Rail-Crossing Violation Warning (RCVW) Software API Description

Version 4.0

Battelle

505 King Avenue

Columbus, Ohio 43201

April 25, 2024

Table of Contents

Page

[Table of Contents 2](#_Toc164926084)

[Chapter 1. High-Level Architecture 5](#_Toc164926085)

[Chapter 2. Component Design 7](#_Toc164926086)

[TMX® Core Software 9](#_Toc164926087)

[Apache Kafka 9](#_Toc164926088)

[AMQP 9](#_Toc164926089)

[Asynchronous I/O (AIO) 9](#_Toc164926090)

[Simple Network Management Protocol (SNMP) 9](#_Toc164926091)

[VBS Software 10](#_Toc164926092)

[Message Receiver Plug-in 10](#_Toc164926093)

[GNSS Plug-in 10](#_Toc164926094)

[Differential Global Positioning System (GPS) Plug-in 10](#_Toc164926095)

[RCVW Plug-in 11](#_Toc164926096)

[Human-Machine Interface (HMI) Plug-in 11](#_Toc164926097)

[RBS Software 11](#_Toc164926098)

[MAP Plug-in 11](#_Toc164926099)

[HRI Status Plug-in 11](#_Toc164926100)

[RTCM Plug-in 11](#_Toc164926101)

[RSU Immediate Forward Plug-in 11](#_Toc164926102)

[CBS Software 12](#_Toc164926103)

[Azure Function App(s) 12](#_Toc164926104)

[Azure Container App(s) 14](#_Toc164926105)

1. Overview

The Rail-Crossing Violation Warning (RCVW) is a prototype Vehicle-to-Everything (V2X) software application built upon the V2X-Hub solution provided by the US Department of Transportation (USDOT) Federal Highway Administration (FHWA)[[1]](#footnote-1). The V2X-Hub is itself a collection of intercommunicating software components for an Intelligent Transportation System (ITS) that uses Battelle’s Transportation Message eXchange (TMX®) platform.

The TMX® software architecture provides a robust, flexible, scalable, and extensible interface allowing multiple connected vehicle (CV) applications to co-exist in a single platform, while sharing critical resources between applications. TMX® is hardware-agnostic and can be modified to work with current roadside unit (RSU) or after market on board unit (OBU) hardware to facilitate CV message transmission and receipt.

The critical operations of OBUs and RSUs include sending and receiving SAE J2735 messages over the Cellular Vehicle-to-Everything (C-V2X) wireless protocols, and many CV applications require interfaces to external systems such as Global Navigation Satellite System (GNSS) positioning information, vehicle controller area network (CAN) data, traffic signals[[2]](#footnote-2), and cloud services. The TMX® middleware provides CV applications with the ability to exchange messages and external data efficiently and effectively across software components using a publish-subscribe model. The critical communication functions required for developing comprehensive safety and mobility applications are all implemented natively within the TMX® platform, leaving the application programmers to focus on coding the *business logic*. This ultimately leads to rapid, productive development of CV safety and mobility applications.

The goals of the TMX® platform and its V2X-Hub derivative is to provide:

* A ubiquitous means of message Tx and Rx over wired and wireless transport, including the licensed C-V2X bands and unlicensed spectrum such as cellular and Wi-Fi.
* A built-in means for efficient encoding and decoding of critical information without the need for custom data structures and code generation.
* A prescribed component architecture to maximize scalability and reusability.
* Native support for common CV and ITS functions such as simple network management protocol (SNMP), CAN, SAE J2735, National Transportation Communications for ITS Protocol (NTCIP), Global Navigation Satellite System (GNSS), and specialized functions such as geofencing and driver alerting.

The FHWA maintains a version of V2X-Hub that was originally built on the release 3.0 of TMX®, which is limited to a singular custom communication protocol built specifically for the US DOT prototype projects. While new components have been added to V2X-Hub, the underlying platform was never enhanced to take advantage of compiler evolution, or to include additional communication protocols and associated performance improvements. In order to meet the requirements of the RCVW Phase III project, Battelle has developed release 4.0 of the TMX® architecture and partial port of the necessary V2X-Hub components. This document details the updated TMX® architecture and the overall RCVW software design.

1. High-Level Architecture

Battelle has developed TMX® to be interoperable with a large range of hardware specifications, and to accommodate the plug-ins and applications that represent the information needed to produce and transmit/receive CV application messages. Figure 1 below illustrates the basic tiered architecture for development of a V2X ecosystem built using the TMX® platform.

Figure 1 – TMX® V2X Architecture

Within the TMX® architecture, each software application or service is set up as a participant of the message exchange, dubbed a plug-in. This architecture is optimized for rapid development of the application software, i.e. the business logic, as the platform abstracts the complexities of message encoding and transmission as well as facilitating cooperation and data sharing amongst the various components. This includes data exchange between plug-ins specifically for in-vehicle, roadside infrastructure, or other road users such as pedestrians or bicyclists, plus any supporting software required for each.

