixteen-phase controller, or whatever? Only require what you need. The more requirements you add, the more likely you're going to limit the vendors, and you're going to increase costs. The specification needs to insure that the device can respond fast enough for the central system and efficiently enough for the limited communications network you might have. And then, finally, your specifications should define acceptance and payment conditions. So does the vendor get paid on delivery? After testing? After maintenance for a year? What are the conditions where he gets paid?

That brings us to our final question of the day. Which of the following is typically not part of the interface specifications: (a) project PRL; (b) testing requirements; (c) environmental requirements; or (d) communications stack. So go ahead and make your answer then we'll review the answers.

So the correct answer is: environmental requirements. They are not typically part of an environmental specification. The project PRL should be a part of your specification. That is where you identify the user needs and requirements that you want on your project. The testing requirements—you will, at least, have requirements for testing and specify what those are. You won't necessarily have all of the test procedures defined but you should have the testing requirements defined. Likewise, we emphasized that there is a need to include the communications stack in your specification to make sure your two devices can communicate.

So that concludes our presentation. We have—we went through—we identified the standard requirements in the standard. We looked at the purpose and benefits of the RTM. We also prepared a Project-Level RTM which we discovered is primarily just supplementing that standard version. And then we went through the process of preparing an ASC Specification.

As I mentioned, this is the Part 1 of the course. Part 2 will follow this course, and we'll focus on managing some special considerations related to NTCIP 1202 v03, the special considerations for signal controllers, in general, where we are in the technology life cycle, etc. And then we'll also look at incorporating requirements not supported by standardized objects and how that can be done. If you have any feedback, you can provide it with the link below. Thank you for participating in this module and we'll look for you next time. Thanks.