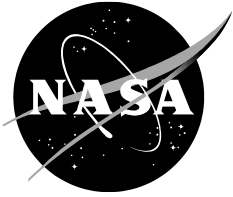


NASA/TM–20230009254



# Air Traffic Management TestBed: Non-Java Programming Language Support

*Chok Fung Lai*  
*Ames Research Center, Moffett Field, California*

*Jimmy Nguyen*  
*Optimal Synthesis, Inc., Moffett Field, California*

*Phu V. Huynh*  
*Universities Space Research Association, Moffett Field, California*

*Huu V. Huynh*  
*Crown Consulting, Inc., Moffett Field, California*

---

**June 2023**

## NASA STI Program ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

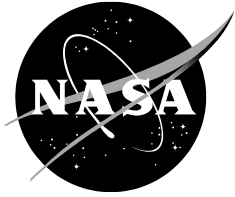
- **TECHNICAL PUBLICATION.** Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.
- **TECHNICAL MEMORANDUM.** Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- **CONTRACTOR REPORT.** Scientific and technical findings by NASA-sponsored contractors and grantees.
- **CONFERENCE PUBLICATION.** Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- **SPECIAL PUBLICATION.** Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- **TECHNICAL TRANSLATION.** English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at <http://www.sti.nasa.gov>
- E-mail your question to [help@sti.nasa.gov](mailto:help@sti.nasa.gov)
- Phone the NASA STI Information Desk at 757-864-9658
- Write to:  
NASA STI Information Desk  
Mail Stop 148  
NASA Langley Research Center  
Hampton, VA 23681-2199

NASA/TM–20230009254



# Air Traffic Management TestBed: Non-Java Programming Language Support

*Chok Fung Lai*  
*Ames Research Center, Moffett Field, California*

*Jimmy Nguyen*  
*Optimal Synthesis, Inc., Moffett Field, California*

*Phu V. Huynh*  
*Universities Space Research Association, Moffett Field, California*

*Huu V. Huynh*  
*Crown Consulting, Inc., Moffett Field, California*

National Aeronautics and  
Space Administration

*Ames Research Center*  
*Moffett Field, CA 94035-1000*

---

**June 2023**

## **Acknowledgments**

The authors thank Dr. Kee Palopo and Alan Lee for the Air Traffic Management TestBed project support. The authors would also like to thank Dr. Kee Palopo, Dr. Gano Chatterji, Dr. Heather Arneson, and Lindsay Stevens for reviewing this technical memorandum.

This report is available in electronic form at  
<http://ntrs.nasa.gov/>

## **Abstract**

The Air Traffic Management (ATM) TestBed provides a simple and easy capability to connect high-fidelity simulations for supporting National Aeronautics and Space Administration (NASA) and community research. Simulation components are connected to the TestBed via plugin adapters which can be publishers, subscribers, or both. Though the plugin adapters are written in Java programming language, connectivity between TestBed and non-Java applications are supported. This document describes procedures to access the TestBed data exchange messages using external applications such as MATLAB and web browsers, as well as non-Java programming language such as C, Python, and JavaScript. Example simulation layouts are presented. Step-by-step instructions to run adapters, and to connect to the external tools are also provided.

This page intentionally left blank.

## Table of Content

1. Introduction .....	7
2. TestBed Architecture .....	7
2.1. Communication Middleware.....	8
2.2. Plugin Adapters.....	8
2.3. Socket Adapters.....	9
3. Simulation Setup.....	10
4. Data Connectivity.....	13
4.1. MATLAB.....	14
4.2. Python .....	16
4.3. Socket.....	18
4.4. WebSocket.....	20
5. References.....	22
A. Appendix .....	23
A1. Building and Uploading Traffic Publisher to Local Repository .....	23

## List of Figures

Figure 2.1. Four-layer Architecture.....	8
Figure 2.2. Class Hierarchy of Socket Adapters .....	9
Figure 3.1. Creating a Simple Simulation Layout in Five Steps .....	11
Figure 3.2. Traffic Viewer.....	13
Figure 4.1. Message pass through: (a) Socket Adapter does not support; (b) WebSocket Adapter supports. ....	14
Figure 4.2. Using MATLAB Adapter .....	14
Figure 4.3. Simulation Outputs: (a) MATLAB Demonstration Program and (b) Traffic Viewer ...	16
Figure 4.4. Using Python Adapter .....	17
Figure 4.5. Showing Vehicle from Python Demonstration Program on Traffic Viewer.....	18
Figure 4.6. Using Socket Adapter.....	18
Figure 4.7. Showing Vehicle from C Demonstration Program on Traffic Viewer .....	19
Figure 4.8. Using WebSocket Adapter .....	20
Figure 4.9. Showing Vehicle via WebSocket on Traffic Viewer .....	21

## List of Listings

Listing 3.1. Running Simulation Architect from Console 1 .....	10
Listing 3.2. Running Apache ActiveMQ from Console 2 .....	12
Listing 3.3. Running Simulation Manager from Console 3.....	12
Listing 3.4. Running Component Manager from Console 4.....	12
Listing 3.5. Running Local Execution from Console 5 .....	13
Listing 4.1. Running Demonstration Program in MATLAB.....	15
Listing 4.2. Running Demonstration Program in Python from Console 6 .....	17
Listing 4.3. Building and Running Demonstration Program in C from Console 6.....	19
Listing 4.4. Running wscat from Console 6.....	20
Listing 4.5. Receiving Messages via WebSocket Using wscat.....	20
Listing 4.6. Publishing Messages via WebSocket Using wscat.....	21
Listing A.1. Extracting Gradle Build Tool Archive File.....	23
Listing A.2. Opening TrafficPub.java via Editor.....	24
Listing A.3. Modifying TrafficPub.java .....	24
Listing A.4. Building and Uploading Traffic Pub.....	24

## List of Tables

Table 3.1. Environment Variables .....	10
--	----



## 1. Introduction

Data connectivity is a critical factor during a simulation run. The Air Traffic Management (ATM) TestBed, or TestBed in short, provides the capability to connect high-fidelity simulations in an easy manner to support National Aeronautics and Space Administration (NASA) and community research [1]. Core functionalities are provided for ATM researchers by standardizing connectivity and data access between the TestBed and external applications. The TestBed supports data connectivity among simulation plugins (or in TestBed terminology, components)—including applications, assets, and data streams—using communication middleware. Components are connected to the TestBed via plugin adapters. Plugin adapters, or adapters in short, are currently written in Java programming language [2]. The main motivation for developing the plugin capability is to leverage existing assets, models, systems, and tools that have been developed using other programming languages and services [3, 4, 5, 6]: C [7] and C++ [8] have been used for decades for developing programs and applications. Python [9] has an extensive list of algorithms and libraries for data science and machine learning, which can be used for developing powerful applications. Researchers and teams have implemented many packages and functions using MATLAB [10]. Numerous web applications have been developed that use the WebSocket protocol [11, 12] for data exchange.

To exchange simulation data between the TestBed and external tools, one method is to create adapters to bridge for data connectivity, and the other method is to leverage existing socket adapters to provide data connectivity via standard sockets and protocols. The former method allows component sharing by hosting tool-specific adapters in the TestBed component repository since creating an adapter adds an abstraction layer on top of the underlying communication implementation. The latter method allows a quick way to prototype a tool without building an adapter, though the capability would be limited to the existing socket adapters allowing less control over the speed performance and network security. It is also possible to combine the two methods by creating adapters that extend existing socket adapters so that the simulation data can be shared with other components/applications and provide more control over the way in which communication takes place.

Section 2 provides background information for readers, including researchers, simulation users, and software developers, to understand the core TestBed architecture. It provides an overview of TestBed for those who are not familiar with TestBed. For a full description of TestBed, refer to [1]. Commands and steps are listed in Section 3 so that the readers can follow the instructions to configure and run simple simulations. Finally, Section 4 discusses access of simulation data from external, non-Java applications via existing socket adapters, with examples demonstrating steps to add noise to flight tracks using external/non-Java programs (MATLAB, Python, Socket, and WebSocket).