The communication of messages between plug-ins occurs using the Publish/Subscribe (Pub/Sub) architecture. Some plug-ins produce messages while others consume them, but all messages are routed through some message broker, known as the TMX® Core. See Figure 2 for the general message flow through TMX®.

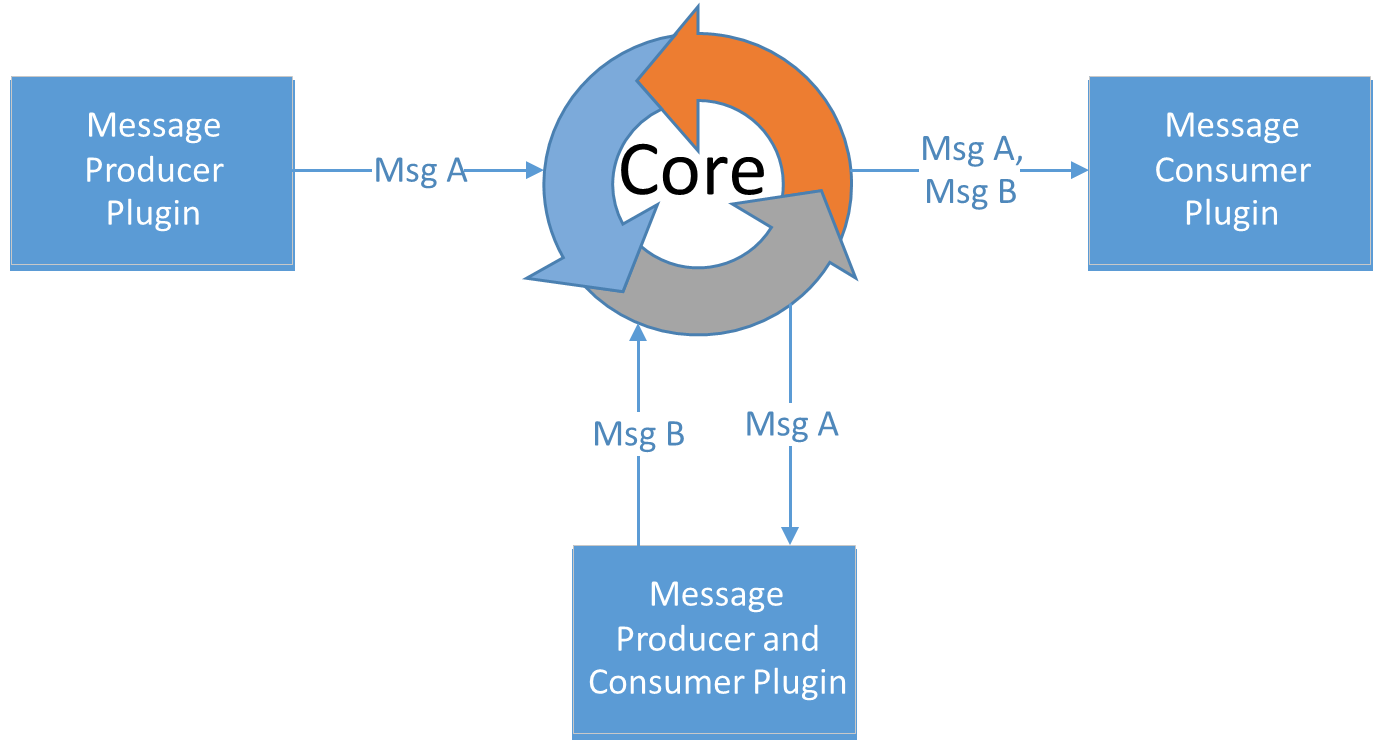


Figure 2 – The TMX™ Publish/Subscribe Architecture

Previous versions of TMX® generally supported a single core message broker per device, even if the plug-ins could operate on remote systems. Additionally, this broker was a custom prototype built for the purpose of the project demonstration. While it functions adequately, far more capable software now exists which performs more advanced functions. Therefore, the purpose of this architecture update is to integrate the TMX® plug-ins across a wide range of cloud-native message brokers that operate independently of the V2X specific applications. In other words, TMX® 4.0 is a refresh of the architecture meant to decouple the broker from the plug-ins.

1. Component Design

Effectively, the TMX® software is middleware that provides an Application Programming Interface (API) to facilitate the rapid development of the applications. While conceptually, the goal is to build a complete V2X application, this is accomplished by building and combining plug-ins. For example, consider the three basic components of the RCVW application:

1. **Vehicle-Based Subsystem (VBS)**: Responsible for determining unsafe conditions and issuing alerts based on the RBS information and vehicle specific telemetry.
2. **Roadside-Based Subsystem (RBS)**: Responsible for reporting the geometry and pre-emption status of the highway-rail grade crossing.
3. **Cloud-Based Subsystem (CBS)**: Responsible for disseminating rail-crossing information to other subscribers.

