

***CONNECTED VEHICLES CORE SERVICES (CVCS)***

***– Data Distribution Subsystem (DDS)***

**Operations Manual**

Version *<1.2>*

*<09/28/2017>*

|  |  |
| --- | --- |
| Client: | *FHWA Intelligent Transportation Systems Joint Program Office –*  *Ariel Gold* |
| Date: | *9/28/17* |
| Project Number: | *DTFH-61-15-R-00037* |
| Work Location(s): | *Michigan* |
| Team | *Leidos* |

**VERSION HISTORY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version #** | **Implemented By** | **Date** | **Approved By** | **Date** |
| 1.0 | *Matt Vahlberg, Erik Hamilton, Diane Newton* | *07/14/2017* | *<name>* | *<mm/dd/yy>* |
| 1.1 | *Matt Vahlberg, Erik Hamilton, Diane Newton* | *09/15/2017* | *<name>* | *<mm/dd/yy>* |
| 1.2 | *Matt Vahlberg, Erik Hamilton* | *09/28/2017* | *<name>* | *<mm/dd/yy>* |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

[1 Introduction 5](#_Toc494899531)

[2 System Overview 5](#_Toc494899532)

[3 Software Summary 5](#_Toc494899533)

[3.1 Software Description 5](#_Toc494899534)

[3.2 Commercial Services and Software Components 6](#_Toc494899535)

[3.3 Developed Components for integration into a CV Environment 7](#_Toc494899536)

[3.4 System Architecture 8](#_Toc494899537)

[3.4.1 Situation Data Clearinghouse (SDC) System 9](#_Toc494899538)

[3.4.2 Situation Data Warehouse (SDW) System 10](#_Toc494899539)

[3.4.3 Situation Data Processing Center (SDPC) System 12](#_Toc494899540)

[3.4.4 Distribution System 13](#_Toc494899541)

[3.4.5 Standalone Applications 15](#_Toc494899542)

[4 Running the Systems 18](#_Toc494899543)

[4.1 Initiate Forwarders 18](#_Toc494899544)

[4.2 Building the Systems 18](#_Toc494899545)

[4.2.1 Upload Plugins 18](#_Toc494899546)

[4.2.2 Setup Data Models 18](#_Toc494899547)

[4.2.3 Build the System 18](#_Toc494899548)

[4.2.4 Start the System 19](#_Toc494899549)

[4.2.5 Verify 19](#_Toc494899550)

[4.3 Complete Forwarders 19](#_Toc494899551)

[5 Monitoring 19](#_Toc494899552)

[5.1 Management Console and System Monitor 19](#_Toc494899553)

[5.2 Warehouse Tools 20](#_Toc494899554)

[5.3 Test Routine: Data Type ISD, VSD, etc. 20](#_Toc494899555)

[5.4 Notification Scheduling 20](#_Toc494899556)

[6 Updating OSS Licensing 20](#_Toc494899557)

[7 Updating SSL Certificate 20](#_Toc494899558)

[8 Troubleshooting 21](#_Toc494899559)

[8.1 Ticketing System 21](#_Toc494899560)

[8.2 Uptime Dashboard 22](#_Toc494899561)

[8.3 DigitalEdge Console 22](#_Toc494899562)

[8.4 AWS Console 22](#_Toc494899563)

[8.5 Rackspace Console 22](#_Toc494899564)

[8.6 Maintenance 22](#_Toc494899565)

[9 DigitalEdge Data Models 22](#_Toc494899566)

[9.1 Vehicle Situation Data Model 23](#_Toc494899567)

[9.2 Intersection Situation Data Model 26](#_Toc494899568)

[9.3 Traveler Information Data Model 30](#_Toc494899569)

[10 Configuring DigitalEdge Systems 34](#_Toc494899570)

[10.1 Configure the Clearinghouse System using System Builder 34](#_Toc494899571)

[10.1.1 Adding the Vehicle Situation Data Transports 35](#_Toc494899572)

[10.1.2 Adding the Intersection Data Message Transport 36](#_Toc494899573)

[10.1.3 Adding the SituationDataRouter Data Sink 37](#_Toc494899574)

[10.2 Configure the Warehouse System Using System Builder 38](#_Toc494899575)

[10.2.1 Adding the Vehicle Situation Data JMS Listener Transport 39](#_Toc494899576)

[10.2.2 Adding the Traveler Information Data Transport 40](#_Toc494899577)

[10.2.3 Adding the Intersection Situation Data JMS Listener Transport 41](#_Toc494899578)

[10.2.4 Adding the MongoDB Data Sink 42](#_Toc494899579)

[Appendix A: List of Acronyms 45](#_Toc494899580)

[Appendix B: Maintenance Plan 46](#_Toc494899581)

[B.1 Software Updates 46](#_Toc494899582)

[B.1.1 Updating Plugins 46](#_Toc494899583)

[B.1.2 Updating Software Libraries Directly 46](#_Toc494899584)

[B.1.3 Rebuild and Cycle Affected System 47](#_Toc494899585)

[B.1.4 Update Forwarders 47](#_Toc494899586)

[B.2 Continued Operations & Infrastructure Maintenance 47](#_Toc494899587)

[B.2.1 AWS VM Retirement 47](#_Toc494899588)

[B.3 Periodic Diagnostics 48](#_Toc494899589)

[B.3.1 DigitalEdge Management Console 48](#_Toc494899590)

[B.3.2 DigitalEdge System Monitor 48](#_Toc494899591)

[B.3.3 AWS Console 48](#_Toc494899592)

[B.4 Backup Procedure for the Situation Datawarehouse and Situation Data Processing Center 48](#_Toc494899593)

[B.4.1 Volume Snapshots 48](#_Toc494899594)

[B.5 Maintenance Issues 49](#_Toc494899595)

[B.6 Security Monitoring and Patching 49](#_Toc494899596)

[B.6.1 Software 49](#_Toc494899597)

[B.6.2 Instance Access 49](#_Toc494899598)

[Appendix C: Background 50](#_Toc494899599)

[Appendix D: DigitalEdge™ Framework 51](#_Toc494899600)

[D.1 Team 51](#_Toc494899601)

[D.2 Licensing 51](#_Toc494899602)

[D.3 Documentation 51](#_Toc494899603)

[Appendix E: DigitalEdge™ Configuration Guide, Chapter 4 – *Defining the Data Model* 53](#_Toc494899604)

[Appendix F: DigitalEdge™ Configuration Guide, Chapter 5 – *Building the System* 122](#_Toc494899605)

[Appendix G: DigitalEdge™ Operation Guide, Chapter 5 – *Managing DigitalEdge from the Management Console* 164](#_Toc494899606)

[Appendix H: DigitalEdge™ Operation Guide, Chapter 6 – *System Monitoring* 185](#_Toc494899607)

List of Tables and Figures

[Table 1. Dynamic Plugin Parameter Values 19](#_Toc494899608)

[**Figure 1- Cloud Environments for the SDC and SDW** 6](#_Toc494899609)

[**Figure 2- Situation Data Clearinghouse DigitalEdge System** 9](#_Toc494899610)

[**Figure 3- Situation Data Warehouse DigitalEdge System** 10](#_Toc494899611)

[**Figure 4- Situation Data Processing Center DigitalEdge System** 12](#_Toc494899612)

[**Figure 5- Distribution DigitalEdge System** 13](#_Toc494899613)

[**Figure 6- Advisory Situation Data Poller Test application** 16](#_Toc494899614)

[**Figure 7- Intersection Situation Data Poller Test Application** 18](#_Toc494899615)

[Figure 8- Vehicle Situation Data Model Input Fields 24](#_Toc494899616)

[Figure 9- Vehicle Situation Data Model Data Sources 25](#_Toc494899617)

[Figure 10- Vehicle Situation Data Model Enrichments 25](#_Toc494899618)

[Figure 11- Intersection Situation Data Model Input Fields 27](#_Toc494899619)

[Figure 12- Intersection Situation Data Model Data Sources 28](#_Toc494899620)

[Figure 13- Intersection Situation Data Model Enrichments 28](#_Toc494899621)

[Figure 14- Traveler Information Data Model Input Fields 30](#_Toc494899622)

[Figure 15- Traveler Information Data Model Data Sources 31](#_Toc494899623)

[Figure 16- Traveler Information Data Model Enrichments 32](#_Toc494899624)

[Figure 17- Clearinghouse System Components 35](#_Toc494899625)

[Figure 18- Vehicle Situation Data Transport Parameters 36](#_Toc494899626)

[Figure 19- Intersection Data Transport Parameters 37](#_Toc494899627)

[Figure 20- Situation Data Router Parameters 38](#_Toc494899628)

[Figure 21- Data Models for Clearinghouse System Situation Data Router 38](#_Toc494899629)

[Figure 22- Warehouse System Components 39](#_Toc494899630)

[Figure 23- Vehicle Situation Data Listener Transport Parameters 40](#_Toc494899631)

[Figure 24- Traveler Information Transport Parameters 41](#_Toc494899632)

[Figure 25- Intersection Data Listener Transport Parameters 42](#_Toc494899633)

[Figure 26- MongoDB Data Sink Parameters 43](#_Toc494899634)

[Figure 27- Data Models for Warehouse System 44](#_Toc494899635)

[**Figure 28- System Architecture** 50](#_Toc494899636)

# 1 Introduction

This document provides information concerning operations of the Data Distribution Subsystem for the United States Department of Transportation (USDOT) Southeast Michigan Connected Vehicles Core Services (CVCS) project. Operating these systems requires periodic interaction with the underlying server-side systems, which run in Amazon Web Services (AWS). This document reflects the Version 2.3 CVCS systems developed by Leidos.

# 2 System Overview

The Situation Data Clearinghouse (SDC), Situation Data Warehouse (SDW), and Situation Data Processing Center (SDPC) are systems designed to accept, process, and disseminate vehicle situation data, intersection data, and traveler information data to authorized users.

The SDC is designed to provide situation and intersection data to subscribers in near real-time. The SDC validates data message formats and verifies that messages fall within the geographic region associated with the SDC. For messages satisfying the input criteria, the SDC makes them available via Java Messaging Service (JMS) topic subscriptions to authorized consumers for data falling within the subscriber’s desired geographic region.

The SDW accept the same message types as the SDC. In addition, the SDW accepts traveler information data. The SDW differs from the SDC in that the valid input messages are stored in a short-term database rather than made immediately available to subscribers. Messages are stored for a configurable amount of time before expiring. The SDW is currently set to age off messages after 30 minutes. Validated users can query stored messages from the database according to message type and geographic region.

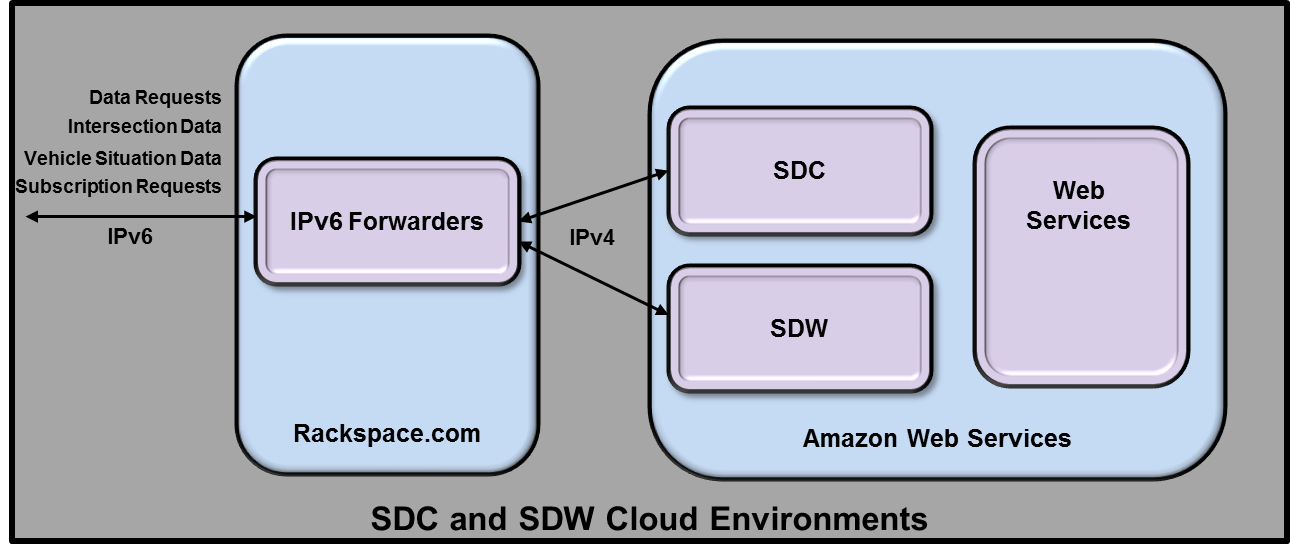
The SDPC is a clone of the SDW. There are two main differences between the SDPC and the SDW. The first difference is that the SDPC does not accept traveler information data. The second is that the SDPC is configured to store messages for 9 months, instead of the SDWs 30 minutes.

A number of web applications and services are used to support the SDC and SDW. The applications support visualization of subscription data from the SDC and data queries into the SDW. The applications are enabled by web services running in AWS that connect to the SDC and SDW systems. These tools are useful for demonstrations and for monitoring and troubleshooting purposes.

# 3 Software Summary

## 3.1 Software Description

The SDC, SDW, and SDPC systems operate in the Amazon Web Services (AWS) and Rackspace Cloud environments as shown in Figure 1. Rackspace is used to accept, translate, forward, and distribute User Datagram Protocol (UDP) packets over Internet Protocol Version 6 (IPv6). This is the interface to the systems from roadside units (RSUs), which receive vehicle situation data and intersection data from devices within range of the RSU. The forwarders running in Rackspace translate the packets from IPv6 to the more commonly used Internet Protocol Version 4 (IPv4) for data ingestion into the AWS cloud. Within AWS, the SDC, SDW, and SDPC are systems that run on the Leidos DigitalEdge framework[[1]](#footnote-1). DigitalEdge provides general-purpose ingest, processing, and data storage services for large, complex systems. Additionally, AWS provides web services for client applications that consume data from the SDC, SDW, and SDPC.

****

**Figure 1- Cloud Environments for the SDC and SDW**

## 3.2 Commercial Services and Software Components

As noted above, the SDC, SDW, and SDPC run in a public cloud infrastructure. The SDC/SDW/SDPC environment also contains libraries and certificates that enable encoding and decoding per the ASN.1 UPER format and encrypting and decrypting IEEE1609.2 standard messages. This section will describe an overview and introduction for each of the various commercial services and software components, for more details refer to the Software Architecture section of this document.

*Amazon Web Services (AWS)* – Enables the operations of the SDC, SDW, SDPC, and web applications via cloud infrastructure. The cloud resources utilized through AWS are Elastic Computing (EC2), Elastic Block Storage (EBS), Simple Storage Service (S3), and Route53.

* EC2 – Handles processing dialogs and situation data messages
* EBS – Stores the situation data before distribution in the SDC or stores the situation data for long-term uses by the SDW and SDPC
* S3 – Stores the SDC, SDW, SDPC configuration files, software components, and other miscellaneous artifacts, such as the Java keystore and truststore files to connect to the SDC
* Route53 – Provides DNS access to the services

*DigitalEdge1 –* DigitalEdge is a big data platform developed by Leidos that runs in AWS. DigitalEdge runs the four systems that comprise the CVCS data distribution and support services subsystems. Included within DigitalEdge are management tools that support data modeling, system configuration and construction, system operation and status, and system performance monitoring. To meet the needs of a CV environment, the Leidos team created multiple running DigitalEdge systems.

*OSS ASN.1 Studio* – The SDC, SDW, and SDPC use a software library provided by OSS Nokalva to handle encoding and decoding the Connected Vehicle dialog and situation data messages in UPER format. The studio is a desktop application that comes with the OSS Nokalva software package and is used to generate the Java source which the SDC, SDW, and SDPC use in AWS. In order to use the studio, users must acquire the Connected Vehicle and Dedicated Short-Range Communication (DSRC) ASN.1 specification files.

* DSRC ASN.1 Specification - <http://standards.sae.org/j2735_201603/>
* Connected Vehicle ASN.1 Specification - <https://gitlab.com/connectedvehicles/fedgov-cv-asn1/blob/master/src/main/resources/SEMI_v2.3.0_070616.asn>

*1609.2 Certificates* – All messages into and out of the SDC, SDW, and SDPC will be encrypted following the IEEE 1609.2-2016 standard. Certificates for both the SDC, SDW, and SDPC will need to be obtained from the Security Certificate Management System (SCMS) in order to decrypt any ASN.1 UPER messages coming into the systems.

## 3.3 Developed Components for integration into a CV Environment

The native DigitalEdge platform alone does not meet the specific system needs for the SDC, SDW, and SDPC, thus, several plug-in components were developed using the DigitalEdge software development kit (SDK). DigitalEdge supports integration of custom components into the platform using a simple upload feature. Below is a description of the various software components developed to ensure integration of SDC, SDW, and SDPC into a real-world CV Test Bed environment.

*CVCS Components – These are components specifically developed for the USDOT Southeast Michigan project. While the following components in this subsection may be useful for others implementing similar services, it is important to note that they were developed for integration with a specific CV Test Bed. The components run in DigitalEdge to support specific functionality required for the SDC, SDW, and SDPC in AWS.*

* fedgov-cv-asn1: This plugin is generated from using the OSS Studio desktop tool using the Connected Vehicle and DSRC ASN.1 specification files. It also contains common utility classes used to assign the encoding and decoding of the Connected Vehicle messages.
* fedgov-cv-common: This plugin contains common Java classes used by the other projects.
* fedgov-cv-transport-udp: This plugin contains the data receiver component of the SDC, SDW, and SDPC. The transport handles all Connected Vehicle dialog messages.
* fedgov-cv-transport-listener: This plugin is used by the SDW and SDPC to connect to the SDC’s Publish/Subscribe Server to listen for all vehicle and intersection situation data.
* fedgov-cv-parser-j2735: This plugin is for the SDC, SDW, and SDPC to parse vehicle, intersection, and traveler information situation data messages (Note: the traveler information messages go to the SDW only) and translate them according to mappings specified in the DigitalEdge data models.
* fedgov-cv-processor-broadcast-instructions: This plugin is used by the SDW to enrich the traveler information input data with broadcast instructions if the traveler information is of type RSU.
* fedgov-cv-processor-geojson: This plugin is for the SDW to enrich intersection and traveler information input data with GeoJSON information before it gets stored in the MongoDB database.
* fedgov-cv-datasink-router: This plugin is for the SDC to route vehicle and intersection situation data to the appropriate subscriber topic.
* fedgov-cv-datasink-subscription: This plugin is for the SDC to create subscriptions. The subscription information is stored on the DigitalEdge internal H2 database.
* fedgov-cv-mongodb: This plugin is for the SDW and SDPC to store the vehicle, intersection, and traveler information situation input data in the MongoDB database.
* fedgov-cv-query-processor: This plugin is for the SDW and SDPC to service query requests for intersection and traveler information messages.

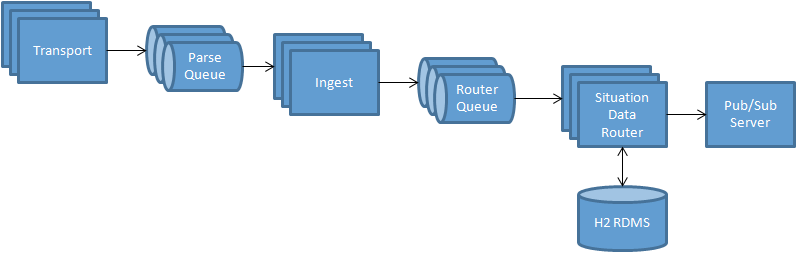
*Standalone Applications – This is a set of test applications developed to send and receive data from the SDC, SDW, and SDPC for integration testing and demonstration purposes.*

* Java command-line applications
  + An application to poll for advisory broadcast messages from the SDW
  + An application to poll for intersection messages from the SDW
  + An application to send intersection situation data to the SDC
  + Warehouse Client application: performs deposits/queries or subscribes for a running stream of data from a Warehouse This client can be embedded directly in Java code as well

## 3.4 System Architecture

In order to meet the requirements of the SDC, SDW, and SDPC, four DigitalEdge systems run in the AWS cloud environment. The first system (Clearinghouse) ingests and processes vehicle and intersection situation data. The second system (Warehouse) ingests, processes, and stores vehicle, intersection, and traveler information situation data. The third system (Distribution) handles subscription requests and traveler information and intersection query data requests from the SDW. The reason for the third system is to separate data requests from situation data deposits, so subscription and query requests can be fulfilled in a timely manner. The fourth system (SDPC) ingests, processes, and stores vehicle and intersection situation data for a long period of time. Components developed specifically for the Southeast Michigan project are denoted in italic font. Native DigitalEdge components are identified in regular font. It should be noted that there is overlap on the various instances used per system. For ease of system use, each system has been separated out below.

### 3.4.1 Situation Data Clearinghouse (SDC) System



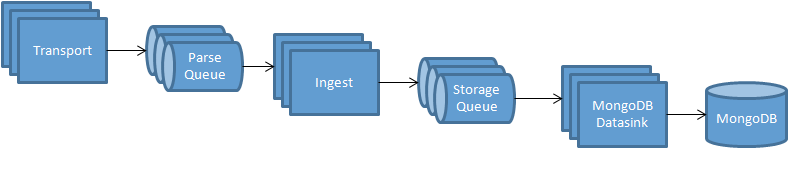
**Figure 2- Situation Data Clearinghouse DigitalEdge System**

The Clearinghouse System, shown in Figure 2, is a DigitalEdge system. Each box and cylinder in the diagram above represents a virtual DigitalEdge instance running in the AWS environment. The boxes and cylinders that are stacked indicate that one or more instance of the same type can run in parallel depending upon the configuration of the Clearinghouse system. The ability to create additional instances of the same type provides a way to horizontally scale the system to meet the performance requirements.

The Clearinghouse system receives vehicle and intersection situation data messages and routes them to the appropriate subscriber. Below is a description of each instance.

* Transport – This instance runs the *fedgov-cv-transport-udp* custom component. It receives and responds to vehicle and intersection situation data ASN.1 UPER dialog messages. When a VehSitDataMessage or IntersectionSituationData message is received, the transport will decrypt it and validate it to ensure that it is properly formed and within the specified service region of the system before placing it on the parse queue for processing. One or multiple Transport instances can be configured to run at once in the Clearinghouse.
* Parse Queue – This instance runs an ActiveMQ JMS Server and is responsible for storing the vehicle and intersection situation data until the Ingest instance picks it up for processing. This ensures that no messages are lost when the Ingest instance is busy. One or three of this type of instance can be configured to run at once in the Clearinghouse.
* Ingest – This instance type runs the *fedgov-cv-parser-j2735* and *fedgov-cv-processor-geojson* custom components, parses vehicle and intersection situation data messages, and maps extracted data fields into JavaScript Object Notation (JSON) records as configured in the appropriate DigitalEdge data model. Afterwards, a GeoJSON structure will be added to each data record. One or multiple Ingest instances can be configured to run at once or the Clearinghouse can be configured to scale the number of Ingest instances dynamically depending on the load of the system.
* Router Queue - This instance type runs an ActiveMQ JMS Server and is responsible for storing the vehicle and intersection situation data records until a Situation Data Router instance accepts it for processing. This ensures that no messages are lost when a Situation Data Router instance is busy. One or three Router Queues can be configured to run in the Clearinghouse.
* Situation Data Router – This instance type runs the *fedgov-cv-datasink-router* custom component and routes vehicle and intersection situation data to the appropriate subscriber topic on the Pub/Sub Server. It compares each situation data record to all active subscriptions and determine which ones match. When a subscription matches, the Situation Data router will encrypt the data using the subscriber’s 1609.2 certificate and place the encrypted message on the subscriber’s JMS topic. All subscription information is stored in the H2 RDMS instance, and the Situation Data Router periodically polls H2 to see if new subscriptions have been added or existing subscriptions removed. If a new subscription is added, the Situation Data Router creates the subscriber’s unique JMS topic on the Pub/Sub Server. One or multiple Situation Data Router instances can be configured to run at once, or the Clearinghouse can be configured to scale the number of instances dynamically depending on the load of the system. If more than one instance of this type exists, only the first instance will create or remove JMS topics for subscriptions.
* H2 RDBMS – This instance type runs the H2 relational database management system used internally by DigtalEdge and stores all the subscription information. Only one instance of this type exists for the Connected Vehicle tenant. The H2 RDBMS also supports the other two DigitalEdge systems.
* Pub/Sub Server – This instance type runs an ActiveMQ JMS Server and is responsible for pushing vehicle and intersection situation data to subscribers connected to a topic. Only one instance of this type exists in the Clearinghouse system.

