Task Order 13: Prototype Operational Data Environment   
*Users Guide*



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

**October 16, 2015**

**Version: 1**

Table of Contents

[1. Introduction 1](#_Toc432848706)

[1.1. P-ODE Background 1](#_Toc432848707)

[2. ODE Collector 2](#_Toc432848708)

[2.1. ODE Agent 3](#_Toc432848709)

[2.2. Registration Request Data 4](#_Toc432848710)

[2.3. Registration 4](#_Toc432848711)

[2.4. Parser 4](#_Toc432848712)

[2.5. Sanitizer 4](#_Toc432848713)

[2.6. Formatter 5](#_Toc432848714)

[2.7. Data Target 5](#_Toc432848715)

[3. Collector Configuration 5](#_Toc432848716)

[4. Publish and Subscribe 7](#_Toc432848717)

[4.1. Subscribing to the ODE 7](#_Toc432848718)

[4.2. Registration Information 9](#_Toc432848719)

[4.3. Publishing to the ODE 11](#_Toc432848720)

[4.3.1. ODE Collector 11](#_Toc432848721)

[4.3.2. ODE Agent 11](#_Toc432848722)

[4.3.2.1. Parser 11](#_Toc432848723)

[4.3.2.2. Sanitizer 11](#_Toc432848724)

[4.3.2.3. Formatter 11](#_Toc432848725)

[4.3.2.4. Data Target 12](#_Toc432848726)

List of Figures

[Figure 1: High-level Physical Diagram of the P-ODE and DMA Application Emulator 2](#_Toc432848727)

[Figure 2: Data Collector 3](#_Toc432848728)

[Figure 3: ODE Agent 3](#_Toc432848729)

# Introduction

The purpose of this document is to describe how to publish and subscribe to the Prototype Operational Data Environment (P-ODE) that was developed as part of the Saxton Lab Task Order 13 to support the Data Capture Management (DCM) Program. The P-ODE is intended to extend the scope of the DCM 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, performs 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 to 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 comprised 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 can be utilized to calculate 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. SPaT\Map

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

Data Sources



Deployed for P-ODE Data Collection

P-ODE System

Emulator

# ODE 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 ODE 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 : Data Collector

Data Collector

Data Source

ODE Agent

Data Source

P-ODE Core

# 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 Agents role in the specific Data Collector. Figure 3 depicts the architecture of an ODE Agent.

Figure : 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.

# 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, 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, and SPaT\Map. 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 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 un-necessary 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 Formater 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 six data types supported by the P-ODE; Speed, Volume, Occupancy, Travel Time, Weather, and SPaT\Map. 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 without the need to recompile the code every time. Below is an example of a Spring Framework configuration for a Collector and Agent.

<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 six different data sets: Speed, Volume, Occupancy, Travel Time, Weather, 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. 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 Java Message Service (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 ODE Collector component. The ODE Collector was designed to function as either a publishing tool, or data subscriber simply by changing which sub components it is configured with. Configure an ODE Collector to Subscribe or Publish to the P-ODE is explained in the following sections.

# Subscribing to the 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 an 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. 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. If the user requires more complex functionality they will 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" valuec:/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 unique identifier for this specific agent. agentId can be up to 45 alphanumeric characters

**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

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

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

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

**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.

**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 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.

# ODE 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 custome Formatter. To create a new formatter, a new implementation 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.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.