## 2. TestBed Architecture

The TestBed architecture consists of four layers: (1) application, (2) framework, (3) platform, and (4) infrastructure. These layers are depicted in Figure 2.1. The first two layers are application aware. Researchers run applications, such as simulation and visualization tools, in the application layer. Software and hardware developers design and implement functions and services for air traffic management operations in the framework layer. The latter two layers are application agnostic. Vendors provide communication middleware for data exchange in the platform layer. Finally, cloud service providers offer network interconnect capability in the infrastructure layer.

A benefit of using this architecture design is flexibility to switch any component from one provider to another without affecting the other components. The design anticipates that more advanced, powerful hardware and tools will be developed and will be available in the future. For example, the existing communication middleware could be replaced with a better one with lower messaging latency and higher throughput performance, without affecting the functions and services running in the framework and application layers.

In Figure 2.1, the horizontal purple dashed line indicates the separation of roles. The components running in the layers above the dashed line will be created and maintained by the researchers and developers; the components running in the layers below the dashed line will be developed and maintained by the TestBed team. The separation of roles allows researchers to focus on their goals and needs, with minimum effort on the functionality and capability support provided by the other layers.

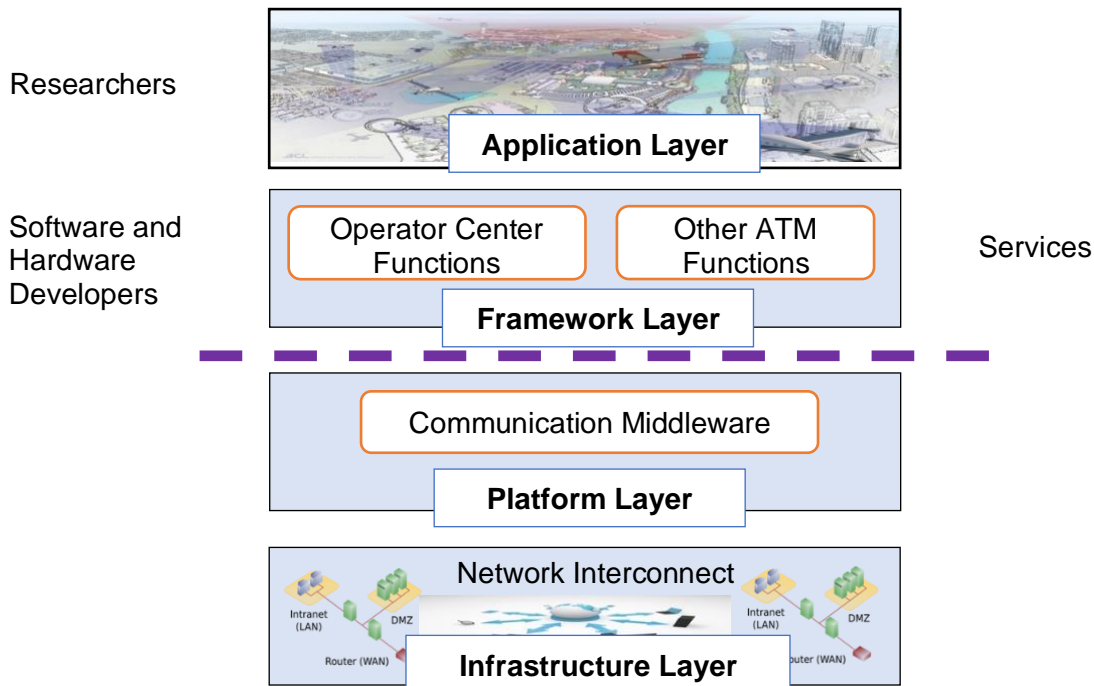


Figure 2.1. Four-layer Architecture

## 2.1. Communication Middleware

The communication middleware running in the platform layer provides a means to exchange data among simulation components. Communication middleware currently supports Apache ActiveMQ [13], Apache Kafka [14], Real-Time Innovations' Connxt [15], and Neural Autonomic Transport System [16] messaging middleware systems. TestBed provides an abstraction layer called *Messaging Support* to facilitate the data exchange. The Messaging Support is a core feature that allows switching from one messaging middleware to another without a need to rebuild the simulation components. One of the TestBed's design goals is to ensure the messaging support is flexible and extensible. The design assumes that better middleware solutions will be available in the future and existing tools should not need to be modified to benefit from them.

## 2.2. Plugin Adapters

In the TestBed environment, a plugin adapter represents a bridge between a simulation component and the communication middleware. A plugin adapter can be either a publisher, a subscriber, or both. Examples include data feeds, message filters, and visualization tools. The main functions of plugin adapters include subscribing messages from their publishers, processing information, and publishing messages to their subscribers. In addition, each plugin adapter has capabilities to record incoming and outgoing messages in a simulation run, and then play back the recorded messages in other runs. Though plugin adapters are written in Java programming language, the simulation component can be any asset, application, or data stream

regardless of the programming language it is written in and the operating system it runs on. A role of the plugin adapter is to convert messages between the TestBed Data Exchange Model [17] and application-specific format so that components are communicated using a standard data format.

### 2.3. Socket Adapters

In addition to plugin adapters, external assets or applications may communicate with the TestBed via socket connections. These external applications may be written in any programming languages such as Java, C/C++, Python, and MATLAB, provided that standard socket is supported. The TestBed provides four specific socket adapter implementations. The class hierarchy among these adapters is shown in Figure 2.2, where the up arrow (↑) indicates extension. Each of them targets a specific group such as researchers, algorithm developers, and simulation engineers:

1. MATLAB Adapter (see Section 4.1) is mainly for researchers using MATLAB.
2. Python Adapter (see Section 4.2) is created for Python developers.
3. Socket Adapter (see Section 4.3) is developed for programming languages and software applications with standard socket support.
4. WebSocket Adapter (see Section 4.4) is implemented for web connectivity.

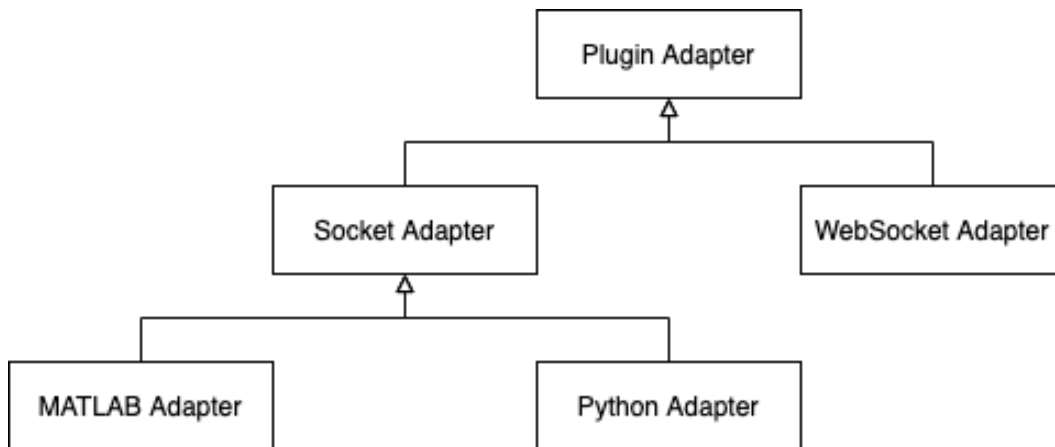


Figure 2.2. Class Hierarchy of Socket Adapters

The Socket Adapter and its subclasses, MATLAB Adapter and Python Adapter, provide connectivity between external applications and the messaging layer via socket connections. During simulation execution, each of these adapters, if used, starts a socket server that sends messages to and receives messages from the external applications. Thus, the external applications are considered clients to the socket adapters.