### 3.4.2 Situation Data Warehouse (SDW) System

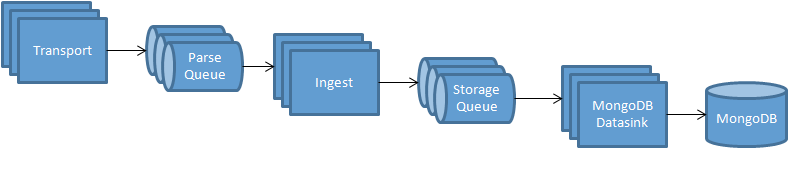


**Figure 3- Situation Data Warehouse DigitalEdge System**

The Warehouse System is the second DigitalEdge system, shown in Figure 3. Each box and cylinder in the diagram above represents a virtual DigitalEdge instance running in the AWS environment. The boxes and cylinders that are stacked indicate that one or more instance of the same type can run in parallel depending upon the configuration of the Warehouse system. The ability to create additional instances of the same type provides a way to horizontally scale the system to meet performance requirements for large amounts of input data.

* Transport - This instance type runs the *fedgov-cv-transport-udp* and *fedgov-cv-transport-listener* custom components. It receives and responds to traveler situation data ASN.1 UPER dialog messages. In addition, it handles vehicle and intersection situation data from the Clearinghouse system for storage into the MongoDB database. Additionally, the Transport processes AdvisorySituationData messages. When an AdvisorySituationData message is received, the transport decrypts and validates it to ensure that it is properly formed and within the specified service region of the system before placing it on the parse queue for processing. One or multiple Transport instances can be configured to run at once in the Warehouse.
* Parse Queue – This instance runs an ActiveMQ JMS Server and is responsible for storing the vehicle and intersection situation data until the Ingest instance picks it up for processing. This ensures that no messages are lost when the Ingest instance is busy. One or three of this type of instance can be configured to run in the Warehouse.
* Ingest – This instance type runs the *fedgov-cv-parser-j2735*, *fedgov-cv-processor-broadcast-instructions*, and *fedgov-cv-processor-geojson* custom components. It parses the vehicle, intersection, and traveler information situation data and maps extracted data fields into JSON records as configured in the appropriate DigtalEdge data model. If the message contains traveler information situation data and the distribution type is RSU, a broadcast instruction JSON structure will be added to the data record. If the message contains vehicle or intersection situation data, a GeoJSON structure will be added to the data record. One or multiple Ingest instances can be configured to run at once, or the Warehouse can be configured to scale the number of Ingest instances dynamically depending on the load of the system.
* Storage Queue - This instance type runs an ActiveMQ JMS Server and is responsible for storing vehicle, intersection, and traveler information situation data messages until the MongoDB data sink instance accepts them for processing. This ensures that no messages are lost when the Data sink instance is busy. One or three Storage Queues can be configured to run in the Warehouse system.
* MongoDB Datasink – This instance type runs the *fedgov-cv-datasink-mongodb* custom component, and it handles storage of the vehicle, intersection, and traveler information situation data messages into the MongoDB database. This instance creates the time-to-live and geospatial indexes for the “expireAt” and “region” fields of the data model. One or multiple MongoDB Datasinks can be configured to run at once, or they can be configured to scale up dynamically depending on the load of the system. If more than one MongoDB Datasinks exist, only the first instance will monitor and create the time-to-live and geospatial indexes.
* MongoDB – This instance runs the actual MongoDB database. The geospatial and time-to-live functionality provided by the database are used to meet the geospatial query and automatic removal requirements of the Connected Vehicle project.

### 3.4.3 Situation Data Processing Center (SDPC) System

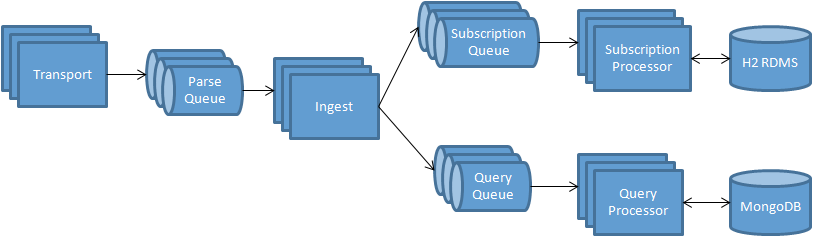


**Figure 4- Situation Data Processing Center DigitalEdge System**

The Processing Center System is the third DigitalEdge system, shown in Figure 3. Each box and cylinder in the diagram above represents a virtual DigitalEdge instance running in the AWS environment. The boxes and cylinders that are stacked indicate that one or more instance of the same type can run in parallel depending upon the configuration of the Warehouse system. The ability to create additional instances of the same type provides a way to horizontally scale the system to meet performance requirements for large amounts of input data.

* Transport - This instance type runs the *fedgov-cv-transport-listener* custom component. It receives vehicle and intersection situation data from the Clearinghouse system for storage into the MongoDB database. One or multiple Transport instances can be configured to run at once in the Processing Center.
* Parse Queue – This instance runs an ActiveMQ JMS Server and is responsible for storing the vehicle and intersection situation data until the Ingest instance picks it up for processing. This ensures that no messages are lost when the Ingest instance is busy. One or three of this type of instance can be configured to run in the Processing Center.
* Ingest – This instance type runs the *fedgov-cv-parser-j2735*and *fedgov-cv-processor-geojson* custom components. It parses the vehicle and intersection situation data and maps extracted data fields into JSON records as configured in the appropriate DigtalEdge data model. If the message contains vehicle or intersection situation data, a GeoJSON structure will be added to the data record. One or multiple Ingest instances can be configured to run at once, or the Processing Center can be configured to scale the number of Ingest instances dynamically depending on the load of the system.
* Storage Queue - This instance type runs an ActiveMQ JMS Server and is responsible for storing vehicle and intersection situation data messages until the MongoDB data sink instance accepts them for processing. This ensures that no messages are lost when the Data sink instance is busy. One or three Storage Queues can be configured to run in the Processing Center system.
* MongoDB Datasink – This instance type runs the *fedgov-cv-datasink-mongodb* custom component, and it handles storage of the vehicle, intersection, and traveler information situation data messages into the MongoDB database. This instance creates the time-to-live and geospatial indexes for the “expireAt” and “region” fields of the data model. One or multiple MongoDB Datasinks can be configured to run at once, or they can be configured to scale up dynamically depending on the load of the system. If more than one MongoDB Datasinks exist, only the first instance will monitor and create the time-to-live and geospatial indexes.
* MongoDB – This instance runs the actual MongoDB database. The geospatial and time-to-live functionality provided by the database are used to meet the geospatial query and automatic removal requirements of the Connected Vehicle project.

### 3.4.4 Distribution System



**Figure 5- Distribution DigitalEdge System**

The third and final DigitalEdge system is the Distribution System, shown in Figure 5. Each box and cylinder in the diagram above represents a virtual instance running in the AWS environment. The boxes and cylinders that are stacked indicate that one or more instance of the same type can be configured to run depending upon the configuration of the Distribution system. The ability to create additional instances of the same type provides a way to horizontally scale the system to meet the performance requirements.

* Transport - This instance type runs the *fedgov-cv-transport-udp* custom component, and it receives and responds to subscription and data requests from ASN.1 UPER dialog messages. When a DataSubscriptionRequest, RsuAdvisorySituationDataRequest, or IntersectionSituationDataRequest message is received, the Transport will decrypt and validate it to ensure that it is properly formed and within the specified service region of the system before placing it on the parse queue for processing. One or multiple Transport instances can be configured to run at once in the Distribution system.
* Parse Queue - This instance type runs an ActiveMQ JMS Server and is responsible for storing the subscription and data request messages until the Ingest instance picks them up for processing. This ensures that no messages are lost when the Ingest instance is busy. One or three Parse Queue instances can be configured to run in the Distribution system.
* Ingest - This instance type runs the *fedgov-cv-parser-j2735* custom component. It parses the subscription and data request messages and maps the extracted information into JSON data records as per the associated DigitalEdge data model. One or multiple Ingest instances can be configured to run at once, or the Distribution system can be configured to scale the number of Ingest instances dynamically depending on the load of the system.
* Subscription Queue - This instance runs an ActiveMQ JMS Server and is responsible for storing subscription request JSON data records until a Subscription Processor instance can accept those records for processing. This ensures that no messages are lost when the Processor instance is busy. One or three Subscription Queue instances can be configured to run in the Distribution system.
* Subscription Processor – This instance type runs the *fedgov-cv-processor-subscription* custom component. It adds new subscriptions or actively and passively removes subscriptions from the H2 RDBMS. Active removal pertains to an action initiated by a subscriber via the DataSubscriptionCancel ASN.1 UPER message. Passive removal pertains to expired subscriptions either by the subscription end time reached or by expiration of the subscriber certificate. When a new subscription is added, the Subscription Processor assigns the subscriber an ID between 10,000,000 and 99,999,999; this is also the name of the topic the subscriber will connect to on the Pub/Sub Server. Only one Subscription Processor is recommended, but more than one can be configured.
* H2 RDBMS – This is referring to the same instance described for the Clearinghouse system. The Subscription Processor places a new subscription and actively (initiated by the client) or passively (expired) removes the subscription from the H2 RDBMS instance.
* Query Queue - This instance type runs an ActiveMQ JMS Server and stores data request situation JSON records until a Query Processor instance accepts them for processing. This ensures that no messages are lost when a Query Processor instance is busy. One or three Query Queue instances can be configured to run in the Distribution system.
* Query Processor - This instance type runs the *fedgov-cv-datasink-query-processor* custom component. It queries the MongoDB database for vehicle and intersection situation data. After receiving the query response, the Query Processor packages the result into one or more RsuAdvisorySituationDataBundle or IntersectionSituationDataBundle ASN.1 UPER messages depending on the query type. It encrypts the messages using the requestor’s 1609.2 certificate, and sends it back to the requestor over UDP. One or multiple Query Processor instances can be configured to run at once, or the Distribution system can be configured to scale the number of Query Processor instances dynamically depending on the load of the system.
* MongoDB – This is referring to the same MongoDB database instance specified in the Warehouse system. The Query Processor connects to the MongoDB instance to perform queries for advisory and intersection situation data.

### 3.4.5 Standalone Applications

#### RSUAdvSitDataPoller

The RSUAdvSitDataPoller was developed to assist obtaining the advisory message and instructions on how to broadcast to the roadside unit where they are broadcast out to nearby vehicles. It also helps to test and demonstrate the ad-hoc query functionality of the SDW. The application was written in Java 1.6 and runs on the command line.

This application is located in the fedgov-cv-apps project and building the project produces the fedgov-cv-apps-<version>-jar-with-dependencies.jar file. Before running the application the settings.properties file needs to be configured and placed in the same folder as the jar file. Below is an example of the settings.properties file.

# The IP address of the situation data warehouse.

WarehouseIP=10.11.148.213

# The port number of the situation data warehouse.

WarehousePort=46754

# The port number on your localhost to receive warehouse response,

# must be different from the WarehousePort.

ResponsePort=46761

# The service region you are interested in advisory message(s).

NWLatitude=48.997519

NWLongitude=-125.595703

SELatitude=24.790723

SELongitude=-68.203125  
# The maximum age in minutes of data to return, this property is optional .

TimeBound=30

# The directory to write the output file on your localhost, this directory

# must be created before the application runs.

# SavePath=/tmp – example on a linux environment

SavePath=C:\\tmp

# The name of the output file.

Filename=out.txt

# The time in seconds between each poll requests. If this value is set too low

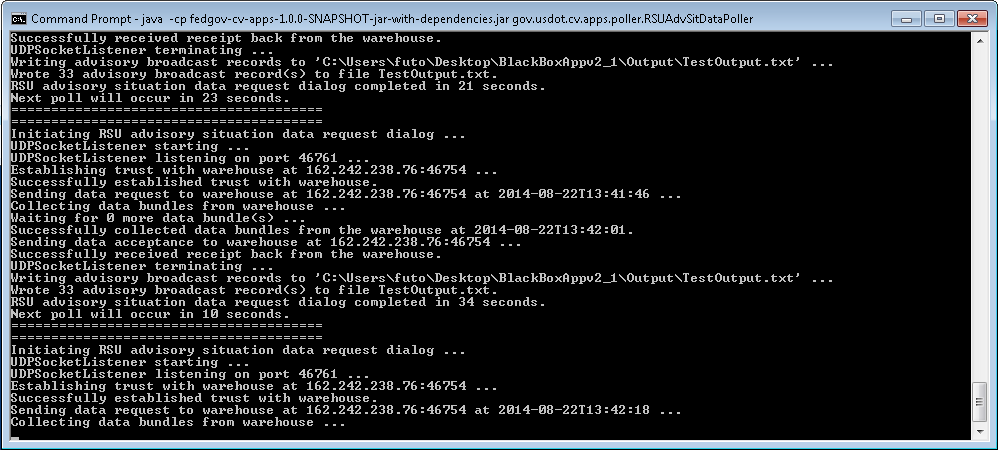
# the application will just poll immediately after finishing the current run sp

# set this value appropriately.

PollInterval=45

To run the RSUAdvSitDataPoller execute the following. Sample output appears in Figure 6.

java -cp fedgov-cv-apps-1.0.0-SNAPSHOT-jar-with-dependencies.jar gov.usdot.cv.apps.poller.RSUAdvSitDataPoller



**Figure 6- Advisory Situation Data Poller Test application**

The application will start up by loading in the configuration file and then submits an advisory situation data request to the SDW and continue to do so until the application is stopped. The result returned from the query will be written to the configured file with a single line representing an advisory message and its broadcast instruction. Below is an example of a single line output. The Payload property is the HEX encoded value of the traveler information message from the DSRC ASN.1 specification.

Version=0.5;Type=TIM;PSID=00008003;Priority=2;TxMode=ALT;TxChannel=178;TxInterval=1;DeliveryStart=07/21/2014, 20:06;DeliveryStop=07/28/2014, 20:06;Signature=true;Encryption=false;Payload=3081c68001108109000000000000003714830101a481ae3081ab800102a11ba119a0108004194fba1f8104ce45ce2382020a0681020006820102820207de830301c17084027d00850102a6108004194fc1988104ce45da4082020a008702016e880100a92430228002000ea21ca01aa31804040ce205a104040ada04f70404068004d60404034d0704aa3aa0383006a004800235293006a0048002010c3006a004800231283006a004800222113006a0048002010c3006a004800231203006a0048002221185021001;

#### IntersectionSitDataPoller

The IntersectionSitDataPoller was developed to test and demonstrate the ad-hoc intersection situation data query functionality of the SDW. The application was written in Java 1.6 and runs on the command line.

This application is located in the fedgov-cv-apps project and building the project produces the fedgov-cv-apps-<version>-jar-with-dependencies.jar file. Before running the application the settings.properties file needs to be configured and placed in the same folder as the jar file. Below is an example of the settings.properties file.

# The IP address of the situation data warehouse.

WarehouseIP=10.11.148.213

# The port number of the situation data warehouse.

WarehousePort=46754

# The port number on your localhost to receive warehouse response,

# must be different from the WarehousePort.

ResponsePort=46761

# The service region you are interested in intersection situation data.

NWLatitude=48.997519

NWLongitude=-125.595703

SELatitude=24.790723

SELongitude=-68.203125  
# The maximum age in minutes of data to return, this property is optional.

TimeBound=30

# The directory to write the output file on your localhost, this directory

# must be created before the application runs.

# SavePath=/tmp – example on a linux environment

SavePath=C:\\tmp

# The name of the output file.

Filename=out.txt

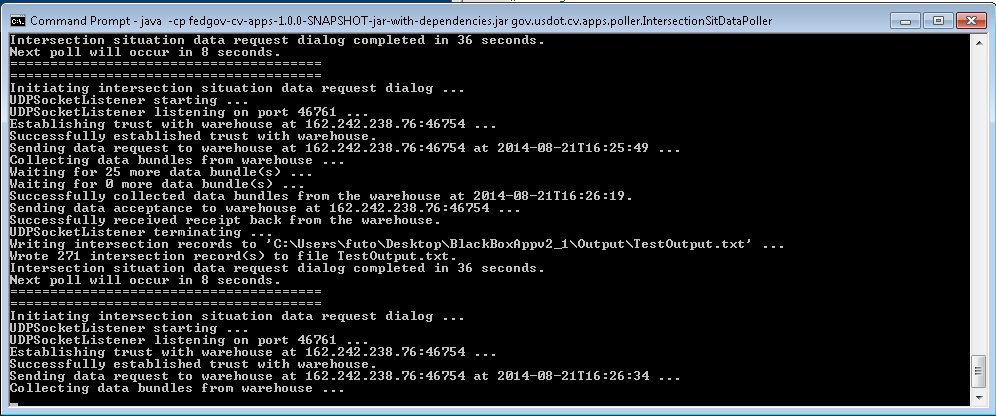
# The time in seconds between each poll requests. If this value is set too low

# the application will just poll immediately after finishing the current run sp

# set this value appropriately.

PollInterval=45

To run the IntersectionSitDataPoller execute the following. Sample output appears in Figure 7.

****java -cp fedgov-cv-apps-1.0.0-SNAPSHOT-jar-with-dependencies.jar gov.usdot.cv.apps.poller.IntersectionSitDataPoller

**Figure 7- Intersection Situation Data Poller Test Application**

The application will start up by loading in the configuration file and then submits an intersection situation data request to the SDW and continue to do so until the application is stopped. The result returned from the query will be written to the configured file with a single line representing the HEX encoded value of the intersection record. Below is an example of a single line output.

*3060a00a80010781010187020000a152a013800207dd81010c82010983010984011e85011ea13b801753616d706c6520496e74657273656374696f6e205b305d81040000035982010183022710840102a50a30088202010286022715860101870102*

# 4 Running the Systems

## 4.1 Initiate Forwarders

Initiate the forwarders as described in the CVCS-SSS Operations Manual.

## 4.2 Building the Systems

### 4.2.1 Upload Plugins

Please refer to the plugins tab of *CVCS Configuration Report 20170912.xlsx**[[2]](#footnote-2)* document for the full list of plugins required for the CVCS DigitalEdge Systems. Ensure that all the plugins have been uploaded into DigitalEdge using the DigitalEdge Management Console. Refer to the DigitalEdge Operations Guide[[3]](#footnote-3), Chapter 5, section *Managing the component repository* for instructions on uploading plugins to DigitalEdge.

### 4.2.2 Setup Data Models

For each CVCS system to be built (SDC, ORDS, etc.), please refer to the corresponding named System tab of the *CVCS Configuration Report 20170912.xlsx* document for the list of Data Models required for that system. Ensure that all required Data Models are uploaded to DigitalEdge. For instructions on uploading Data Models, refer to the DigitalEdge Configuration Guide[[4]](#footnote-4), Chapter 4, section *Upload and Download Data Models Between System.* After all required Data Models have been uploaded; open the Data Model using the DigitalEdge Data Model Editor to ensure that all Field Types and Translations match what is listed in the *CVCS Configuration Report 20170425.xlsx* document for each Data Model. Please refer to the DigitalEdge Configuration Guide4, Chapter 4, *Defining the Data Model* for detailed information about working with Data Models in DigitalEdge.

### 4.2.3 Build the System

For each CVCS system to be built, refer to the corresponding named System tab of the *CVCS Configuration Report 20170912.xlsx* document for the list of required plugins and system build parameters. For instructions on configuring and building a system, refer to the DigitalEdge Configuration Guide4, Chapter 5, *Building the System*. Please note that the systems should be built in the following order: SDC, ORDS, SDW, and finally SDPC.

Most plugin parameter values are static and can be entered into DigitalEdge System Builder exactly as they are shown in *CVCS Configuration Report 20170912.xlsx*, but the following values are dynamic and must be updated in DigitalEdge prior to building each system.

Table 1. Dynamic Plugin Parameter Values

|  |  |  |  |
| --- | --- | --- | --- |
| System | Plugin | Field | Value |
| SDC | CvUDPTransportService | forwader-host-name | IPv4 DNS address of SDC/ORDS Forwarder |
| SDW | JMSTopicListenerTransportService | target-host | IPv4 DNS address of PubSub Server |
| SDW | CvUDPTransportService | forwader-host-name | IPv4 DNS address of SDW Forwarder |
| SDPC | JMSTopicListenerTransportService | target-host | IPv4 DNS address of PubSub Server |

### 4.2.4 Start the System

After building a system, go to the Management Console, find the newly built system and press the start button. The systems cannot be started simultaneously, start one system at a time in the following order: SDC, then ORDS, then SDW, and finally SDPC. For instructions on using Management Console and starting a system, please refer to the DigitalEdge Operations Guide3, Chapter 5, *Managing DigitalEdge from the Management Console*.

### 4.2.5 Verify

When starting subsequent systems verify that the status of the system, that was just started, is OK before starting the next one. For details on system status, please see the DigitalEdge Operation Guide3, Chapter 6, *System Monitoring*.

## 4.3 Complete Forwarders

Complete the forwarders as described in the CVCS-SSS Operations Manual.

# 5 Monitoring

## 5.1 Management Console and System Monitor

To manage and monitor the DigitalEdge systems, please refer to the DigitalEdge Operations Guide3 Chapter 5, *Managing DigitalEdge from the Management Console* and Chapter 6, *System Monitoring.*

## 5.2 Warehouse Tools

For detailed information on the *Warehouse Tools Set*, please refer to the Warehouse Tools description in the CVCS-SSS Operations Manual.

## 5.3 Test Routine: Data Type ISD, VSD, etc.

For detailed information on how to use the Warehouse Tools Set to perform the test routine, please refer to the Test Routine section of the CVCS-SSS Operations Manual.

## 5.4 Notification Scheduling

For a detailed description of the System Monitor, how it monitors the systems, and its configuration options, please refer to the CVCS-SM Operations Manual.

# 6 Updating OSS Licensing

1. Install new OSS liscense (“ossinfo” file) into the base folder where the OSS executable is located
2. Restart OSS Studio
3. Compile the ASN to Java using OSS Studio
4. Delete the old compiled code from the fedgov-cv-asn1-generated-j2735 Git repository: src/main/java/gov/usdot/asn1/generated/j2735
5. Copy the new compiled code to the same location in the Git repository
6. Build the fedgov-cv-asn1-generated-j2735 project
7. Build the rest of the projects that have dependencies on the generated code in the following order:
   1. fedgov-cv-asn1
   2. fedgov-cv-common
   3. fedgov-cv-parser-j2735
   4. fedgov-cv-processor-broadcast-instructions
   5. fedgov-cv-transport-listener
   6. fedgov-cv-transport-udp

The build process will generate the DigitalEdge plug-ins for each of these projects.

1. Upload the plug-ins to the Plug-Ins section of the DE Management Console. This will overwrite the previous version of the plug-ins.
2. In the System Builder, build a new system for each of the DDS systems.
3. Start the newly built systems using the Management Console.
4. When the new systems are started, transfer all publically facing domain names to point to the new systems using the AWS Route53 service.

# 7 Updating SSL Certificate

The CV Environment uses commercially signed certificates to communicate securely. When these certificates expire, they need to be replaced with newly signed certificates for the environment to continue to function.

1. Obtain new signed certificates.
2. Download the old Java keystores from the AWS S3 locations.
3. Follow standard Java keytool usage to remove the old signed certificates from the keystores.
4. Follow standard Java keytool usage to load the new signed certificates to the keystore.
5. Replace the old Java keystores in AWS S3 with the newly created Java keystore.
6. In the System Builder, build a new system for each of the DDS systems.
7. Start the newly built systems using the Management Console.
8. When the new systems are started, SCP a copy of the new keystore to both the SDC and SDW transport EC2 instances to the same location as defined in the CvWebSocketTransportService keystore-path parameter for each system. This value can be found by viewing the system in System Builder.
9. Using the DE Management console, for each system, restart the Transport:
   1. Select the system from the “Systems” list
   2. Double click on the transport process group in the “Process Groups” list at the bottom
   3. Select the transport node on the list that pops up
   4. Under the “Actions” dropdown menu, select “Restart”
10. Merge the PubSub’s ActiveMQ keystore and trust stores into the new keystore and truststore:
    1. SCP two copies of the new keystore to the PubSub server of the SDC. One copy will be the keystore for ActiveMQ and the other will be the truststore.
    2. The old keystore/truststore are located at /usr/local/apache-activemq/conf/external-ssl-keystore and /usr/local/apache-activemq/conf/ssl-truststore respectively.
    3. Export their certificates using standard Java keytool functionality.
    4. Load the certificates from the old keystore into the new keystore using standard Java keytool functionality.
    5. Load the certificates from the old truststore into the new truststore using standard Java keytool functionality.
    6. Replace the old keystore and truststore with the newly merged keystore and truststore.
    7. Restart ActiveMQ
11. Transfer all publically facing domain names to point to the new DDS systems using the AWS Route53 service.

