**Vincent Valdes:** ITS Standards can facilitate the deployment of interoperable ITS systems, and make it easier to develop and deploy regionally integrated transportation systems. Transit standards have been developed by transit professionals like you at a national level to encourage competition and limit costs within our industry. However, these benefits can only be realized if you know how to write them into your specifications and test them. There are now a series of modules for public transportation providers that cover practical applications for promoting multi-modalism and interoperability in acquiring and testing standards-based ITS Transit systems.

**Scott Altman:** This is Module 14, Part 2. Applying GTFS-realtime to your Agency. Hello, my name is Scott Altman, and I have been a member of the Technical Staff at Consensus Systems Technologies, we're ConSysTec since 2011. I have experience with various specifications for transit data, including transit schedule data and realtime information. I'm currently working with Transcom, a consortium of 16 transportation agencies in the New York, New Jersey, Connecticut region to develop a GTFS-realtime feed, which integrates transit data for multiple agencies in the region. I have a Bachelor's degree in Civil Engineering from Rensselaer Polytechnic Institute, and I'm registered as an Intern Engineer, or EIT in New York. I also wish to acknowledge my colleague Bruce Eisenhart, Vice President of Operations at ConSysTec, who is the Co-Developer of this course. Our Learning Objectives today are 1) Define the scope, uses for and users of the GTFS-realtime specification; 2) Apply transit source applications to GTFS-realtime; and, 3) Implementation of GTFS-realtime. Our first Learning Objective is: Define the scope, uses for and users of the GTFS-realtime realtime specification. Our key points are: background of GTFS-realtime; benefits and uses of GTFS-realtime; and target devices for realtime applications. Today we will begin with a story before we learn about the GTFS-realtime specification. Consider the scenario of being a bus passenger, and you are about to start your commute to work, it's raining and you'd like to minimize the time you're standing in the rain. From your daily commuting experience, you know that it takes about two minutes to get from your house to the bus stop. Fortunately, there are both web and smartphone applications available to tell you if you have to wait for your bus, or if it's on time. You use one of these locations, and click on the stop seen by the red arrow. A pop-up comes up as we see on the left, and we can click on where it says Departure Board, as noted by the green arrow. We then get a list of all of the buses that are coming to that stop. We see the current time is 7:28 AM. And the next bus is at 7:31 AM. So we know that the bus will be there in three minutes. We also know the bus is on time, because the time is green. Therefore, it's time to leave and get to the bus. There are multiple applications that transit customers may utilize to get realtime information on the status of their transit trip. Because of the variety in application, agencies have a need to export realtime information in a common format that all of these applications can use. The GTFS-realtime specification that we will learn about right now is one of those formats. So first we will go over an overview of what this specification is. In Module 14, Part 1, we introduced GTFS, or the General Transit Feed Specification. As we learned, GTFS is a specification for handling static transit data, or the data that remains existing over a period of time, this typically being the agency's schedule. GTFS-realtime specification that we are learning about today supplements GTFS by providing realtime information, which builds upon the information in the GTFS specification. GTFS-realtime is primarily focused on providing data related to passenger facing applications so when GTFS-realtime specification that we are learning about was originally launched in 2001. Six cities were the original participants in Google Live Transit Updates program, and these cities were Portland, Oregon; San Francisco, California; San Diego, California; Boston, Massachusetts; Madrid, Spain; and Turin in Italy. It was intended to provide up to the minute transit information to passengers. In order to support the live transit updates program GTFS-realtime specification was developed to feed this data into the application. In August of 2011 the first version of the specification was released. Now the specification is maintained by Google, and it is important to know that it is a specification, not a standard. We will talk about what that means in a moment. Three overall types of information are provided in the GTFS-realtime specification. Trip updates tell us when the vehicle for the trip will arrive and depart from each stop on the trip. The vehicle position gives us the location of the vehicle and alerts TeleSyStar any planned or unplanned events that affect service. This might be an incident of some sort that is causing delays. So let's take another look at understanding what the GTFS-realtime specification is and is not. In the previous slide, we find GTFS-realtime as a specification for providing realtime transit information. Another meaning of this is that is a format for sharing realtime transit information. If you are a transit agency, and want to share up to the minute information about your transit service, this is one possible way that you can do this. Another way to think about what GTFS-realtime actually is. Is that it is a set of rules for encoding the data. So essentially we can think of the GTFS-realtime as a file, and there's a specification that said exactly what needs to go in this file. That's what GTFS-realtime is. It's also important to know what GTFS-realtime is not. GTFS-realtime is not software. It does not provide realtime information. It is also not an application that uses realtime information. In fact, GTFS-realtime does not technically do anything for you at all. It's simply a way to share information between systems that either create or use real time information. We have this diagram here that hopefully will better explain this point. On the left, the box on the left is a software system that produces realtime transit information. That might be software hosted by a transit agency. On the right side we see software that uses realtime information. This might be a smartphone application that you use to check the status of your bus or train. GTFS-realtime is the thing in the middle that is going between the two systems. And GTFS-realtime feed is simply the pile that the information is shared in as it is transmitted from one system to another. So the GTFS-realtime specification as we've already said is officially maintained by Google. In this role, Google hosts the specification website, moderates the discussion group, publishes changes and develops open source tools that can be used with the specification. Google also maintains the protocol buffer format which is the format that GTFS-realtime is encoded in. We will discuss what that means a little later. As we previously said GTFS-realtime is not a standard. This means that it does not go through a formal standardization process. This contrasts to other ITS transit specifications, where a set of changes are defined, and there is an extensive review and balloting process. The update process for GTFS-realtime is a little less formal than the standard, and it is nearly identical to the format used for the static GTFS specification that we learned about in Module 14, Part 1. Changes are proposed and discussed in an online forum or discussion group, which is hosted as part of Google Groups. Membership is open to anyone who can claim some relationship to GTFS. For example, transit agency staffs, application developers, planning tool users or enthusiasts can all join and  propose changes. There are advantages and disadvantages to this process that is used. The major advantage is flexibility. Changes can be made on an as-needed basis and there are relatively low barriers to change. A second advantage is that GTFS-realtime can be extended to meet the needs of agencies and data users, whether or not the specification formally described in their needs. Flexibility, however, is also a major disadvantage. Different classes of users have different needs, and there's not necessarily consistency with regards to extensions that specific agencies may make and add to their GTFS-realtime feeds. This has the potential to lead to interoperability issues as consumers of the data, you need to be able to use these extensions. A second disadvantage is that because this is not a standard, there is potential for ambiguity in interpretation of the specification. This could also lead to interoperability issues. Finally, because of a loose structure of the update process, there not be a clear path to reaching a consensus on the direction of the specification. Now we will see what the process is that GTFS-realtime specification goes through when a change is proposed. The process starts by somebody proposing a change. This could be a field or a message that they would like to see added. This Proposal is then discussed in the discussion forum and feedback is provided to revise the Proposal as necessary. If there is general consensus, we move forward to the third step, which is testing. In order to accomplish this, one data producer, such as a transit agency and one data consumer, such as an application that uses whatever this change will be, needs to prove that it actually works, and demonstrate it. After testing there are final comments to discuss. You'll have the actual change performed and if there is agreement to move forward. And if there is agreement that the change should be made to the specification, it is implemented and is published on the specification website and it is used by the GTFS-realtime community. However is there is not consensus to move forward, the Proposal may go back to the drawing board and go over again. We will now shift gears and briefly discuss some alternatives to GTFS-realtime. As we go through this module we will learn about what the GTFS-realtime specification does and does not cover. If GTFS-realtime does not meet your needs, one of these alternatives may better suit you. The first alternative is Transit Communication Interface Profiles, or TCIP. This is a standard that is maintained by the American Public Transportation Association, or APTA. While GTFS-realtime is primarily focused around passenger information, TCIP also focuses on internal and operational communications that might be used by a transit agency. This is in addition to passenger information. You can learn more about TCIP by viewing ITS Transit Standards Modules 3 and 4, Transit Communication Interface Profiles, TCIP, Part 1 and