Each client utilizes the TestBed Socket Client Library to connect to the socket server associated with the socket adapter. The client library provides application programming interfaces for making connection to the socket server, as well as for receiving messages from and sending messages to the socket server. External application developers will use the client library to communicate with the TestBed. Depending on the version of the Java programming language, the client library is in a corresponding directory. For the current TestBed implementation, it is in one of these directories (in Linux's path format):

- Java 8: \$SNTB\_HOME/TestBedPlugins/SocketAdapter/SocketClientJava8/
- Java 11: \$SNTB\_HOME/TestBedPlugins/SocketAdapter/SocketClient/

Note that the design of the current socket adapters for non-Java programming languages, algorithms, and tools is an initial one. The design is expected to evolve over time. At its current state, whenever a socket adapter starts a socket server, it opens a connection to any socket

client within the network. Thus, other socket clients can connect to the socket server. Due to the current design, the socket adapters must be used with caution. Before running a socket adapter, make sure a secure network is used, firewall is enabled, and the exchanged data are non-sensitive. The socket adapters are useful for local simulation executions due to lack of security support. Future versions will include secure socket connections using a token-based authentication, data encryption, and the Transport Layer Security [18].

### 3. Simulation Setup

Running a TestBed simulation requires setting up a simulation layout. This section briefly describes how to set up a simple simulation using the TestBed Simulation Architect, a graphical user interface tool for composing simulations using blocks representing components and links representing message channels [19].

*Table 3.1. Environment Variables*

Variable Name	Description	Example Value
<b>ACTIVEMQ_HOME</b>	Home directory of the Apache ActiveMQ server.	/Users/xxx/apache-activemq-5.15.8/
<b>SNTB_HOME</b>	Home directory of the TestBed Software Development Kit.	/Users/xxx/testbed-sdk/smart-nas/

Table 3.1 lists the required environment variables for running TestBed simulations. This document assumes both the Apache ActiveMQ and the TestBed Software Development Kit with socket adapters are available<sup>1</sup>. Listing 3.1 shows the commands to run the Simulation Architect on Linux, macOS, and Windows platforms. The commands assume the environment variable SNTB\_HOME is properly set.

*Listing 3.1. Running Simulation Architect from Console 1*

Platform	Command
Linux, macOS	\$ cd \$SNTB_HOME/TestBedCore/SimulationArchitect/ \$ ./bin/run_sim_architect.sh
Windows	> cd %SNTB_HOME%\TestBedCore\SimulationArchitect\ > .\bin\run_sim_architect.bat

After executing the commands, an application frame titled “Sim1 – Simulation – SNTB Architect” will be shown. A layout of simulation involving two components is presented:

1. *Traffic Pub*: a simple publisher that publishes a track message every second.
2. *Traffic Viewer* [20]: a two-dimensional visualization tool for showing location of the vehicle defined in the published track message.

---

<sup>1</sup> Please contact sn-testbed-developers@lists.nasa.gov for software support.

Follow these five steps to create the simulation layout as shown in Figure 3.1:

1. Press  $\text{⌘}$  <F> (Linux and macOS) or <Ctrl> <F> (Windows) shortcut key to show the *Find Blocks* bar located at the top of the palette pane.
2. Enter the term “traffic” (without quotes) to find traffic-related blocks.
3. Drag and drop the *Traffic Pub* block from the palette pane to the editor workspace.
4. Drag and drop the *Traffic Viewer* block from the palette pane to the editor workspace.
5. Drag and drop the output arrow (→) of the Traffic Pub block to the Traffic Viewer block. The directed link represents the message flow between the publisher and the viewer, which is the subscriber.

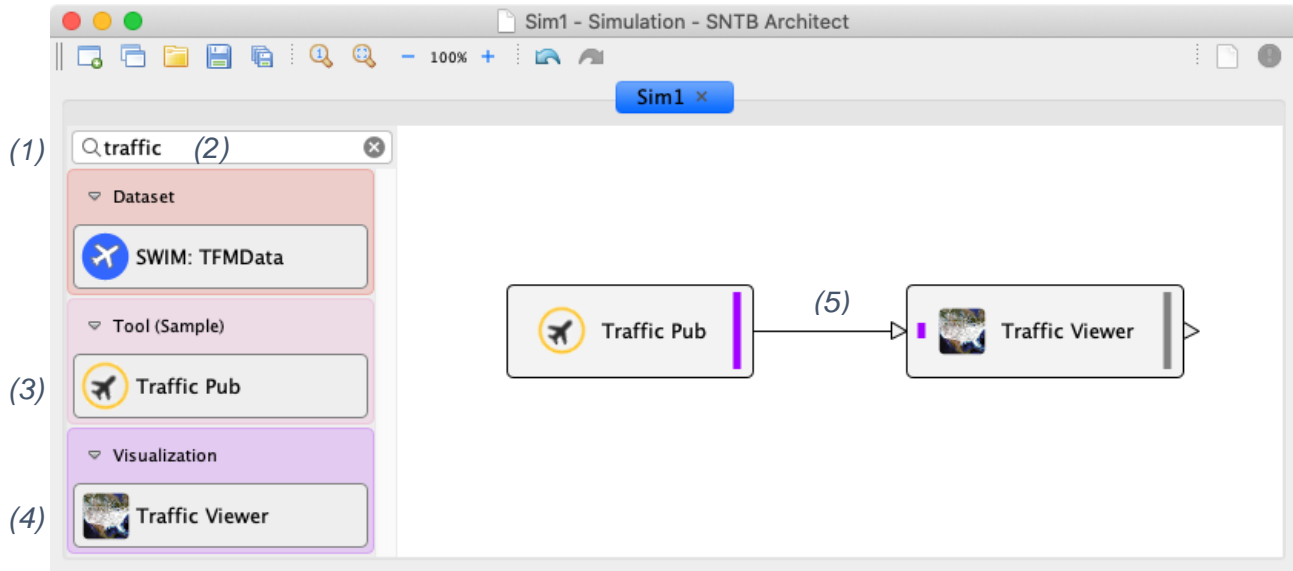


Figure 3.1. Creating a Simple Simulation Layout in Five Steps

After composing the simulation, either click on the Save button on the toolbar, or press  $\text{⌘}$  <S> (Linux and macOS) or <Ctrl> <S> (Windows) shortcut key, to save the configuration to a file. This document assumes that the default file name `Sim1.sim` is used. Other file names may be used.

To execute a local simulation, applications and managers are started on six separate consoles<sup>2</sup> for clarity. The following console numbers are also mentioned in the listing captions.

Console 1: Simulation Architect for composing the simulation.

Console 2: Apache ActiveMQ server for exchanging simulation data messages.

Console 3: Simulation Manager for managing simulations.

Console 4: Component Manager for managing plugin adapters.

Console 5: Local Execution for launching simulations on local network.

Console 6: External application simulation data access.

Note that this document focuses on the usage of non-Java programming language support and assumes the readers have experience building and uploading the Traffic Publisher component to a local repository. Please refer to Appendix A1 for the steps to modify, build, and upload the adapter.

<sup>2</sup> A console represents a Terminal tab (Linux and macOS) or a Command Prompt application (Windows).

*Listing 3.2. Running Apache ActiveMQ from Console 2*

Platform	Command
Linux, macOS	<pre>\$ cd \$ACTIVEMQ_HOME/bin/ \$ ./activemq console</pre>
Windows	<pre>&gt; cd %ACTIVEMQ_HOME%\bin\ &gt; .\activemq.bat start</pre>

Listing 3.2 lists the commands to start the ActiveMQ server. These commands assume the environment variables `ACTIVEMQ_HOME` is properly set. The ActiveMQ server is one of the messaging middleware supported by the TestBed. A Simulation Manager manages the execution of the simulation; it can be started by following the commands listed in Listing 3.3. When the manager starts successfully, the console shows the TestBed core library version, system environment version, host key of the manager, and a prompt message “Press `<ctrl>` `<c>` to quit.”