# 8 Troubleshooting

## 8.1 Ticketing System

If a problem is detected, visit the Samanage portal and view the current system status by opening the incident report indicated on the alert email sent out.

<https://cvcs.samanage.com/login>

## 8.2 Uptime Dashboard

Go to CVCS monitoring dashboard to view the system status. When this is available, this should be the first step in troubleshooting. In the meantime, DigitalEdge console, and AWS console and Rackspace console should be used for reference.

## 8.3 DigitalEdge Console

One way to monitor the state of each system is to go to the Tenant Console <https://default.tms.aws-dev.deleidos.com/tenantconsole/>. Verify that the status of a system is OK; if in the Warning state please refer to the DigitalEdge Operations Guide3 under the sections *System Monitoring->Viewing system status* and *Troubleshooting*.

Another way to monitor each system is in DigitalEdge System Monitor <https://default.tms.aws-dev.deleidos.com/systemmonitor/>. Generally if there is zero for the Average Throughput or a very steep incline, the system is in a bad state. For more information please refer to the DigitalEdge Operations Guide3 under the section *System Monitoring->Checking system metrics*

## 8.4 AWS Console

In AWS Console <https://console.aws.amazon.com/console/home>, instances can be monitored by checking the Instance State of each Instance to verify that it is running. If an instance is in the warning state or if there is a scheduled downtime please refer to either the support center <https://console.aws.amazon.com/support/home?region=us-east-1#/> or the documentation for ec2 instances <https://aws.amazon.com/documentation/ec2/>.

## 8.5 Rackspace Console

Login to Rackspace console <https://mycloud.rackspace.com/> to verify that all of the servers are green. If there are any warning signs (not showing up green) or if there is any scheduled downtime, please click on the *Support* link on the console.

## 8.6 Maintenance

For maintenance of the systems, please refer to Appendix B: Maintenance Plan.

# 9 DigitalEdge Data Models

This section provides a detailed discussion of the configuration of data models used for the DigitalEdge systems that comprise the SDC, SDW, and SDPC. The data models were created using the DigitalEdge Data Model Editor tool. There are no operational requirements associated with the models, as they are part of the deployed systems. This section is informational only.

A DigitalEdge data model specifies how the data looks, how it is enriched, how input is mapped to it, and how dimensions are used to enrich the data. The data model includes:

* Input model – A simple JSON-based model that specifies how the data will look when it enters the system and is mapped into a normalized data model. The input model can support multiple versions simultaneously.
* Data source mapping – Specifications for parsing the incoming data and mapping each field to the input model.
* Enrichments – Rules for enhancing the data with additional context and meaning. Enrichments can be created with data from a secondary database, with a regex formula, or a mathematical equation.

See the DigitalEdge *Overview Guide*, chapter on *Data Model*, for a more detailed explanation of the data model components.

Two DigitalEdge GUI tools are used to create and maintain a data model:

* Data Model Editor – Used to specify fields to parse and extract from data sources, to create translation statements for mapping data to the input model, and for specifying enrichments to apply to the data
* Table Manager – Used to specify the dimension tables and fields that are used in dimensional enrichment definitions

See the DigitalEdge *Configuration Guide*, chapter on *Defining the Data Model*, for the procedures to use these tools when specifying a data model.

A data model is needed to support the situation data deposit and data request ASN.1 UPER messages that enter the Clearinghouse, Warehouse, and Distribution system. The data model is used internally in the system to represent the ASN.1 UPER message and is used to provide the required data deposit, subscription, and data request functionalities of the SDC, SDW, and SDPC.

## 9.1 Vehicle Situation Data Model

The vehicle situation data model is used to represent the VehSitDataMessage ASN.1 UPER message within the Clearinghouse system and Warehouse system. The data model is in JSON format and is used internally to help route the vehicle situation data to the appropriate subscriber in the Clearinghouse system and to store in the MongoDB database in the Warehouse system.

The following Connected Vehicle components must be uploaded to the DigitalEdge platform before this data model can be constructed.

* fedgov-cv-parser-j2735
* fedgov-cv-processor-geojson

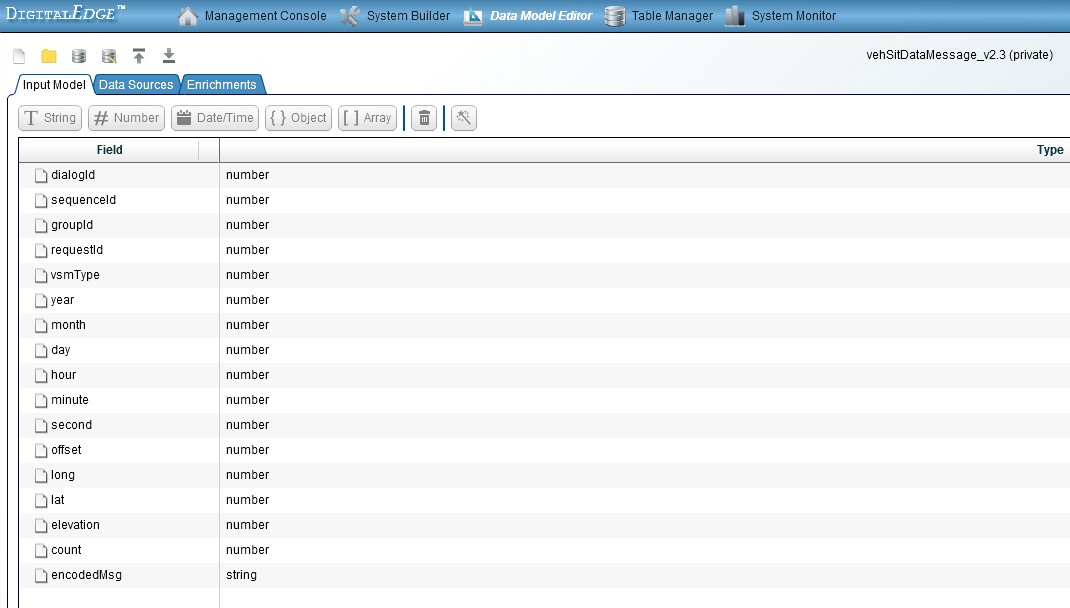


Figure 8- Vehicle Situation Data Model Input Fields

The data model contains a number of fields, described below.

* dialogId – The dialog ID of the vehicle situation data deposit dialog.
* sequenceId – The sequence ID of the vehicle situation data deposit dialog.
* groupId – The group ID of the vehicle situation data deposit dialog.
* requestId – The request ID of this vehicle situation data deposit.
* vsmType – The type of vehicle situation data message.
* year – The year the vehicle situation data message was captured.
* month – The month the vehicle situation data message was captured.
* day – The day the vehicle situation data message was captured.
* hour – The hour the vehicle situation data message was captured.
* minute – The minute the vehicle situation data message was captured.
* second – The second the vehicle situation data message was captured.
* offset – The offset for the vehicle situation data message.
* long – The longitude of the vehicle when the message was captured.
* lat – The latitude of the vehicle when the message was captured.
* elevation – The elevation of the vehicle when the data was captured.
* count – The number of vehicle situation records in the message.
* encodedMsg – The base64-encoded value of the vehicle situation data.

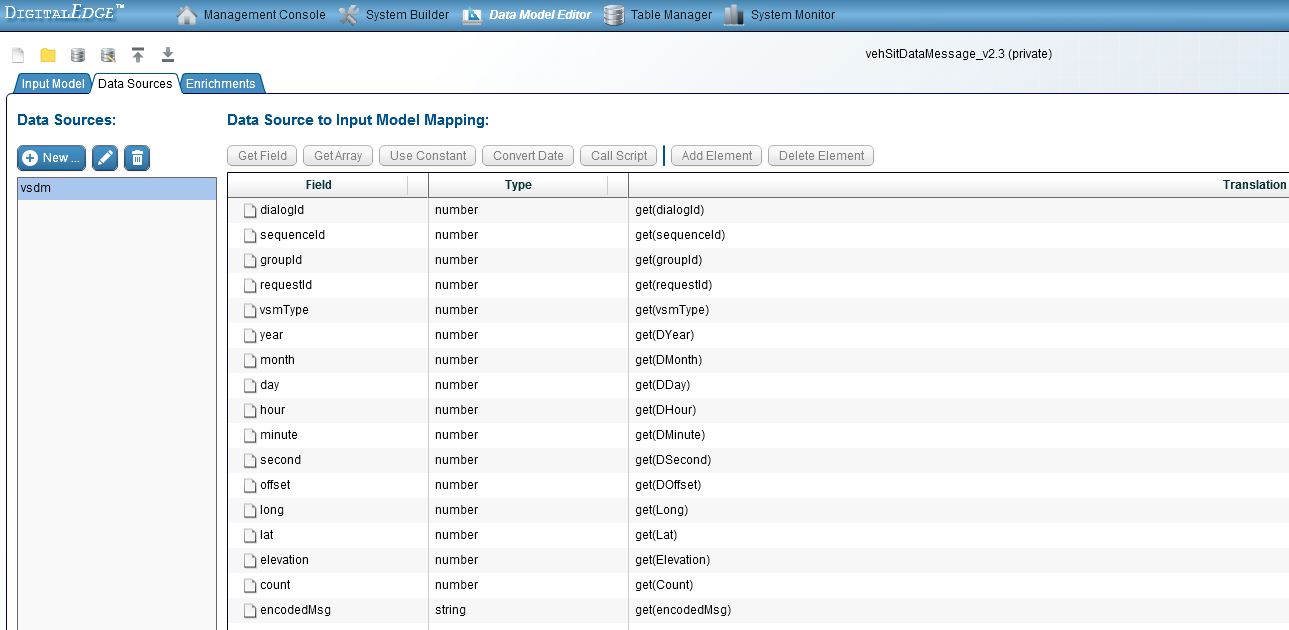


Figure 9- Vehicle Situation Data Model Data Sources

Once the vehicle situation data model fields are defined, they must be mapped to a particular value by using the “Get Field” button so the DigitalEdge ingest pipeline knows how to fill in the values for each field. Internally, the DigitalEdge ingest pipeline will decode the vehicle situation data UPER message, extract the needed values, and store them in a map. The pipeline will then use this map to create and populate the data model.

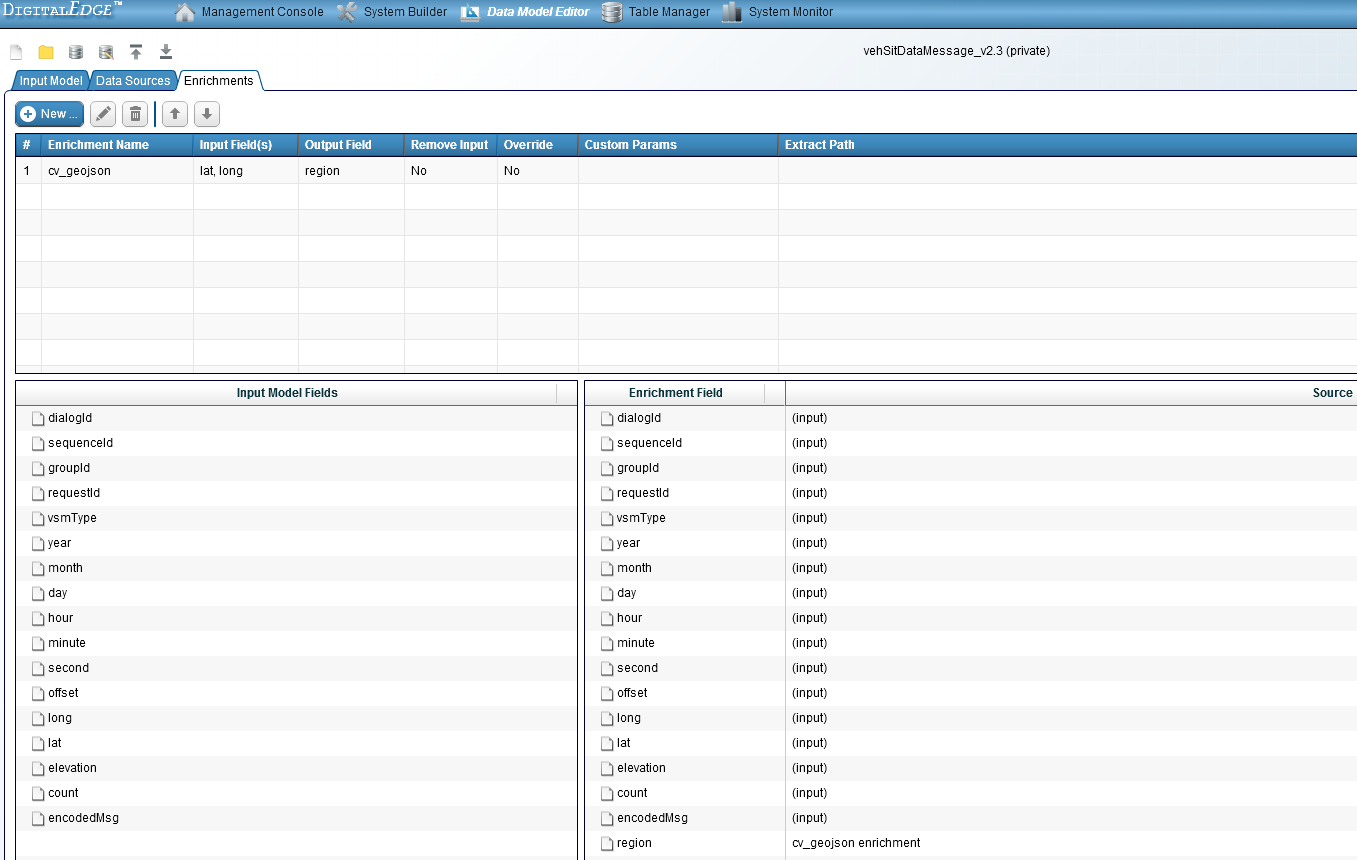


Figure 10- Vehicle Situation Data Model Enrichments

Once the data model is created, a GeoJSON enrichment will be performed to create a new field called “region” that follows the GeoJSON structure. In order to create this structure, the latitude and longitude must be supplied as the input fields so the enrichment processor knows how to add them to the data model. The field is used by the SDW and SDPC to build a geospatial index to support geospatial queries.

Two special fields, “createdAt” and “expireAt” (not shown here), are created when the vehicle situation data model is stored in MongoDB. The former serves to keep track of when the data was stored while the latter is used to build a time to live index so the database will remove it when the time arrives.

Here is a sample vehicle situation data model when it is stored in the MongoDB database.

{

"\_id" : ObjectId("53d3c3deba11b0e435a17e05"),

"standardHeader" : {

"uuid" : "64844995-4148-4f43-a625-144b63053ef5",

"source" : "UNKNOWN",

"accessLabel" : "UNCLASSIFIED",

"modelName" : "vehSitDataMessage",

"modelVersion" : "1.4"

},

"dialogId" : 154,

"sequenceId" : 5,

"requestId" : -1985229329,

"vsmType" : 2,

"year" : 2014,

"month" : 7,

"day" : 25,

"hour" : 14,

"minute" : 6,

"second" : 33472,

"long" : -83.0458529,

"lat" : 42.3279744,

"elevation" : 1492,

"region" : {

"type" : "Point",

"coordinates" : [

-83.0458529,

42.3279744

]

},

"count" : 1,

"encodedMsg" : "MIGmgAIAmoEBBYIEiavN74MCB4CkOKAnoBWAAgfegQEHggEZgwEOhAEGhQMAgsCBBM6ANV+CBBk6vICDAgXUggEBowqDCAAAAAAAAAAApVUwU4AEOe8NgqEVgAIH3oEBB4IBGYMBDoQBBoUDAILAohCABBk6vICBBM6ANV+CAgXUoyKAAuHWgQITG4IBf4MHAGYH0TIAwoQCCAClCIACALCBAgHGhgIZnw==",

"createdAt" : ISODate("2014-07-26T15:06:06.067Z"),

"expireAt" : ISODate("2015-01-26T15:06:06.067Z")

}

## 9.2 Intersection Situation Data Model

The intersection situation data model is used to represent the IntersectionSituationData ASN.1 UPER message within the Clearinghouse system and Warehouse system. The data model is in JSON format and is used internally to help route the intersection situation data to the appropriate subscriber in the Clearinghouse system and to store in the MongoDB database to support query requests from third party applications in the Warehouse system.

The following Connected Vehicle components must be uploaded to the DigitalEdge platform before this data model can be constructed.

* fedgov-cv-parser-j2735
* fedgov-cv-processor-geojson

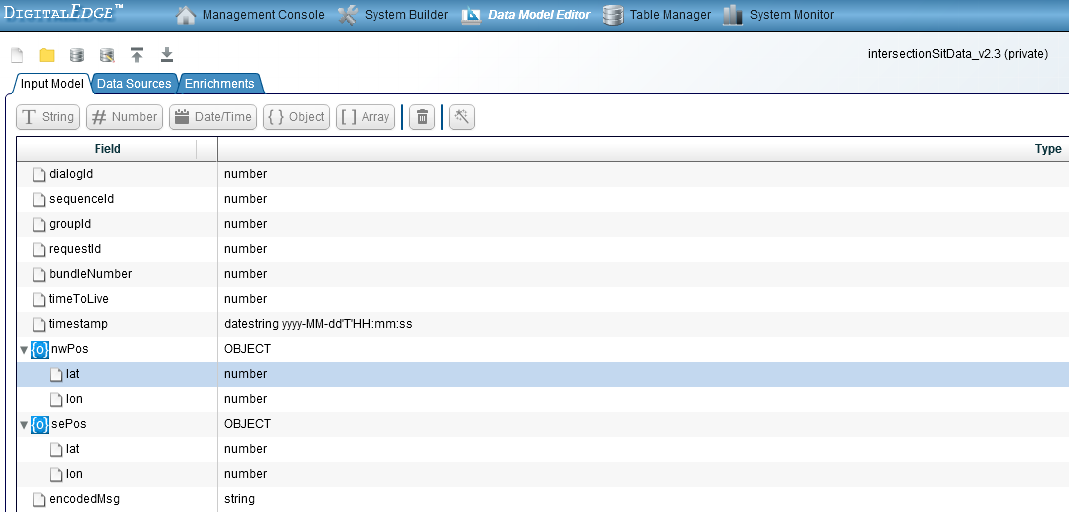


Figure 11- Intersection Situation Data Model Input Fields

The data model contains a number of fields described below.

* dialogId – The dialog ID of the intersection situation data deposit dialog.
* sequenceId – The sequence ID of the intersection situation data deposit dialog.
* groupId – The group ID of the intersection situation data deposit dialog.
* requestId – The request ID of this intersection situation data deposit.
* bundleNumber – The bundle number of this intersection situation data deposit.
* timeToLive – The amount of time the intersection situation data will be stored in the SDW and SDPC. See the latest SEMI.asn spec to get a mapping of the value to the amount of time in minutes(s).
* timestamp – The SPaT UTC timestamp stored in the intersection situation data.
* nwPos & sePos – The northwest and southeast coordinates, the service region that the intersection situation data covers.
* encodedMsg – The base64-encoded value of the intersection situation data.

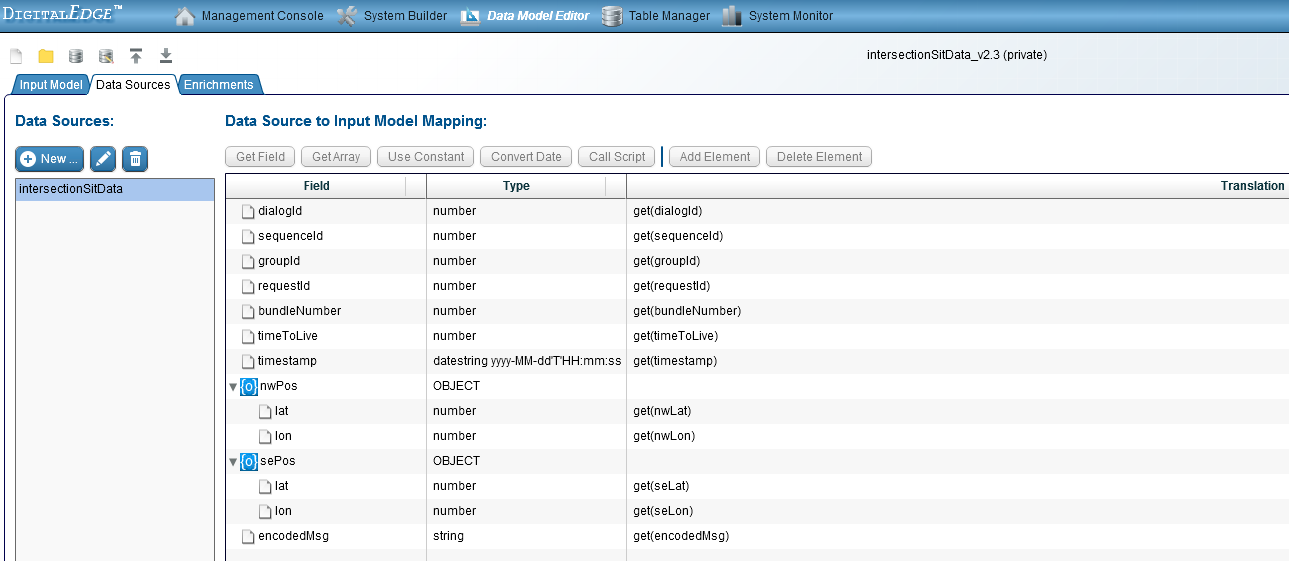


Figure 12- Intersection Situation Data Model Data Sources

Once the intersection situation data model fields are defined, they must be mapped to particular values by using the “Get Field” button so the DigitalEdge ingest pipeline knows how to fill in the values for each field. Internally, the DigitalEdge ingest pipeline will decode the intersection situation data UPER message, extract the needed values, and store them in a map. The pipeline will then use this map to create and populate the data model.

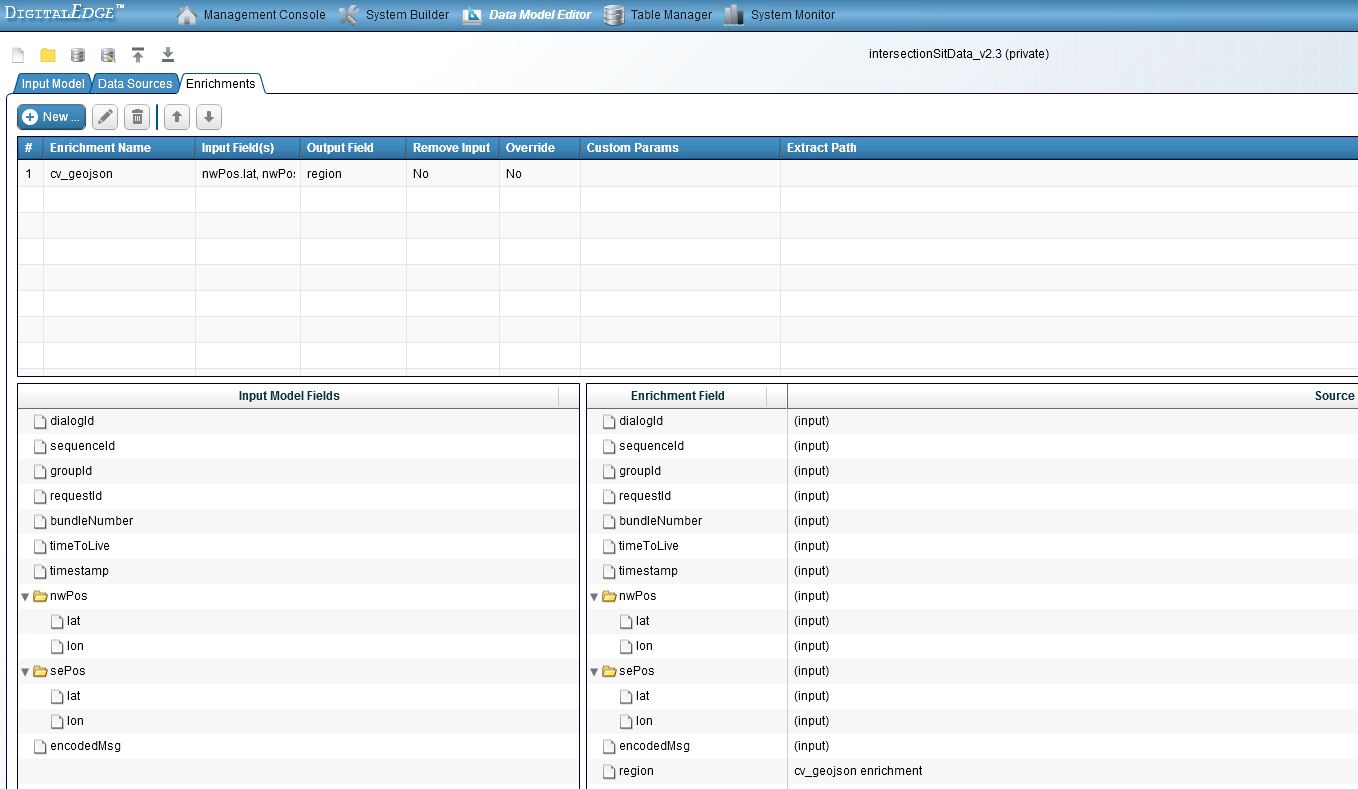


Figure 13- Intersection Situation Data Model Enrichments

Once the data model is created, a GeoJSON enrichment will be performed to create a new field called “region” that follows the GeoJSON structure. In order to create this structure, the latitude and longitude of the northwest and southeast corners must be supplied as the input fields so the enrichment processor knows how to add it to the data model. The field is used by the SDW and SDPC to build a geospatial index to support geospatial queries.