2. The second alternative is the Service Interface for Real Time Information, or SIRI. This is a European Standard that is maintained by the European Committee for Standardization or CEN. Like GTFS-realtime, SIRI also focuses on passenger information, although it provides a greater level of detail, which may or may not be necessary for your agency. Although the names are similar, it is important to note that there is no relationship between the SIRI standard we are discussing here, and SIRI that you might use on your Apple iPhone. And although this is a European standard, there are several well-known deployments in the United States. One of the most common being New York City Transit buses. It is important to note that TCIP and SIRI are both standard maintained by standards development organizations. GTFS-realtime is not a standard, as we've mentioned. Although it is often referred to as a de facto standard, or a specification that is used so widely it is used as if it were the standard. Now we will discuss some of the Uses and users of the GTFS-realtime specification. GTFS-realtime has one primary use and that is customer facing information. Several standards and specifications in this for providing information. Many of these provide operational data and data that is useful for various parts of the transit agency. GTFS-realtime is only designed to focus on passenger information. So that keeps the scope specific. However, that does not mean that although the data is designed for one specific purpose, that does not mean it cannot be used for other purposes. Performance measurement is another way that GTFS-realtime has been used. In such cases GTFS-realtime data is collected and archived and used for performance analysis, such as measuring on time performance of buses or trains. An example of an agency that uses GTFS-realtime for performance analysis purposes is the Massachusetts Bay Transportation Authority, or MBTA in the Boston, Massachusetts region. As we mentioned early-on, there are three types of feeds in GTFS-realtime. And these feeds are the Trip Update, Vehicle Position and Alerts, and we will learn about how these feeds are used shortly. Finally, we should acknowledge the users of GTFS-realtime. The primary users of this feed are third-party developers who, in turn, develop applications that transit customers may use to access the data. So if customers don't use the data directly, but third party developers translate it in a way that customers to read, typically doing a smartphone or computer application. Agency Staff, Consultants and Systems also might use this data and use it for archiving purposes, or they may use it to create an application maintained by the system that provides realtime information to customers. Of course, the third-party developers are the largest group of users, because that is who is really who the specification is designed around. We will now look at how GTFS-realtime might fit into an agency's operations by looking at parts of the National ITS architecture. A Regional ITS Architecture is a blueprint for Intelligent Transportation Systems in the region, and it also defines the interfaces between these systems. Then National ITS architecture is a template for developing a Regional ITS architecture. We see a service diagram on the slide which comes from the National ITS architecture. This one's showing a model for how systems involved with Transit Vehicle Tracking might interact with each other. It is not necessary to understand every detail of this diagram, however we will point a few things out. We see in the center is a Transit Management Center as represented by the purple box in the middle. This takes

information in and sends the information out to other entities. On the right we see a transit vehicle that is sending two types of information into the Transit Management Center. Transit  Vehicle location data and Transit Vehicle Scheduled Performance. Once this data is received by the Transit Management Center, it can be sent out to other parties, such as Information Service Providers, who might then provide the information to the public. We see the orange arrow points to a flow called Transit Schedule Adherence Information. It is this flow of GTFS-realtime specification could be used as a standard or a specification to share information along this flow. Here is a second service package diagram which shows that archived data and how archived data can be collected. This diagram discusses many modes, not just transit. However, we see the orange arrow at the bottom, showing data from a Transit Management Center going into an archived Data Management Center. And that transit archived data denoted by the orange arrow is another place where the GTFS-realtime specification might be used. You can see a larger version of this diagram in your student supplement. There are some benefits, of course, to implementing the GTFS-realtime specification. A key benefit and probably the most well-known benefit is that it provides a conduit to transmit data that should ultimately be provided to the passenger. This has the ability to provide a higher level of customer satisfaction to the customers. As a result of data provided in the GTFS-realtime format, developers can create dynamic software applications and websites that provide passengers up to the minute transit status information. Additionally transit agencies can provide data to Google Transit which is part of Google Maps. And that's a free service that can make realtime information more accessible to a wide base of customers. Another benefit of the GTFS-realtime specification is that an agency might already have the required data inputs. Many agencies have already invested in computer aided dispatch or automatic vehicle locations systems. This data is used to create GTFS-realtime feeds. Third benefit is that there are open source tools available. Google, the organization that maintains the GTFS-realtime specification provides open source tools that work with the protocol buffer format. This is the format that GTFS-realtime is stored in. Open source software is software where the code behind the software is freely available and the software may be reused by other users in their own software, typically within the parameter of a license agreement released with the software. It should be noted, however, that using these tools does require some software development skills. And there are tools available, other tools, such as those maintained by agencies, and some other open-source tools that can aid in the creation of GTFS-realtime feeds. And a final benefit of GTFS-realtime is that is becoming widely accepted. Many applications are currently being developed which read the GTFS-realtime feed and in turn provide information to the users using data in this feed. Target devices for Realtime Information. Since the intent of GTFS-realtime is customer specific, the primary applications are those that provide realtime information to customers. In the story we presented at the beginning, the passenger uses his or her web browser to look up realtime information about the status of the bus. Applications like this whether for a smartphone or a web browser are the primary way that GTFS-realtime data is used. The graph on the right shows an example of a customer facing application. GTFS-realtime feeds can get information not only to devices owned by a customer, but agencies may also have electronic signage, such as dynamic messaging signs at stops and stations which show realtime information to customers. GTFS-realtime might be the input to these devices that provide the information in turn to customers. And as we've already mentioned, although the GTFS-realtime specification is designed for customer facing applications, this does not mean that there are no other uses. GTFS-realtime can be used as inputs to performance analysis tools, which as we said earlier is done by MBTA in Boston. GTFS-realtime can be used to share realtime information between transit agencies, for example, as well. All these are good uses of the feed. And widely used uses of the feed. It's important to keep in mind that they are not the intended purpose, so there may be gaps in the information contained and defined by the specification that must be accounted for by the agencies if they wish to share such information. And while there are many applications that provide realtime information to passengers, it is important to make Google Maps Transit, which was the original application that consumed GTFS-realtime. Google Maps is a popular tool for planning transit trips, and now also provides realtime information. In the screenshot that we see here is a shot from Google Maps Transit. We now have our first activity. And we have a question, the question is, "Which of the following groups any participate in developing the GTFS-realtime specification?" And the choices are: a) Transit Agency Staff; b) Planning Tool Users; c) Smartphone Application Developers; or, d) All of the Above. And let's review our answers. The correct answer is d) All of the Above. Anyone who can claim involvement with GTFS-realtime specification may participate. And the other answers are all incorrect, because these are all one group out of the many that may participate in the GTFS-realtime development activities.