*Listing 3.3. Running Simulation Manager from Console 3*

Platform	Command
Linux, macOS	<pre>\$ cd \$SNTB_HOME/TestBedCore/SimulationManager/ \$ ./bin/run_sim_manager.sh</pre>
Windows	<pre>&gt; cd %SNTB_HOME%\TestBedCore\SimulationManager\ &gt; .\bin\run_sim_manager.bat</pre>

A Component Manager manages plugin adapters to be run on an assigned host key, the default value of which is “localhost.” In this simulation setup, since all the adapters are assigned “localhost” host key value, a single Component Manager is needed. Listing 3.4 lists the commands to start a Component Manager. On successful startup, the console shows the versions, the host key, and the prompt message.



*Listing 3.4. Running Component Manager from Console 4*

Platform	Command
Linux, macOS	<pre>\$ cd \$SNTB_HOME/TestBedCore/ComponentManager/ \$ ./bin/run_component_manager.sh</pre>
Windows	<pre>&gt; cd %SNTB_HOME%\TestBedCore\ComponentManager\ &gt; .\bin\run_component_manager.bat</pre>

A local execution command can be used to start the simulation. The commands are listed in Listing 3.5. Note that if a non-default file name (“Sim1.sim”) is used when saving the configuration in the Simulation Architect, the `blueprint` parameter value needs to be updated. The configuration information stored in the blueprint file is sent to the Simulation Manager. The Simulation Manager then sends commands to the Component Manager to deploy, start up, initialize, and execute the plugin adapters.

Listing 3.5. Running Local Execution from Console 5

Platform	Commands
Linux, macOS	<pre>\$ cd \$SNTB_HOME/TestBedCore/ExecutionManager/ \$ ./bin/run_local_execution.sh \   -blueprint ../SimulationArchitect/Sim1.sim</pre>
Windows	<pre>&gt; cd %SNTB_HOME%\TestBedCore\ExecutionManager\ &gt; .\bin\run_local_execution.bat ^   -blueprint ..\SimulationArchitect\Sim1.sim</pre>

If everything is set up correctly, a Traffic Viewer visualization tool will be displayed as shown in Figure 3.2, and a single yellow track icon will be shown. The track icon will move from the San Francisco International Airport towards the north-east direction. Enabling the  Flight Path Layer on the toolbar will show the path flown by the aircraft in yellow; enabling the  Flight Tag Layer will show the tag information such as callsign, altitude, speed, and heading, of the aircraft.

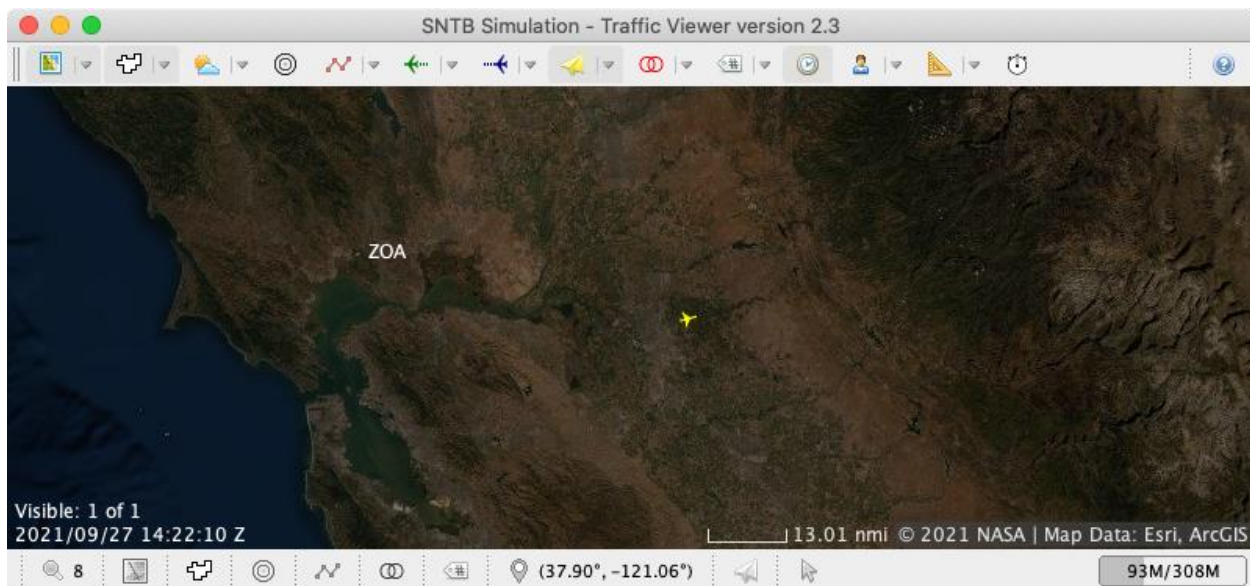


Figure 3.2. Traffic Viewer

To stop the simulation, press `<ctrl> <c>` keys on the Execution Manager console (Console 5) and a stop command will be sent to the Simulation Manager. The Simulation Manager will then send the commands to the Component Manager to shut down, archive, and retire the plugin adapters. A new simulation can be started using the Execution Manager console. When there is no simulation to run, the Component Manager, the Simulation Manager, and the ActiveMQ server may also be stopped in this specific order by pressing `<ctrl> <c>` keys on their consoles.

## 4. Data Connectivity

Simulations using external applications and non-Java programming language are explained using the simple simulation setup described in Section 3. This section describes the insertion of a new adapter between the Traffic Publisher (publisher) and the Traffic Viewer (subscriber). The new adapter is designed to act both as a subscriber and as a publisher. The following new adapters, presented in the subsections, can be found in the following project directories (in Linux's path format):

- \$SNTB\_HOME/TestBedPlugins/SocketAdapter/MatlabAdapter/
- \$SNTB\_HOME/TestBedPlugins/SocketAdapter/PythonAdapter/
- \$SNTB\_HOME/TestBedPlugins/SocketAdapter/SocketAdapter/
- \$SNTB\_HOME/TestBedPlugins/WebAppSupport/WebSocket/WebSocketServerAdapter/

An adapter supports a message pass through feature if all the incoming messages, M1, are also published to the subscribers. The Socket Adapter does not support message pass through (See Figure 4.1(a)), hence both the MATLAB adapter and the Python adapter inherit the feature and they do not support message pass through. The Socket Adapter needs to publish its own messages, M2. On the other hand, the WebSocket Adapter supports message pass through because the WebSocket application clients, such as microservice endpoints, may not be available in the simulation configuration (See Figure 4.1(b)).

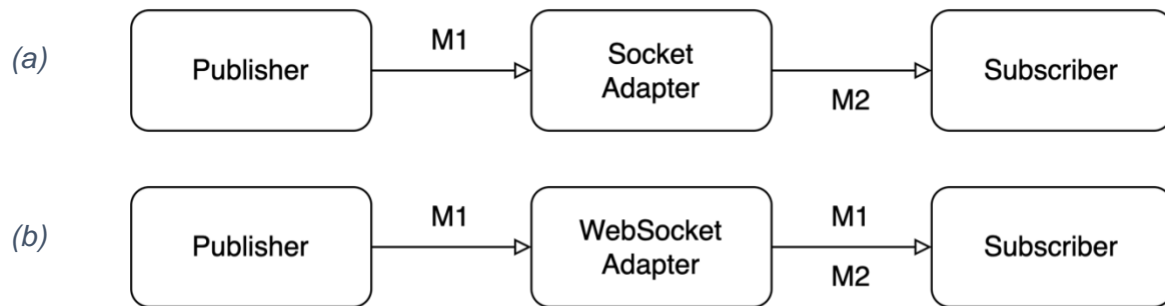


Figure 4.1. Message pass through: (a) Socket Adapter does not support; (b) WebSocket Adapter supports.