The required plug-ins for each of these components, along with the data exchange between them, is shown in Figure 3.

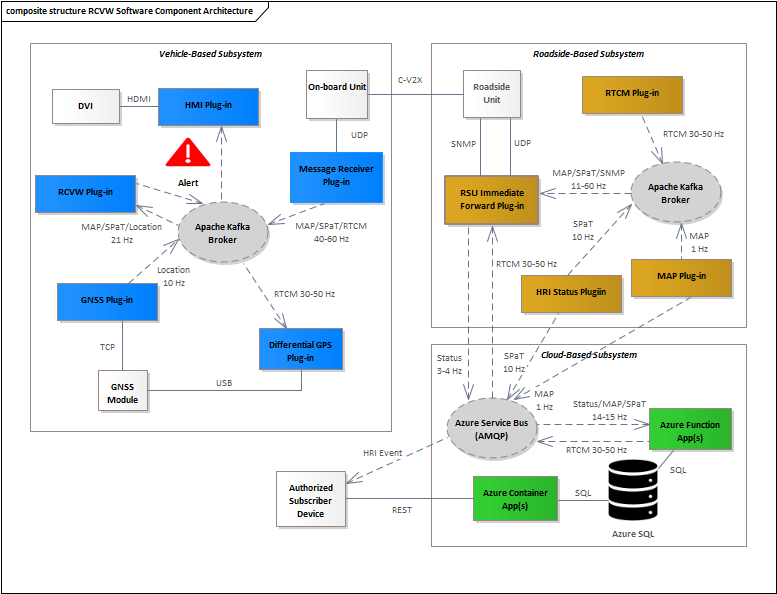


Figure 3 – RCVW Software Component Architecture

RCVW Core Software

The core message exchange within RCVW is provided by TMX® in order to connect all the application plug-ins together. The primary driver for the TMX® v4.0 architecture is to decouple the core messaging protocol from the plug-ins, and to support multiple different brokers simultaneously. This is key for RCVW application because the roadside component for Phase III was designed to send its messages both to the local radio and to the cloud-based component.

While latency of the message delivery is critical for optimal performance, there are a number of other factors such as scalability that may guide specific implementations at different junctions of the system or for particular applications. Therefore, every plug-in must be unaware of the underlying technology implemented in the core it is connected with. Likewise, a plug-in may be connected to multiple cores at once in order to publish to multiple exchanges with different scalability needs, it was more practical to achieve those results with proper architecture of the core broker technology itself, such as using clusters and bridging. The only real requirement of the core technology is some basic support for the Pub/Sub architecture. This section lists some messaging or general communication technologies that have been adapted to operate within TMX® for the RCVW project.

Apache Kafka

While there may be more general-purpose Internet of Things (IoT) messaging protocols, it is hard to deny that the opensource Apache Kafka streaming service is prevalent in its use. After previous research determined that Kafka provides very low latency with minimal overhead and a simple to use C++ API, the Battelle team decided that Apache Kafka would be the default opensource broker for TMX® applications, particularly for the internal routing of messages within a subsystem. Therefore, this means that a separate Kafka broker service is required for both the RBS and the VBS.

AMQP

The Advanced Message Queuing Protocol (AMQP) is a standardized, distributed system similar in function to the Unix POSIX message queues. Whereas the Apache Kafka service is to be used for intra-subsystem messaging, the AMQP standard is already supported in a number of cloud services such as Microsoft Azure Service Bus, which provides a near real time solution for cloud ingest. The Apache Qpid Proton toolkit provides the underlying cloud enabled C++ implementation for the AMQP broker in TMX®.

Asynchronous I/O (AIO)

Although not explicitly depicted in Figure 3 as a TMX® Core technology, the platform provides a broker implementation for access of file system and/or network resources using an asynchronous interface. These include Internet Protocol (IP) access to Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) services, as well as data on serial interfaces such as the Universal Serial Bus (USB). While many libraries for asynchronous access to file and network resources exist already, this particular implementation adheres to the specific Pub/Sub architecture that a TMX® plug-in expects, and thus allows for the same kind of data handling. The sole purpose of this message exchange within RCVW is to support the V2X-Hub default message receiver plug-in, which is used to ingest incoming SAE J2735 messages from the external radio unit.

