Task Order 29: Prototype Operational Data Environment   
*Users Guide*



**Project Number: DTFH61-12-D-00020**

**June 1, 2016**

**Version: 3 DRAFT**

# Table of Contents

[1 Table of Contents i](#_Toc456794683)

[2 List of Figures ii](#_Toc456794684)

[3 Introduction 1](#_Toc456794685)

[4 P-ODE Background 1](#_Toc456794686)

[4.1 Technical Background 2](#_Toc456794687)

[5 Operators Guide 4](#_Toc456794688)

[5.1 Compilation 4](#_Toc456794689)

[5.2 Glassfish Configuration 4](#_Toc456794690)

[5.3 Database Configuration 4](#_Toc456794691)

[5.4 Installation 5](#_Toc456794692)

[6 Developers Guide 6](#_Toc456794693)

[6.1 Data Collector 6](#_Toc456794694)

[6.2 Data Source 7](#_Toc456794695)

[6.3 ODE Agent 7](#_Toc456794696)

[6.3.1 Registration Request Data 7](#_Toc456794697)

[6.3.2 Registration 8](#_Toc456794698)

[6.3.3 Parser 8](#_Toc456794699)

[6.3.4 Sanitizer 8](#_Toc456794700)

[6.3.5 Formatter 8](#_Toc456794701)

[6.3.6 Data Target 9](#_Toc456794702)

[6.4 Collector Configuration 9](#_Toc456794703)

[6.5 Publish and Subscribe 11](#_Toc456794704)

[6.5.1 Subscribing to the P-ODE 11](#_Toc456794705)

[6.5.2 Registration Information 13](#_Toc456794706)

[6.5.3 Publishing to the ODE 15](#_Toc456794707)

[7 Consumers Guide 18](#_Toc456794708)

[7.1 P-ODE Data Format 18](#_Toc456794709)

[7.2 P-ODE URLs 18](#_Toc456794710)

[7.3 P-ODE Subscription Communications Protocol 18](#_Toc456794711)

[7.4 Reusing P-ODE Components for Subscribing 19](#_Toc456794712)

# List of Figures

[Figure 1: High-level Physical Diagram of the P-ODE and DMA Application Emulator 2](#_Toc453680873)

[Figure 2: Data Collector 6](#_Toc453680874)

[Figure 3: ODE Agent 7](#_Toc453680875)

[Figure 4 Sequence diagram illustrating the message flow for subscription 19](#_Toc453680876)

# Introduction

The purpose of this document is to describe how to operate, extend, and consume data from the Prototype Operational Data Environment (P-ODE) that was developed as part of the Saxton Transportation Operations Lab contract to support the Connected Data Systems (CDS) Program. The P-ODE is intended to extend the scope of the CDS program to include near real-time data sources and publish-subscribe interfaces that will make the data available to users in near real-time.

# P-ODE Background

The P-ODE system receives data from multiple sources in real-time, has the potential to perform sanitization checks, transforms the data into a consistent format, and makes the data available to applications as well as stores the data in the USDOT Research Data Exchange (RDE). The P-ODE provides a Publish-and-Subscribe interface to enable users to subscribe to only the data they need, in near-real time. This streamlines the data collection and distribution process and provides users with a fresher data set for applications and in conducting research in general. It also eliminates the need for researchers to manage data collection logistics and formats for multiple data sources, letting researchers focus strictly on application and algorithm development.

The P-ODE also has a “playback” feature that enables users to retrieve data from the RDE as if it was coming from the source in real time such that applications can test different processes\algorithms using the same data scenario.