**Scott Altman:** We move on now to our second Learning Objective, Apply transit source applications to GTFS-realtime. Our key points are, describe the data necessary to create a GTFS-realtime realtime feed. Discuss issues with providing stop time updates or ETA data. GTFS-realtime content and structure. Translating source applications to GTFS files. And related tools for GTFS-realtime. So recall that earlier we mentioned the three types of data that are contained in the GTFS-realtime specification, also known as the three feed types. We are going to start looking at this information in more detail. And as we've said these are Trip updates which provide the estimated times of arrival or departure to a stop. Vehicle positions, which tell you where a transit vehicle is. And alert information which provides information about events that are occurring. Why is this information important? It's important because it answer the current status of transit service. This is information that a transit customer might want to know. So in a little more detail we look now at three message types. Or the three feed types. A Trip Update message tells us when will a trip arrive at what location, at a specific location. It also answers the question of what is the delay of the trip. Knowing the delay of the trip is particularly useful, because there are cases when we may not be able to predict the estimated time of arrival at a stop. This could be due to unpredictable traffic conditions. Lack of a comprehensive prediction model, or possibly even lack of data being provided to give us prediction information, such as a driver not turning on their AVL system in their bus. It's important to note that delay is not only used to describe a trip running behind schedule in the GTFS-realtime specification, it could also tell us if a trip is running ahead of schedule, therefore we have the possibility to have a negative delay if a trip is ahead of schedule. And we can also see how this works illustrated. We have on the bottom a bus and the bus travels from one stop to the other, and we have predicted times for both of those. The second feed type is the Vehicle Position message. The primary question this asks is, "Where is the vehicle right now?" To answer this question we might want to know the GPS coordinates, or latitude and longitude of the vehicle. That gives a pretty precise location where a vehicle might be. We also might need to know what stops a vehicle is en route to, approaching, or physically stopped at. This is another way to tell our customers where a vehicle is. Vehicle information is included in many GTFS-realtime feeds. It's slightly more common in bus feeds, because they often have better data to provide this, but route feeds do get to include it  too. A final message type is the Alert message. And this answers a bunch of questions about unplanned events or planned events that are happening. It answers the questions of what is happening? Is there a sick passenger that's causing delays? Is there construction, which may be planned or unplanned? We also want to know where is the event happening? Where is not necessarily a specific location, but it could be groups that are affected by the service, such as a specific route, a specific stop, or a specific trip. We also want to answer the question of when is this event happening? Is it happening now? Is the construction that's happening in the future? And finally, we answer the question of how is service affected? Is service canceled? Are there schedule changes? Are there delays? Alerts answer these questions for us. So having discussed the three main feed types, we'll talk about the sources of GTFS-realtime data. Unlike the fixed or static GTFS feeds that we've learned about in the previous Module 14, Part 1, GTFS-realtime feeds must use data that is generated on an up-to-the-minute basis. Data typically must come from some system maintained somewhere in the agency. Often this a combination of the Computer Aided Dispatch, or CAD system. Or the Automatic Vehicle Location, or AVL system. An AVL system is one that tracks the location of the transit vehicle operated by agencies, and relies on equipment onboard each transit vehicle. For buses, this is usually GPS-based system. Computer Aided Dispatch systems are systems used for dispatching and monitoring transit vehicles, combining these two systems track vehicles and transit system performance. Look at agencies that operate a rail sometimes have sensors embedded in the tracks, and these can provide locations. Typically these are related to signaling systems used by train and rail systems. Systems which provide vehicle location, as we've just discussed provide the most important data to have. Obviously, we can't tell you the location of a vehicle without knowing or even having a system that tells us that. Knowing location is necessary to provide an estimated time of arrival for a trip to a stop as well. It should be noted, however, that location's not explicitly needed to come from such a system. There are more modern applications that can collect location data from a cellphone onboard a transit vehicle. And translate this data so that a GTFS-realtime feed can be generated. There are some other systems that are used. Incident Management