#### 4.1. MATLAB

MATLAB is a programming and a numeric computing platform developed by MathWorks. It enables matrix algebra, implementation of algorithms, plotting, user interface development and connecting to programs written in other languages. It is used widely in academia, government, and industry. The *MATLAB Adapter* was added to the Component Library in the TestBed Build 2.3 to allow connectivity between TestBed and MATLAB.

Figure 4.2 shows a simulation configuration using three components: Traffic Pub, MATLAB Adapter, and Traffic Viewer. In this setup, the MATLAB Adapter receives the track messages from the publisher/Traffic Pub, modifies the messages, and publishes the modified messages to the subscriber/Traffic Viewer. By default, the adapter uses the port number 50001 for data exchange. The port number can be configured via the “Edit Properties” dialog by double-clicking the MATLAB block in the Simulation Architect layout and selecting the “User” property group.

Incoming messages to the MATLAB Adapter do not pass through the adapter. Thus, MATLAB code needs to publish the incoming messages or custom messages to subscribers by calling the `TestBedSocketClient.publish(Sndem)` method. If the message pass through feature is needed, a directed link can also be created from the publisher (e.g., the Traffic Pub) to the subscribers (e.g., the Traffic Viewer).

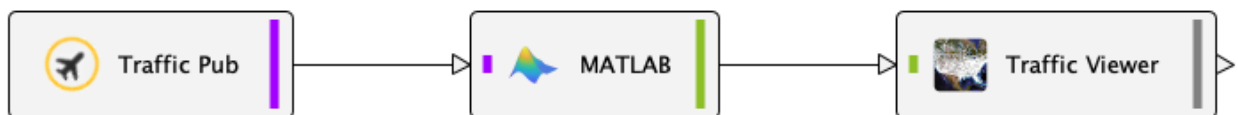


Figure 4.2. Using MATLAB Adapter





After the simulation starts, launch the MATLAB application on the machine that runs the MATLAB Adapter. In the Command Window, run the three commands listed in Listing 4.1. Make sure to replace the macro `$SNTB_HOME`<sup>3</sup> with the user's specific path because MATLAB does not accept environment variables. Also note that MATLAB supports Java 11 since R2023a [21]. The TestBed socket client library built with Java version 8 should be used.

*Listing 4.1. Running Demonstration Program in MATLAB*

Step	Command
1	% Add Java Path >> javaaddpath('\$SNTB_HOME/TestBedPlugins/SocketAdapter/SocketClientJava8/lib/TestBedSocketClient_j8.jar')
2	% Change Working Directory >> cd '\$SNTB_HOME/TestBedPlugins/SocketAdapter/MatlabAdapter/demo/'
3	% Run Demonstration Program >> TestBedSocketClientDemo

Step 1 adds the Java path. Step 2 changes the current working directory to the folder named “demo.” Step 3 starts the MATLAB demonstration program. This program performs the following operations:

1. Receives the track messages published by the Traffic Publisher.
2. Plots the ground speed profile of the track on a separate graphical window.
3. Modifies the vehicle identifier (callsign) of the track from “NASA123” to “MAT456.”
4. Adds noise to the track’s geo-location data.
5. Publishes the noisy track message to the Traffic Viewer via the TestBed.

After starting the demonstration program, press the “Start” button to start the data collection and noise generation. Figure 4.3(a) shows the graphical window of the track’s ground speed profile. The Traffic Viewer receives the noisy track messages, and the path flown is shown in Figure 4.3(b). Note that both  Flight Path Layer and  Flight Tag Layer buttons on the Traffic Viewer toolbar are enabled. The source code of the program provides an example for the readers to learn and understand modification of incoming messages and subsequent publication of the modified messages.

---

<sup>3</sup> All the macros are marked in blue throughout this document. Please replace the macro with its actual environment variable value.

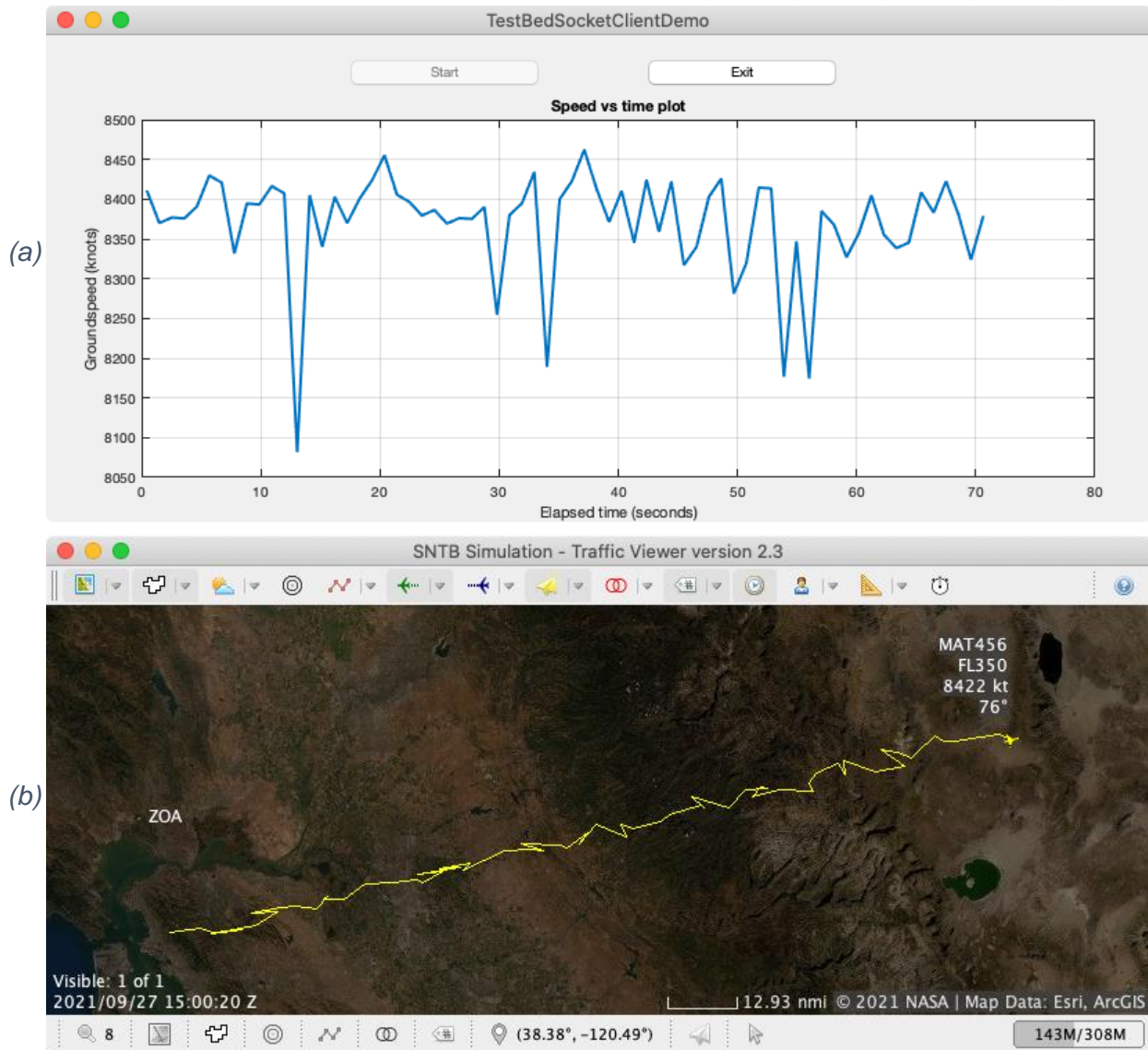


Figure 4.3. Simulation Outputs: (a) MATLAB Demonstration Program and (b) Traffic Viewer

To stop the MATLAB demonstration program, press the “Exit” button on the graphical window. To stop the simulation, press `<ctrl> <c>` keys on the Execution Manager console (Console 5).