Two special fields, “createdAt” and “expireAt” (not shown here), are created when the intersection situation data model is stored in MongoDB. The former serves to keep track of when the data was stored, while the latter is used to build a time to live index so the database will remove it when the time arrives. Also, the timestamp field is converted from a string type into the MongoDB ISODate type when it is stored to support datetime queries.

Here is a sample intersection situation data model that is stored in the MongoDB database.

{

"\_id" : ObjectId("53f4eb0b6e95b0e4b10e7eb7"),

"standardHeader" : {

"uuid" : "c2034b3e-0d18-4978-9fc6-8f4bb6fe65d8",

"source" : "UNKNOWN",

"accessLabel" : "UNCLASSIFIED",

"modelName" : "intersectionSitData",

"modelVersion" : "1.2"

},

"dialogId" : 162,

"sequenceId" : 5,

"requestId" : 1,

"bundleNumber" : 1,

"timeToLive" : 1,

"timestamp" : ISODate("2013-12-09T09:30:30Z"),

"nwPos" : {

"lat" : 43,

"lon" : -85

},

"sePos" : {

"lat" : 41,

"lon" : -82

},

"encodedMsg" : "MIGTgAIAooEBBYIEAAAAAYMBAYQBAaUcoAyABBmhR4CBBM1WB4ChDIAEGHAagIEEzx/LAKZgoAqAAQeBAQGHAgAAoVKgE4ACB92BAQyCAQmDAQmEAR6FAR6hO4AXU2FtcGxlIEludGVyc2VjdGlvbiBbMF2BBAAAA1mCAQGDAicQhAECpQowCIICAQKGAicVhgEBhwEC",

"region" : {

"type" : "Polygon",

"coordinates" : [

[

[

-85,

43

],

[

-82,

43

],

[

-82,

41

],

[

-85,

41

],

[

-85,

43

]

]

]

},

"createdAt" : ISODate("2014-08-20T18:38:03.491Z"),

"expireAt" : ISODate("2014-08-20T19:08:03.492Z")

}

## 9.3 Traveler Information Data Model

The traveler information data model is used to represent the AdvisorySituationData ASN.1 UPER message within the Warehouse system. The data model is in JSON format and is used internally to store in the MongoDB database to support query requests from third party applications.

The following Connected Vehicle components must be uploaded to the DigitalEdge platform before this data model can be constructed.

* fedgov-cv-parser-j2735
* fedgov-cv-processor-broadcast-instructions
* fedgov-cv-processor-geojson



Figure 14- Traveler Information Data Model Input Fields

The data model contains a number of fields described below.

* dialogId – The dialog ID of the traveler information data deposit dialog.
* receiptId – A unique ID that is generated by DigitalEdge to represent a receipt for a single traveler information message.
* sequenceId – The sequence ID of the traveler information data deposit dialog.
* groupId – The group ID of the traveler information data deposit dialog.
* requestId – The request ID of this traveler information data deposit.
* recordId – The record ID of this traveler information data deposit. This ID is used to replace a traveler information message in the MongoDB database if one exists with the same record ID.
* timeToLive – The amount of time the traveler information data will be stored in the SDW and SDPC. See the latest SEMI.asn spec to get a mapping of the value to the amount of time in minutes(s).
* nwPos & sePos – The northwest and south east coordinates, the service region that the traveler information data covers.
* advisoryDetails – The content of the traveler information message.
* encodedMsg – The base64-encoded value of the traveler information data.

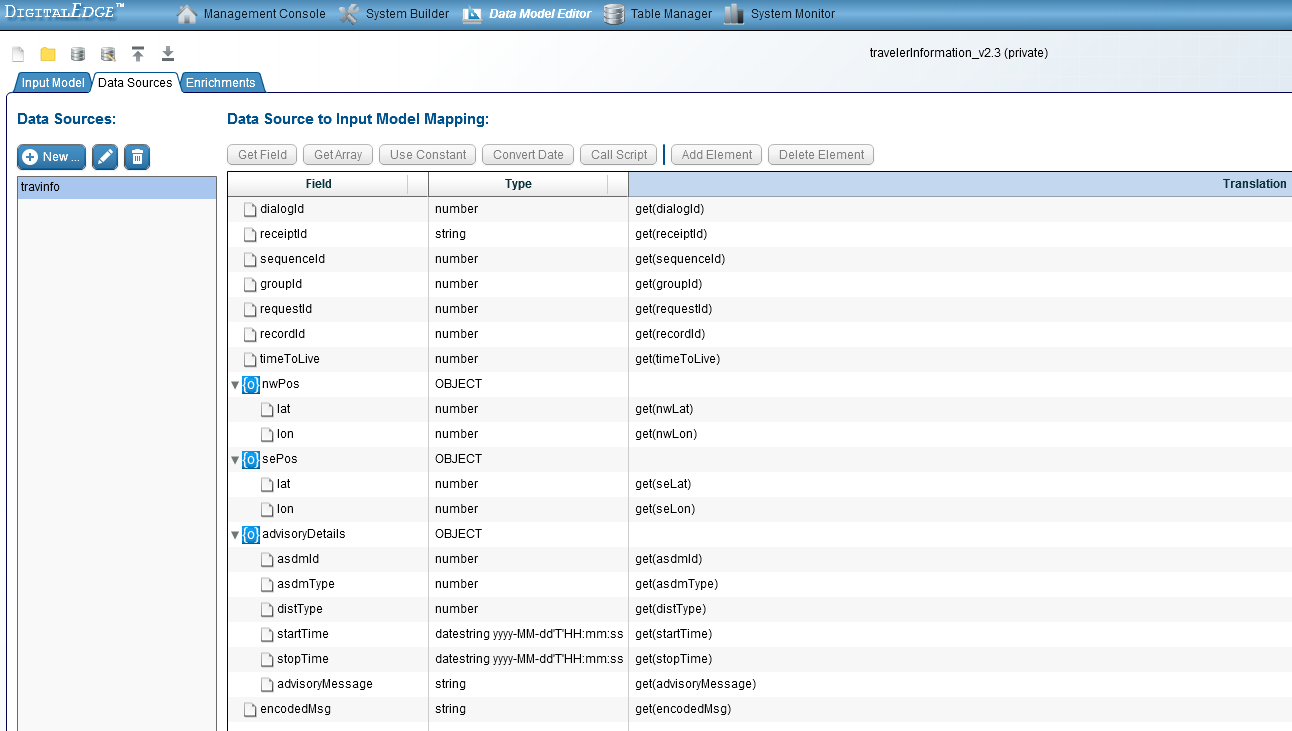


Figure 15- Traveler Information Data Model Data Sources

Once the traveler information data model fields are defined, they must be mapped to particular values by using the “Get Field” button, so the DigitalEdge ingest pipeline knows how to fill in the values for each field. Internally, the DigitalEdge ingest pipeline will decode the traveler information data BER message, extract the needed values, and store them in a map. The pipeline will then use this map to create and populate the data model.

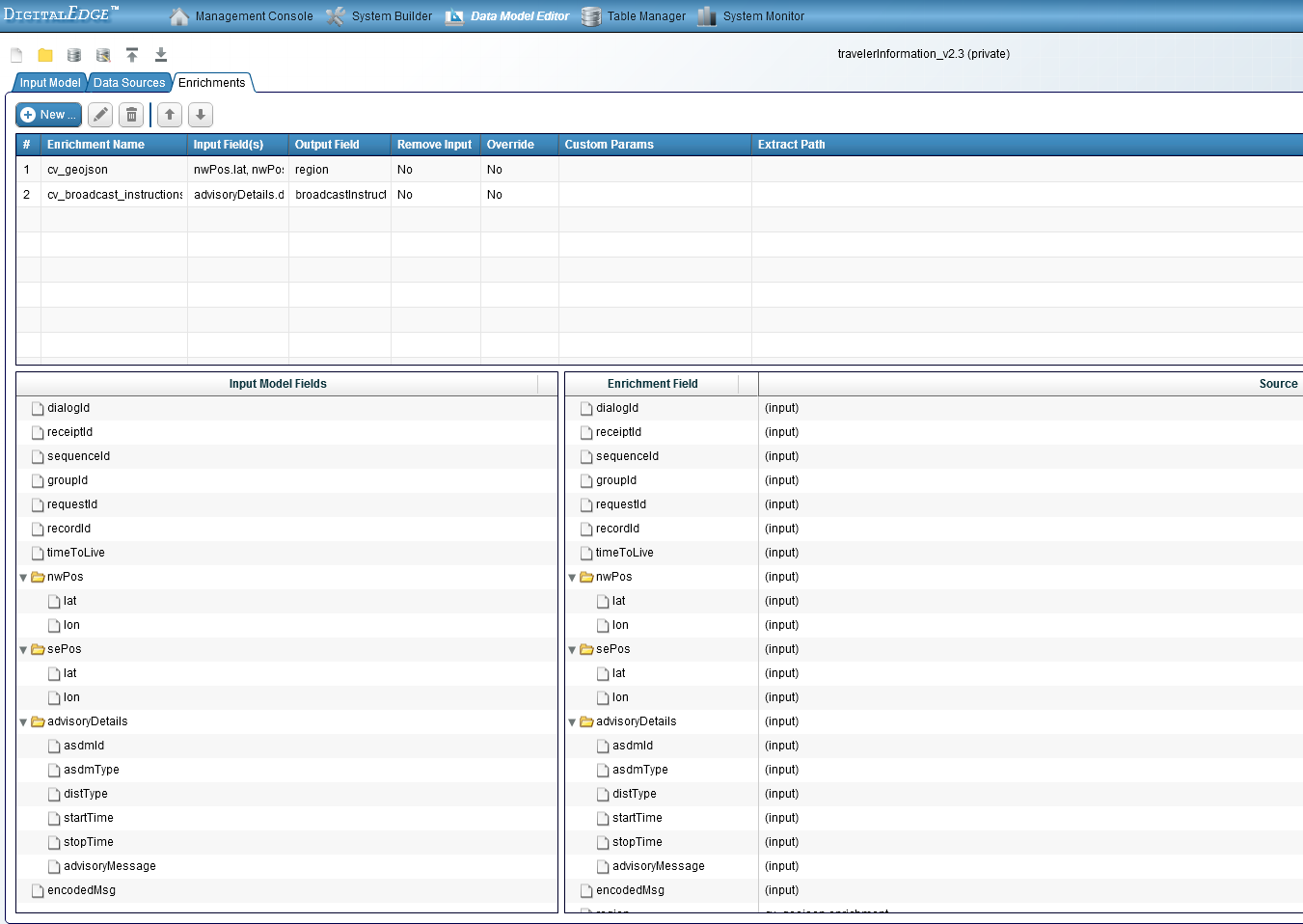


Figure 16- Traveler Information Data Model Enrichments

Once the data model is created, a GeoJSON enrichment and a broadcast instruction enrichment will be performed to create the new fields “region” and “broadcastInstructions”.

The GeoJSON enrichment will create a new field called “region” that follows the GeoJSON structure. In order to create this structure, the latitude and longitude of the northwest and southeast corners must be supplied as the input fields so the enrichment processor knows how to add it to the data model. The field is used by the SDW and SDPC to build a geospatial index to support geospatial queries.

The broadcast instructions enrichment will create a new field called “broadcastInstructions” which contains instructions to roadside units on how to broadcast the traveler information message to nearby vehicles. Broadcast instructions will only be added if the traveler information distribution type is RSU. If the distribution type is not RSU, the “broadcastInstructions” field will not be added to the data model.

Two special fields, “createdAt” and “expireAt”(not shown here), are created when the traveler information data model is stored in MongoDB. The former serves to keep track of when the data was stored while the latter is used to build a time to live index so the database will remove it when the time arrives.

Here is a sample traveler information data model that gets stored in the MongoDB database.

{

"standardHeader" : {

"uuid" : "3f2935bc-0dc8-4ad6-8e2b-195d17d6b4a7",

"source" : "UNKNOWN",

"accessLabel" : "UNCLASSIFIED",

"modelName" : "travelerInformation",

"modelVersion" : "1.4"

},

"dialogId" : 156,

"receiptId" : "7897851e-5d27-42a6-8165-2c12b0d4fa56",

"sequenceId" : 5,

"requestId" : 1001,

"timeToLive" : 1,

"nwPos" : {

"lat" : 43,

"lon" : -85

},

"sePos" : {

"lat" : 41,

"lon" : -82

},

"advisoryDetails" : {

"asdmId" : 5555,

"asdmType" : 2,

"distType" : 1,

"advisoryMessage" : "3081c68001108109000000000000003714830101a481ae3081ab800102a11ba119a0108004194fba1f8104ce45ce2382020a0681020006820102820207de830301c17084027d00850102a6108004194fc1988104ce45da4082020a008702016e880100a92430228002000ea21ca01aa31804040ce205a104040ada04f70404068004d60404034d0704aa3aa0383006a004800235293006a0048002010c3006a004800231283006a004800222113006a0048002010c3006a004800231203006a0048002221185021001"

},

"encodedMsg" : "MIIBCYACAJyBAQWCBAAAA+mEAQGlHKAMgAQZoUeAgQTNVgeAoQyABBhwGoCBBM8fywCmgdiABAAAFbOBAQKCAQGFgckwgcaAARCBCQAAAAAAAAA3FIMBAaSBrjCBq4ABAqEboRmgEIAEGU+6H4EEzkXOI4ICCgaBAgAGggECggIH3oMDAcFwhAJ9AIUBAqYQgAQZT8GYgQTORdpAggIKAIcCAW6IAQCpJDAigAIADqIcoBqjGAQEDOIFoQQECtoE9wQEBoAE1gQEA00HBKo6oDgwBqAEgAI1KTAGoASAAgEMMAagBIACMSgwBqAEgAIiETAGoASAAgEMMAagBIACMSAwBqAEgAIiEYUCEAE=",

"region" : {

"type" : "Polygon",

"coordinates" : [

[

[

-85,

43

],

[

-82,

43

],

[

-82,

41

],

[

-85,

41

],

[

-85,

43

]

]

]

},

"broadcastInstructions" : {

"type" : 2,

"psid" : 32771,

"priority" : 2,

"txMode" : 1,

"txChannel" : 5,

"txInterval" : 1,

"deliveryStart" : "2014-08-18T18:32:17",

"deliveryStop" : "2014-08-25T18:32:17",

"signature" : true,

"encryption" : false

},

"createdAt" : ISODate("2014-08-18T18:32:18.393Z"),

"expireAt" : ISODate("2014-08-18T19:02:18.393Z")

}

# 10 Configuring DigitalEdge Systems

This section discusses the configurations for each of the three DigitalEdge systems- Clearinghouse, Warehouse, and Distribution. Similar to the data models in Appendix A, this section is informational as there are no operational requirements beyond the initial configuration and building of the systems.

## 10.1 Configure the Clearinghouse System using System Builder

DigitalEdge includes a System Builder application for construction of systems. For detailed information about building systems, refer to the DigitalEdge Configuration Guide. The Clearinghouse system runs in the AWS us-east-1 service region and us-east-1c availability zone. The system size is medium and the software version is a release cut on September 15, 2016. The system itself consists of the two data models described in Section 8, four data transports, and a data sink.

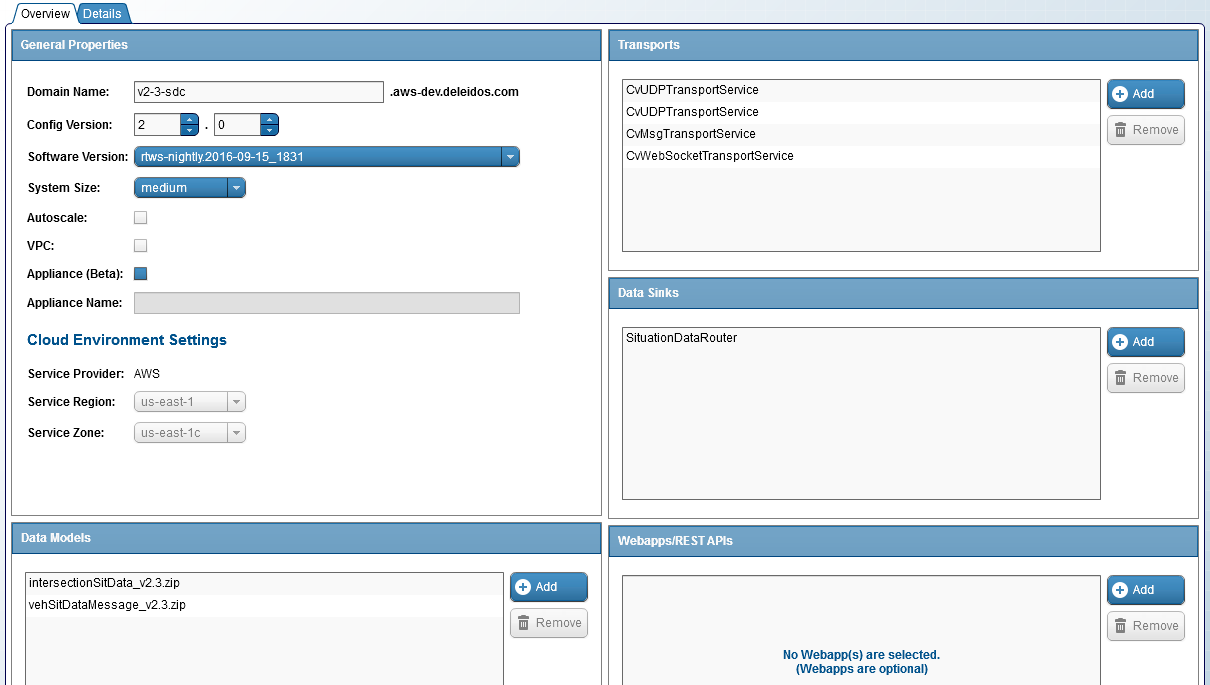


Figure 17- Clearinghouse System Components

### 10.1.1 Adding the Vehicle Situation Data Transports

The CvUDPTransportService provides the mechanism for receiving UDP data messages from the forwarder running in Rackspace.com. The user needs to add the two transports in the System Builder application and then configure specific parameters for the transport. The differences between the two transports are that one should have the message-format parameter configured for unencrypted messages and the other should be configured for encrypted, and because of this, they should be configured with different forwarder-port and listen-port values. The following parameters are used.

* forwarder-host-name: Host name or IP address of Rackspace forwarder.
* forwarder-port: Port of Rackspace forwarder.
* input-format: Input source associated with the DigitalEdge data model (vsdm for vehicle situation data).
* listen-port: Listen port for messages not coming through the Rackspace forwarder.
* low-volume-flush-strategy: Allows for the system to push along low volumes of data quicker.
* max-packet-size: Maximum packet size of incoming UDP packets (default 65535).
* max-time-between-flush: Fine tuning of how long the system waits to push data.
* message-format: Format of incoming messages- DSRCJ2735 (unencrypted) or IEEE1606.2 (encrypted).
* receipts-topic: JMS topic for output of the processed packets (cv-receipts is used for the clearinghouse system).
* svc-northwest-latitude: Latitude of northeast corner of the SDC geographic region.
* svc-northwest-longitude: Longitude of northeast corner of the SDC geographic region.
* svc-southeast-latitude: Latitude of southeast corner of the SDC geographic region.
* svc-southeast-longitude: Longitude of southeast corner of the SDC geographic region.



Figure 18- Vehicle Situation Data Transport Parameters

### 10.1.2 Adding the Intersection Data Message Transport

The intersection data transport receives and processes intersection data messages destined for the SDC. The DigitalEdge transport for this is the CvMsgTransportService. As with the previous transport, the user adds this in the System Builder application. This transport has four configurable parameters as follows.

* input-format: Input source associated with the DigitalEdge data model (intersectionSitData for intersection data).
* low-volume-flush-strategy: Allows for the system to push along low volumes of data quicker.
* max-time-between-flush: Fine tuning of how long the system waits to push data.
* message-type: Type of message to process within the intersection data transport (intersectionSitDataDep for this transport).



Figure 19- Intersection Data Transport Parameters

### 10.1.3 Adding the SituationDataRouter Data Sink

The SituationDataRouter is a process that runs in every Router instance and it is responsible for finding one or more subscriptions that match the vehicle and intersection situation data, and places the data on the appropriate subscriber topic for consumption.

Go to the “Data Sinks” panel and click “Add”. Select the SituationDataRouter data sink and click “OK”. You will be presented with a modal window to enter in parameters for the data sink.

* database-filter-table-name: This is the name of the subscription filter table which contains criteria for a subscriber’s vehicle or intersection subscription.
* database-subscriber-table-name: This is the name of the subscriber table which contains information about the subscriber such as the subscriber ID, its 1609.2 certificate, and the subscriber’s destination IP Address and Port for sending a subscription response.
* models-to-filter: This is a comma-delimited list of input data models that the data sink supports for subscriptions.

For the database-filter-table-name and database-subscriber-table-name parameters, choose one that is not already used by another Clearinghouse system. Note: the DigitalEdge platform lets you create multiple Clearinghouse systems to support development, testing, and operations.

For the models-to-filter parameter, enter the vehicle and intersection situation data model names here.

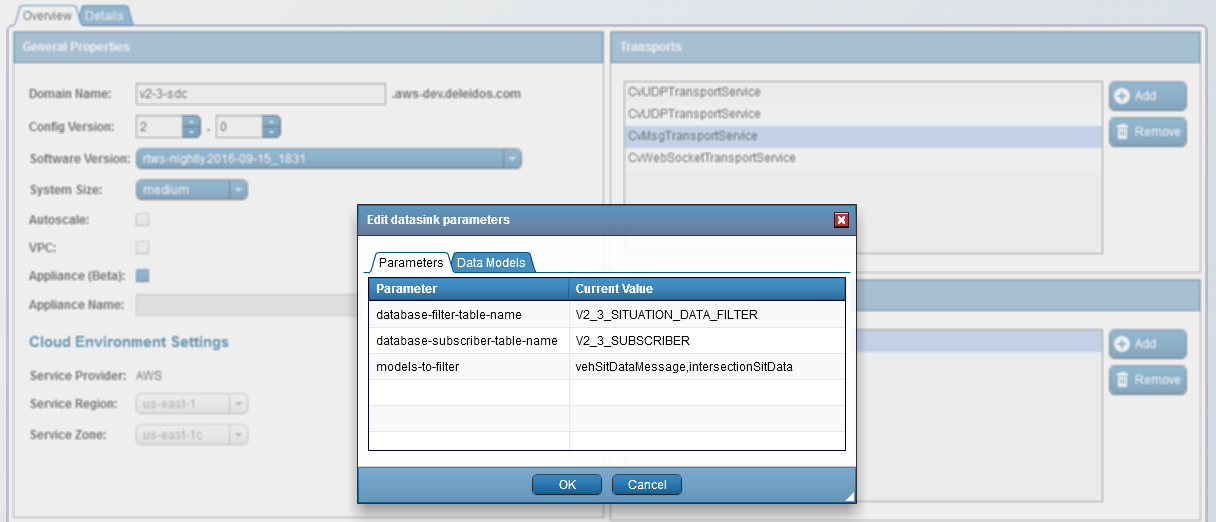


Figure 20- Situation Data Router Parameters

Next, go to the “Data Models” tab and ensure that the vehicle and intersection situation data models are selected. Then click “OK” to finish entering the parameters for the SituationDataRouter data sink.