Systems are another source of data. Specifically for the Alert message. An Incident Management System is a system maintained by an agency to allow incidents that are affecting the transportation network, and often to share in this information with other agencies within the region. These rely on dispatchers to manually input event information, and it can be used as a source of data for GTFS-realtime feeds specifically with the Alert message. Another source of data, although an indirect source is Existing Data Feeds. Agencies might have realtime transit data stored in a different format other than GTFS-realtime. There are multiple standards and specifications, including some of our vendors specifically are, as well as specifications and agencies have developed to transmit realtime data. There are cases when it might be beneficial to convert this data into GTFS-realtime. We'll learn a little bit about this in a case study later. Finally all GTFS-realtime feeds rely on static scheduled data. A GTFS-realtime feed must reference a static schedule. There are certain fields in the GTFS-realtime specification that must be explicitly referenced from a static GTFS file, such as a stop identifier. And it is important to note for all of these systems that in order to produce a GTFS-realtime feed, the specific data available by a system must be understood. For example, a CAD or AVL system might not have the ability to predict realtime stop arrival times. In such a case, an agency might elect not to provide the time, and might just report the measured delay of the trip, even though delay could change. It might be better than guessing on the time. So most importantly an agency simply needs to understand the data that it can provide. We've talked a bit about GTFS-realtime feed provide estimated times of arrival to a stop. We'll talk here a little bit about how this can be predicted. Here we see a diagram showing different sources of data for predicting the estimated time of arrival. On the left, we have the current location, the current delay, and events and incidents that are occurring. These are up to the minute, you know, sources of data. They change on a minute-by-minute basis. So that if you have location, we can tell where a vehicle is. And if we know how far delayed it is, we can run it through algorithms to propagate that delay to the future. And also through our incidents we can possibly calculate how that affects service. On the right, we have two other sources of data. A static schedule, an historical travel time. These are not changing on the minute. You know, they change only at defined periods of time. And this data can be used and merged with the up to the minute data to provide a relatively accurate incident-- or a relatively accurate estimated time of arrival. Note that if we are estimating a departure time from a stop, we would have to add the dwell time, which is the time that a vehicle is stopped at a stop. Typically we notice from the static schedule, however if there are factors that are causing this to be delayed or changed, we could factor that in. In the previous slide, we discussed how we can predict the estimated time of arrival to a stop. The data is used to provide stop-time updates in GTFS-realtime feeds. If a trip is delayed to a specific point, the delay is often propagated to future points. However, it may not be consistent across the whole trip, because vehicles do not necessarily move at a constant speed, and different incidents or events might be occurring at a future point in the trip, if there's a downstream incident, for example. So therefore, we have to be mindful of how we are creating, you know, a prediction. Predictions do depend heavily on the data available. This means that an agency must have the appropriate data necessary to predict arrival and departure times accurately and precisely. If an agency has no way to provide an accurate prediction of a transit vehicle's estimated time of arrival, then it should not make such predictions. High reliability data is data where there is confidence, high confidence in the accuracy of the data. High confidence occurs when data is received frequently, is accurate and precise. When determining the estimate time of arrival for a transit vehicle, this comes down to having both accurate location data and a method of estimating time to the stop. The method of estimating time of arrival is a prediction algorithm, which combines many variables, including historical travel time, and a static schedule to fill gaps, and still provide a prediction as we just saw on the previous slide. Vehicle locations, of course, change on a second-by-second basis. So accurate location data is data that is obtained frequently, typically meaning several times per minute. Additionally, data must be highly precise, or have a high resolution. We see on the map different levels of a precision. We see on the map, different levels of a precision for a specific location. In general, data can only be shown if it is reliable. In the case where a transit arrival time cannot be reliably predicted, the current delay might be a better and more accurate field to display. While this may not allow the passenger to know the exact time of arrival, we can measure this value as it is at present, and release it and disseminate to downstream users. We are now moving into a part of the module where we will talk about the details of the GTFS-realtime specification. These next several slides are a little more technical in nature, so therefore it's not necessarily to understand every line of detail, but more important to understand the overall concepts. Much of the data we'll talk about can be looked up in a reference and had we done at your own pace that you were comfortable with. So again, just, it's important to understand the high level concepts that we are discussing. So the next concept we will discuss is the Protocol Buffer format. This is the format that GTFS-realtime feeds are encoded in. In other words, for those who are not familiar with the concepts of encoding data, this is a structure and rules that describe how the GTFS-realtime file is constructed. So as we said GTFS-realtime feeds are stored and transported I this Protocol Buffer format. Google also maintains this specification, like the GTFS-realtime specification. This format is used because the size of the files are smaller and faster to transport than other formats that are used for other standards. Protocol Buffer file is not human readable. The contents are ultimately converted into what's known as a Binary String, or a series of zeros or ones. You could not read that, I could not read that. That's not something that is human readable. Therefore, we need software of some sort to translate the files to and from a text format that can be read. The structure of a specific protocol file. A Protocol Buffer file is defined by a .proto file. This is something like a template or a schema, if you have used .xml text formats. This can then be used as an input to software that can read and write protocol buffer files. Because as we said, GTFS-realtime files are encoded in protocol buffers, which are a series of zeros and ones. We won't use that format to discuss the GTFS-realtime feed. We will look at the text based or human readable format, so that we can understand the data that is encoded within the GTFS-realtime file. We will now see an example of a Protocol Buffer file. In the previous side, we mentioned that protocol buffer are stored as series of zeros and ones, but can be translated into a readable text form, which is what we will see here. Protocol Buffers are coded into structures known as messages. Messages has curly braces as we see. So we see the line on top says "entity" and there's a curly brace after that and there's a curly brace at the bottom, so everything that appears between these two curly braces are part of the entity message. We see that there are fields contained within the message, and we see that there is, on the fourth line, it says vehicle, and there's another curly brace. That is another message contained within the higher level message. The vehicle message is contained within the entity message. Therefore we see that we have messages stacked within messages, so the message may consist of fields or messages. Fields, this may be some sort of data, or other message. For example, the bottom line says "stop_id: "96"" that's one field within the greater message. We'll take another look at how this format exists and how messages are stacked within messages. Some might be a little more clear. So as we said, GTFS-realtime consists of a series of messages. If you look at this graphic on the right, you see the outermost level of the specification is the message. So our different data frames within the message, including the higher level messages, which are also called messages. Messages contain fields, which may contain a single value, or other messages. So if we look at this graphic, we see there is the green layer that says "Message." That contains three fields. The three fields are in blue. Field 1 and 2 just have some sort of data next to them. And Field 3 has another message within it. Field 3 is another message. That message contains two other fields that contain some sort of data. So we see from this graphic how messages are encoded within other messages. It's important to note that the GTFS-realtime specification does to officially support the request and response structure seen in other data standards and specifications. In particular, there is no defined request message or message request data or information about one part of the data. Rather, the entire feed is published or broadcast. If an agency has a GTFS-realtime feed, the entire feed containing all of the active trips, all of the vehicle locations, and all of the alerts as published as one large dataset. This contrasts with other standards where we might request information just for a specific trip, or a specific vehicle. One final note is that it is impossible to extend GTFS-realtime specification. In the event that an agency has a need to add an additional field or message, the Protocol Buffer specification has a mechanism to enable this. Well, now we'll hit the details of GTFS-realtime messages of the GTFS-realtime specification. We are looking at some overarching definitions that we will see as we go through the specification. Again, it's important to just understand the basic concepts here. A message is the fundamental organization of data. As we said, a message contains a series of fields, and these fields may have sort of data in them, or other messages. A message is the highest level of specification as well. So a GTFS-realtime feed is a message that contains other messages and fields within it. Messages, as we said are composed of fields. Each field has a type, which describes what that field is. Each field also has a property of being acquired optional or repeated. A field that is repeated appears zero or more times. Of course, a required or optional field would only appear once in each sequence. Finally, each field has a description that describes the context so we know exactly what data should be included. We should note time in the