Simple Network Management Protocol (SNMP)

Many ITS systems use SNMP to manage devices, and V2X applications are no different. The SNMP broker is specifically used to pull management data off of National Transportation Communications for ITS Protocol (NTCIP) enabled devices. In the case of RCVW, the RSU radio is managed via SNMP like other NTCIP devices, and that is how one can obtain statistics on the current operational state. The TMX® SNMP broker software provides the general *get* and *set* operations to perform on any SNMP agent, but a plug-in must configure what data from the Management Information Base (MIB) to control. The RBS RSU plug-in in RCVW uses this exchange to retrieve radio diagnostics from the external radio using the RSU 4.1 MIB, which are relayed to the CBS and monitored.

VBS Software

Most of the safety logic is built into the plug-ins that run inside the vehicle. For this reason, there are some unique considerations for these. First, latency of the message delivery has a real impact on the operation of the plug-in. For example, every tenth of a second (100 milliseconds) wasted in transmission for a vehicle traveling at 50 mph directly implies 7.33 feet less of distance required to stop. Therefore, the messaging system should be optimized for the lowest latency, which means a kernel implemented loop-back interface for messages to be passed internally around the vehicle as opposed to traversing a serial network interface. Secondly, the asynchronous messages that are used to determine crossing state and vehicle telemetry must all be aligned to a common time frame based on the time created, and not received, to ensure data consistency.

Since one of the key benefits of the TMX® software is to be able to run anywhere, there is also a wide range of possible deployment hardware for vehicle plug-ins, including existing systems like C-V2X OBUs. However, it may be difficult to find OEM partners willing to run Battelle software on their equipment. Therefore, the RCVW project also supplies VBS prototype hardware that is optimized for performance of the on-board safety system. It is meant to be used as a black box solution for demonstration without integration within a specific vehicle.

Message Receiver Plug-in

CV applications are primarily defined by the exchange of SAE J2735 messages over dedicated radio spectrum in the 5.9 GHz band. The V2X-Hub Message Receiver plug-in is designed to accept the J2735 payload on a UDP server and forward directly to a TMX® core instance, in this case the default Apache Kafka broker on the VBS. For convenience, the RCVW III prototype architecture depicts the Message Receiver Plug-in directly on the VBS computing platform, but it may also be placed on separate hardware as necessary, including co-located with the OBU radio if the hardware allows.

GNSS Plug-in

The V2X-Hub GNSS Plug-in, formerly known as the Location Plug-in, is responsible for obtaining positioning data from the GNSS module on-board the VBS. The plug-in software connects to the module to the *gpsd* server process over a TCP socket connection. The plug-in encodes the resulting position information as a V2X-Hub Location message, that is sent to the VBS Apache Kafka broker instance for use in other plug-ins. Per requirements, the Location messages must be issued at least 10 times per second.

Differential Global Positioning System (GPS) Plug-in

This RCVW specific plug-in works along with the GNSS Plug-in in order to improve the accuracy of the positioning data. This is done by injecting Radio Technical Commission on Maritime Services (RTCM) Real-Time Kinematics (RTK) correction messages directly into the VBS GNSS module over its USB interface. The correction messages are expected to be received over C-V2X radio communications, and while it is conceivable that the bytes be relayed through the *gpsd* server, this was declined in lieu of the more efficient and direct method.

RCVW Plug-in

Previous phases of the RCVW project focused on developing an effective algorithm for issuing rail-crossing informational and warning alerts to the driver. The RCVW plug-in implements this algorithm based on the position and speed of the vehicle, along with the geometry and state information from the RBS. Therefore, it consumes the highest number of messages from the local Apache Kafka broker, while only producing the alert messages that need to be displayed.

Human-Machine Interface (HMI) Plug-in

This plug-in drives the HDMI monitor or infotainment display on behalf of the RCVW plug-in by turning the specific alert into an image to present on the screen. While the operational displays were not significantly altered for this phase of the project, this code now must be modified to read messages directly from the VBS Apache Kafka broker.

RBS Software

The job of the roadside plug-in is to communicate with ITS infrastructure devices, such as the rail-crossing controller system, and generate TMX® messages to pass to the vehicles over the radio and to external subscribers via the cloud-based component. For RCVW, the RBS must communicate the geometry and state of the Highway-Rail Intersection (HRI).

MAP Plug-in

The geometry of the HRI is summarized within a SAE J2735 MAP message, that lists the position of the intersection, and the approach lanes. For RCVW, the MAP is configured to include a tracked vehicle lane that exists in a distinct *signal group* from the vehicle approach lanes that it crosses. The V2X-Hub MAP plug-in provides the ability to specify the precise bytes (in hexadecimal) that should be broadcast every second for this HRI. Each MAP message is published to the local Apache Kafka broker and over AMQP to the Azure Service Bus (see Figure 3).