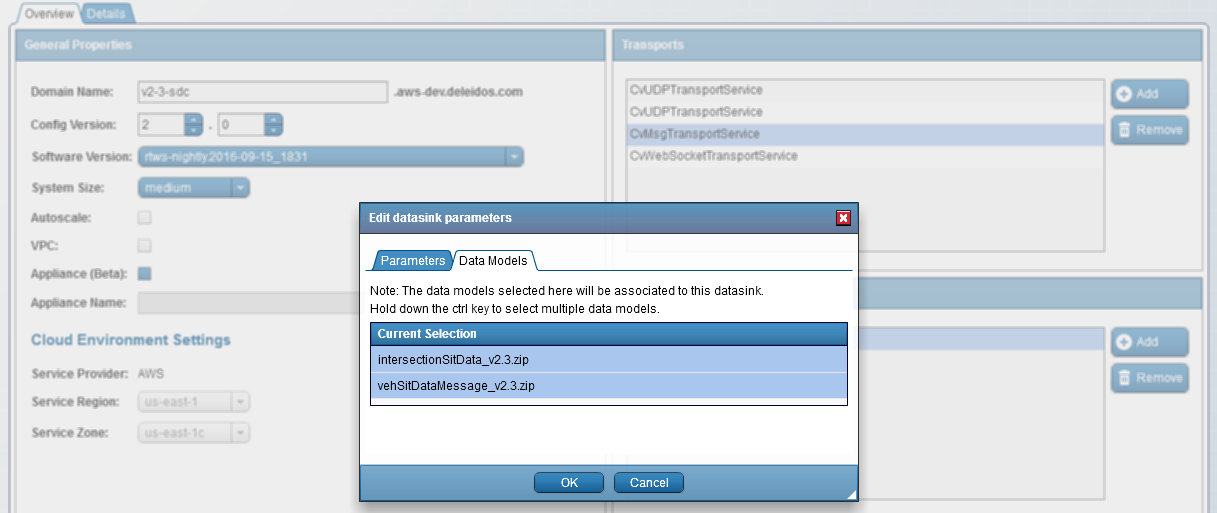


Figure 21- Data Models for Clearinghouse System Situation Data Router

## 10.2 Configure the Warehouse System Using System Builder

As described for the clearinghouse system, the DigitalEdge System Builder application constructs the warehouse system. The warehouse system receives intersection and vehicle situation data from the clearinghouse system and traveler information from the Rackspace forwarder. For detailed information about building systems, refer to the DigitalEdge Configuration Guide. The warehouse system runs in the AWS us-east-1 service region and us-east-1c availability zone. The system size is medium and the software version is a release cut on September 15, 2016. The system itself consists of three data models described in Section 8, five data transports, and a data sink.

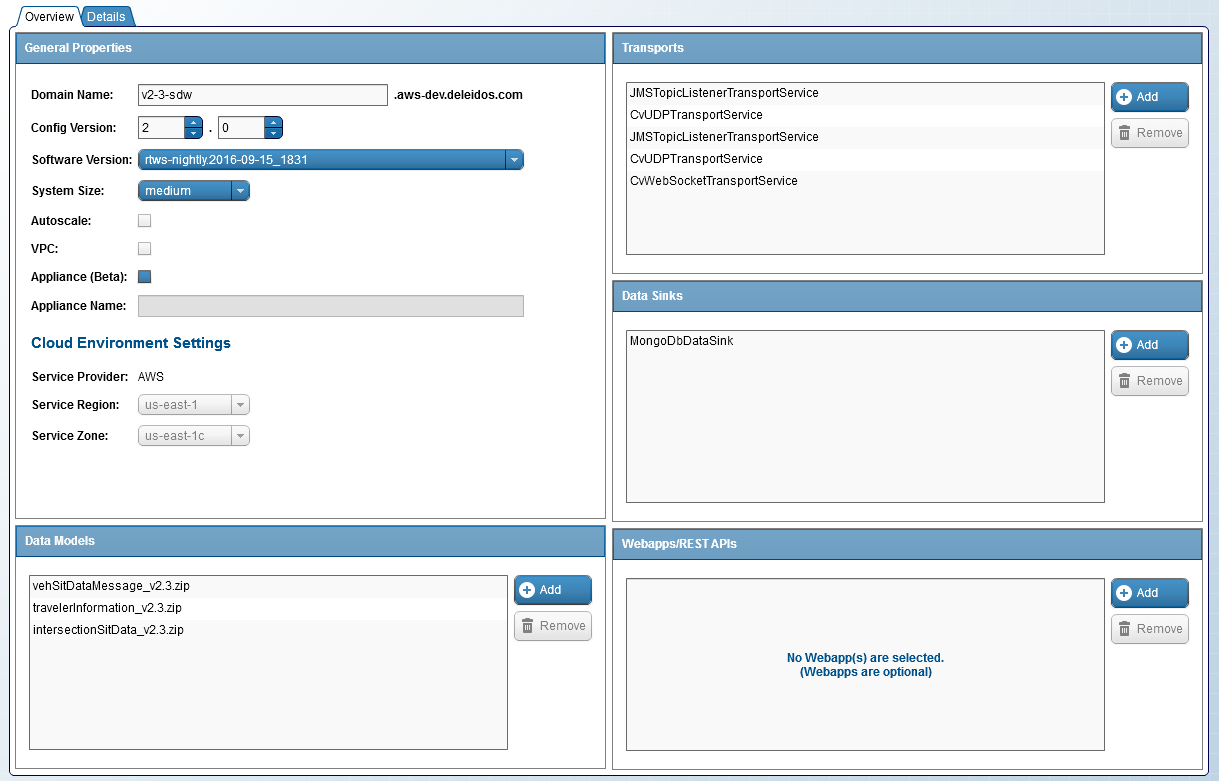


Figure 22- Warehouse System Components

### 10.2.1 Adding the Vehicle Situation Data JMS Listener Transport

The JMSTopicListenerTransportService is a process that runs in every transport instance; it is responsible for connecting to the Clearinghouse system’s Pub/Sub server to get vehicle situation data messages and feed them into the Warehouse system for storage.

Go to the “Transports” panel and click “Add”. Select the JMSTopicListenerTransportService, and click “OK”. You will be presented with a modal window to enter in parameters for the data sink.

* input-format: Select the vehicle situation data model for this parameter.
* keystore-path: Enter the location to retrieve the keystore file. This transport supports pulling the file from S3 or from the local directory.
* truststore-path: Enter the location to retrieve the truststore file. This transport supports pulling the file from S3 or from the local directory.
* protocol: The protocol to use to connect to the Clearinghouse system’s pub/sub server.
* sdc-topic-name: The name of the topic to connect to in the Clearinghouse system’s pub/sub server.
* target-host: The IP Address or hostname of the Clearinghouse system’s pub/sub server.
* target-port: The port to connect to in the Clearinghouse system’s pub/sub server.

For the keystore-path and truststore-path, to get the keystore and truststore files, go to the “Security” tab in the Management Console and download the files. It is recommended that you store the files in S3 and have the process retrieve it and use it when it starts up.

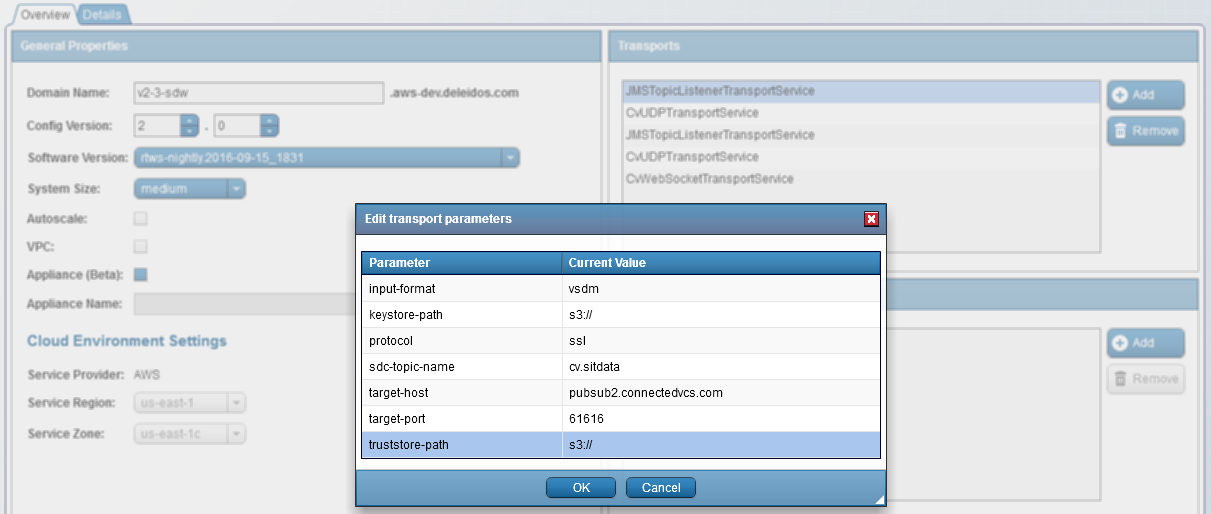


Figure 23- Vehicle Situation Data Listener Transport Parameters

### 10.2.2 Adding the Traveler Information Data Transport

The CvUDPTransportService provides the mechanism for receiving UDP data messages from the forwarder running in Rackspace.com. The user needs to add the transport in the System Builder application and then configure specific parameters for the transport. The following parameters are used.

* forwarder-host-name: Host name or IP address of Rackspace forwarder.
* forwarder-port: Port of Rackspace forwarder.
* input-format: Input source associated with the DigitalEdge data model (travinfo for traveler information data).
* listen-port: Listen port for messages not coming through the Rackspace forwarder.
* low-volume-flush-strategy: Allows for the system to push along low volumes of data quicker.
* max-packet-size: Maximum packet size of incoming UDP packets (default 65535).
* max-time-between-flush: Fine tuning of how long the system waits to push data.
* message-format: Format of incoming messages- DSRCJ2735 (unencrypted) or IEEE1606.2 (encrypted).
* receipts-topic: JMS topic for output of the processed packets (cv-receipts is used for the clearinghouse system).
* svc-northwest-latitude: Latitude of northeast corner of the SDC geographic region.
* svc-northwest-longitude: Longitude of northeast corner of the SDC geographic region.
* svc-southeast-latitude: Latitude of southeast corner of the SDC geographic region.
* svc-southeast-longitude: Longitude of southeast corner of the SDC geographic region.

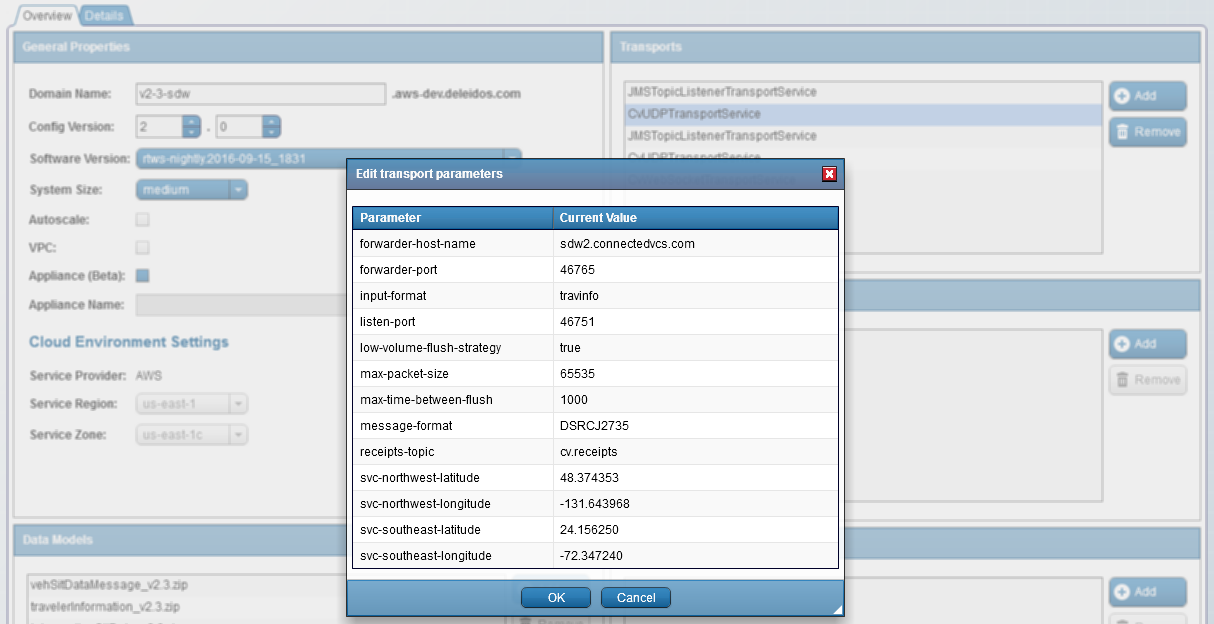


Figure 24- Traveler Information Transport Parameters

### 10.2.3 Adding the Intersection Situation Data JMS Listener Transport

The JMSTopicListenerTransportService is a process that runs in every transport instance; it is responsible for connecting to the Clearinghouse system’s Pub/Sub server to get intersection situation data message and feed it into the Warehouse system for storage.

Go to the “Transports” panel and click “Add”. Select the JMSTopicListenerTransportService, and click “OK”. You will be presented with a modal window to enter in parameters for the data sink.

* input-format: Select the intersection situation data model for this parameter.
* keystore-path: Enter the location to retrieve the keystore file. This transport supports pulling the file from S3 or from the local directory.
* truststore-path: Enter the location to retrieve the truststore file. This transport supports pulling the file from S3 or from the local directory.
* protocol: The protocol to use to connect to the Clearinghouse system’s pub/sub server.
* sdc-topic-name: The name of the topic to connect in to the Clearinghouse system’s pub/sub server.
* target-host: The IP Address or hostname of the Clearinghouse system’s pub/sub server.
* target-port: The port to connect to in the Clearinghouse system’s pub/sub server.

For the keystore-path and truststore-path, to get the keystore and truststore files, go to the “Security” tab in the Management Console and download the files. It is recommended that you store the files in AWS S3 and have the process retrieve it and use it when it starts up.

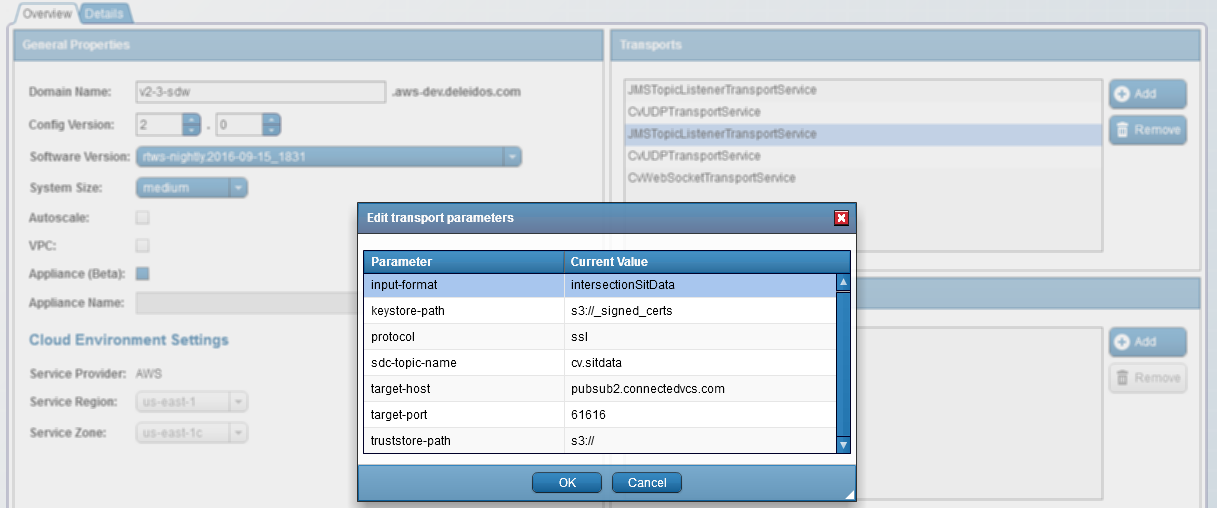


Figure 25- Intersection Data Listener Transport Parameters

### 10.2.4 Adding the MongoDB Data Sink

The MongoDBDataSink is a process that runs in every MongoDB data sink instance and it is responsible for storing the vehicle, intersection, and traveler information situation data in the MongoDB database.

Go to the “Data Sinks” panel and click “Add”. Select the MongoDbDatasink, the one with the package “gob.usdot.cv.mongodb.datasink”, and click “OK”. You will be presented with a modal window to enter in parameters for the data sink.

* auto-connect-retry: A flag to indicate if the MongoDB client should automatically retry connection.
* collections: A comma-delimited list of Connected Vehicle collections. The collection name is the same as the input data model name; currently, we use it to create and monitor the time to live and geospatial indexes for each collection.
* connect-timeout-ms: The interval to wait, in milliseconds, before retrying a connection failure to the database.
* database-name: The name of the database that will store all situation data.
* index-definition-list: Defines the indexes to create on each collection. Indexes are defined as <fieldName>:<value> and are separated by a comma. Use a space between the indexes for compound indexes.
* ignore-message-time-to-live: A flag indicating whether to ignore the input data model timeToLive field or not.
* mongo-server-host: The hostname or IP Address to use to connect to the MongoDB database. If the special value “mongodb.%s” is used, the data sink will fill in the %s with the system’s domain name and use that as the hostname to the MongoDB instance.
* mongo-server-port: The port to use to connect to the MongoDB database.
* time-to-live-field-name: The field in the input data model to create the ttl index in the database.
* time-to-live-unit: The ttl unit. If the ignore-message-time-to-live flag is true, this field is used.
* time-to-live-value: The ttl value. If the ignore-message-time-to-live flag is true, this field is used.
* topic-name: The topic name to use to send the receiptId. The transport instance in the Warehouse uses the receiptId to send a DataReceipt ASN.1 BER message to the client to complete the dialog.

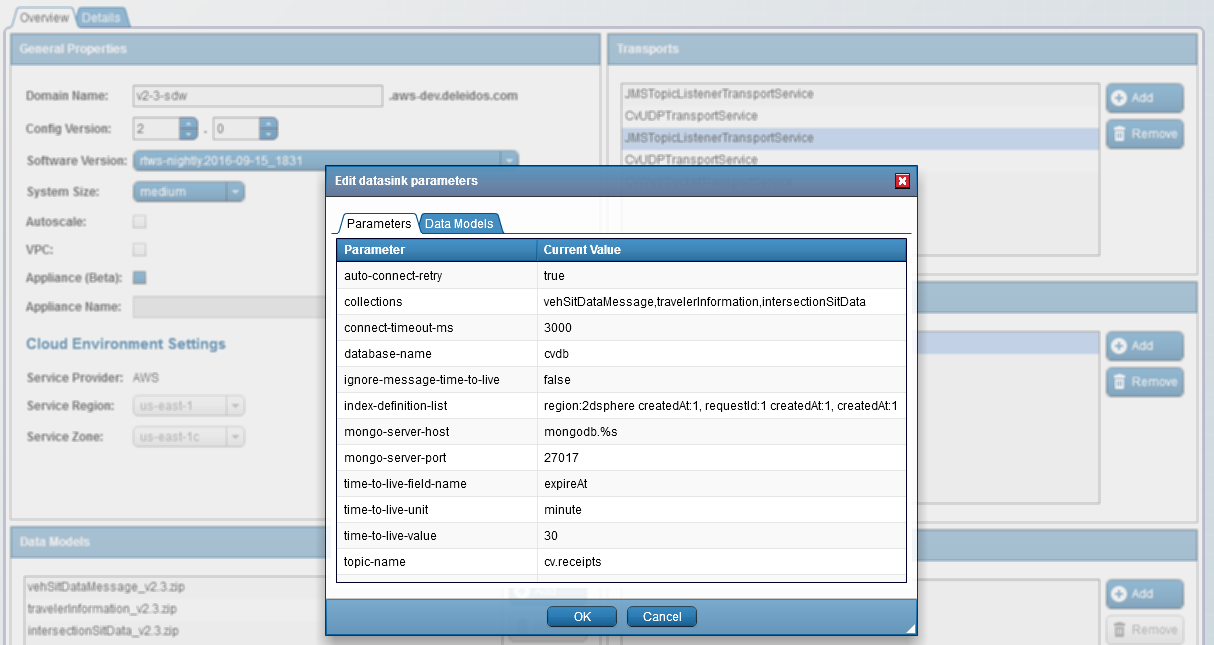


Figure 26- MongoDB Data Sink Parameters

Next, go to the “Data Models” tab and ensure the vehicle, intersection, and traveler information situation data models are selected. Then click “OK” to finish entering the parameters for the MongoDbDataSink.

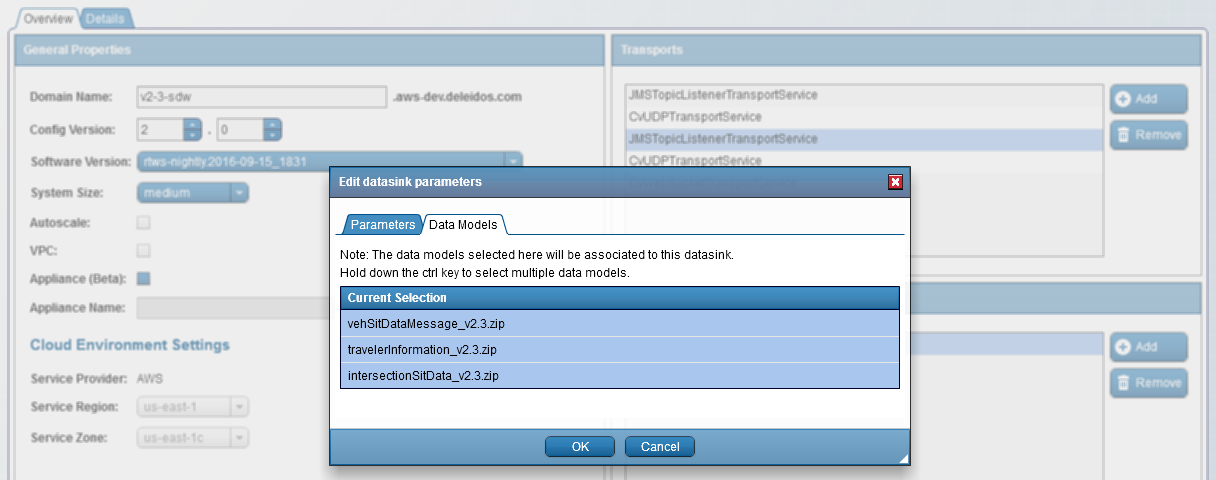


Figure 27- Data Models for Warehouse System

# Appendix A: List of Acronyms

|  |  |
| --- | --- |
| ARC-IT | Architecture Reference for Cooperative and Intelligent Transportation |
| AWS | Amazon Web Services |
| CVCS | Connected Vehicles Core Services |
| DB | Database |
| DSRC | Dedicated Short-Range Communications |
| EBS | Elastic Block Store |
| EC2 | Elastic Compute Cloud |
| IEEE | Institute of Electrical and Electronics Engineers |
| IP | Internet Protocol |
| IPv4 | Internet Protocol version 4 |
| IPv6 | Internet Protocol version 6 |
| ISD | Intersection Data |
| JMS | Java Messaging Service |
| RSU | Roadside Unit |
| SAE | Society of Automotive Engineers |
| SCP | Secure Copy Protocol |
| SDC | Situation Data Clearinghouse |
| SDK | Software Development Kit |
| SDW | Situation Data Warehouse |
| SSH | Secure Socket Shell |
| USDOT | United States Department of Transportation |
| UPD | User Datagram Protocol |
| VM | Virtual Machine |

# Appendix B: Maintenance Plan

## B.1 Software Updates

The software systems running within CVCS are operational systems. Therefore, determining reasonable software updates is and should be biased toward ensuring maximum uptime.

### B.1.1 Updating Plugins

Before updating plugins, please note that plugins are Java based software components and familiarity with Java will be required.

If an existing DigitalEdge plugin needs to be updated, or a new plugin needs to be created, refer to the DigitalEdge Software Development Kit (SDK) Guide[[5]](#footnote-5) for details on developing and updating a plugins. Once the updated or new plugin is ready it must be uploaded to the DigitalEdge repository before it can be used in a DigitalEdge system. Refer to the DigitalEdge Operations Guide3, Chapter 5, section *Managing the component* repository for instructions on uploading plugins.

### B.1.2 Updating Software Libraries Directly

Sometimes a software update is required for a library that isn’t a DigitalEdge plugin. In these situations it is necessary to access the DigitalEdge nodes directly. The nodes are accessible via a Secure Socket Shell (SSH). In order to do so, the security group for the instance needs to be opened via the AWS Console for the Internet Protocol (IP) of the machine accessing the instance[[6]](#footnote-6) and a copy of the public key from the key pair used to start to instance must be obtained. To upload new software onto the instance, any Secure Copy Protocol (SCP) client can be used. Once the instance is accessed and the new software library files are uploaded, the software update can be performed.