GTFS-realtime specification. Time is not, in most cases, time is not shown in the format that we are used to, which will be, you know, time such as twelve o'clock AM. Instead, time in GTFS-realtime is included in what is called POSIX TIME, also known as UNIX time or EPOCH time. This is an integer value. It measures the number of seconds since 12 o'clock AM, UTC, on January 1st, 1970. This is a common way to represent time in various computer operating systems. Because it takes up less space, and provides an exact universal time, independent of time zones. However, because most of us don't know the number of seconds the current time is since January 1st, 1970 at midnight on that day, there are websites available to help us convert that time. In true time we are familiar with. Here's an example on the bottom. We take 12:00 AM on January 1st, 2016 Eastern Standard Time (EST). That translates to 1464104462 sec since 12:00 AM on January 1st, 1970 Greenwich Mean Time, or UTC. This is more familiar to us. However the GTFS-realtime specification requires us to convert time into POSIX TIME with all except, with one exception and we will show what that exception is as we go through the specification. One final note is that some fields are considered experimental in the GTFS-realtime specification. This means that their use has not been formally adopted. And they are subject to change, and the usages are subject to change. Here shows some of the data types that we will see within the GTFS-realtime specification. Each field reference with some type of data. The first time is a message. Message is simply a collection of other fields, that could be other messages or other data types. We've seen an example of a message here. Some types of fields are called enum, or enumerated values. This is a list of specific values that can appear. Some data types are strings which are a set of alphanumeric characters, such as the phrase "express1" that we see here. We also have various ways of specifying numbers in the GTFS-realtime specification. The first two, "uint32" or "uint64" mean an integer that is either zero or positive. The 32 and 64 are important for computer developers to know. However we don't need to worry about them for the purpose of this module. On the second line, if we see "int32" or "int64", that means that we have an integer value that could be either negative, positive or zero. Again the 32 and 64 are not important for this Module, they have to do with the size--maximum size of the data. Finally we will see "float" and "double" appear. Those are numbers that could have a decimal after. And those values get float double, have to do with the size of the value. But that's not important for this Module. And finally, the final type that we will see is called a "bool" and that means either true or false. This graphic that we will see next, shows the high level of organization of the GTFS-realtime specification. We'll look at the details of each of these messages in the upcoming slides. Of course, as we've mentioned the highest level of structure in the GTFS-realtime specification is the feed message. Is the actual GTFS-realtime feed, which is messaged as a feed message. This message contains one message called with a field called "header," which is a feed header message. And it also contains the entity message, which is message type feed entity. The entity field, which is message type feed entity. An entity is repeated for each thing we need to discuss or describe in the GTFS-realtime feed. So every trip update there's going to be an entity message every vehicle position is going to be an entity message, and every alert is going to be an entity message. So the entity message, entity field is repeated many times within a message. And each entity as we said could have, will have either the trip update field, the vehicle field or the alert field. Each entity never has more than one. Again, if we need to show, you know, a trip update gets its own entity. Every vehicle gets its own entity. Every alert gets its own entity. And that's an important concept to understand as we go through this. So now we will go through the specific messages contained with the GTFS-realtime specification. And as we do that on each slide we will see this table. This table shows us the field we're talking about, whether it's required, optional or repeated, or the type of field, which might be a message, it might be a specific data point. It's not important to memorize every field that we see. It's important to understand the overall intent of each field, and not every message. If you would like to know more about each detail, you can always go back later and look at the details. And GTFS-realtime specification reference provides a good source of the detailed information. So we will just understand the concept that each message is telling us. The first message as we see here is the FeedMessage. This is the message that contains all other messages within the GTFS-realtime feed. And so a message has a header, which describes data about the feed, and a repeated entity message which shows each point of information that we are describing. Here's an example of the Feed Messages. So theoretically this could be a GTFS-realtime feed, because the Feed Message is synonymous with the feed itself. So we see the top, the header which describes some information about the feed, such as the version and the timestamp. And then we see the first entity, which shows, this one shows the trip update. The next and need to show another trip update, or it can show the vehicle information. And now we see the FeedHeader message itself. Again, this just describes what's known as metadata, or information about the feed. So we can describe the version of the feed which is currently 1.0 for everything, because GTFS-realtime is only at version 1.0. We're also gonna put a timestamp on this to say when the message was released. And we see a value called incrementality, this is an experimental field right now. It's not commonly used, but it does have to do if we're releasing a whole feed or just part of the feed. But because the use is not really formalized right now, we don't need to worry about that, understanding details of how it works right now. We see at the bottom an example of the header feed, which gives us a version for GTFS-realtime incrementality and a timestamp. The FeedEntity message is next. As we said the FeedEntity message for the entity field, which references the FeedEntity message is repeated for many, many times. So every trip update will be within its own entity message. Every vehicle will be within its own entity message. Every alert will be within its own entity message. And each entity can only contain one type of feed. So the entity could not contain a trip update and a vehicle message, those would need to be separate. Here's an example of an entity message showing a vehicle position. Here we see the entity has an identifier at top, and it is the leading field which is related to incrementality, and still experimental, and not necessary to understand at this point. And then we see the details about the vehicle position. We now move in and learn about the first of the three feed types. TripUpdate as we've discussed is the feed type that describes when a vehicle or when a trip will arrive or depart from a specific stop. So this message contains the information to disseminate this.

So we can describe the trip or the vehicle, and then we can repeat a stop time update message, which gives us specific details about each stop, and we repeat that as frequently as needed for all the stops on the trip. We can also report the delay of the trip in this message. Here's an example. We have TripUpdate at the top of what we have information describing specifically what trip we're discussing. And then we have the stop time update message that is repeated. It shows arrival times and departure times in a specific stop for trip along, the trip being described. So every trip will go into one trip update message. So this message here would describe one trip. We have a message called the TripDescriptor message, and this message describes a specific trip. This is a way for us to know what trip we're talking about. It's referenced in all three of the feed types. Trip Update, Vehicle Position and Alerts. All in some point reference have this message contained within it, so that we can describe to our downstream users what trip we're talking about. We can describe a trip with a trip ID, with a route ID, or a direction ID. These values typically should link to a static GTFS feed, so if we give a trip ID, we should be able to look that up in the static GTFS and know what trip we're talking about. The same with the route or the direction. We can also describe a trip by the time it starts. You know, if this value, this start time value is the only time that we will see a time that is not in POSIX TIME. This is just in the standard 24-hour clock time. And we can also describe the schedule relationship, so we can describe if a trip is scheduled in the static GTFS feed or unscheduled or canceled. That field can be used for that. And here's an example of this message. We see we have a trip_ID, a start time and a start date, which are- and the time of course is not in POSIX TIME. And date is in the format yeah, yeah, year, year, month, month, date, date. Or 20160512, which translates to May 12th, 2016. We see all the fields are included so we can look up, for example, Trip_ ID:121 in the static GTFS feed. We can look up Route_ ID: 2 in a static GTFS feed. Our next message is the StopTimeUpdate message. This is the message that is repeated for every stop on a trip. So we can describe a trip by either the stop sequence, which would relate to the static GTFS trip, or the stop identifier, which would be a stop that we could look up in the GTFS feed. Therefore, the reference is very important that we can match static GTFS. We can also describe an arrival and departure time for that trip. Here's an example. We see a one stop time update on a specific trip. This is Stop_Sequence number 20. Which is also Stop_ID: 34. We know what those mean from static GTFS feed. And we have an arrival time and a departure time. Notice there is no delay on the trip. And the arrival and departure times appear in POSIX TIME, so the value we see there is the number of seconds since January 1st, 1970. The StopTimeEvent is where we actually show the time, the arrival time or the departure time. So we can display for a specific time, either the delay or the specific time or both. We can also include uncertainty, which is a range of effect into the uncertainty, so we see the trip is accurate to within 30 seconds, we can make an uncertainty value of 30. If we don't have a way of factoring in this, we would just leave it out. The second type of feed that we have discussed is the VehiclePosition message. The VehiclePosition message is one of the high level messages that would appear within an entity. And this describes a vehicle location. So we can tie vehicle location to a specific trip or a specific vehicle. And for each vehicle, we can describe a specific position, which will be a coordinate position, or stop sequence along the route or trip, or we can describe a specific stop that a vehicle might be, either incoming at, in transit to, or stopped at. We also have some other values that describe information about the congestion of the route, or the occupancy status. These are still experimental or not widely used fields. An example of what a VehiclePosition message looks like. We see towards the top of the top of the message, we identify the trip. We have a trip ID that we can relate to. We also provide a position, this being specific latitude and longitude coordinates, as well as a bearing or direction on the compass that the trip is facing. That the vehicle was facing and traveling at. And we also describe a current stop sequence, whether or not the trip has stopped at that stop, we see here that it is. And an identifier for that stop. We also at the bottom have information describing the specific physical vehicle. So from that we could look at a vehicle and see a number on the side of the vehicle and know whether or not that's the right vehicle. Here's a VehicleDescriptor message, which describes whether or not, which describes a specific vehicle. We can describe a vehicle by an identifier value used internally. By the label, which would be the public number on the side of the bus or train. Or the license plate of the vehicle if we have that. And these are just ways to help customers know what specific vehicle to look for. Another message that appears within the VehicleMessage is the Position. Here we can describe a vehicle location by its latitude or longitude. Of course we can also describe it by the speed it's traveling at, the bearing or direction which on a compass, zero degrees being North, 90 being East, 180 degrees being South, and 270 being West. We can also describe the odometer value if we have that. We see an example which just shows latitude and longitude, and bearing.