HRI Status Plug-in[[3]](#footnote-3)

This RCVW specific plug-in reads the HRI preemption signal status and forms a J2735 SPaT message for broadcast 10 times per second. A SPaT message only contains the current state for each signal group at the HRI, thus both MAP and SPaT are needed to decode the current preemption state of the HRI tracked lane. When the preemption signal is active, the tracked lane signal group is in a *protected movement allowed* phase, similar to a green light, while the vehicle lanes are in a *stop and remain* phase, similar to a red light. When the preemption signal is off, the phases of these signal groups are reversed. Each SPaT message is published to the local Apache Kafka broker and over AMQP to the Azure Service Bus (see Figure 3).

RTCM Plug-in

The primary purpose of the V2X-Hub RTCM plug-in is to receive binary RTCM data and convert those to an SAE J2735 message that can be broadcast by the roadside radio. However, the RBS may be configured to receive RTCM data in various different ways[[4]](#footnote-4). While the plug-in itself can be used for directly connecting to a Network Transport of RTCM via IP (NTRIP) caster and pulling RTCM messages, this option also is dependent on the V2X-Hub Location plug-in in order to receive boot-strap location information (NMEA). Additionally, previous RCVW iterations used the Location plug-in to pull RTCM data directly from a GNSS receiver configured to act as a local base station. While both of these options are functional, they also utilize additional RBS resources and network. Therefore, this plug-in can also be configured to receive RTCM data from the Azure Service Bus, which is supplied by the proxy server in the CBS, thus eliminating the need for the Location plug-in. In any case, the resulting J2735 messages are pushed along the local Apache Kafka broker to be broadcast over C-V2X. It is still up to the receiving vehicle to apply the corrections to its local GNSS receiver. This is, of course, done automatically by a VBS-equipped vehicle.

RSU Immediate Forward Plug-in

The RSU Immediate Forward is a V2X-Hub plug-in that receives messages from a core broker and pushes them to a connected RSU radio using the RSU 4.1 immediate forward specification. It is expected for RCVW that each MAP, SPaT, and RTCM message is to be broadcast, however the plug-in will only forward message types configured in the RSU immediate forward table. Others are dropped. Additionally, the radios are configured within the plug-in as SNMP agents so this plug-in can also monitor the radio performance. This information is published to the Azure Service Bus in the CBS via plugin status messages using AMQP (see Figure 3).

CBS Software

The cloud-based subsystem is not so much a TMX® subsystem as it is a Microsoft Azure subsystem that understands TMX® communication protocol. Therefore, the software is not developed using the standard TMX® libraries. This is an example of how a V2X ecosystem can be built using software that is not specifically built using V2X-Hub.

There are two basic methods for a subscriber to obtain data from the CBS. First, the polling method requires the subscriber to query directly from the CBS database. This is facilitated through the use of a RESTful API built into an application container deployed within Azure. While all the stored data within the CBS is available to be polled, it does require the subscriber to periodically execute queries in order to keep up to date with the fluctuating data, such as HRI pre-emption signal status and RBS operational status. Additionally, the web-based connectivity of such a poll makes it naturally slower, thus increasing potential latency concerns for certain applications of the data. Therefore, there is also a push method available, which lets the cloud components that detect the HRI pre-emption signal status as well as the RBS operational state, to forward resulting AMQP events through the Azure Service Bus. This is a highly efficient manner of distribution, but has the drawback of potentially missing deliveries if the client is not available to receive them. Fortunately, those issues can generally be remedied by proper configuration of the Azure Service Bus. However, even event driven applications also will require polling for general information about the HRI.

The cloud-based component of RCVW is built into two categories. The first are programs that execute upon triggers based on Azure events, such as receipt of incoming messages on the Service Bus. Each of these use the Function App capabilities in Microsoft Azure and are programmed in Python. Note that if no events are triggered, then an Azure Function App is never executed, thus making its resource usage highly efficient. Some software components, however, must be run at all times. Thus, the second cloud-based category of Azure Container Apps are used instead for those. A Container App generally are standalone components built based on another software tool, such as the Azure Data API Builder or programming language.

Azure Function App(s)

The Microsoft Azure cloud services include applications that can automatically be triggered by events in the cloud, including messages arriving on the Azure Service Bus. The CBS provides some default Azure Function Apps that record incoming messages into an SQL database. This function also decodes the message using a portion of V2X-Hub software so that the HRI preemption signal information can be used. Other functions then are used to update the current signal state and, if necessary, generate an HRI event message that is put back on the Azure Service Bus for downstream consumption by authorized subscribers. The following is a description of the Azure function apps.

rcvw-cbs-azure-ingest-azure

This is an Azure Function App that is built into a Docker container and used for writing records to the RCVW CBS data log, which for Azure deployments is an Azure SQL Database. This Python function is automatically triggered when a message arrives at the Azure Service Bus subscription and includes a portion of compiled TMX® software for decoding the SAE J2735 messages.