The location of the software depends on which software is being updated. Any DigitalEdge software libraries can be found in the sub-directories under /usr/local/rtws/\*. The libraries will be located in “lib” directories and there will be scripts to stop and start the various services located in corresponding “bin” directories. On transport instances, DigitalEdge transport specific software is located in /usr/local/rtws/transport/\*. DigitalEdge datasink and enrichment specific software is located on their corresponding ingest instances under the directory /usr/local/rtws/ingest/\*. Non-DigitalEdge datasink software locations depend on how the instance is configured to install the software. For example, a MongoDB Datasink instance is configured to store the Database (DB) on a mounted volume under /mnt/rdafs/mongodb/. JMS instances key software is Apache ActiveMQ located at /usr/local/apache-activemq/\* with scripts to start and stop the service located under /usr/local/rtws/commons-core/bin/boot/.

In order to perform the update, after locating the correct software library, the service needs to be stopped, the correct library files need to be replaced with new copies, and the service needs to be started again. It is recommended to create backup copies of the original library files before replacing them with the new version. This will ensure old versions can be restored if there is an issue with the new software.

### B.1.3 Rebuild and Cycle Affected System

After new or updated plugins have been uploaded or a virtual machine (VM) has been retired, the effected DigitalEdge system(s) must be rebuilt and restarted for the software updates to take effect. To do this, first open the target system in System Builder. Add and/or update the required plugins (including plugin parameters) for the system and give the system a new unique name. This system, with the software updates and new name, will run in parallel with the old system until it can be confirmed the changes were successful. Build the new system and start it in the Management Console. For detailed instructions on building and starting a DigitalEdge system, please refer to the DigitalEdge Configuration Guide[[7]](#footnote-7), Chapter 5, *Building the System*.

### B.1.4 Update Forwarders

After starting the new system, the respective forwarder’s settings.properties files need to be updated. Refer to the *CVCS Configuration Report 20170912.xlsx2* file for the names and locations of these files. Change the TransportIP in each of the files from the public IP of the Transport of the old system to the public IP of the Transport of the new system. After doing this, restart the forwarder. To verify that data is getting through, use the webtools (such as Warehouse Tools Set). It is recommended to track Intersection Data (ISD) messages through the system to verify the software update has worked. If successful the old system can be stopped.

## B.2 Continued Operations & Infrastructure Maintenance

### B.2.1 AWS VM Retirement

Periodically, AWS needs to retire (shutdown, restart, etc.) one or more of the virtual machines (nodes) that make up a DigitalEdge system. If there is ever an issue with an AWS node, an email will be sent to the AWS account administrator with details about the node in question. In addition to the email, EC2 instances which require maintenance will have a scheduled event listed for them on the AWS Console. To view which EC2 instances have scheduled events, please refer to the AWS documentation[[8]](#footnote-8). When this happens, use the DigitalEdge or AWS Console to determine the DigitalEdge system(s) that will be affected. Rebuild and restart the affected system as discussed in sections

B.1.3 Rebuild and Cycle Affected System and B.1.4 Update Forwarders.

There is also a way to restart the EC2 instance without restarting the whole system. To do so, first go to the AWS Console. Right click on the EC2 instance that requires restarting. Hover over the “Instance State” context menu option and select the “Terminate” option from the menu. While the instance is terminating, open an SSH terminal connected to the system’s Master node. When connected to the master node, run “ps aux | grep ClusterManager” to determine the Process Identifier number. Return to the AWS Console and wait for the EC2 instance that is terminating to reach a state of “Terminated”. When it is listed as terminated, return to the Master node SSH terminal and run “sudo kill -9 <PID>”, where <PID> is the process identifier number from the “ps aux …” command that was ran. At this point, the Master process should be down. Next change directories to the Master’s scripts: “cd /usr/local/rtws/master/bin”. Restart the Master process: “sudo –u rtws ./start\_master.sh”. The Master process will start and attempt to connect to any existing nodes in the system. This will re-establish a connection to the rest of the system. When the Master process looks for the terminated instance, it will detect that it is missing and start a new EC2 instance part to replace the old. You will be able to see the new EC2 instance starting on the AWS Console; it may take a bit for the EC2 instance to start. Once the EC2 instance starts, it will also take some time for it to download and start the DE software.

## B.3 Periodic Diagnostics

There are a few places where diagnostics can be done for a running system. To view the status the DigitalEdge system as a whole, there are two tools available: the DigitalEdge Management Console and the DigitalEdge System Monitor. To view the health of individual instances that compose the systems, the AWS Console can be used.

### B.3.1 DigitalEdge Management Console

The DigitalEdge Management Console provides two ways to perform diagnostics for a system. The first is the system view list which provides a quick snapshot of a running system, including what process groups are running, the number of instances and types, and the number of volumes and sizes. The second tool is the log viewer which allows access to the logs of each node in a DigitalEdge system. The use of both of these is described in the DigitalEdge Operations Guide3, Chapter 6, in sections *Viewing system status* and *Checking log files*.

### B.3.2 DigitalEdge System Monitor

The DigitalEdge System Monitor is a dynamic console tool for monitoring system health and potential problems in real-time by visually depicting system activity and resource scaling. It provides the capability to see the throughput of ingest instances, the amount of data on the internal system queues, and the current storage capacity of the datasink instances. The use of the DigitalEdge System Monitor is outlined in the DigitalEdge Operations Guide3, Chapter 6, section *Checking system metrics*.

### B.3.3 AWS Console

The AWS Console can be used to check the health and status of the individual instances that compose the DigitalEdge systems. Using the AWS Elastic Compute Cloud (EC2) service, a web service that provides secure, resizable compute capacity in the cloud, each instance can be selected in order to show: its current state, a detailed description of the instance and its attributes, status checks, monitoring, and much more. Detailed documentation on how to use AWS EC2 is available through AWS EC2 Documentation[[9]](#footnote-9). For ease of system use, the AWS Elastic Block Store (EBS) is also utilized to provide persistent block storage volumes for use with AWS EC2 instances.

## B.4 Backup Procedure for the Situation Datawarehouse and Situation Data Processing Center

In the SDW and SDPC, the technology used to store the data is the NoSQL document database, MongoDB. The database is stored on RAIDed EBS volumes attached to the DigitalEdge datasink instance. Since the database is completely stored on volumes separate from the instance, this makes using an AWS EBS snapshot the ideal way to create a backup of the data.

### B.4.1 Volume Snapshots

Creating a volume snapshot is an easy process using the AWS console and is documented by AWS EC2 Documentation[[10]](#footnote-10). Since multiple volumes are used and RAIDed by the DigitalEdge system, it is necessary to complete the snapshot process for each volume on the instance. Once the process is completed, the snapshots will remain stored by AWS indefinitely until manually deleted from the AWS EC2 Console.

## B.5 Maintenance Issues

Maintenance issues are currently reported by customers using the ticketing system from the Samanage Service Desk[[11]](#footnote-11). The service desk is accessible from the “Support” link in the top right corner of the Connected Vehicles Tool Library page. Once a ticket is submitted, it will be addressed as quickly as possible and closed upon resolution.

## B.6 Security Monitoring and Patching

### B.6.1 Software

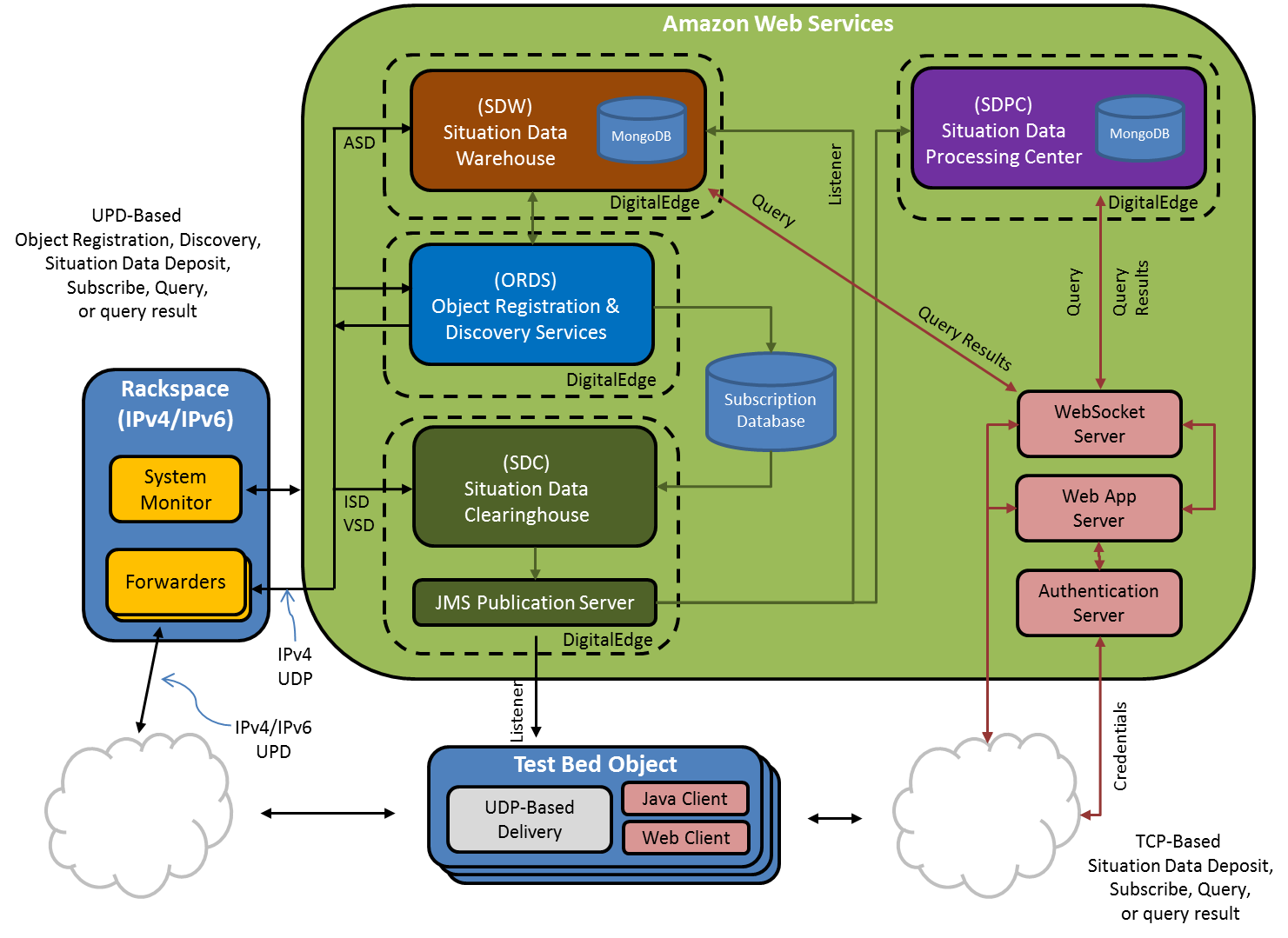
Due to the nature of Connected Vehicle environments, security must be built into every facet of a Connected Vehicle network. The SDC, SDW, and SDPC were developed using the security libraries, recommended by the USDOT and industry experts, in the following industry standards: IEEE 1609.2-2016 and SAE J2735. Because security is built directly into the software, any patches or updates to the security of the system can be applied using the, above, instructions for software updates in Section B.1.

### B.6.2 Instance Access

Security for the access of the DigitalEdge instances is provided by multiple means. When a new instance is launched, it is done so using the public key from an RSA key pair. This public key is controlled by AWS and not accessible to anyone else. In order to access an EC2 instance launched using the public key, a copy of the private key that was generated with the public key must be used for authentication. Without the private key, it is not possible to access the instance. In addition to the use of key pairs, AWS implements Security Groups. A Security Group is a virtual firewall that can be used to control traffic to instances. The Security Group maps IPs, IP ranges, or other Security Groups to specific port or port ranges. Each instance must be part of a Security Group, and only specified IPs or other instances in specified Security Groups will be allowed to access the instance on the specified ports. Therefore, the Security Group needs to be maintained to only ensure specific access to the instance. Description on how to configure the Security group can be found in AWS Documentation6.

# Appendix C: Background

In 2007, the United States Department of Transportation (USDOT) established a Connected Vehicle (CV) Test Bed in Oakland County, Michigan (known as both the Unified Implementation of the Reference Architecture and the Southeast Michigan Test bed – referred herein as the SEMI Test Bed) as a test facility for Proof of Concept (POC) testing of Dedicated Short-Range Communications (DSRC) using allocated bandwidth within 5.9 GHz. The SEMI Test Bed comprises all of the required elements for a functional V2V and V2I/I2V, 2016 standards compliant, real-world deployment. To date the SEMI Test Bed has been used as a testing facility for stakeholders to evaluate prototype equipment, applications, and services. In order to meet the evolving needs of the CV industry and remain a leader in the implementation of best practices and advances in technology and architecture, the Test Bed has implemented numerous enhancements, such as various approaches and improvements in data distribution systems, advances in mapping tools, and updates to the most up to date standards.



**Figure 28- System Architecture**

# Appendix D: DigitalEdge™ Framework

DigitalEdge is a cloud-based platform for creating big data solutions. It offers:

* *Scalable, pluggable, flexible, pre-integrated, data management platform* to enable quick implementation and simplified management of near real time (or batch) big data applications
* *Software* that leverages the power and dynamism of cloud computing to support rapid out-of-the-box creation and integration of IT systems
* *Data enrichment engine* that standardizes, integrates, and enhances diverse data sources to clean up imperfect data and transform it into information, rich with meaning.
* *Administration tools* that support rapid creation and spawning of systems that perform pre- ingest and enrichment

## D.1 Team

The DigitalEdge platform was developed and is maintained by Leidos. The DigitalEdge Team is comprised mostly of software engineers, based in the National Security Sector of Leidos, and contains a depth of experience ranging from college interns to 30+ years creating large scale enterprise systems.

The DigitalEdge Team prefers a Linux based environment, leveraging Docker in Amazon Web Services, Microsoft Azure, Eucalyptus and Rackspace. Favorite programming languages include Java and AngularJS.

## D.2 Licensing

DigitalEdge is provided for use under the Apache 2.0 Open Source License (https://raw.githubusercontent.com/deleidos/digitaledge-platform/master/LICENSE). The source information for DigitalEdge is available on GitHub at [https://github.com/deleidos/digitaledge-platform](https://webmail.leidos.com/owa/redir.aspx?SURL=hBHXcWQeF-nMmF0Jz8jDe4XkE0szysv0nJzt3mHNg9_KDPaNC-rSCGgAdAB0AHAAcwA6AC8ALwBnAGkAdABoAHUAYgAuAGMAbwBtAC8AZABlAGwAZQBpAGQAbwBzAC8AZABpAGcAaQB0AGEAbABlAGQAZwBlAC0AcABsAGEAdABmAG8AcgBtAA..&URL=https%3a%2f%2fgithub.com%2fdeleidos%2fdigitaledge-platform).

## D.3 Documentation

The below documentation may have been referenced in this manual.

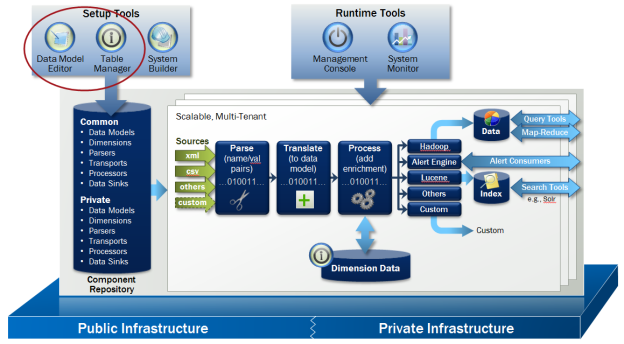
* DigitalEdge Overview Guide
* https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Overview\_Guide.pdf
* DigitalEdge Operations Guide
  + https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Operations\_Guide.pdf
* DigitalEdge Configuration Guide
  + https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Configuration\_Guide.pdf

Additional documentation can be found at:

https://github.com/deleidos/digitaledge-platform/tree/1.2.1/docs

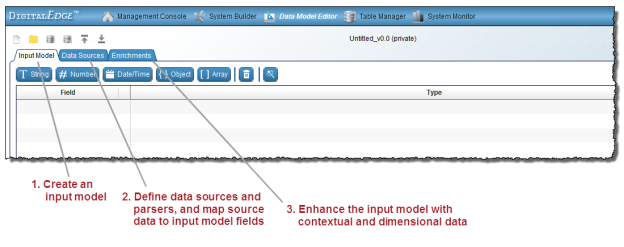
# Appendix E: DigitalEdge™ Configuration Guide, Chapter 4 – *Defining the Data Model*

Once you understand your data sources and you have decided how to map the data to the input model, you are ready to use DigitalEdge’s Setup Tools to configure your system.



Use the Data Model Editor to define the three parts of a DigitalEdge data model:

* Input model: This is the normalized data model into which all incoming data is mapped
* Data sources: How source data fields are parsed and mapped to the input model
* Enrichments: Specifications for adding meaning and context to the data



You can work with common data models, provided with DigitalEdge, as a starting point for your configuration work.  Or you can create private data models that are only available to your site.

If you have a simple CSV or JSON data source file that you would like to quickly map into DigitalEdge, you can use the Data Model Creation Wizard. The Wizard builds a flat input model without any objects, arrays, nesting, or enrichments.

Step 1: Define an Input Model

The goal of this configuration step is to specify all possible fields for the input model.  The input model can include a set of fields that are common to all data sources, and extrinsic fields which are unique to one or several data sources. You should define the data fields that will live in DigitalEdge and specify the properties of those fields.

Choose an approach for defining an input model:

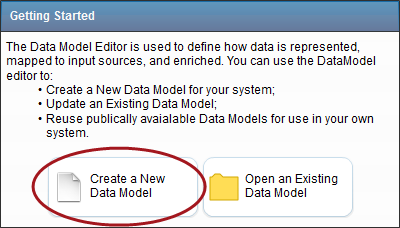
* Create a new input model from completely from scratch. (See " Create a new input model" on page 55)
* Search for and work with an existing input model as a starting point for your system. You can edit a model, delete fields, add fields, and change the specifications to meet your needs. DigitalEdge provides several common data models that you can use for this purpose, including models for sales data, CEF data, and transportation systems, to name a few. (See "Work with an existing input model " on page 60)
* Upload an input model from another DigitalEdge system. If you created a data model in another DigitalEdge system, you can download that input model to a hard drive and then upload it to this DigitalEdge system. (See " Upload and Download Data Models Between Systems" on page 120)
* Quickly create a simple data model from a CSV or JSON file. (See "Create a Data Model with the Wizard" on page 115)

You may want to start with a common data model and create prototype system for test purposes. Once you have worked with DigitalEdge for a while, you can then create your own site-specific data model.

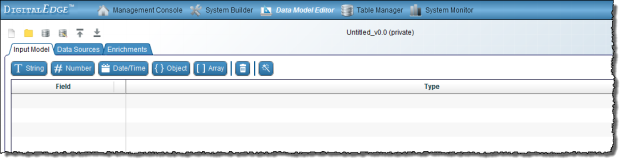
DigitalEdge uses a data model versioning system that lets you keep multiple versions of a data model. Every data record is tagged with its data model name and version, allowing multiple versions to exist in the same data store. This lets you easily migrate data from one version to the next on a timeframe that meets your needs. You can have both old and new version data records in the same system simultaneously. You create multiple versions by editing the current version and saving it as a new version number. When you save a new input model, be sure to use a unique and meaningful version number to keep and distinguish multiple versions.

Create a new input model

1. Access the Data Model Editor tool   from the Management Console Tools list: https://<your-domain-name>/tenantconsole
2. In the Getting Started dialog box, click Create a New Data Model.



1. The Input Model workspace appears with empty Field and Type columns. You will build a data model in JSON representation, including arrays, objects, and fields.

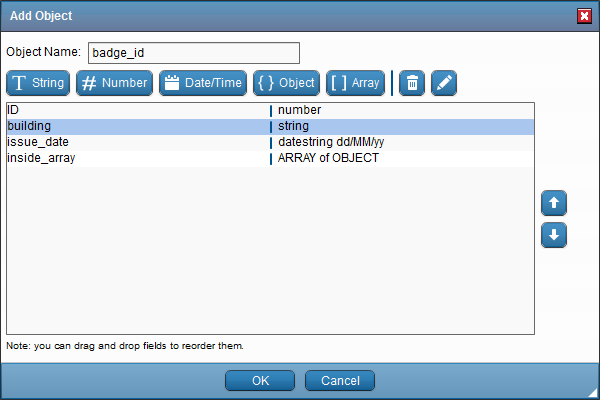


Create a new field

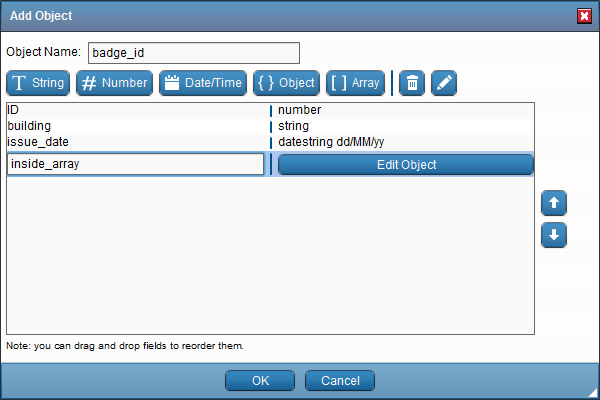
1. Highlight the field below which the new field should appear.
2. Click a Type button:
   * String 
     1. Enter the Field Name using letters, numbers and underscores.
     2. Click OK.
   * Number 
     1. Enter the Field Name using letters, numbers and underscores.
     2. Click OK.
   * Date/Time 
     1. Enter the Field Name using letters, numbers and underscores.
     2. Click OK.
     3. Select a format from the list of Common Date Formats, Common Time Formats, or Custom Formats.
     4. To create a custom format, build a Format String by selecting element(s) from the Custom Format drop-down menu and optionally entering delimiter(s) in the Format String box. As you make your selections, the custom format that you are building appears in the Format String box. For example, to enter a credit card expiration date, select MM from the Custom Formats list, enter / in the Format String, then select yyyy from the Custom Formats list. Your custom format appears as: MM/yyyy in the Format String box. Note that text enclosed in singled quotes is ignored.
     5. Click OK.

DigitalEdge is very flexible in handling dates. Dates in original data sources may be in one format, dates in your input model may be converted to another format, and dates handled in enrichments can be in a third format. At no time does DigitalEdge impose a specific date format on your input data.

* + Object 
    1. Enter the Object Name using letters, numbers and underscores.
    2. Enter at least one Field by clicking a Type button, entering the field name, and hitting the Enter key. Continue to define as many fields as needed.



* + - * Use the arrows on the right to re-order a field up or down. You can also drag and drop a field in the list.
      * To insert a new field in a list, highlight the field that should appear below the new field and click a Type button.
      * When you create a Date/Time field, a datestring text box is provided. Click  to access the Date Time Format dialog box (follow the Date/Time instructions).
      * When you create a new field or an array of Object type, a second Add Object dialog box opens to define the fields in the sub-object. After the sub-fields are defined, access the list by highlighting the field and clicking  and .



* + 1. Click OK.
  + Array 
    1. Enter the Array Field Name using letters, numbers and underscores.
    2. Select the Element Type (string, number, or OBJECT).
    3. If you selected OBJECT as the Element Type, enter at least one field (follow the Object instructions).
    4. Click OK.

**Configuring an input model associated with a dimension data sink as its data source**

If your input model specifies an enriched data record in a dimension data sink that will be sourced in another DigitalEdge system, you must include two special fields in the input model definition to associate the table in the tenant database and its keys with the input model and fields.

You should create two systems in System Builder when creating a dimension data sink (or a JSON to JDBC data sink) that will be used as a data enrichment source for another data stream. System 1 would define the input model for the dimension data sink (on the Data Model Editor/ Input Model tab), specify the dimension table (in Table Manager), define the dimension table enrichment (on the Data Model Editor/Enrichments tab), map the dimension columns to the input model fields (on the Table Manager/Enrichment tab), and define the dimension data sink in System Builder. System 2 would read the dimension data sink in system 1 as an enrichment source for its input model. See "Example: Planning for multiple data sources and enrichments" on page **Error! Bookmark not defined.** for an example scenario.