**Scott Altman:** Now we will discuss the third and final feed type, this being the Alert message. In an Alert message we can describe a time of the alert, when it starts and stops. We could describe an entity that is affected, as well as the cause and effect, and some other text-based details about the event. Here are lists of possible enumerated causes and effects. Enum of causes. And a list of effects on the right. If we don't-- if our specific cause for an alert or an event is not covered, we can always use "Other" cause or "Other" effects. Here's a sample of an Alert menu. We see at the top it has an active period start or end. Then we have an informed entity, which is the essentially the objects that are affected, so the agency or the route affected by the trip. And we see there's a cause and effect, as well as text-based descriptions of the event that is occurring. So we can describe a time range for an event by a starting or ending-- starting and/ or ending time. If we know when the event started, we can transmit that. If we know don’t know that, then we leave it out. If we know when a trip is going to end, we can add that. If we don't know it, we can leave it out. So if the ending period is blank, we treat that as unknown. And if the starting time is blank, we treat that as unknown. There are cases when we can't predict how long an event lasts, or we didn't record exactly when it started. This EntitySelector message allows us to select the entity affected, or describe what entities are affected by a message. For example, a specific agency might be affected by an alert, a specific route, route type, trip or stop ID might be affected and alert. We can describe all of those using this message. So for example, in the example we see below, we see an agency ID of one, which we can look up in the GTFS. And affected, route_ID 36 was affected, and Route_Type 3 is affected. Inside GTFS Route_Type 3 means it's a bus. Another way to think of the informed entity field or the entity selector message is that these are the groups that must be informed of this message. So users of this agency, this route, and this route type, they should be told about this event, because it might affect their trip. The TranslatedString message is used to describe text-based information. If you recall in the slide where we discussed the field contained in the Alert message, we could describe a URL, a header text, or a description text. Each of those fields has a message Translated String. This allows us to provide multiple translations of a specific message. For example, in a bilingual environment that you speak English and Spanish, we have a mechanism to disseminate a message to customers in both of these languages. A translation, therefore, as we see in this message can be repeated throughout a TranslatedString. So in the messages below we only have one translation, that being English, and being a standardized code for the English language. However, if we also wanted to disseminate this in another language, say, Spanish, we would translate, we'd have a second translation within that message, and we have that same message in Spanish, and a language code for Spanish. So this allows us to get information out to as many customers as possible.

**Scott Altman:** So now that we have discussed the data that is contained within a GTFS-realtime feed, we'll begin to talk about how we translate that data from its data source into GTFS-realtime. We mentioned earlier the types of data that are required, and the systems that this data comes from. The first step, of course, so we can evaluate the data and see what we have available and what we could use to create GTFS-realtime from. Once we have that data, we export the data into some whatever format it's available in. If we are not already exporting it in GTFS-realtime, then we have to translate into GTFS-realtime. And finally we need to reference a static GTFS file, because that is a requirement of the specification. Now if we go back to the exporting step, it's possible that the systems that contain the data, such as an AVL system may already have the ability to export GTFS-realtime data. Now there are some tools that can be used to work GTFS-realtime. However, unlike GTFS, static GTFS feed, there are no standalone off the shelf tools that are widely implemented for this. This, of course, can be attributed to the fact that GTFS-realtime need to integrate with other systems contained by an agency. Google, the maintainer of GTFS-realtime does provide open source tools that can read the Protocol Buffer format and can be integrated with other custom software tools to write and read GTFS-realtime feeds. If you are familiar with software programs, these tools are officially available in Java C++ and Python. Using these tools does require development, software development experience. So therefore you may need to rely on someone who has that experience if you do not have it. CAD or AVL systems and other systems that an agency has may provide location information, as these are the major source of data for

GTFS-realtime feeds. These systems may include modules for exported GTFS-realtime. These might be an add-on that you can purchase, or they might already be there, and you just need to turn it on. Finally, it may be necessary to generate custom software to provide GTFS-realtime feeds. In the screenshot we see a custom software tool, and the software for it. That was written to work with GTFS-realtime using open source tools that Google already provides. Understanding the software code that you see on the screen is not important for this Module. This just shows us that there is software that works with GTFS-realtime feeds.

**Scott Altman:** So we come to our next Activity, and we have a question. And the question is, "Which of the following formats is used for encoding a GTFS-realtime feed? Our choices are: a) Extensible Markup Language or XML; b) JavaScript Object Notation, or JSON; c) Protocol Buffers; or, d) Comma Separated Values, or CSV. Our correct answer is C, Protocol Buffers. And these are the format used for GTFS-realtime. All other choices, a) XML is not used for GTFS-realtime; JSON is not used officially for GTFS-realtime; and CSV are not officially used for GTFS-realtime. We have a second question. And this question is, "Which of the following is not a way to show location in GTFS-realtime?" Our choices are a) Latitude/Longitude; b) Stop sequence on a trip; c)Stop identifier; or, d) Distance to destination. And our correct answer is d) Distance to destination. There is no field for this in GTFS-realtime. However, if your agency needed this, you could extend the GTFS-realtime feed to show it. The other options, the other choices, a) Latitude and Longitude is incorrect, because the field latitude and longitude are used for this, similar to current vehicle position message. b) Stop sequence on a trip is incorrect, because the field current stop sequence is used to show location based on stop sequence. And c) Stop identifier was incorrect, because the field stop_ ID is used to show the stop identifier, or the stop that a trip is at.