A separate trigger exists for each subscription for the specific Azure Service Bus topic that this App is listening for. The j2735.map rsu-incoming subscription handles J2735 MAP messages from the RBS Map Plug-in. The j2735.spat rsu-incoming subscription handles J2735 SPaT messages from the RBS HRI Status Plug-in. Likewise, there is a trigger for the rsu-incoming subscription of status and error topics for those plugins plus the RBS Immediate Forward Plug-in. Since all status and error messages are already JSON encoded, those require no additional decoding and are stored as-is in the Data Log.

Every message is written to the RBS\_INCOMING\_MESSAGE table, but the decoded MAP and SPaT messages are additionally written to RBS\_INCOMING\_MAP and RBS\_INCOMING\_SPAT, respectively.

rcvw-cbs-event-mgr-azure

This Azure Function App monitors the current state of the RBS. This includes checking that the RBS messages are being received at the appropriate rate, and independently detecting the current state of the HRI preemption signal. For convenience, the detection is done post-logging, thus may be lagging real-time by a few seconds. Additionally, each check is automatically scheduled by the running App, at first when a new RBS is detected, and subsequently every half second.

There are two scheduled checks. First, the RBS check is done to verify that:

1. All J2735 messages (MAP, SPaT, and RTCM) are being received at the expected rate.
2. All RBS plugins are reporting status at the expected rate.
3. All configured RSUs are operational and are currently sending radio broadcasts.
4. All expected IEEE-1570 heartbeat messages are being received, if applicable[[5]](#footnote-5)

If any of these checks fail, then the RBSOperational status is set to False (0). Only after all of these criteria are met will the RBSOperational status be set back to True (1).

Secondly, a specific check for the HRI preemption signal status is done. If the RBS is in the operational state, then the HRI Active state will be set to:

1. True (1) if the HRIActive field was False (0) but now has preemption active
2. True (1) if the HRIActive field was True (1) and still has preemption active
3. False (0) if the HRIActive field was True (1) but now has preemption inactive
4. False (0) if the HRIActive field was False (0) and still has preemption inactive

The main purpose of this App is to send events to subscribers based on the above criteria. A CBS subscriber application must obtain an Azure Service Bus connection string and program its own client of the Service Bus using any of the supported AMQP client libraries, such as the available C# .Net or Python Azure libraries. A general topic called cbs.events is where every HRI event message is sent, but each subscriber application should use the HRI-filtered subscription for monitoring their specific HRI. For example, while the rsu-outgoing subscription will hear all HRI event messages, the HRI\_10785 subscription is specifically set up to hear only the events for HRI number 10785. Each RBS-equipped HRI will have its own topic subscription.

Every CBS message will have a JSON-encoded body (contentType application/json) for maximum flexibility of the data. The subject of the AMQP message will be the HRI ID (e.g., 10785) and there will be an additional custom property called timestamp that provides the Linux timestamp when the message was created (**not** received by Service Bus).

Any error detected in the RBS, whether through the event manager checks or directly incoming on any Plug-in error topic will result in an event message body in the form of:

{

"HRI": 10785,

"code": 121,

"message": "RBS J2735 MAP rate dropped below the minimal operational threshold of 8 messages per second"

}

The HRI field denotes the crossing and should be identical to the message subject. The code is a numeric error code, and the message is a summary of the problem. There may also be an exception field included if the error was a result of a program run-time error. Note that a code of 0 indicates that this message is informational, and no error actually occurred. For example:

{

"HRI": 10785,

"code": 120,

"message": "RBS J2735 message rates restored above minimum operational thresholds"

}

The HRI Active event is not an error message at all, thus contains no code, but simply shows the preemption state transition from Off to On or vice versa. For example:

{

"HRI": 10785,

“code”: 1,

"active": true,

"message": "Preemption signal activated"

}

Or

{

"HRI": 10785,

“code”: 0,

"active": false,

"message": "Preemption signal deactivated"

}

Table 1 below describes each of the possible HRI event codes.