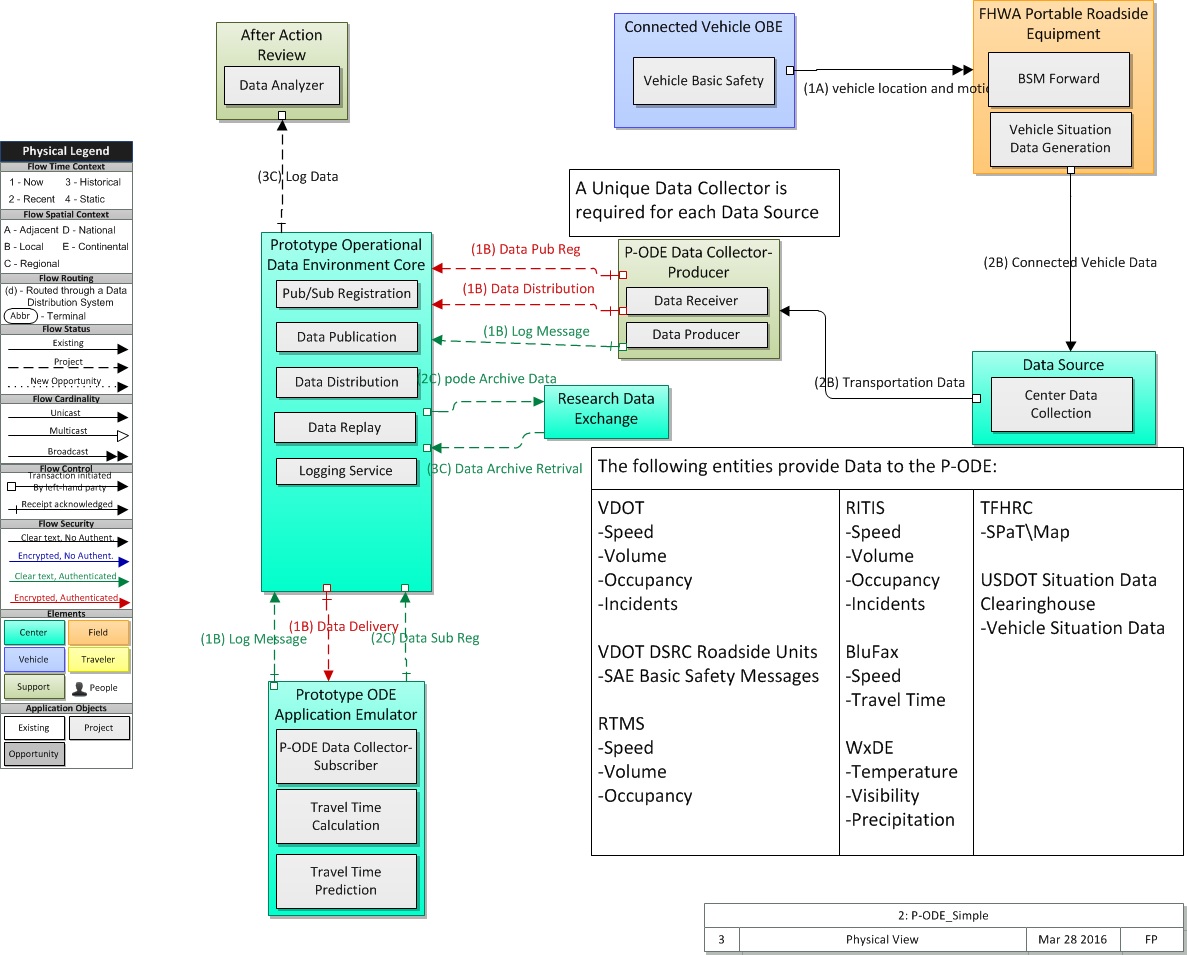
The P-ODE system is composed of two main components; Data Collectors and the Core. The Data Collectors interface directly with each data source and are responsible for data sanitization, translating the source data into a common format, and sending data to the Core for distribution. A dedicated Data Collector is required for each unique Data Source. The Core manages user subscriptions and distributes data to users based on those subscriptions.

To demonstrate the capability of the P-ODE, the system collects basic data that is used to emulate an application that calculates Travel Times along a given segment of roadway. As such, the P-ODE supports the following Data Elements:

1. Speed
2. Volume
3. Occupancy
4. Travel Time
5. Weather information
6. Incident information
7. Signal Phase and Timing\MAP messages

Figure 1 presents the high-level USDOT Connected Vehicle Reference Implementation Architecture (CVRIA) of the P-ODE system, including data sources, interfaces, Roadside Equipment deployed for Data Collection, and an Application Emulator.

Figure 1: High-level Physical Diagram of the P-ODE and Application Emulator



## Technical Background

The P-ODE is programmed in Java 7 and Java 8 and uses the Spring Framework to facilitate configuration of components via XML files. The P-ODE source code provides composable and reusable classes that can be used to assemble new data sources to bring data into the P-ODE as well as subscriber clients that listen to P-ODE output. The Spring configuration files determine how these components get put together at run time. The following sections will go into detail describing these components and how they fit together and what a sample configuration might look like for common use cases.

# Operators Guide

## Compilation

Since the P-ODE software is distributed only as source code it is necessary to compile it prior to installation. A version (or versions) of the Java Development Kit capable of compiling both Java 7 and Java 8 must be present on the system where compilation will take place. Also required is Apache Maven. Prior to the beginning of compilation, the BinaryNotes (provided in the source distribution) library must be installed into the local Maven repository with the command (in the root source directory):

*mvn install:install-file -Dfile=libs/binarynotes.jar -DgroupId=binarynotes -DartifactId=binarynotes -Dversion=1 -Dpackaging=jar*

In addition to BinaryNotes, the RDE API library and Saxton Utilities library must also be installed to the local maven repository in the same way:

*mvn install:install-file -Dfile=libs/rdeapiclient-3.1.45-SNAPSHOT-jar-with-dependencies.jar -DgroupId=org.rde.rdelive -DartifactId=rdeapiclient-jar-with-dependencies -Dversion=3.1.45-SNAPSHOT -Dpackaging=jar*

*mvn install:install-file -Dfile=libs/saxton-utils-1.0-SNAPSHOT.jar -DgroupId=gov.dot.fhwa.saxton -DartifactId=saxton-utils -Dversion=1.0-SNAPSHOT -Dpackaging=jar*

In addition to installing the dependencies, the “ode-api” package should be configured so the collectors have the correct IP address for the P-ODE Core server. This is accomplished by editing the “ode-api/src/main/resources/META-INF/ODE-API-Context.properties” file. The “leidos.ode.reg.baseurl” property should be changed so that the IP address and port correspond to the IP address and port the P-ODE Core will be deployed to.