**Scott Altman:** We will now discuss our third and final learning objective, Implementation of GTFS-realtime. Our key points are, keep testing GTFS-realtime files and GTFS-realtime validation tools; illustrate how an agency implements GTFS-realtime; and we will close with a case study. All along we've been talking about the importance of providing high quality data. High quality data is data that meets the requirements of the GTFS-realtime specification, properly references the static GTFS, is accurate, complete precise, high resolution and high frequency. What does that really mean? That means what I just said that data that does not conform to the specification cannot be processed, and data that is not overall high quality and accurate is useless to downstream users. Therefore, we need our data to be conformant and appropriately reference the static GTFS feed, so that we can properly use it and make it available. Testing exists to make sure that data is correct, and there are two major levels of testing that should take place for a GTFS-realtime feed. The first is specification conformance. This includes ensuring that all mandatory objects are present, that all the values in each field are the correct type. That references to the static GTFS feed are correct, that each entity only has one type of message, and that all fields are in the appropriate message. Each of these will produce errors if they are not correct. Additionally, there should be tests that each field is recognized, which should produce a warning if there's a field that is not recognized. If there's a field that's not recognized, the Protocol Buffer template for the GTFS-realtime feed will need to be extended to accommodate for this field. It's a good idea to search for optional fields that may not be present, as this also might provide an information about lapses in data. The second area of testing is to check the data accurately reflects operations. This often is manually verified by staff to ensure that what a feed is showing is the same as what is happening in real life. Often we might randomly sample a test to do this. Make a random test to check one part of the feed. In addition, we can automate this process to check GTFS-realtime data against other realtime data held by the agency. Finally, we need to test that a GTFS-realtime feed appropriately references the static GTFS feed. For example, we see on this graphic here, the stop IDs after reference the stop ID, then stops that text file on the static GTFS feed. If the reference is incorrect, then downstream users will not know what stop you are discussing in your GTFS-realtime feed. Like any other data specification, it is important to test that the feed being produced is conformant to the defined specification, and that it accurately represents what it describes. There are no off the shelf tools commercially available for the sole purpose of testing GTFS-realtime feeds. Any tools available are either experimental or designed for a specific implementation on a GTFS-realtime. Testing tool are often therefore built into the process for producing the feed, again built into the process for consuming the feed. If Google open source tools for working with Protocol Buffers are used to write and read GTFS-realtime feeds, there is some level of testing built into them. This being they won't allow invalid or undefined value to be imported into whatever tool you're building. However, the don't provide clear errors or warnings, so we might need to build that in. And they also don't check that a GTFS-realtime feed references a static GTFS feed, so you would need to build that into your application to ensure the consistency. On the screen we see test app for some custom test groups that were used to test GTFS-realtime file, and these test scripts that we see the output from, well, we're using Google's open source tools. Now we'll talk about how an agency actually implements GTFS-realtime. First we will discuss the implementation strategy for an agency. As part of the strategy, we must look at the checklist of items that must be considered as part of the implementation. First, an agency that wishes to implement GTFS-realtime must determine the need for realtime information. It may, or maybe in most cases, an agency's customers will benefit from realtime information. But this is not always the case. So we should only implement GTFS-realtime if there is an actual need. If customers won't use the information provided there, it might not be worth investing in. We also might want to consider, if it might  be useful to provide dynamic message signs at stops that rely on GTFS-realtime data. As part of the implementation of GTFS-realtime, an agency must inventory its existing systems in the data it has available. An agency should ask if it already has a way to produce GTFS-realtime. If it does not, then it might need to procure a system. We should also determine feasibility. Is it feasible to implement GTFS-realtime feed? Are the funding resources available for procurement? Are staff resources available to maintain the feed?  Data lifecycle requirements specific to the agency shall be determined around this time. Finally once everything is determined and the need is determined, we could procure and/or develop the feed. This might require development of custom software or might require a procurement process as specified by the agency. And finally once all systems are in place, we can produce data and provide it to downstream users. So on the previous slide we presented a checklist for implementation. Here, we look at some questions that must be addressed by an agency that is considering providing GTFS-realtime feeds to downstream users. And we do that by asking you a question, "Should GTFS-realtime be implemented?" To answer this question, we ask some other questions. First, we need to establish the need. The first set of questions is about establishing a need and identifying your usage. So we ask, "Will customers benefit from realtime information?" There might be a case where customers do not benefit from realtime information. For example, if service almost never varies, then there may not be a need to provide up to the minute information. As a secondary use, we could ask, "Will archiving the realtime information be useful?" For example, will we be having planning tools that analyze on-time performance, using archived realtime data? Finally, we should ask, "Are there downstream applications that will use the data?" Are there applications that customers will use to view the data? So we need to identify if there is a need in usage. We also need to determine whether or not it is feasible to implement GTFS-realtime? Can existing systems be leveraged? Are there systems in place? Do we need to procure systems, if we don't already have systems? Will funding exist for procurement if we need to procure mid-process? And finally, if we're producing GTFS-realtime feeds, can they be maintained? Do we have staff resources available? All these questions should be asked to determine whether or not we can produce a GTFS-realtime feed. So if we answered yes to these two groups of questions, there's one finally question that we need to ask ourselves, "Is GTFS-realtime the correct format to use?" Earlier in the module we discussed the other formats that are used for sharing realtime transit information. One of these might better meet your needs, and that is okay. However, if GTFS-realtime does meet your needs, and downstream users are using that feed, that might be the correct format to use. But each agency that wants to implement a GTFS-realtime must ask themselves this question. System Procurement is required if you want to export GTFS-realtime feeds. We will discuss how that’s done. We will always recommend for ITS procurements using a systems engineering process to manage risk. To ensure the system is built correctly, and ensure that the correct system is built. And that's important to keep your costs under control and make sure you are building and getting the exact system you need. This, of course, is recommended when using systems engineering to start with a concept of operations or ConOps. This allows us to describe the user needs, such as the example user need. Downstream users need to receive trip update information. By the time we've completed a concept of operations, or ConOps, we should know the goals and objectives for procurement. We then define requirements. System requirements that we would use, or functional requirements and performance requirements. Functional requirements describe what a system must do. An example here states, "The system shall export a GTFS-realtime feed, containing the TripUpdate, VehiclePosition, and Alert messages." That's a specific requirement that could go into a procurement document. Whilst that performance requirements that describe how well a system must perform. For example, the system shall provide an updated GTFS-realtime feed every 30 seconds. That could also go into a procurement document. And of course that could be used as a requirement that a contractor must follow when providing the system. For more examples of procurement language, please see your student supplement. We provide some examples that you could turn into specific procurement language for your agency if applicable. And I also encourage you to visit ITS Transit Standards Module #1, Introduction to ITS Transit Standards. This provides you information about the systems of engineering process that you might find useful. We'll talk about some data life cycle requirements and strategies. The lifecycle of a GTFS-realtime feed is quite short. Transit Systems Operations change on a second-by-second basis. And this must be accounted for in GTFS-realtime feeds. The first process/first step in the lifecycle process is to collect data by either a CAD system, an AVL system, or other systems on a frequent basis. This data must then be merged into a GTFS-realtime feed. And exported at a defined frequency. This might be every 30 seconds, for example. Testing should also always occur on an exported feed, particularly to inform that the feed that feed you're exporting conforms to the requirements of the GTFS-realtime specification. We don't necessarily test the accuracy every time unless we have a fully automated process for doing that. Finally, data is provided to downstream users, such as customer facing applications, operation staffs, archiving systems, or third-party developers who use the data. In terms of lifecycle strategy it is important to ensure that the data being imported through GTFS-realtime feed is sufficiently up to date. It is not useful to export the feed every 30 seconds, as vehicle information can only be imported every ten minutes. On the other hand, releasing data more frequently than needed requires more bandwidth, and can lead to data overload and cause problems with downstream applications. The key strategy to takeaway is to ensure a good balance between providing up to the minute information without inundating downstream users. So of course the strategy we have discussed here is repeated, we just have to define frequency as we see, and that's really what's most important, make sure data is fresh and repeated frequently. And our final step of implementation is making data available to the users who use it. Typically GTFS-realtime feeds are always they're updated in realtime. Therefore users must be given constant access. Typically a GTFS-realtime feed is placed in a fixed location with an existing URL that can be accessed. The GTFS-realtime specification does not define request messages. Instead agency is defined the request protocol that's going to be used, also known as an application programming interface, or API. And API is the set of rules and messages that define how to get information through from the system. Although this is undefined by the GTFS-realtime specification, typically Hypertext Transfer Protocol, or HTTP methods are used for a consumer to request data. HTTP is an application layer protocol commonly used to send and receive data over the internet. Typically the data consumer, such as a third-party developer can set their application that consumes GTFS-realtime to check a specific URL or location at a defined frequency and then the data is returned. Alternatively, the system exporting GTFS-realtime may also automatically provide data to known recipients as data is updated. Often, in order to receive access to a GTFS-realtime feed, it's required for a user, and the user is provided a code, which they then submit as part of their request, which says that they authorized to receive the data. And when providing GTFS-realtime feeds, it is important to make sure that the corresponding static GTFS feed is available as GTFS-realtime feeds do need it to reference it.