**Table 1: HRI Event Codes**

|  |  |
| --- | --- |
| HRI Event Code | Meaning |
| **0** | HRI pre-emption signal was deactivated. This event will be sent when the signal transitions from on to off and additionally when the RBS operational state is restored at a time when the signal is not active. The activeflag is also set to false. |
| **1** | HRI pre-emption signal was activated. This event will be sent when the signal transitions from off to on and additionally when the RBS operational state is restored at a time when the signal is active. The activeflag is also set to true. |
| *1XX* | Some change occurred in the expected messaging. These are in order of priority, meaning only the lowest value (highest priority) notice will be issued. |
| **110** | J2735 SPaT messaging rates that were below the operational thresholds have been restored above the minimum. |
| **111** | J2735 SPaT messaging rates that were above the operational thresholds have dropped below the minimum. |
| **120** | J2735 MAP messaging rates that were below the operational thresholds have been restored above the minimum. |
| **121** | J2735 MAP messaging rates that were above the operational thresholds have dropped below the minimum |
| **130** | J2735 RTCM messaging rates that were below the operational thresholds have been restored above the minimum. |
| **131** | J2735 RTCM messaging rates that were above the operational thresholds have dropped below the minimum |
| **160** | RBS diagnostic messaging that was absent from the HRI status plug-in has been restored. |
| **161** | RBS diagnostic messaging that was being received from the HRI status plug-in is now absent. |
| **170** | RBS diagnostic messaging that was absent from the MAP plug-in has been restored. |
| **171** | RBS diagnostic messaging that was being received from the MAP plug-in is now absent. |
| **180** | RBS diagnostic messaging that was absent from the RSU plug-in has been restored. |
| **181** | RBS diagnostic messaging that was being received from the RSU plug-in is now absent. |
| **190** | RBS diagnostic messaging that was absent from the RTCM plug-in has been restored. |
| **191** | RBS diagnostic messaging that was being received from the RTCM plug-in is now absent. |
| 2XX | Some change occurred at the RSU radio. These are in order of priority, meaning only the lowest value (highest priority) notice will be issued. |
| **210** | The RSU clock that was stopped is increasing. This likely denotes that the RSU is back on-line. The RSU host information will also be supplied in the event, if available. |
| **211** | The RSU clock that was increasing has now stopped. This likely denotes that the RSU has gone down. The RSU host information will also be supplied in the event, if available. |
| **220** | The RSU mode state that was non-operational is now operational. This likely denotes that the RSU was taken out of stand-by mode. The RSU host information will also be supplied in the event, if available. |
| **221** | The RSU mode state that was operation is now non-operational. This likely denotes that the RSU was manually put into stand-by mode. The RSU host information will also be supplied in the event, if available. |
| **250** | The RSU transmission rate that was below the operational threshold has been restored above the minimum. This likely denotes recovery from transmission error. The RSU host information will also be supplied in the event, if available. |
| **251** | The RSU transmission rate that was above the operational threshold has dropped below the minimum. This likely denotes some transmission error. The RSU host information will also be supplied in the event, if available. |
| 3XX | Some change occurred in the train detection system status. Note that this currently is only available if the HRI is equipped with IEEE-1570 connectivity. These are in order of priority, meaning only the lowest value (highest priority) notice will be issued. |
| **310** | The train detection system that was disconnected is now actively connected. |
| **311** | The train detection system that was actively connected is now disconnected. |

Azure Container App(s)

Another deployment option in the Microsoft Azure environment is to create a custom Docker container and let the cloud system manage its operation and resource consumption. For RCVW, the primary Azure Container App is used to expose the SQL database tables as a RESTful API for consumption by authorized subscribers. This custom container utilizes the Azure Data API Builder (DAB) base container but also incorporates a custom authentication mechanism to ensure only subscribers can use the API. The following is a description of the Azure container apps.

rcvw-cbs-rtcm-azure

This Azure Container App is used for forwarding RTCM messages from the CORS network to the Azure Service Bus at an HRI specific subscription of the j2735.rtcm topic. Just like for the HRI events, the message subject should be the target HRI. The RBS RSU Immediate Forward Plug-in must be configured in order to receive those messages and then forward them to the RSU. Because the CORS network is only accessible through local State DOT subscriptions, this App must be pre-configured for subscriptions in every State that an RSU is deployed. The currently supported States include Florida, Michigan, Missouri, and Ohio.

rcvw-cbs-api-azure

This Azure Container App is used as a quick way for subscribers to obtain details regarding the rail crossing itself. It is currently populated by the FRA database, with tables and fields that should match that schema. Built using a DAB container, however, allows for web-based queries of the data. For example, to list a specific HRI:

[https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings?$filter=HRI\_ID%20gt%200&uuid=<SUBSCRIBER\_ID](https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings?$filter=HRI_ID%20gt%200&uuid=%3cSUBSCRIBER_ID)>

Or, by FRA crossing identifier:

[https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings/CrossingID/867751N?uuid=<SUBSCRIBER\_ID](https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings/CrossingID/867751N?uuid=%3cSUBSCRIBER_ID)>

To list all the HRIs in a given state:

[https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings?$filter=StateName%20eq%20%27MI%27&uuid=<SUBSCRIBER\_ID](https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings?$filter=StateName%20eq%20%27MI%27&uuid=%3cSUBSCRIBER_ID)>

Or, within a certain latitude and longitude boundary:

[https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings?$filter=Latitude%20gt%2041.7%20and%20Latitude%20lt%2041.8%20and%20Longitude%20lt%20-83.5%20and%20Longitude%20gt%20-83.6&uuid=<SUBSCRIBER\_ID](https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api/Crossings?$filter=Latitude%20gt%2041.7%20and%20Latitude%20lt%2041.8%20and%20Longitude%20lt%20-83.5%20and%20Longitude%20gt%20-83.6&uuid=%3cSUBSCRIBER_ID)>

Note that a unique SUBSCRIBER\_ID must be obtained for each CBS subscriber application.