As an example, if the input model defines a customer record and the dimension data sink serving as the data source has a table name of CUSTOMER, your input model specification would include fields similar to:

FirstName: string

LastName: string

DateOfBirth: date

tableName: string

tableKey: string

Note that the tableName and tableKey fields must be specified as part of the input model definition, to identify the dimension data sink (or JSON to JDBC data sink) that is serving as the data source for this input model.

and the input data source definition would include:

FirstName: get(firstName)

LastName: get(lastName)

DateOfBirth: get(DOB)

tableName: CONSTANT("CUSTOMER")

tableKey: CONSTANT("FirstName, LastName")

Note that the get statements reference the dimension table column names as defined in Table Manager.

So, if a record in DigitalEdge currently exists as:

FirstName: John

LastName: Doe

DateOfBirth: 1/1/1980

and the data source gets updated as:

FirstName: John

LastName: Doe

DateOfBirth: 4/3/1986

the dimension data sink in System #1 will update the row in the DigitalEdge tenant database with the new DateOfBirth, because the names match based on the tableKey definition.

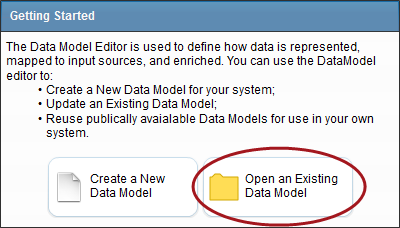
Save an input model as a private component

1. Click the Save icon on the Data Model Editor title bar.   
2. In the Save Data Model As dialog box, enter a new name for the model in the Model Name box.
3. Select a Major Version number.
4. Select a Minor Version number.
5. Select the private Area to keep the data model confidential.
6. Click OK.
7. To log out of the Data Model Editor, click your user icon/name in the upper right corner. Select Sign Out.

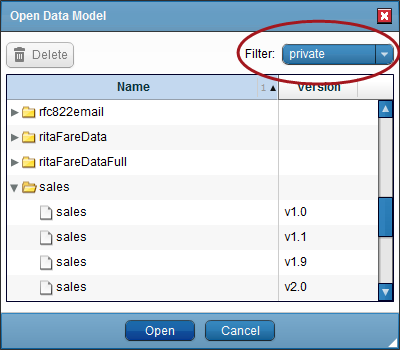
You can save a partially edited session and return to the Data Model Editor later to continue your configuration work.

Work with an existing input model

1. Access the Data Model Editor tool   from the Management Console Tools list.
2. In the Getting Started dialog box, click Open an Existing Data Model.



1. In the Open Data Model dialog box, select a data model from one Filter area:

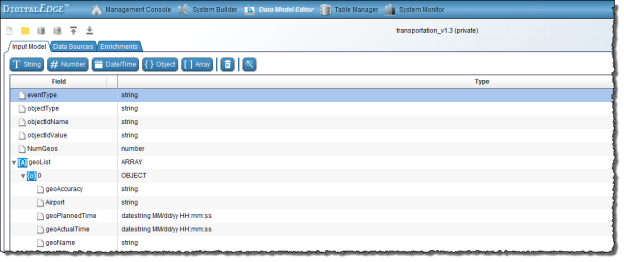


* + Common: Common data models are pre-defined and supplied with DigitalEdge as samples to help get you started. When you select a common data model, it is copied into your environment as a private data model (for your use only).
  + Private: Private data models are only resident in your DigitalEdge system, not shared with other organizations. Whenever you work with and edit a common model with the intention of using it as the foundation of your system, you save it as a private model. Note: When you assemble your DigitalEdge system with System Builder, you can only use a private data model.

1. Expand the data model Name to see a list of available data model Versions.
2. Select a Name or a Version and click Open.

When you select a data model Name, DigitalEdge automatically opens the latest Version of that data model.

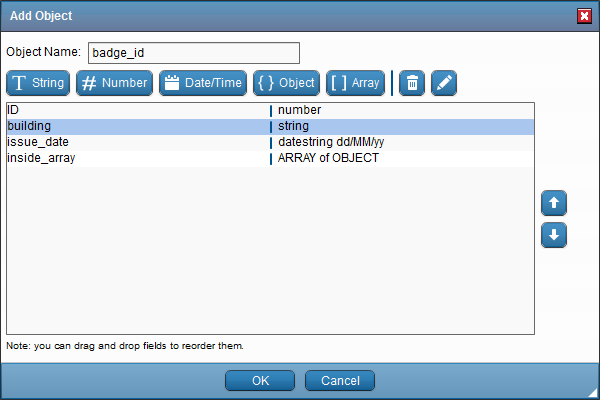
1. The Input Model screen lists the data model Fields and their data Types:
   * String
   * Number
   * Date/Time
   * Object
   * Array



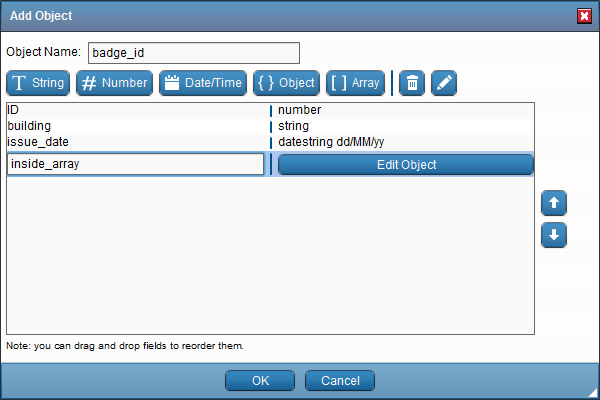
Data models are specified in JSON representation, including arrays, objects, and fields. You can expand or close an array entry with the arrowheads on the left.

Create a new field

1. Highlight the field below which the new field should appear.
2. Click a Type button:
   * String 
     1. Enter the Field Name using letters, numbers and underscores.
     2. Click OK.
   * Number 
     1. Enter the Field Name using letters, numbers and underscores.
     2. Click OK.
   * Date/Time 
     1. Enter the Field Name using letters, numbers and underscores.
     2. Click OK.
     3. Select a format from the list of Common Date Formats, Common Time Formats, or Custom Formats.
     4. To create a custom format, build a Format String by selecting element(s) from the Custom Format drop-down menu and optionally entering delimiter(s) in the Format String box. As you make your selections, the custom format that you are building appears in the Format String box. For example, to enter a credit card expiration date, select MM from the Custom Formats list, enter / in the Format String, then select yyyy from the Custom Formats list. Your custom format appears as: MM/yyyy in the Format String box. Note that text enclosed in singled quotes is ignored.
     5. Click OK.
   * Object 
     1. Enter the Object Name using letters, numbers and underscores.
     2. Enter at least one Field by clicking a Type button, entering the field name, and hitting the Enter key. Continue to define as many fields as needed.



* + - * Use the arrows on the right to re-order a field up or down. You can also drag and drop a field in the list.
      * To insert a new field in a list, highlight the field that should appear below the new field and click a Type button.
      * When you create a Date/Time field, a datestring text box is provided. Click  to access the Date Time Format dialog box (follow the Date/Time instructions).
      * When you create a new field or an array of Object type, a second Add Object dialog box opens to define the fields in the sub-object. After the sub-fields are defined, access the list by highlighting the field and clicking  and .



* + 1. Click OK.
  + Array 
    1. Enter the Array Field Name using letters, numbers and underscores.
    2. Select the Element Type (string, number, or OBJECT).
    3. If you selected OBJECT as the Element Type, enter at least one field (follow the Object instructions).
    4. Click OK.

Edit a field name

1. In the list of input fields, double-click on an entry in the Field column.
2. Edit the field name.

To edit field specifications, you must delete a field and re-configure it as a new field.

Delete a field

1. In the list of input fields, highlight the Field.
2. Click the Delete button.

Save an input model as a private component

1. Click the Save As icon on the Data Model Editor title bar.   
2. In the Save Data Model As dialog box, enter a new name for the model in the Model Name box.
3. Select a Major Version number.
4. Select a Minor Version number.
5. Select the private Area to keep the data model confidential.
6. Click OK.
7. To log out of the Data Model Editor, click your user icon/name in the upper right corner. Select Sign Out.

You can save a partially edited session and return to the Data Model Editor later to continue your configuration work.

If you edit a data model that is being used in a running system, you must go to the Management Console and Update the system that is using the data model. 

Step 2: Define Data Sources, Parsers, and Data Source Mappings

Next, use the Data Model Editor to define each data source and to map all the fields in the data sources into your new input model. Each incoming field must be mapped to and translated into a field in your input model. The goal of this configuration step is to build a normalized data model. This step can take a lot of time, depending on the number of data sources and fields you have to specify.

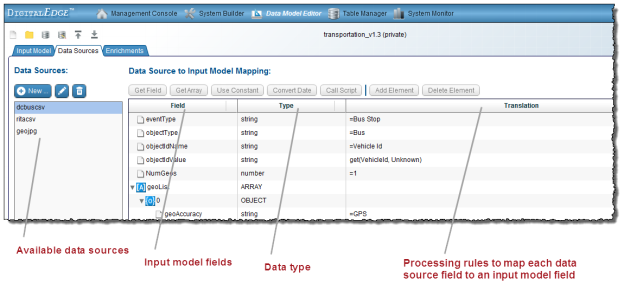
For each data source, you will:

* Name the source
* Select the type of parser that will extract the data fields for DigitalEdge input
* Name and specify each incoming data field
* Map each data source field to a field in the input model
* Specify how to translate, process, or convert the data in each incoming field

DigitalEdge parsers accept incoming data in several different formats. Each parser creates name-value pairs from a specific data source to feed the translation stage of the pipeline. The translation stage maps the name-value pairs to the input model according to the processing rules that you define. This model represents normalized data, regardless of the data source.

The Data Sources screen lists:

* Field: each field defined in the currently selected input model
* Type: the data type of each input model field
* Translation: a specification for each data source field that will be mapped into an input model field. The translation statement describes how the incoming data will be processed or converted so that it is normalized into the DigitalEdge input format.

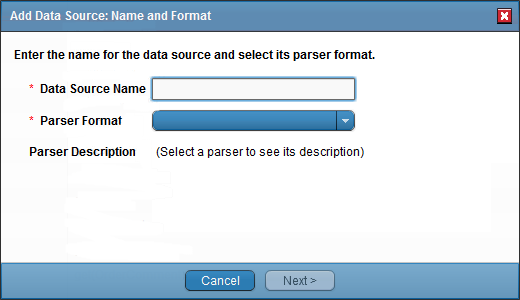


As you specify each field, the model is built in JSON representation.

Add a data source and parser to work with

First, name a data source as input to DigitalEdge and specify the type of parser to extract data.

1. Select the Data Sources tab.
2. Click the New icon.   

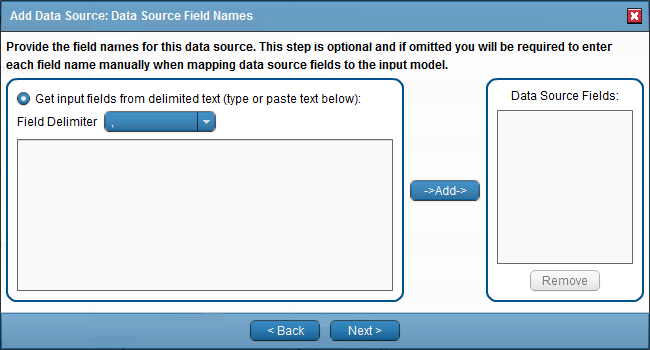


1. Enter a Data Source Name. The name should represent both the content and the parser type for the data source. For example, IDTrackingCSV or LocationsJPG.
2. Select  the type of parser to use when extracting data from this source by choosing a Parser Format option from the drop-down menu. (Note that this list includes the parsers that are included in the core release. You may also have custom parsers created for your site.)
   * BinaryParser: a user-configurable parser that takes any binary file encoded in Base 64, interprets the file as a string of bytes, and extracts portions of that file as fixed fields which should be saved and ingested
   * CEFParser: Common Event Format from ArcSight®, an open standard for logging security-related information in a common event log format
   * CsvParser: a frequently used Comma-Separated Values configurable parser to extract delimited plain text
   * DNS PCAP: parser that reads and extracts DNS packets, prefixed with DigitalEdge data; requires the PcapSnifferTransport in System Builder
   * EmailParser: parser that works with the RFC 822 ARPA email text format; RFC 822 specifies the structure of email messages, including attachments, to and from fields, etc.
   * EXIFParser: configurable parser that works with the Exchangeable Image File format used for handling image and sound files from digital cameras
   * JSONParser: configurable parser extracting data from text formatted in the JavaScript Open Notation standard
   * LibpCapParser: parser that works with packet captures in the UNIX Libpcap library
   * LogParser: parser that reads any log file (including firewall logs) by extracting the timestamp, based on a regex pattern you provide, and the text of the log entry
   * SNMP PCAP: parser that reads and extracts SNMPv1 or SNMPv2 packets, prefixed with DigitalEdge data; requires the PcapSnifferTransport in System Builder
   * Unstructured file: beta parser that works with unstructured files such as Word, Excel, and PDFs to extract content and metadata
   * XMLParser: configurable parser extracting data from the Extensible Markup Language format
3. Click Next.

Define the fields in a data source

Next, name and specify each incoming data source field on the Data Source Field Names dialog box. This is an optional task. If you skip this step, you must enter data source field names when you map data source fields to input model fields (See " Map data source fields to the input model" on page 68).

1. In the Get input fields from delimited text (type or paste text below):
   1. Select the appropriate Delimiter that you are using to separate field names.
   2. Enter a text string from which the field names can be parsed. Field names cannot include the following characters: single quote ('), double quote ("), or comma (,). For example: ID,Name,Product,Cost



1. Click Add to transfer the input fields to the Data Source Fields box.
2. Click Next.
3. Review the list of Parameters and Current Values.

* Double-click on any entry to make corrections.
* Click Back…Cancel to discard the specification.

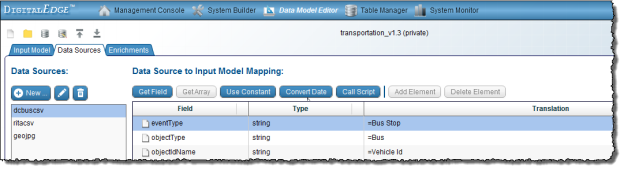
1. Click Finish.

For details about parser parameters, See "Parser parameters" on page 71.

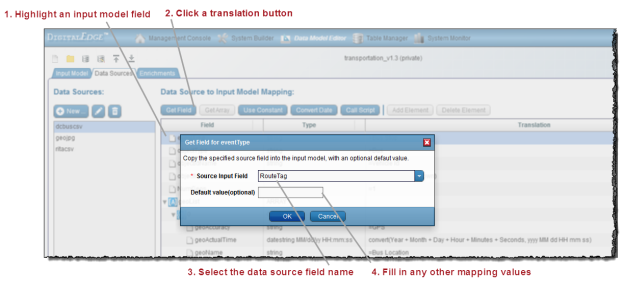
Map data source fields to the input model

Next, associate each incoming data field with a field in the DigitalEdge input model and specify how each field should be parsed, converted, formatted, or translated for the normalized model.

1. Click to highlight an input model Field. Several Translation buttons appear to help you specify the data Type for a field:



1. Click the appropriate Translation button and enter translation rules:



* + Get Field: To map data directly from a data source field into the input model without any data manipulation.
    1. Enter or select the name of the Source Input Field. Field names cannot include the following characters: single quote ('), double quote ("), or comma (,).
    2. Optionally enter a Default Value.
  + Get Array: The Get Array translation is most useful with the XML and JSON parsers. Use it to copy all child or repeated elements from a source field into your data model (you cannot get and map child elements individually). For example, in an XML schema, Get Array will copy a sequence of elements into the data model; if you do not use Get Array, only the last element from the sequence will be mapped into your data model.
    1. Select the name of the Source Input Field. Field names cannot include the following characters: single quote ('), double quote ("), or comma (,).

The Get Array button is only available for a field if you created it on the Input Model tab as an Array; input and output field specifications must match as arrays for input model mapping to succeed.

* + Use Constant: To set a specific value for an input model field that is not mapped from an incoming data source field.
    1. Enter constant value.
  + Convert Date: To convert an incoming date format to the input model’s date format.
    1. In the Source Field(s) box, enter the name of the data source field that is a date/time field. Separate multiple field names with +.

OR

From the Copy Field drop-down menu, select Input Field(s) (generated from the list created in step 1 under [Define the fields in this data source](file:///C:\Users\vahlbergm\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\JKPRDWJA\DigitalEdge%20Configuration%20Guide%201.3.doc#157551151)).

* + 1. In the Input Date Format box, click the Select … button and choose the appropriate date/time format(s) from the available Select a format lists.
    2. Click OK.
  + Call Script: To process the incoming data with a custom script. You can use a generalized DigitalEdge script, or you can create your own script and store it in your System Repository. (Use the Management Console > Plug-ins > Scripts > Private Scripts to upload a script to your repository.)

When you write a script, the parameters use Data Source Field names (incoming fields from originating in the external data source), not input model field names created for the DigitalEdge data model.

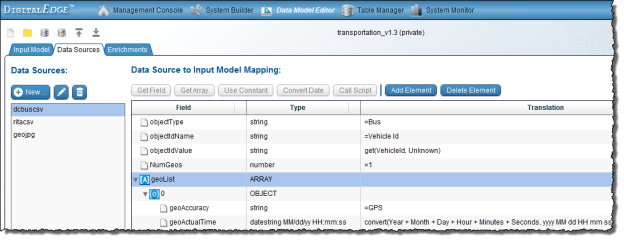
* + 1. Select the Script Name.
    2. In the Parameters list, double-click in a Data Source Field row and select the incoming data field. Field names cannot include the following characters: single quote ('), double quote ("), or comma (,).

1. Click OK. Your new field mapping and translation statement appears in the Data Sources list.

Map array elements to the input model

Mapping arrays to the input model follows a different procedure.

1. Select an Input Field with Type = ARRAY. Several Translation buttons appear that are specific to array elements:



1. Click Add Element. A new OBJECT or field is added to the selected array as the last element in the list. Placeholder values are added to the new element.
2. Select the new OBJECT or field and click a Translation button that is appropriate for the new element’s data type. Follow the steps above for the translation Type you selected.
3. Click OK. The new element and translation statement appear in the Data Source Mapping list.

You can also remove an element from an array with the Delete Element button.

Edit a source field translation statement

1. Select a source from the drop-down Data Source list. The previously defined field specifications appear.
2. Double-click an input model Field.
3. Edit the Translation statement following the guidelines above.

Delete a data source

1. Select the source from the drop-down Data Source list.
2. Click the Delete icon.    
3. Click OK.

Save your work

1. Click the Save icon on the Data Model Editor title bar.   
2. To log out of the Data Model Editor, click your user icon/name in the upper right corner. Select Sign Out.

You can save a partially edited session and return to the Data Model Editor later to continue your configuration work.

Parser parameters

Each parser includes a set of parameters to control its operation.

You can access parser parameters by creating or editing a data source in the Data Model Editor. See " Step 2: Define Data Sources, Parsers, and Data Source Mappings " on page 65 for more information.

Here are detailed lists of parser parameters, descriptions, and values for the parsers included in the core release. You can also hover over a parameter name in DigitalEdge for tool-tip help.

Binary parser

This configurable parser works with binary files to extract fixed fields. The parser reads any Base 64 encoded binary file, interprets the file as one string of bytes, and extracts portions of that string as fixed fields that should be saved and input for DigitalEdge. You define the field definitions and extraction algorithms to specify the fields that are ingested into your data model. This parser can be used with any Base 64 encoded binary file that includes fixed fields, such as a data feed or messages coming in over a wire. Use this parser when none of the other format-specific parsers meet the needs of the incoming data. While it may take some time to configure the parser, you can customize it exactly to your site-specific needs.

When selecting a transport to use with the binary parser, the transport's record-format parameter must be set to PCAP to work with the binary parser, since the PCAP record format type outputs Base 64 encoded data.

| Binary Parser | |
| --- | --- |
| Parameter Name | Explanation |
| custom-config | Double-click the Current Value entry to create a custom configuration that DigitalEdge should use to parse the incoming data. User instructions will appear in the dialog box; delete the instructions or cut and paste portions of the instructions to help create your custom specifications.  For each incoming field in a binary file that you want to ingest and save in DigitalEdge, you must include one line to define the field. Use one of the following types of field definitions to define each field. Note that fieldName references one of the fields that you specified on the previous screen, in the Add Data Source dialog box in Data Model Editor (Get input fields from...).   * READTOEND:fieldName:startPos:fieldType - To locate this fieldName, read data starting with the byte in startPos and reading data to the end of the data string. Specify a fieldType of ASCII, NUMBER, or BYTES. For example, in a data string that is 20 bytes long, to extract a field called Title in the last 10 bytes of a string as ASCII data, specify:   READTOEND:Title:10:ASCII   * FIXED:fieldName:startPos:length:fieldType: reverse - Use this definition when all incoming fields are the same length. Locate this fieldName by reading data starting with the byte in startPos and reading a fixed number of bytes specified in the length parameter. If you want to read data starting from the end of the data string, use the reverse parameter (valid values = true, reverse, r, or t); otherwise, leave the reverse parameter out of the definition. Specify a fieldType of ASCII, NUMBER, or BYTES. For example, to locate a field named taxrate in the middle of a 16 byte string, you may specify:   FIXED:taxrate:8:2:NUMBER   * BACKREFERENCE:fieldName:startPos:otherField: fieldType:reverse - Use a BACKREFERENCE definition to use a previously defined field in the definition for another field. The BACKREFERENCE line must always appear after the definition of the otherField that is used in the definition. If you want to read data starting from the end of the data string, use the reverse parameter (valid values = true, reverse, r, or t); otherwise, leave the reverse parameter out of the definition. For example, if you extracted location data in a READTOEND definition that included a three-character airport code and a state abbreviation starting in position 20, and you then want to isolate the state data in that string, you would specify:   READTOEND:location:20:ASCII  BACKREFERENCE:state:3:location:ASCII   * REGEX:fieldName:regex:targetField:fieldType - Use this definition to write a regular expression as a fieldName specification. The last capturing group of the regex will be returned as the value for this field. The targetField is used to reference the fieldName defined in a preceding field definition line. A REGEX line must always follow the line that defines the field needed for this definition. For example, to extract the second number from a string of delimited fields such as:   ABC123;456;789DEF  specify:  READTOEND:data:0:ASCII  REGEX:second:\w+?\d+\D(\d+)\D:data:NUMBER  FieldType valid values include ASCII, NUMBER, or BYTES.  Here's a simple example that illustrates the use of all four definition types:  Incoming data string representing positions 0-9: ARDELP07WI   | Field Definitions | Results | | --- | --- | | READTOEND:rawdata:0:ASCII | ARDELP07WI | | READTOEND:type:8:ASCII | WI | | FIXED:transaction:3:4:ASCII | DELP | | FIXED:montha:6:2:NUMBER | 07 | | REGEX:monthb:\w+(\d+)\D:rawdata:ASCII | 07 | | FIXED:branch:0:2:ASCII | AR | | BACKREFERENCE:loc:0:branch:ASCII | A | |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |

CEF parser

The CEF parser works with ArcSight's Common Event Format, an open standard for logging security-related information in a common event log format.

| CEFParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |

CSV parser

The comma-separated values parser is a frequently used parser to extract fields from delimited text files.

The CSV parser requires a header to work properly.

| CsvParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |
| delimiter | Character used to separate fields in the incoming data  Default = , |
| mode | Indicates the mode in which the parser will run:   * Parse = parse and extract fields * Metrics = scan each record and collect statistics about the ingested data without parsing and extracting fields \*   Default = Parse |
| stop-on-missing-field | Specifies if DigitalEdge should stop processing a record when is encounters a missing field; double-click to select true or false from the drop-down menu  Default = true |

\* This parser can also be run in Metrics mode to generate statistics about the ingested data, to help determine how valuable and valid the data are. Metrics are collected by the parser for each processed file, such as:

* The total number of records processed
* Records that failed to be parsed by the DigitalEdge CSV parser
* Records in which the field count does not match the file header
* Records with non-ASCII characters
* Records with empty fields

A single record can generate multiple metrics. For example, a record could contain both non-ASCII characters and empty fields, and would be counted in the metrics as both having non-ASCII characters and empty fields.

To use this parser in Metrics mode, you must build a DigitalEdge system that also includes these components:

* Data Model: csv\_parser\_metrics
* Transport: One of the following:
  + DirectoryCrawlerTransportService
  + DirectoryWatcherTransportService
  + S3FileTransportService
* Data sink: MongoDbDataSink

For detailed instructions on building and using a system with CSV metrics, see "Generating metrics for CSV files" in the Operations Guide.