## 4.2. Python

Python programming language is popular with data science engineers. It provides an extensive list of modules for numerical analysis and machine learning. To support Python and the TestBed connectivity, the *Python Adapter* was added to the Component Library in the TestBed Build 2.2a.

Figure 4.4 shows the simulation configuration using three components: Traffic Pub, Python Adapter, and Traffic Viewer. In this setup, the Python Adapter receives the track messages from the publisher, modifies the messages, and publishes the modified messages to the subscriber. Like the other socket adapter implementations, incoming messages do not pass through the Python adapter. To publish the messages or any custom messages to the subscribers, the `TestBedSocketClient.publish(Sndem)` method needs to be called from the Python program. If the message passing through between a publisher and a subscriber is needed, a direct link may be created from the publisher to the subscribers. When running the simulation with the

proposed setup, the Traffic Viewer does not show any vehicle track on the display because the Python Adapter does not pass through the incoming messages to its subscriber.



Figure 4.4. Using Python Adapter

By default, the Python Adapter uses the port number 50 002 for data exchange. The port number can be configured via the “Edit Properties” dialog by double-clicking the Python block and selecting the “User” property group. Run the Python demonstration program to receive and publish track messages following the commands listed in Listing 4.2. The program uses the JPyype1 module [22] to access Java from within Python. The program performs the following five operations:

1. Receives the track messages published by the Traffic Publisher.
2. Modifies the vehicle identifier (callsign) of the track from “NASA123” to “PYT123.”
3. Reverses the vehicle’s heading by adding 180 degrees.
4. Shifts the vehicle location by a three-degree latitude.
5. Publishes the modified track message to the Traffic Viewer via the TestBed.

Listing 4.2. Running Demonstration Program in Python from Console 6

Platform	Command
Linux, macOS	<pre>\$ cd \$SNTB_HOME/TestBedPlugins/SocketAdapter/PythonAdapter/demo/ \$ python3 -m pip install JPyype1 \$ python3 tb_socket_client_demo.py</pre>
Windows	<pre>&gt; cd %SNTB_HOME%\TestBedPlugins\SocketAdapter\PythonAdapter\demo\ &gt; python3 -m pip install JPyype1 &gt; python3 tb_socket_client_demo.py</pre>

After starting the demonstration program, the Traffic Viewer will show the vehicle state from the Python modified track messages. Note that the path flown of the vehicle is shifted towards the North and the heading is reversed.

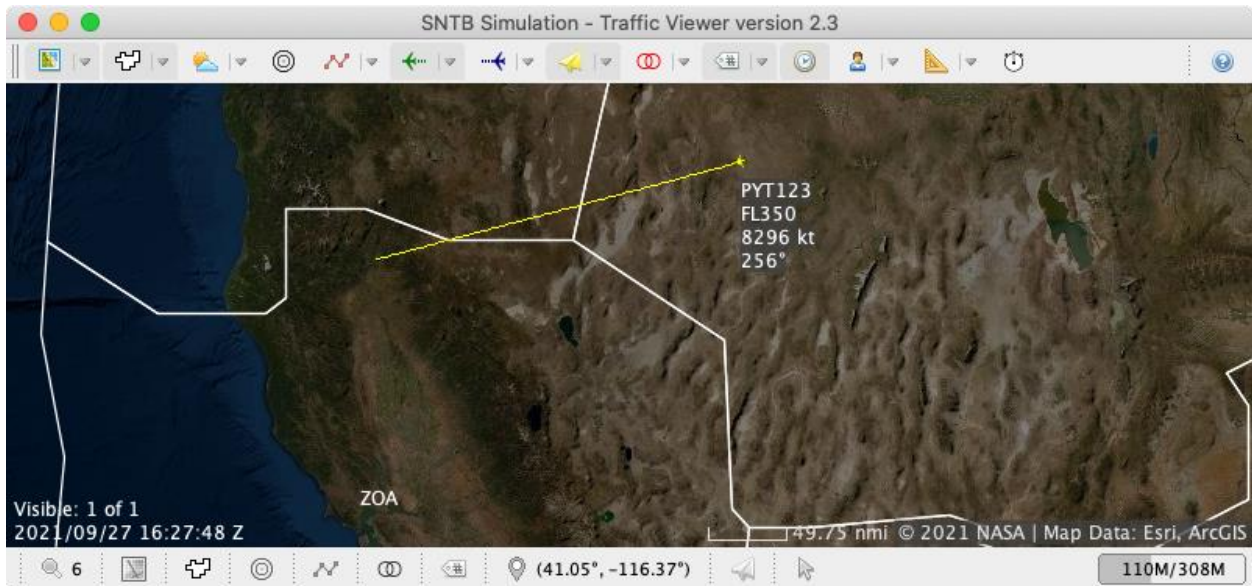


Figure 4.5. Showing Vehicle from Python Demonstration Program on Traffic Viewer

This section described the capability to connect Python application program and the TestBed using the Python Adapter. To stop the Python demonstration program, press `<ctrl> <c>` on the Python console (Console 6). To stop the simulation, press `<ctrl> <c>` keys on the Execution Manager console (Console 5).

### 4.3. Socket

To support connectivity between the TestBed and external application programs, written in different programming languages, via Transmission Control Protocol (TCP) [23] network sockets, the *Socket Adapter* was added to the Component Library in the TestBed Build 2.3. A Socket Adapter is a specific plugin adapter that provides connectivity between an external application and the TestBed communication middleware via the socket connection. Like both the MATLAB and Python Adapters discussed earlier, the Socket Adapter does not pass through the incoming messages to the subscribers. To do so, either the `TestBedSocketClient.publish(Sndem)` method needs to be called or a direct link between the publisher and subscriber needs to be created.

In this section, the usage of the Socket Adapter is discussed for interacting with a C demonstration program running on macOS. The build commands for the example presented here may be modified for the other operating systems such as Linux and Microsoft Windows. Figure 4.6 depicts a simulation configuration with the connection between the publisher, Traffic Pub, and the subscriber, Traffic Viewer, via the Socket Adapter. In this setup, the Socket Adapter receives the track messages from the publisher, logs the message content, and then publishes the messages to the subscriber.



Figure 4.6. Using Socket Adapter

The configurable port number 50 000 is used for data exchange in the Socket Adapter. The port number can be configured via the “Edit Properties” dialog by double-clicking the Socket block and selecting the “User” properties group.

After the simulation starts, the Traffic Viewer does not show the vehicle icon because the Socket Adapter does not directly pass through any incoming messages to the Traffic Viewer. Follow the commands listed in Listing 4.3 to build the executable application of the C program on the machine that runs the Socket Adapter. Note that the commands provided are for the macOS platform. Readers need to modify the compiler parameters to build the executable application on Linux; a compiler such as Microsoft Visual C++ [24] can be used for Windows.

Listing 4.3. Building and Running Demonstration Program in C from Console 6

Platform	Command
macOS	<pre>\$ cd \$SNTB_HOME/TestBedPlugins/SocketAdapter/SocketAdapter/demo/ \$ gcc -I\$JAVA_HOME/include/ \       -I\$JAVA_HOME/include/darwin/ \       -L\$JAVA_HOME/lib/server \       -ljvm \       -Wl,-rpath,\$JAVA_HOME/lib/server \       tb_socket_client_demo.c \       -o tb_socket_client_demo.out \$ ./tb_socket_client_demo.out</pre>

The example program discussed here performs the following operations:

1. Receives an incoming message published by the Traffic Publisher.
2. Logs the JavaScript Object Notation (JSON) [28] string of the received message.
3. Publishes the message to the Traffic Viewer via the TestBed.