The full capability of the RESTful API is documented on the Azure website:

<https://learn.microsoft.com/en-us/azure/data-api-builder/rest>

Appendix A: CBS Resources

|  |  |  |
| --- | --- | --- |
| Resource Name | Resource Type | Description |
| rcvw-db-server | SQL Server | rcvw-db-server.database.windows.net |
| rcvw-db | SQL Database | Contains the tables for the CBS Data Log |
| rcvwcbssb | Service Bus Namespace | Holds the topics and subscriptions for the RCVW CBS Azure Service Bus |
| cbs.events | Topic | The topic for all outgoing HRI event messages. The cbs-outgoing subscription can be used to receive all HRI event messages, with the message subject indicating the HRI ID that the event pertains to. Additionally, a unique subscription within this topic will exist for each HRI ID an easy filter for events specific to that crossing. |
| cbs.scheduler | Topic | DO NOT USE This topic is used internally to trigger the event manager checks. |
| j2735.map | Topic | The rsu-incoming subscription holds all incoming J2735 MAP messages from each RBS. Used to trigger the cloud ingest operations. |
| j2735.spat | Topic | The rsu-incoming subscription holds all incoming J2735 SPaT messages from each RBS. Used to trigger the cloud ingest operations. |
| j2735.rtcm | Topic | The rsu-incoming subscription holds all incoming J2735 RTCM messages from each RBS. Used to trigger the cloud ingest operations. |
| tmx.message.v2x  .rtcm.sc10403\_3 | Topic | The topic for all outgoing RTCM 3.3 event messages. The cbs-outgoing subscription can be used to receive all RTCM messages, with the message subject indicating the HRI ID that the event pertains to. Additionally, a unique subscription within this topic will exist for each HRI ID an easy filter for events specific to that crossing. |
| tmx.plugin.hristatus.error | Topic | The rsu-incoming subscription holds all incoming error messages from each RBS HRI Status Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin.hristatus.status | Topic | The rsu-incoming subscription holds all incoming diagnostic and status messages from each RBS HRI Status Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin.map.error | Topic | The rsu-incoming subscription holds all incoming error messages from each RBS MAP Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin.map.status | Topic | The rsu-incoming subscription holds all incoming diagnostic and status messages from each MAP Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin .rsuimmediateforward.error | Topic | The rsu-incoming subscription holds all incoming error messages from each RBS RSU Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin .rsuimmediateforward.status | Topic | The rsu-incoming subscription holds all incoming diagnostic and status messages from each RBS RSU Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin.rtcm.error | Topic | The rsu-incoming subscription holds all incoming error messages from each RBS RSU Plug-in. Used to trigger the cloud ingest operations. |
| tmx.plugin.rtcm.status | Topic | The rsu-incoming subscription holds all incoming diagnostic and status messages from each RBS RSU Plug-in. Used to trigger the cloud ingest operations. |
| rcvw-cbs-api | Container App | The RESTful API server for the CBS Data Log. Uses Azure Data API Builder  <https://dab-container-app.redwave-53404dd1.eastus.azurecontainerapps.io/api> |
| rcvw-cbs-event-mgr | Function App | A function that detects and relays HRI events. It is triggered periodically when there is an HRI to manage. |
| rcvw-cbs-ingest | Function App | A function that writes incoming messages to the **rcvw-db** database. Note that this app is developed as a Function App but is actually deployed as a container since it contains custom C++ code used to decode the J2735 messages. |
| rcvw-cbs-rtcm | Container App | The proxy RTCM server for the RCVW RBS systems. Only connects when there is an HRI to manage. |
| rcvw-cbs-containers | Container App Environment | The common Azure environment to deploy the Container Apps to. |
| rcvw-cbs-functions | Function App Environment | The common Azure environment to deploy the Function Apps to. |

1. <https://github.com/usdot-fhwa-OPS/V2X-Hub> [↑](#footnote-ref-1)
2. TMX® works on NTCIP-compliant traffic signal controllers. [↑](#footnote-ref-2)
3. More detailed information regarding the HRI pre-emption signal detection options can be found in the RCVW Architecture and Design Document [↑](#footnote-ref-3)
4. More detailed information regarding the position correction options can be found in the RCVW Architecture and Design Document [↑](#footnote-ref-4)
5. The IEEE-1570 connectivity is optional for RCVW (see the requirements), but provides the ability to detect connectivity issues between the RBS and the train detection system. [↑](#footnote-ref-5)