Once the dependencies are installed and the collectors are configured compilation may be initiated by running

*mvn install -Dmaven.test.skip=true*

in the root directory of the source tree. The compilation process may take several minutes.

## Glassfish Configuration

Once the P-ODE JARs are compiled, the Glassfish server must be configured to run the core. One Java Messaging Service (JMS) topic must be set up per data type in the system, with name “DatatypeTopic” i.e. SpeedTopic, WeatherTopic, VolumeTopic, OccupancyTopic, TravelTimeTopic, and IncidentTopic. Documentation on how to create these topics is available here: <https://docs.oracle.com/cd/E19798-01/821-1758/create-jmsdest-1/index.html>

In addition to the topics, a javax.jsm.TopicConnectionFactory named ODETopicConnFact must be created in the Glassfish configuration. Documentation on how to create this topic factory is available here: <https://docs.oracle.com/cd/E19798-01/821-1751/giotd/index.html>

## Database Configuration

The P-ODE software as distributed depends on two database instances: MongoDB and MySQL. The MongoDB database is used for logging of messages through the system for performance analysis. As MongoDB is a schemaless document store, all that is required is initial setup and ensuring that the P-ODE is configured with the correct credentials, documentation on how to setup accounts for MySQL is available here: <http://dev.mysql.com/doc/refman/5.7/en/default-privileges.html>. The MySQL database should be instantiated with the schema described in “schema.sql” (by running schema.sql from the MySQL command line) from the P-ODE source distribution. This sets up the “ODE\_REGISTRATION” for sub/pub registration as well as sets up the “MSG\_QUEUE” to store data target information. Rows must be added to the “MSG\_QUEUE” table to point subscription and publication requests to the correct URLs and IP addresses of the P-ODE Core server. One row must be added per data type with the following semantics:

**ID:** Unique ID for row

**MSG\_TYPE:** One of (‘SPEED’, ‘VOLUME’, ‘OCCUPANCY’, ‘TRAVELTIME’, ‘WEATHER’, ‘SPATMAP’, ‘INCIDENT’)

**REGION:** Integer ID of region data corresponds to, must match publishers/subscribers (no formal definition, though “1” is used for Northern VA). Publishers configure their own region number which must match a row defined in this table.

**QUEUE\_NAME:** Topic name for MSG\_TYPE, configured in Glassfish above.

**QUEUE\_CONN\_FACT: ‘**ODETopicConnFact’

**TARGET\_ADDRESS:** IP address of P-ODE Core server

**TARGET\_PORT:** IP port the P-ODE Core process is listening on

**WS\_HOST:** Host address of HTTP endpoint used to receive data publications by the P-ODE Core

**WS\_URL:** URL path to publish data controller for datatype in P-ODE Core server

The WS\_HOST and WS\_URL columns for each row should point to the address of the physical server on which the P-ODE Core is being setup and the URL path to the PublishDataController appropriate for the data type.

## Installation

Once all the necessary databases and servers are configured, the application can be installed via the Glassfish administration web interface. The “ode-core-web.war” file should be deployed as the “ode-web” application and the “ode-emulator-web.war” file should be deployed as “ode-emulator-web”. The logs for the core may be viewed in the standard Glassfish domain log file.

The collectors may be installed on the same physical server (not in the Glassfish instance, however) as the core or on a separate device as long as they are configured with the correct IP address to communicate with the core. Simply copy the “ode-api-1.0-SNAPSHOT-shaded.jar” file to the desired server and run it with the

*java –jar ode-api-1.0-SNAPSHOT-shaded.jar <PATH\_TO\_COLLECTOR\_RUNNER\_PROPERTIES>*

command, where *<PATH\_TO\_COLLECTOR\_RUNNER\_PROPERTIES>* is the path to collector\_runner.properties. The collector\_runner.properties file specifies which collectors are to be started using binary on/off configuration values. The default location for this file is “ode-api/src/main/resources/collector\_runner.properties”, though if the working directory for the above command is not the root of the P-ODE source tree this path will have to be updated to match. The logs for this process may be viewed in the “ODE.log” file that will be created in the local directory.

Once the P-ODE server is deployed it is important to inform potential clients of the IP address of the P-ODE Core server to allow them to connect.

# Developers Guide

## Data Collector

The P-ODE was designed as a “pluggable” architecture, meaning the underlying framework of the system allows users to customize the collectors by writing new components and plugging them into the framework. The ODE Collector is the driving component for both Subscribers and Publishers.