**Scott Altman:** We will now close our Module today with a Case study. This Agency Case Study describes the implementation of a GTFS-realtime feed by Transcom, which is a consortium of 16 separate transportation related agencies in the New York, New Jersey and Connecticut Region. Historically, they have provided traffic related data feeds affecting the highway network, and they are now also implementing transit related feeds. Transcom is integrating realtime transit feeds to its member public transit agencies into its Data Fusion Engine, and then making this data available in two formats. One of them being GTFS-realtime and one being SIRI. This is the least of their data exchange system. Transit alert data about ongoing events is also collected by another incident management system that they have called Open Reach. The motivation behind this effort is to promote sharing of public transit information, including for two integrated corridor management projects that they are providing data for. As well as other operational uses by the member agencies, such as providing dynamic message signs that have realtime in transit information. GTFS-realtime feeds are available to downstream users who are given access. It should be noted that each system, each transit system's feed is maintained at a separate location by Transcom, because they are all referencing static-- different static GTFS feeds. It's important to know, of course, which static GTFS feed is being referenced. Additionally, Transcom is in the process of developing a transit dashboard that provides performance metric information based on realtime transit information that is collected and archived as part of this effort. We see a diagram of Transcom system. It's not important to understand details of this system, it's just important to understand how data gets from its source to its final location. So data comes from member agencies that provide realtime information. Some agencies provide GTFS-realtime feeds already, others provide other formats including a custom format provided by one agency. This data goes into their Data Fusion Engine which fuses many different types of transportation related data. It is then exported, it is then created, exported-- once it's loading into the database, it is exported as a GTFS-realtime feed, as well as a SIRI feed, which is the other transit format that is being used. Other data is also important from their Open Reach Incident Management System. So this shows how data is translated in some cases into a common format. Here we see the agency that Transcom is integrating as part of the GTFS-realtime feed. Year 1, which previously included integrating New Jersey Transit Rail data, which uses a custom XML format. That format was translated into GTFS-realtime. The second year, which is ongoing as of the time of this module is integrating data from New Jersey Transit Bus, New York City Transit Subway, Long Island Rail Road, the Metro North Railroad. New Jersey Transit Bus also uses a custom

XML format which must be translated, and the three MTA agencies have GTFS-realtime feeds. Although they have already GTFS-realtime, some translation needs to occur, because at each feed, data is represented slightly differently in different fields, actually are used to mean different things. So they are normalized into one format so that they can be unequivocally interpreted by downstream applications. And also you should note that these are rail agencies, and they collect location data from rail based networks that embed sensors into the rail network as opposed to a GPS-based system. And finally, Year 3 would include MTA's New York City Transit Bus, and that is a SIRI format that would be translated into GTFS-realtime. So are lessons that are also learned from this case study. That effort had to uncover some issues with data integration. This effort includes five feeds that must be unified into a common format. The biggest problem is that there are issues inconsistency in the data. For example, one of the agencies provides GTFS-realtime data that uses the time field to represent the scheduled time of the trip as opposed to the estimated time of arrival, and they only use the delay to tell you the estimated time of arrival. Several agencies also don't use the trip ID value. This creates problems with interpretation. So the two key lessons are one, the meaning of data must be well-documented, so other users know what is meant by data point. And two, when integrating data from multiple sources, it's important to understand the meaning of each field.

**Scott Altman:** We have our final Activity, and we have a question. The question is, "Which of the following methods is most commonly used to access GTFS-realtime feeds?" The choices are: a) Email; b) FTP; c) HTTP; or d) Telephone. And the correct answer is C. HTTP methods are commonly used to share GTFS-realtime feeds. Email is not used because it is too slow. FTP is not typically used. And Telephone is not used because that is not an electronic format.

**Scott Altman:** So let's review what we have learned today. One, we learned the background of GTFS-realtime specification, it's benefits and its uses. Two, we learned how GTFS-realtime feeds are structured, the content of the feeds, and how feeds are created. And three, we learned how agencies test and implement GTFS-realtime feeds. Overall, this module taught us about the it specification and how it can be used. So we do thank you for completing this module and learning about GTFS-realtime today. We do ask that you submit your Feedback, and there's a link provided to you so that we can improve training and understand what you liked and did not like about training. So we thank you very much for both your both your feedback and for participating today. Thank you.

#### End of 2016_09_14_11.07_Mod_14_P2_Final_Record.mp4 ####