After running the simulation, the Traffic Viewer displays the vehicle state originating from the Traffic Publisher. To stop the demonstration program, press <ctrl> <c> keys on the C console (Console 6). To stop the simulation, press <ctrl> <c> keys on the Execution Manager console (Console 5).

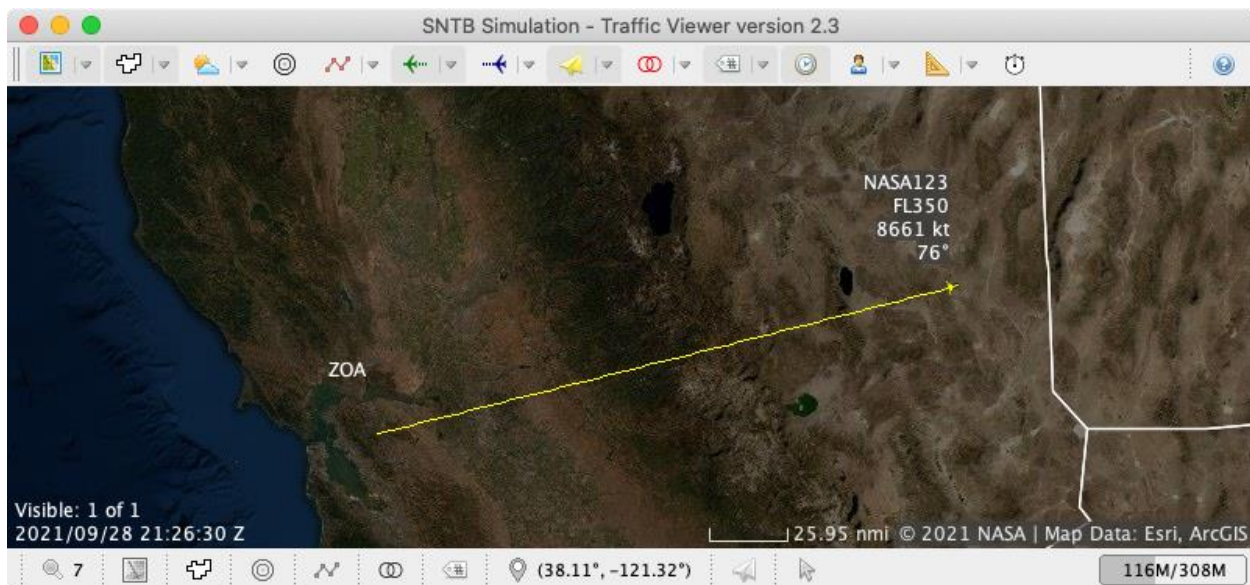


Figure 4.7. Showing Vehicle from C Demonstration Program on Traffic Viewer

Although connectivity to the TestBed from a C program was described here, the Socket Adapter can be used to connect to programs written in any programming language that supports the standard socket.

## 4.4. WebSocket

The WebSocket protocol is widely supported by the web browsers. The *WebSocket Adapter* was added to the Component Library in the TestBed Build 1.5a. This adapter has been used for connecting with the Boeing Ground Station Console during live flight tests of the Boeing B777 ecoDemonstrator in 2018 [25] for conducting a system check.



Figure 4.8. Using WebSocket Adapter

Figure 4.8 shows the simulation configuration demonstrating the connection between the publisher, Traffic Pub, and the subscriber, Traffic Viewer, via a WebSocket server. Unlike the other socket adapters discussed above, incoming messages to the WebSocket Adapter pass through to the subscribers. This is because a TestBed adapter of the WebSocket application such as a microservice endpoint may not be created. A simulation engineer would not be able to connect such the WebSocket application to the subscriber using the Simulation Architect tool.

For the WebSocket Adapter, the Uniform Resource Locator [26] for receiving and publishing messages has the format

```
wss://<host>:<port>/testbed/sndem/<type>
```

where <host> represents the hostname (default value is “localhost”) of the machine running the adapter, <port> represents the port number (default value is 1234) defined in the adapter properties, and <type> represents the simple class name of the data exchange model message.

Listing 4.4. Running wscat from Console 6

Platform	Command
Linux, macOS	\$ wscat -nc wss://localhost:1234/testbed/sndem/track
Windows	> wscat -nc wss://localhost:1234/testbed/sndem/track

Listing 4.4 lists the commands to receive track messages from the publishers via the WebSocket using a tool called *WebSocket cat (wscat)* [27]. On running the commands, incoming track messages are shown on the console as a JSON string (see Listing 4.5).

Listing 4.5. Receiving Messages via WebSocket Using wscat

```
$ wscat -nc wss://localhost:1234/testbed/sndem/track
Connected (press CTRL+C to quit)
< {"vid":"NASA123","latDeg":38.159935,"lonDeg":-120.46988,"altFt":35000.0,"time":1632940741223,"gsKt":8725.599,"crsDeg":75.717545,"vsFpm":0.0,"meta":{"src":"Traffic Pub.0","ver":"2.3","tpub":1632940741224,"tsub":1632940741224}}
< {"vid":"NASA123","latDeg":38.169933,"lonDeg":-120.419876,"altFt":35000.0,"time":1632940742227,"gsKt":8733.165,"crsDeg":75.71567,"vsFpm":0.0,"meta":{"src":"Traffic Pub.0","ver":"2.3","tpub":1632940742230,"tsub":1632940742230}}
< {"vid":"NASA123","latDeg":38.17993,"lonDeg":-120.36987,"altFt":35000.0,"time":1632940743234,"gsKt":8706.025,"crsDeg":75.71378,"vsFpm":0.0,"meta":{"src":"Traffic Pub.0","ver":"2.3","tpub":1632940743236,"tsub":1632940743236}}
```

The wscat tool may also be used to publish messages to the subscribers via the WebSocket. For example, copy the first JSON message from the output shown in Listing 4.5, modify the callsign NASA123 to NASA456, copy and paste the updated JSON message to the wscat console, and then press <enter> key. These steps are shown in Listing 4.6. The published message is indicated in black.

Listing 4.6. Publishing Messages via WebSocket Using wscat

```
$ wscat -nc wss://localhost:1234/testbed/sndem/track
Connected (press CTRL+C to quit)
< {"vid":"NASA123","latDeg":38.159935,"lonDeg":-120.46988,"altFt":35000.0,"
time":1632940741223,"gsKt":8725.599,"crsDeg":75.717545,"vsFpm":0.0,"meta":{"
src":"Traffic Pub.0","ver":"2.3","tpub":1632940741224,"tsub":1632940741224
}}
> {"vid":"NASA456","latDeg":38.159935,"lonDeg":-120.46988,"altFt":35000.0,"
time":1632940741223,"gsKt":8725.599,"crsDeg":75.717545,"vsFpm":0.0,"meta":{"
src":"Traffic Pub.0","ver":"2.3","tpub":1632940741224,"tsub":1632940741224
}}
```

After running the simulation, the Traffic Viewer shows both the vehicle state originating from the Traffic Publisher (callsign NASA123) as well as from the WebSocket Adapter (callsign NASA456). Note that the vehicle with callsign NASA456 stays at the same location because there are no more messages published to the adapter. To stop the wscat session, press <ctrl> <c> keys on its console (Console 6). To stop the simulation, press <ctrl> <c> keys on the Execution Manager console (Console 5).



Figure 4.9. Showing Vehicle via WebSocket on Traffic Viewer

This section discussed connectivity to the TestBed from the WebSocket Adapter using wscat. The adapter can also be accessed by application programs written in any programming language that supports the WebSocket application programming interface.