The Data Collector is composed of two components: a Data Source and an ODE Agent. The Data Source component is responsible for connecting to the data source and polling or listening for incoming data, and informing the Collector when data is received. The ODE Agent is passed all data received from the source, for processing. Figure 2 depicts the high level architecture of a Data Collector.

Figure 2: Data Collector

Data Collector

Data Source

ODE Agent

External Data Source

P-ODE Core

## Data Source

The Data Source component of the Data Collector serves the purpose of communicating with an external data provider to get data for processing by the ODE Agent. The Data Source is responsible for managing resource location, authentication, sending/handling protocol messages, and packing the data received into a byte array for processing by the ODE Agent. The Data Source is run in its own thread which and manages its own execution interval in accordance with the data provider’s documentation. Implementations of Data Sources range from generic – RestPullDataSource, which executes a configured HTTP GET request on a REST resource in a fixed time interval – to provider specific – SDCDataSource, which authenticates with an SDC specific authentication server before opening a Websocket connection to the SDC server and sending a subscription message.

## ODE Agent

The ODE Agent is the work horse of the P-ODE Architecture for publishing and subscribing to data. The Agent also uses a pluggable architecture that is composed of 5 core sub components; Registration, Parser, Sanitizer, Formatter, and Data Target and an additional object depending on the Agent’s role in the specific Data Collector. Figure 3 depicts the architecture of an ODE Agent.

Figure 3: ODE Agent

ODE Agent

Registration

Sanitizer

Formatter

Parser

Data Target

Data Source

P-ODE Core

Registration Request Data

The Core ODE Agent subcomponents are described below.

### Registration Request Data

The Registration Request Data is an Agent data object that contains all necessary information about the Agent that will be sent to the P-ODE upon registration. This data includes what type of registration (Publish or Subscribe), Agent ID, what data sets the Agent is providing/subscribing to, start and end dates, etc. This information is configured in the ODERegistrationRequest class via the application’s Spring configuration file.

### Registration

The Registration component is responsible for informing the P-ODE Core of the Agent’s intent to either subscribe to data or publish data to the Core. Registration performs the initial Service Request, as well as performs the Registration for the Collector by sending the Core all necessary registration information. The P-ODE has already defined the necessary code for the Registration needed to both subscribe and publish to the P-ODE. Users will only need to use the correct implementation of the Registration subcomponent (ODEPublishRegistration or ODESubscriptionRegistration), which is available in the ODE-API library, and configure it correctly through the Spring Framework configuration file.

### Parser

For subscription, the Collector need only implement the Parser class com.leidos.ode.agent.parser.impl.ODEJ2735DataParser from the P-ODE API

For publication, the Parser is responsible for transforming raw source data streams into P-ODE defined Java objects: Speed, Volume, Occupancy, Travel Time, Weather, SPaT\Map, and Incident. Each Parser implementation is dependent upon the data source. When developing a custom publisher, a new abstract ODEDataParser class must be created for the new data. The new class will need to contain the implementation details for parsing the raw source data into a P-ODE defined Java object.

The Parser sends its results to the Sanitizer Step.

### Sanitizer

For publication, the Sanitizer removes Personally Identifiable Information (PII) from the data feed prior to sending it to the P-ODE Core. The sanitization logic is implemented on a source-by-source basis which involves sanitizing a message (or data element) if certain criteria are met. The P-ODE library provides a “Pass through” sanitizer that can be utilized if the data feed does not contain PII. The Sanitizer step can also be used as a “Data Cleansing” step to remove any unnecessary or likely erroneous (determined using source-specific logic) content from the data feed prior to sending it to the Core.

When developing a custom publisher, a new Java class implementing the P-ODE interface ODESanitizer is required to remove data from results of the Parser step that will not be sent to the Core.

The Sanitizer sends its results to the Formatter Step.

### Formatter

For data subscription, a “Pass Through” formatter is deployed to the Agent.

For data publication, the Formatter is responsible for transforming the parsed data feed into one or more of the seven data types supported by the P-ODE: Speed, Volume, Occupancy, Travel Time, Weather, SPaT\Map, and Incident. Every data source requires a customized implementation of the Formatter. This is accomplished by extending the P-ODE abstract class ODEMessageFormatter and providing an implementation of the formatMessage method. A single data source message could result in multiple standardized P-ODE data set messages sent to the Core. The Formatter generates a P-ODE standardized message for each supported data type received from the source.

The Formatter sends its results to the Data Target Step.

### Data Target

The Data Target is responsible for sending data from the Agent to its final destination. When subscribing to the P-ODE, data can be directed to an application for further processing or to a file for archive. A custom Data Target can be created by implementing the P-ODE interface ODEDataTarget. If data is to be written to a file, the Agent can be configured with the ODEFileDataTarget class.

In the case of a Publisher, the Data Target sends the data to the P-ODE Core. The P-ODE library provides the necessary implementation of the Data Target (ODERestTarget) that is plugged into the Agent to send data to the Core.

## Collector Configuration

The P-ODE makes use of the Spring Framework to configure Collectors and Agents. The dependency injection provided by the framework allows for easy configuration of Collectors on a per-source basis without the need to recompile the code every time. Below is an example of a Spring Framework configuration for a Collector and Agent. The values contained in “${…}” are placeholders.

<bean id="basicPubRegistration" class="com.leidos.ode.agent.registration.ODEPublishRegistration">

<property name="registrationBaseUrl" value="${leidos.ode.reg.baseurl}"/>

<property name="registrationEndpoint" value="${leidos.ode.reg.pub.endpoint}"/>

<property name="serviceRequestEndpoint" value="${leidos.ode.reg.pub.endpoint.serviceRequest}"/>

</bean>

<bean id="publishODEAgent" class="com.leidos.ode.agent.PublishODEAgent"/>

<bean id="restTarget" class="com.leidos.ode.agent.datatarget.ODERestTarget" scope="prototype"/>

<bean id="passthroughSanitizer" class="com.leidos.ode.agent.sanitizer.PassthroughSanitizer"/>

<bean id="bsmParser" class="com.leidos.ode.agent.parser.impl.BSMParser"/>

<bean id="bsmPubRegistrationRequest" class="com.leidos.ode.registration.request.ODERegistrationRequest">

<property name="messageType" value="${leidos.ode.reg.bsm.pub.messageType}"/>

<property name="region" value="${leidos.ode.reg.bsm.pub.region}"/>

<property name="registrationType" value="${leidos.ode.reg.bsm.pub.registrationType}"/>

<property name="agentId" value="${leidos.ode.reg.bsm.pub.agentId}"/>

<property name="dataTypes" value="${leidos.ode.reg.bsm.pub.datatypes}"/>

<property name="startDate">

<bean factory-bean="dateFormat" factory-method="parse">

<constructor-arg value="${leidos.ode.reg.bsm.pub.startDate}"/>

</bean>

</property>

<property name="endDate">

<bean factory-bean="dateFormat" factory-method="parse">

<constructor-arg value="${leidos.ode.reg.bsm.pub.endDate}"/>

</bean>

</property>

</bean>

<bean id="bsmMessageFormatter" class="com.leidos.ode.agent.formatter.BSMMessageFormatter">

</bean>

<bean id="bsmPubAgent" class="com.leidos.ode.agent.PublishODEAgent">

<property name="registration" ref="basicPubRegistration"/>

<property name="parser" ref="bsmParser"/>

<property name="sanitizer" ref="passthroughSanitizer"/>

<property name="dataTarget" ref="restTarget"/>

<property name="registrationRequest" ref="bsmPubRegistrationRequest"/>

<property name="formatter" ref="bsmMessageFormatter"/>

<property name="odeLogger" ref="odeLogger"/>

</bean>

<bean id="bsmDataSource" class="com.leidos.ode.collector.datasource.push.UDPPushDataSource">

<property name="hostAddress" value="${leidos.ode.reg.bsm.pub.hostAddress}"/>

<property name="hostPort" value="${leidos.ode.reg.bsm.pub.hostPort}"/>

</bean>

<bean id="bsmCollector" class="com.leidos.ode.collector.ODECollector">

<property name="agent" ref="bsmPubAgent"/>

<property name="dataSource" ref="bsmDataSource"/>

</bean>

## Publish and Subscribe

The P-ODE Publish-Subscribe interface allows data producers to specify what type of data they will be providing to the P-ODE, and data consumers to only request the type of data they are interested in receiving.

The P-ODE supports seven different data sets: Speed, Volume, Occupancy, Travel Time, Weather, Incident, and SPaT/MAP. The P-ODE allows data producers to register a publication intent for any of the data sets. In the response of the registration, the P-ODE provides the web address for each data set registered. The data producer is responsible for sending the correct data set to the correct address. If a data set is sent to the wrong address it is ignored; the P-ODE does not handle routing an incorrectly addressed data set. At this time the P-ODE does not inform the publisher if a data set is sent to the wrong address, such as sending speed data to the volume address. If the publisher attempts to use an address not specified by the P-ODE, it will receive a standard HTTP 404 error. When a data set is received by the P-ODE Core, the data is decoded and checked for accuracy, repackaged in the outbound data frame and placed on a JMS Topic for that data set type.

Data consumers are allowed to subscribe to one or all of the data sets provided by the P-ODE. When a consumer registers a subscription, it provides the data set(s) of interest and the IP address, port, and protocol of where the P-ODE should send the data. Currently the P-ODE supports UDP and TCP communication for subscriptions. For each data type subscription for each consumer, the P-ODE will start a dedicated “Data Distributor”. The Data Distributor connects to the JMS Topic for the data set(s) it has been assigned, receives any messages placed on the topic(s), then sends the message(s) to the subscriber on the requested IP address, and port using the specified protocol.

When publishing or subscribing to the P-ODE, all users will employ the P-ODE Collector component. The P-ODE Collector was designed to function as either a publishing tool or data subscriber simply by changing which sub components it is configured with. Configuration of a P-ODE Collector to Subscribe or Publish to the P-ODE is explained in the following sections.