## 5. References

1. Robinson, J. E., Lee, A., and Lai, C. F., "Development of a High-Fidelity Simulation Environment for Shadow-Mode Assessments of Air Traffic Concepts," *Royal Aeronautical Society: Modeling and Simulation in Air Traffic Management Conference*, London, UK, 14-15 November 2017.
2. "Java SE | Oracle Technology Network | Oracle" (Online), <https://www.oracle.com/java/technologies/java-se-glance.html>. Oracle. Retrieved 2023/06/06.
3. Roychoudhury, I., Daigle, M., Goebel, K., Spirkovska, L., Sankararaman, S., Ossenfort, J., Kulkarni, C., McDermott, W., and Poll, S., "Initial Demonstration of the Real-Time Safety Monitoring Framework for the National Airspace System Using Flight Data," AIAA 2016-4216, *16<sup>th</sup> AIAA Aviation Technology, Integration, and Operations Conference*, Washington, D.C., 13-17 June 2016.
4. Putnam, J. and Littell, J., "Evaluation of Impact Energy Attenuators and Composite Material Designs of a UAM VTOL Concept Vehicle," *Vertical Flight Society Annual Forum and Technology Display*, Philadelphia, PA, May 13, 2019.
5. Rios, J. L., Smith, I. S., Venkatesen, P., Homola, J. R., Johnson, M. A., and Jung, J., "UAS Service Supplier Specification: Baseline requirements for providing USS services within the UAS Traffic Management System," NASA/TM-2019-220376, NASA Technical Memorandum, October 1, 2019.
6. Coppenbarger, R. A., Aweiss, A., and Erzberger, H., "Flight Demonstration of the Tailored Arrival Manager," *AIAA Aviation 2021 Forum*, Virtual Event, August 2-6, 2021.
7. "ISO - ISO/IEC 9899:2018 - Information technology — Programming languages — C" (Online), <https://www.iso.org/standard/74528.html>. International Organization for Standardization. Retrieved 2023/06/06.
8. "ISO - ISO/IEC 14882:2020 - Programming languages — C++" (Online), <https://www.iso.org/standard/79358.html>. International Organization for Standardization. Retrieved 2023/06/06.
9. "Welcome to Python.org" (Online), <https://www.python.org/>. Python Software Foundation. Retrieved 2023/06/06.
10. "MATLAB - MathWorks - MATLAB & Simulink" (Online), <https://www.mathworks.com/products/matlab.html>. MathWorks. Retrieved 2023/06/06.
11. "The WebSocket Protocol" (Online), <https://datatracker.ietf.org/doc/html/rfc6455>. Internet Engineering Task Force. Retrieved 2023/06/06.
12. "'websocket' | Can I use... Support tables for HTML5, CSS3, etc" (Online), <https://caniuse.com/?search=websocket>. Retrieved 2023/06/06.
13. "ActiveMQ" (Online), <https://activemq.apache.org/>. The Apache Software Foundation. Retrieved 2023/06/06.
14. "Apache Kafka" (Online), <https://kafka.apache.org/>. The Apache Software Foundation. Retrieved 2023/06/06.
15. "Connxt: Connectivity Framework | RTI" (Online), <https://www.rti.com/products>. Retrieved 2023/06/06.
16. "NATS.io – Cloud Native, Open Source, High-performance Messaging" (Online), <https://nats.io/>. Cloud Native Computing Foundation. Retrieved 2023/06/06.
17. Lai, C. F., "Air Traffic Management TestBed Data Exchange Model," NASA/TM-20205001252, NASA Technical Memorandum, May 1, 2020.
18. Rescorla, E., "RFC 8446 – The Transport Layer Security (TLS) Protocol Version 1.3" (Online), <https://datatracker.ietf.org/doc/html/rfc8446>. Internet Engineering Task Force. Retrieved 2023/06/06.
19. Lai, C. F., "Air Traffic Management TestBed Simulation Architect: User's Guide," NASA/TM-2019-220196, NASA Technical Memorandum, April 1, 2019.



20. Lai, C. F., “Air Traffic Management TestBed Traffic Viewer: Developer’s Guide,” NASA/TM-2020-220511, NASA Technical Memorandum, March 1, 2020.
21. “MATLAB Release Notes: R2023a” (Online), <https://www.mathworks.com/help/matlab/release-notes.html>. Retrieved on 2023/06/06.
22. “JPype1 · PyPI” (Online), <https://pypi.org/project/JPype1/>. PyPI, Python Software Foundation. Retrieved 2023/06/06.
23. “A Roadmap for Transmission Control Protocol (TCP) Specification Documents” (Online), <https://datatracker.ietf.org/doc/html/rfc7414>. Internet Engineering Task Force. Retrieved 2023/06/06.
24. “Visual Studio IDE With C/C++ for Cross-Platform & Game Development” (Online), <https://visualstudio.microsoft.com/vs/features/cplusplus/>. Microsoft. Retrieved 2023/06/06.
25. “Boeing: ecoDemonstrator” (Online), <https://www.boeing.com/principles/environment/ecodemonstrator>. Boeing. Retrieved 2023/06/06.
26. “Uniform Resource Locators (URL)” (Online), <https://datatracker.ietf.org/doc/html/rfc1738>. Internet Engineering Task Force. Retrieved 2023/06/06.
27. “wscat – npm” (Online), <https://www.npmjs.com/package/wscat>. npm, Inc. Retrieved 2023/06/06.
28. Bray, T., “RFC 8259 – The JavaScript Object Notation (JSON) Data Interchange Format” (Online), <https://tools.ietf.org/html/rfc8259>. Internet Engineering Task Force. Retrieved 2023/06/06.

Figures 3.2, 4.3, 4.5, 4.7, and 4.9 on pages 13, 16, 18, 19, and 21:

Source: [https://services.arcgisonline.com/ArcGIS/rest/services/World\\_Imagery/MapServer](https://services.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer),  
Used with permission according to ESRI terms of service: <https://www.esri.com/en-us/legal/copyright-trademarks>.

## A. Appendix

### A1. Building and Uploading Traffic Publisher to Local Repository

This section lists the steps to update the component before running a simulation involving the Traffic Publisher component.

- Listing A.1: extracting Gradle build tool archive file for building a component.

- Listing A.2: opening an editor to load the Traffic Publisher Java source file.
- Listing A.3: editing the Java source file to publish heading and groundspeed values.
- Listing A.4: building and upload the Traffic Publisher component to the local repository.

*Listing A.1. Extracting Gradle Build Tool Archive File*

<b>Platform</b>	<b>Command</b>
Linux, macOS	<pre>\$ cd software/ \$ unzip gradle-5.6.4-bin.zip</pre>
Windows	<pre>&gt; cd software\ &gt; tar -xf gradle-5.6.4-bin.zip</pre>

*Listing A.2. Opening TrafficPub.java via Editor*

<b>Platform</b>	<b>Command</b>
Linux, macOS	<pre>\$ cd \$SNTB_HOME/TestBedPlugins/TrafficPub/ \$ vi src/main/java/gov/nasa/sntb/trafficpub/TrafficPub.java</pre>
Windows	<pre>&gt; cd %SNTB_HOME%\TestBedPlugins\TrafficPub\ &gt; notepad src\main\java\gov\nasa\sntb\trafficpub\TrafficPub.java</pre>

*Listing A.3. Modifying TrafficPub.java*

<b>Line Number</b>	<b>Code</b>
114 115	<pre>logger.debug("publish: {}", track); publish(track);</pre>
161 162 163 164 165	<pre>track.setGroundspeed((float) speedKnots);  // calculate heading, in deg track.setHeading((float) GreatCircles.getHeadingTo(     oldLatDeg, oldLonDeg, newLatDeg, newLonDeg));</pre>

*Listing A.4. Building and Uploading Traffic Pub*

<b>Platform</b>	<b>Command</b>
Linux, macOS	<pre>\$ cd \$SNTB_HOME/TestBedPlugins/TrafficPub/ \$ gradle buildComponent uploadComponent</pre>
Windows	<pre>&gt; cd %SNTB_HOME%\TestBedPlugins\TrafficPub\ &gt; gradle buildComponent uploadComponent</pre>