### Subscribing to the P-ODE

To configure a Data Collector to Subscribe to the P-ODE, several sub component implementations are provided. The Data Source for the Data Collector will be the P-ODE Core, and should use one of the two provided Data Source implementations in the ODE-API library. These are com.leidos.ode.collector.datasource.push.UDPPushDataSource or com.leidos.ode.collector.datasource.push.TCPPushDataSource. Both of these Data Sources will start up and listen on the configured IP Address and Port; one uses the TCP protocol and the other UDP.

The Spring Configuration for a Data Source is below:

<bean id="weatherDataSource" class="com.leidos.ode.collector.datasource.push.UDPPushDataSource">

<property name="hostProtocol" value="UDP"/>

<property name="hostAddress" value="127.0.0.1"/>

<property name="hostPort" value="16000"/>

</bean>

The Data Collector also contains a P-ODE Agent which has the following subcomponents and configurations:

**Registration:** The Registration sub component should configure and use the class com.leidos.ode.agent.registration.ODESubscribeRegistration. This class is provided by the ODE-API and can be used by any subscriber. This class requires the following four configuration properties:

* registrationBaseUrl – Base URL value for the P-ODE (**http://<IPAddress>:8080/ode-web**)
* registrationEndpoint – Target endpoint for the registration (**registerSubscription**)
* unregisterEndpoint – Target endpoint to unregister a subscription (**unregister**)
* serviceRequestEndpoint – Target endpoint to send the Service Request (**registerSubscribeServiceRequest**)

**Parser:** The Parser should configure and use the com.leidos.ode.agent.parser.impl.ODEJ2735DataParser class. This class is provided by the ODE-API and can be used by any subscriber. The ODEJ2735DataParser does not require a specific configuration. The parser decodes the DER encoded J2735 data frame used defined by the P-ODE. An example of the Spring configuration is:

<bean id="odeMessageParser" class="com.leidos.ode.agent.parser.impl.ODEJ2735DataParser" scope="prototype"/>

**Sanitizer**: The Sanitizer should configure and use the com.leidos.ode.agent.sanitizer.PassthroughSanitizer class. This class is provided by the ODE-API and can be used by any subscriber. The PassthroughSantizer performs no action on the parsed data and does not require a specific configuration. When subscribing to the P-ODE, all PII information has already been removed from the data by the publisher. There is no reason to do any additional sanitization of the data. If the subscriber would like to do additional filtering of data it can be performed in this step. In the current implementation of the P-ODE, no PII information is being received, so there is no sanitization being performed. The system has been built to allow data sanitizing to be performed, and it is currently being used to filter out weather data so only Precipitation Rate, Visibility and Temperature is being sent to the P-ODE. An example of the Spring configuration is:

<bean id="passthroughSanitizer" class="com.leidos.ode.agent.sanitizer.PassthroughSanitizer" scope="prototype"/>

**Formatter**: The Formatter should configure and use the com.leidos.ode.agent.formatter.PassthroughFormatter class. This class is provided by the ODE-API and can be used by any subscriber. The PassthroughFormatter performs no action on the parsed data received and does not require a specific Spring configuration. An example of the Spring configuration is:

<bean id="passthroughFormatter" class="com.leidos.ode.agent.formatter.PassthroughFormatter"/>

**Data Target:** The Data Target will be Subscriber specific and will need to be developed by the user. The ODE-API contains one simple Data Target that will write any data received to a file specified by the user. Users requiring more complex functionality they need to develop a Data Target that interacts with their application. An example Spring configuration for a File Data Target is.

<bean id="fileDataTarget" class="com.leidos.ode.agent.datatarget.ODEFileDataTarget">

<property name="filePath" value=”c:/test/outputfile.txt"/>

</bean>

### Registration Information

Once the Agent has been configured in Spring with all of the needed sub-components, the registration information needs to be populated.

Below is an example Registration Information configuration for subscribing to “Speed”:

<bean id="speedSubRegistrationRequest" class="com.leidos.ode.registration.request.ODERegistrationRequest">

<property name="messageType" value="SPEED"/>

<property name="region" value="1"/>

<property name="registrationType" value="Subscribe"/>

<property name="agentId" value="emuspd1"/>

<property name="dataTypes" value="00000001"/>

<property name="subscriptionReceiveAddress" value="10.10.10.25"/>

<property name="subscriptionReceivePort" value="13000"/>

<property name="subscriptionProtocol" value="UDP"/>

<property name="subscriptionType" value="RealTime"/>

<property name="startDate">

<bean factory-bean="dateFormat" factory-method="parse">

<constructor-arg value="09-19-2015"/>

</bean>

</property>

<property name="endDate">

<bean factory-bean="dateFormat" factory-method="parse">

<constructor-arg value="10-19-2015"/>

</bean>

</property>

</bean>

**messageType**: This field is used internally by the Agent for storing data as it is parsed and formatted. It is not passed to the ODE core, but should be set.

**region**: This field is used to set the region the subscriber is requesting data from. The P-ODE only reports data for one region, so this field needs to always be set to 1.

**registrationType**: This field indicates whether the Agent is a Subscriber of Publisher. In the example above, the Agent is a Subscriber.

**agentId**: This field is a globally unique (amongst ODE agents) identifier for this specific agent. agentId can be up to 45 alphanumeric characters and can otherwise be chosen freely, as long as it is unique.

**dataTypes**: This field is a bit map representation of the data types the agent is subscribing to. Each bit in the string represents a specific data type, with a “1” indicating a subscription to that data type. In the above example only the bit for Speed is set to “1”. Other valid values are:

* Occupancy: 00000010
* Volume: 00000100
* Travel Time: 00001000
* Weather: 00010000
* SPaT/MAP: 01000000
* Incident: 10000000

As another example, dataTypes subscribing to Speed and Occupancy would be 00000011

**subscriptionReceiveAddress**: This field is the IP address to which the P-ODE Core should send the requested data.

**subscriptionReceivePort**: This field is the port to which the P-ODE Core should send the requested data.

**subscriptionProtocol**: This field is the protocol the subscriber wants to use to receive data. The two protocols available are UDP and TCP.

**subcriptionType**: This field indicates if the subscriber wants real time data or replay data from the P-ODE. Valid values are RealTime and Replay.

**startDate**: This field is either the date and time the Subscriber wishes to start a RealTime subscription or the date and time of the start of the data to Replay. Depending on the value of subscriptionType, the P-ODE will determine how to use this value in the registration. The format is MM-DD-YYYY.

**endDate**: This field is either the date and time the Subscriber wishes to end a RealTime subscription or the date and time of the end of the Replay data. Depending on the value of subscriptionType, the P-ODE will determine how to use this value in the registration. The format is MM-DD-YYYY.

The Data Collector can be started once everything is configured within the Spring context. The ODE Collector framework handles all connections and processing of data to the point it leaves the Agent at the Data Target.

### Publishing to the ODE

The same P-ODE architecture used to subscribe to data can be utilized to publish data to the P‑ODE, however, some Data Collector subcomponents will need to be customized to the source data. At a minimum, a custom Parser and Formatter are required and a custom Data Source component may be required as well. This section describes how to customize P-ODE Data Collector subcomponents.

#### Data Collector

**Data Source**: The ODE-API provides two Data Source classes that can be utilized with minimal configuration; a Restful Web Service class that retrieves data at a configured interval and a TCP and UDP protocol class that can receive data as it is sent. All Collector Data Sources must extend the abstract class DataSource and provide implementations for the three abstract methods.  The three methods are pollDataSource(), canPoll(), and cleanUpConnections(). pollDataSource() performs the actual logic to get data from the data source.  canPoll() tests to see if the data source is in a state to poll for data. cleanUpConnections() closes any open connections before the DataSource is stopped.

An example of a pull data source would be a data source that retrieves information from a web service, or FTP server at set intervals.  On a preconfigured interval (i.e. every 15 minutes), this data source would retrieve data from the source and send it to the Agent for processing.  A push data source opens a socket on a specified IP Address and port and waits for a packet in the pollDataSource() method.  Once a packet is received it is send that to the Agent for processing.

#### ODE Agent

##### Parser

The Parser subcomponent implementation is dependent upon the data source and is responsible for converting a raw stream of data into appropriate P-ODE Java objects. A custom Parser is created by extending the P-ODE abstract class ODEDataParser and overriding the method: public ODEDataParserResponse parse(byte[] bytes).

The data parser will parse the incoming data from the data source and return an instance of ODEDataParserResponse. This object contains two fields: an Object with name data (this is where the parser will store the parsed data object) and ODEDataParserReportCode named reportCode. If the parser fails to parse the data feed, the parser should return a reportCode of ODEDataParserReportCode.PARSE\_ERROR, if it succeeds, a reportCode of ODEDataParserReportCode.PARSE\_SUCCESS will be returned.

##### Sanitizer

The Sanitizer subcomponent is responsible for removing any Personally Identifiable Information (PII) or removing data that will not be sent to the P-ODE Core. This could be data out of scope, or extra information provided by the data source that is not supported or required by the Core. To create a new Sanitizer, a new implementation of the P-ODE Java interface ODESanitizer is created overriding the method: public ODEAgentMessage sanitizeMessage(ODEAgentMessage message) throws ODESanitizeException.

This method takes an instance of ODEAgentMessage as a parameter and returns the same type of object. Within the ODEAgentMessage the field of interest to the Sanitizer is Object formattedMessage. This is where the Agent stores the results of the Parser. The Sanitizer then removes data from this field as necessary and places the, now sanitized, results back into the same field of the ODEAgentMessage and returns that value from the method.

##### Formatter

The Formatter is responsible for transforming the parsed and sanitized data feed into one or more of the six data sets supported by the P-ODE. Every new data source being published will need to have a custom Formatter. To create a new formatter, a new implementation of the P‑ODE abstract class ODEMessageFormatter is created overriding the method: public abstract Map<ODEMessageType,List<PodeDataDistribution>> formatMessage(ODEAgentMessage agentMessage, ServiceRequest serviceRequst).

This method takes an instance of ODEAgentMessage as well as an instance of ServiceRequest as parameters and returns a Map where the key for the map is an ODEMessageType and the value is a List of PodeDataDistribution, defined in the P-ODE ASN.1 format. The formattedMessage field of the ODEAgentMessage contains the Parsed and Sanitized data object from the previous two Agent subcomponents. It is this field the formatter will use while constructing the proper PodeDataDistribution objects. If the data source is formatted into multiple P-ODE messages of the same type, each message will be placed into a Java List, the List is then placed into the Map object that is returned by this message with the proper key. If the data source is formatted into different P-ODE messages, each message type is placed into its own list, and each List is then placed into the Map using the proper key value.

Valid Map keys are one of the following Enumerated values:

ODEMessageType.SPEED

ODEMessageType.VOLUME

ODEMessageType.OCCUPANCY

ODEMessageType.TRAVEL

ODEMessageType.WEATHER

ODEMessageType.INCIDENT

ODEMessageType.SPATMAP

##### Data Target

The Data Target is responsible for sending the formatted P-ODE messages to the P-ODE Core. For publication, the Agent should be configured to use the ODERestTarget. This Data Target sends data directly to the P-ODE Core. For data subscribing client this would be where the incoming data should be handled according to appropriate application logic.

# Consumers Guide

## P-ODE Data Format

The specification for the P-ODE data format is available in the P-ODE source distribution as “PODE\_IMPLICIT\_R\_0.2.7\_160525.asn”. This document, in conjunction with *SAE J2735*, describes the logical structure of the data generated by the P-ODE. These logical data elements are then encoded using the BER ASN.1 encoding ruleset described in *ITU-T X.690*. Any standards-compliant ASN.1 parser/compiler should be able to interpret the P-ODE format data, though only *BinaryNotes* has been used and tested with the P-ODE. The P-ODE Core receives (and responds to) protocol messages via HTTP. The messages described in the P-ODE specification are wrapped in XML as xsd:base64Binary elements. Data delivery messages are transmitted as raw binary via either TCP or UDP.

## P-ODE URLs

The P-ODE operator must provide all clients with the IP address of the core server; from this all other URLs needed by the client can be constructed.

http://<IP\_ADDRESS>/ode-web/<ENDPOINT>

To communicate with a service in the P-ODE core, simply substitute the IP address of the core for <IP\_ADDRESS> and the resource name of the service for <ENDPOINT> in the URL above. A list of relevant resource names follows:

**Registration Service**: “registerSubscribeServiceRequest”

**Subscription Service**: “registerSubscription”

**Publish Service**: “publish<DATA\_TYPE>”, where <DATA\_TYPE> is one of “Weather”, “Speed”, “Volume”, “Occupancy”, “TravelTime”, “Incident”

## P-ODE Subscription Communications Protocol

To subscribe to P-ODE output data, it is first necessary to request service from the P-ODE Core by sending a ServiceRequest message (described in the P-ODE format specification). Once this has been processed by the P-ODE Core it will send a ServiceResponse message back, echoing the values in the ServiceRequest to indicate success. Once this handshake is completed, a RegistrationRequest message may be sent to the P-ODE Core to begin the data subscription. Once the RegistrationRequest is processed by the core it will respond with a RegistrationResponse, echoing the values to indicate success and populating the fields it will be using to determine which data to send. At this point a data distributor is created in the core which will attempt to connect to the client on the requested IP address and port (if TCP communication was requested) and any data that matches the subscription request will be sent to the client.

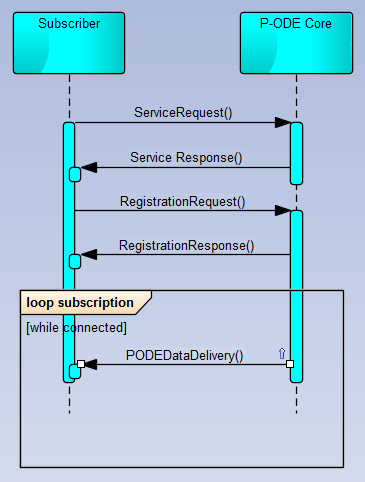


Figure 4 Sequence diagram illustrating the message flow for subscription

## Reusing P-ODE Components for Subscribing

A simple P-ODE client can be built out of existing components contained in the P-ODE source distribution. To do so, the registration request, data source, parser, and data target of a P-ODE data collector must be reconfigured. The registration request must be changed to an implementation that requests to subscribe instead of publish, the data source must be changed to a simple UDP listener, and the parser must be an ODEJ2735Parser instance. The data target then would be the location to implement application specific logic to handle the incoming data. Several examples of this configuration of a collector are available in the “ode-emulator” project contained in the P-ODE source distribution.