DNS PCAP parser

This parser reads DNS packets, extracts each packet from the data stream, and prefixes the packet with DigitalEdge data (including a timestamp). The DNS parser works with the PcapSnifferTransport, using a record-format parameter type of PCAP as a splitter; be sure to configure that transport in System Builder.

| DNSPCAPParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |
| discard-malformed-packet | Specifies if DigitalEdge should ignore malformed DNS packets; true = discard malformed packets; false = keep the packet, set the fields to empty, and add a field that identifies the packet as malformed  Default = true |

Email parser

This parser works with the RFC 822 ARPA email text format.

| EmailParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |

EXIF parser

This parser extracts data from Exchangeable Image Format files, used for handling image and sound files from digital cameras.

| EXIFParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |

JSON parser

This parser extracts data from files formatted in the JavaScript Open Notation standard.

| JSONParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |

Libpcap parser

This parser works with packet captures in the UNIX Libpcap library.

| LibpCapParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |

Log parser

This parser reads any log file (including firewall logs), extracts the timestamp based on a regex pattern you provide, and extracts the remainder of the string as the substantive log entry.

| LogParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |
| input-date-format | Specifies the format of the incoming timestamp value  Default = MMM dd yyyy HH:mm:ss |
| key-value-extract | Indicates if the parser should extract the key/value pairs from the log line(s); double-click to select true or false from the drop-down menu  Default = false |
| multi-line-record | Indicates if the log record spans multiple lines in the log file; double-click to select true or false from the drop-down menu  Default = false |
| output-date-format | Specifies the format to use when transforming the incoming timestamp to an output format  Default = MMM dd yyyy HH:mm:ss |
| timestamp-regex | Defines the regex pattern you want to use for identifying the timestamp in the log entry  Default = ([A-z]{3}\s\d{2}\s\d{4}\s\d{2}:\d{2}:\d{2}) |

SNMP pcap parser

This parser reads SNMPv1 or SNMPv2 traffic, extracts each packet, and prefixes the packet with DigitalEdge data (including a timestamp). The SNMP parser works with the PcapSnifferTransport, using a record-format parameter type of PCAP as a splitter; be sure to configure that transport in System Builder.

| SNMPPCAPParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |
| discard-malformed-packet | Specifies if DigitalEdge should ignore malformed SNMP packets; true = discard malformed packets; false = drop the malformed packet and send a record through that indicates that a malformed packet was identified and dropped  Default = true |

Unstructured file parser

This beta parser reads different unstructured files and extracts both the file's content and metadata (if available). When used with a transport that sends source file names, the UnstructuredFileParser preserves those file names automatically.

File formats that are processed by this parser include:

* HTML
* XML and derived formats
* Microsoft Office documents
* MBOX
* ODF
* PDF
* EPUB
* RTF
* TXT
* Audio
* Video
* Java class files and archives

See [https://tika.apache.org/1.0/formats.htm](https://tika.apache.org/1.0/formats.html)l for more details about these formats.

| UnstructuredFileParser | |
| --- | --- |
| Parameter Name | Explanation |
| content-encoding | Specifies the encoding used for the incoming data:  Default = Base64 |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |
| extracted-content-length | Maximum byte limit of content that should be extracted from a file. Do not set this parameter higher than the amount of available memory space.  Default = 268435456 |

XML parser

This parser works with files in the Extensible Markup Language format.

| XMLParser | |
| --- | --- |
| Parameter Name | Explanation |
| default-access-label | Distinguishes classified and unclassified data coming in; useful for arbitrating user access; valid values = UNCLASSIFIED, or any other classification marker  Default = UNCLASSIFIED |
| default-source | Specifies the source of the incoming data (e.g., IP address, URL, path) |
| field-handling-error-policy | Specifies the action to take when the parser finds a field that has not been defined, is empty, or that does not match your configuration specifications; choose DISCARD\_AND\_LOG to delete the field and create an error message in the log file, or DISCARD\_AND\_IGNORE to delete the field without writing a log entry  Default = DISCARD\_AND\_LOG |
| suffix-path-depth | DigitalEdge builds a key from element names in XML. If your data uses nested elements, all those elements are included in the key by default. If you have a high number of nested elements, the key may become too long for displaying and reading in the Data Model Editor > Data Source Mapping. In that case, you can specify a maximum number of nested elements to keep before DigitalEdge truncates the element list and drops the highest level elements. For example, if your data includes nested elements a, b, c, and d, and you specify 3 levels to keep, DigitalEdge will keep and use elements b, c, and d in the key. If suffix-path-depth = 2 and your data includes 10 nested elements, DigitalEdgewill trim the 8 highest levels and keep just the two lowest level elements.  Default = 0 |

About dimension table enrichments

Data enrichment is one of the most powerful features of DigitalEdge. Records in data sources are often coded, linked to other relational tables that provide contextual data. For example, with sales transactions, records may include data elements such as CUSTOMER\_ID, DEPT\_CODE, and PRODUCT\_ID; dimensional data tables provide the descriptive information keyed to those fields, such as CUSTOMER\_NAME, DEPT\_TITLE, and PRODUCT\_NAME. By specifying dimensional tables and associating them with data sources, you instruct DigitalEdge to pre-join the dimension tables with incoming data sources, enriching the data source records with relevant information in one fully processed input model record.

Configuring data tables that provide enhancement data and defining dimensional enrichments can be complex, depending on the number of enrichments you want to configure. But once data specifications are complete and accurate, dimension table enrichments can save time by:

* reducing the time it takes to receive search results from a complex query
* consolidating related information in one complete record for presentation to an analyst
* eliminating ambiguity with contextual data in pre-joined records

Dimensional enrichments require the specification of dimension tables, which store the enhancement data that you want to add to coded data in the DigitalEdgeinput model. You can define a dimension table at any time, but it is most logical to define the tables with the Table Manager tool before you specify enrichments in the Data Model Editor. In other words, when possible, you should define your source data – including dimension tables - before you configure data use in DigitalEdge. Data tables that are defined in the Table Manager determine what data will go into the dimension table enrichments that you specify in the Data Model Editor.

There are three types of keys in dimension tables:

Primary key: Every table has a master key which aids in read/write performance and with data integrity. This is the primary key. The primary key defines one or more columns which uniquely identify a row within a table. The primary key can be either a natural or a surrogate key. However, best practices dictate that the primary key would be a surrogate key.

Most database systems only allow a single key in a table, the primary key. However, you can have uniqueness indexes which identify secondary keys, which can either be natural or surrogate keys.

Natural key: A natural key defines one or more columns that uniquely identify each row in the table.The columns used in a natural key must be part of the original data set. For example, a unique customer key may be built with the C\_NAME and the C\_SSN columns. Or a table of airport codes may only need one field - the3-letter airport code - as a natural key that is unique for each record. Natural keys are most often defined with two or more columns. This key is used to index the records in the dimension table and to search the table for a matching row.

Surrogate key: A surrogate key is a single column added to the original data set, to uniquely identify each record. Any column type (string, number, alpha-numeric) can be used as a surrogate key as long as it is unique. However, a sequential numeric column is usually used for performance and simplicity. You need a surrogate key to insert a record into DigitalEdge. You do not need a surrogate key if you set the enrichment cache to ignore an unknown record (and not add it to the database). (See "Configure the enrichment cache" on page 91)

About the enrichment cache

When DigitalEdge ingests data source records, the enrichment processor checks input records for matches against the dimension records. It would be too slow to do full look-ups to match every transactional record against the dimension records, so DigitalEdge uses an enrichment cache for faster processing. The enrichment cache can hold dimension records in memory, but if no match is made between an incoming record and the cache, the enrichment engine does a full look-up on the dimension data, using the indexes that you specify for a particular table. Performance could decrease if look-ups occur too frequently. But the cache “learns” as processing continues, keeping the most recently used dimension records in memory and dropping the least frequently matched records out of the cache.

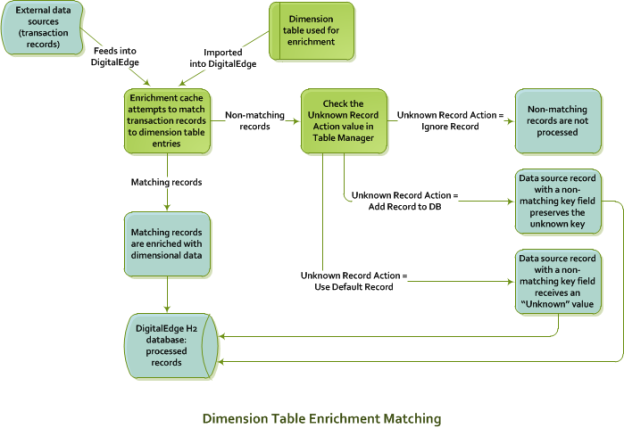
When you configure the enrichment cache, there are several specs that must be defined. First, you must define the size of the cache by specifying how many dimensional records should be held in memory for matching against input records. This number will depend on the size of each dimension table. You should also consider what resources are available on the ingest node for dedication to the enrichment cache. Obviously, there is a trade-off between resources and performance that you should experiment with in test mode.

You must also select a load strategy for the enrichment cache, to specify how the cache is initialized. This is another trade-off that must be weighed: speed of cache startup versus potential cache misses if the cache has not yet reached its maximum size. Load strategy options include:

* Eager: Completely filling the cache as soon as the system starts up the ingest node (for smaller dimension tables )
* Lazy: Creating the cache only as data comes in – this is a good strategy to use if the dimension records are used infrequently (for example, customer names for a big-box store, where customers make purchases infrequently)
* Background: Creating the cache when the system starts up the ingest node, by loading dimension records into the cache in the background or on-demand when a record needs to be processed

When the enrichment engine attempts to match input model records with dimension tables, the system could occasionally locate an unknown record. For example, a new customer may not be in the CUSTOMER database yet, so when a transaction comes in with a newly created CUSTOMER\_ID that does not match a key in the CUSTOMER dimension table, the system will fail to match the record to the CUSTOMER dimension table. When this occurs, DigitalEdge needs instructions for handling the unknown record. These instructions are specified as an Unknown Record Action on the Enrichment tab:

* Ignore Record: Stop processing the record entirely. The record will not be added to the DigitalEdge database. DigitalEdge will take no further action on the record.
* Add Record to DB: Insert the incoming record into the DigitalEdge database under a new surrogate key. The natural key and data values identified in the enrichment configuration will be stored in the table. Using the sales transaction example, a new transaction record is added to the DigitalEdge database with a CUSTOMER\_ID that does not match a record in the CUSTOMER dimension table.
* Use Default Record: Look up a default row in the dimension table and use it to enrich the record. Using the sales example, if a new customer has not yet been added to a dimension table, the CUSTOMER\_ID column in the sales transaction record may not find a match in the CUSTOMER dimension table. In that case, you could instruct DigitalEdge to add a value such as “unknown”or "UNK" to the enrichment field of the transaction record and continue processing other records.



How frequently DigitalEdge encounters an unknown record may depend on the integrity of your data sources. Data errors may occur frequently in a less than robust data source. By adding an “unknown” value to these records, you may be able to identify quality control issues that should be addressed in the application where the data source originated.

When you have the columns, indexes, and keys specified for a dimension table, you must map the dimension fields to the input model fields that will be enriched with data from this table. The Enrichment tab correlates dimensional data with DigitalEdgedata records.

About importing a dimension table

When you finish defining a dimension table, you can import your source enrichment data into the newly created dimension table at any time. A good practice is to import a few test records first, then work in the Data Model Editor to define a limited number of enrichments for prototyping. You can clear the dimension table and re-import records at any time.

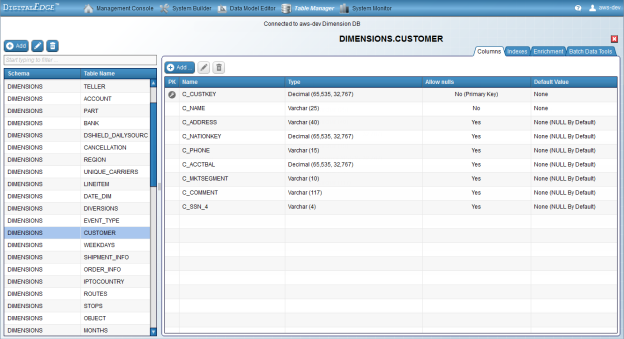
Step 3: Define Dimension Tables

Use the Table Manager to specify dimension tables as sources for dimension table enrichments. You do not have to use the Table Manager for generalized algorithmic enrichments.

Store the dimension tables in your DigitalEdge tables database. For each table that you want to use for enrichments, you must:

* Add a dimension table: to identify and name the source table that will be used for enrichment data (See "Add a dimension table" on page 85)
* Add columns: to specify every column in the table and the column properties (See "Add table columns" on page 86)
* Index a table: to specify keys and indexes for the dimension table that optimize the enrichment engine look-ups (See "Index a dimension table " on page 89)
* Configure the enrichment cache: to improve performance when matching input model records with dimensional enrichment tables (See "Configure the enrichment cache" on page 91)
* Map dimension columns to input model fields: to correlate the input model with the enrichment data sources (See "Map dimension columns to enrichment fields" on page 92)
* Import dimension data: to get your source enrichment data into the newly created DigitalEdge dimension table in your DigitalEdge tables database (See "Import dimension data" on page 94)

The Table Manager screen lists dimension table specifications:



* Database (top of screen): Name and connection of the tenant database in use
* Table list (left panel): List of all the dimension tables specified in the system
* Table specifications (right panel): Table columns, indexes, enrichment cache specifications, and batch data tool options for a selected table

When adding a table, your work is not saved until you click Save.This means you can easily add and edit table details. However, once a table is saved, all edits are atomic - changes must be completed one at a time and automatically saved. When you work with the Enrichment tab, you can Reset the screen to cancel all the changes you have made during that session.

The Table Manager also controls the manipulation of application-level tables for custom user applications. For the purposes of defining dimension table enrichments, work with DIMENSIONS entries only, not APPLICATION tables.

Add a dimension table

While you identify data sources for input records in the Data Model Editor, you identify data sources for enrichment data in the Table Manager. You may have one dimension table for each enrichment you define. Or, one table could feed data into multiple enrichments, using the same or different columns for each enrichment. The ratio of dimension tables to enrichments is not necessarily 1:1. However, all fields for all enrichments must be declared on the Enrichment tab of Table Manager.

If you do not define dimension table enrichments for your system, you do not have to use the Table Manager. Algorithmic enrichments do not require the creation of dimensional tables.

When you first open the Table Manager tool, it will be empty. There are no default dimension tables or pre-defined enrichment tables; you must identify and name dimension tables specific to your organization. A dimension table may originate from an SQL table, located in a corporate data warehouse. It may be a simple CSV file linked to other systems. Or it may be an XML file in a custom application. Whatever the original location, the data table must be loaded into your DigitalEdge tenant database on the gateway node. This tables database is listed in the Table Manager banner as the Connected to database. The sole purpose of the tables database is to store all your dimension tables used for enrichments.

You probably want to duplicate the name and specifications of the original corporate table source here when you Add a table to your tenant database as a dimension table. Think of a dimension table as a staging table, used to pass data from a corporate source to DigitalEdge. The table is an interim step, because selected columns from the table will be extracted for integration into your DigitalEdgeinput model as enrichment data.

To add a dimension table and identify it to the system:

1. Access the Table Manager tool   from the Management Console Tools list.
2. The name of your tables database is displayed at the top of the screen. The left panel will be blank if you have not yet added any tables.
3. Click the Add a Table button   at the top of the left panel.
4. Select a Schema from the drop-down list.   Select DIMENSIONS to define an enrichment table. (Table Manager is also used to manage APPLICATION tables that are needed by DigitalEdgeweb apps and custom user applications.)
5. Enter a Table Name.

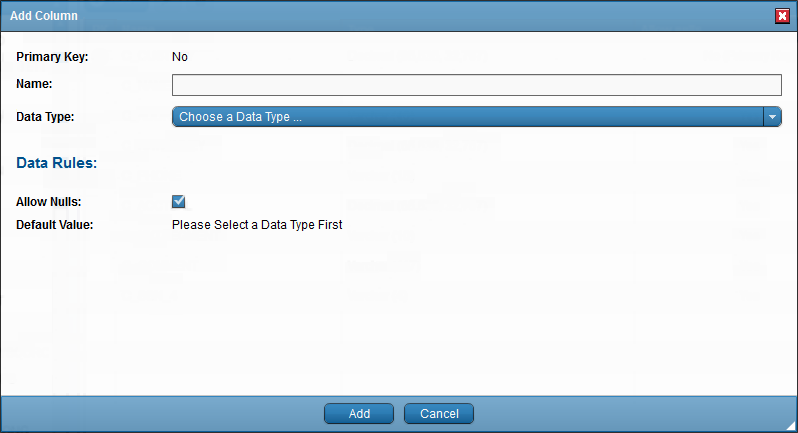
Add table columns

Even though you may just need one table column to enrich an input field, or a subset of fields to augment data in the input model, you should define all the columns in the table for accuracy. Defining columns in the Table Manager (with the Columns tab) is simply a task of naming each column and its properties as they were defined in your corporate source. Nine times out of ten, you will replicate SQL specifications here, including identical names in uppercase and column types. For example:

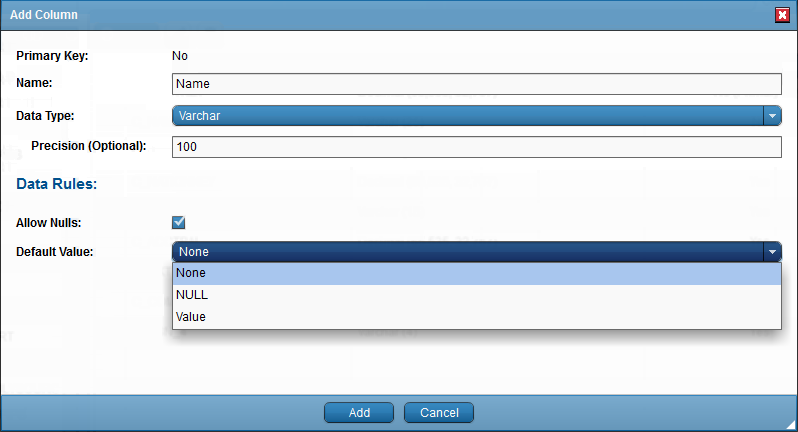
| Column | Type | Property |
| --- | --- | --- |
| C\_CUSTKEY | Decimal | Primary Key |
| C\_NAME | Varchar (25) | No nulls |
| C\_ADDRESS | Varchar (40) | Nulls allowed |
| C\_NATIONKEY | Decimal (65,535, 32,767) | Nulls allowed |
| C\_PHONE | Varchar (15) | Nulls allowed |
| C\_ACCTBAL | Decimal (65,535, 32,767) | Nulls allowed |
| C\_MKTSEGMENT | Varchar (10) | Nulls allowed |
| C\_COMMENT | Varchar (117) | Nulls allowed |
| C\_SSN\_4 | Varchar(4) | Nulls allowed |

To specify all the columns (fields) in the dimension table:

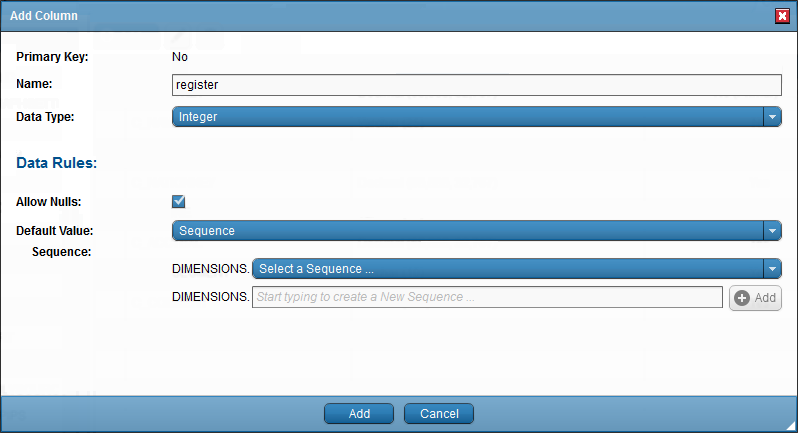
1. Make sure you defined the table Schema and Table Name when you added the table. ([See "Add a dimension table" on page 85)](file:///C:\Users\vahlbergm\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\JKPRDWJA\DigitalEdge%20Configuration%20Guide%201.3.doc#-1802542320)
2. Click the Add a Column button  in the right panel. The Add Column dialog box appears.



1. Specify each column property:
   * Primary Key: Click to designate this column as the table’s primary key. DigitalEdge provides a surrogate key for the table’s primary key, a unique arbitrary number.
   * Name: Enter the name of the column
   * Data Type: Select a data type option from the drop-down menu of tenant database data types (e.g. Boolean, Decimal, Date, Varchar, etc.)
   * Precision (Optional): Enter the max length of a value (for select Data Types only)
   * Scale (Optional): For Decimal types only, the number of places to the right of the decimal point
   * Allow Nulls: Click to permit null values in this column
   * Default Value: Provide a default value type for a dimension table column that is imported or inserted into the tenant database without a value in the column. Be sure to add a default value for the Primary Key in uploaded dimension tables.



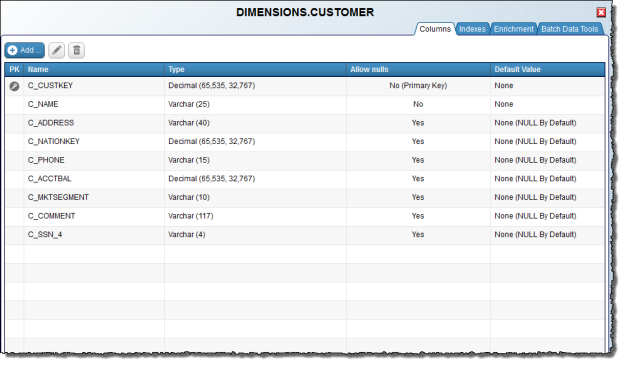
* None: No value is specified
* NULL: NULL is provided by default
* Value: A specific data value that should be assigned to this column when it is empty in an imported record. Enter the default value in the provided Value field.
* Sequence: This Default Value is available for any column specified as a numeric data type. This value is typically assigned to a Primary Key when the next available number in a Sequence should be assigned to the column. Select or create a sequence in the following fields.



* + Sequence: When editing a table, Select a Sequence from those already defined in Table Manager. When creating a new dimension table, enter the sequence name in the provided text box and click Add to include the new sequence in the drop-down list of sequences. A sequence is usually named as TABLENAME\_COLUMNNAME\_SEQ. Sequence names must be unique.

1. Click Add to create the column. Or click Add & New to save this column and create another one.
2. Click Create Table when you have specified all the columns.  

Here is a sample specification for a dimension table named CUSTOMER:



Index a dimension table

Once you have a table and its columns defined, use the Indexes tab to specify keys and indexes to build for the dimension table. When DigitalEdge ingests data source records, the enrichment processor checks input records for matches against the dimension records. Indexes are used when the enrichment engine has to do a full look-up on the dimension records. For more details about table look-ups and the enrichment process, See "Configure the enrichment cache" on page 91

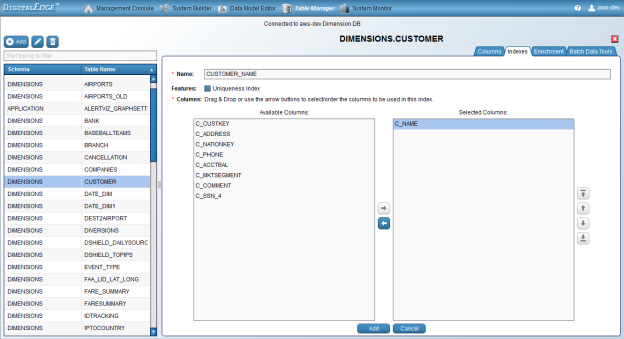
There are three types of keys in play here:

* Natural key: A natural key defines one or more columns that uniquely identify each row in the table. For example, a unique customer key may be built with the C\_NAME and the C\_SSN columns. Or a table of airport codes may only need one field - the 3-letter airport code - as a natural key that is unique for each record. Natural keys are most often defined with two or more columns. This key is used to index the records in the dimension table.
* Surrogate key: A surrogate key is a sequential number unique to each record. You need a surrogate key to insert a record into DigitalEdge. You do not need a surrogate key if you set the enrichment cache to ignore an unknown record (and not add it to the database). (See "Configure the enrichment cache" on page 91)
* Primary key: The primary key could be a natural key or a surrogate key, but it must be unique for each row in the table. A primary key is usually a meaningless, sequential number; it can be used as a surrogate key if that is the case.

Typically, you will have both a natural key and a surrogate key for each dimension table.

To index a dimension table:

1. Double-click on a table name in the left panel. Its column list appears in the right panel.
2. Click the Indexes tab. If the selected table already has an index configured, the index specifications appear.
3. Click Add… to create a new index on this table.   The Indexes dialog appears.



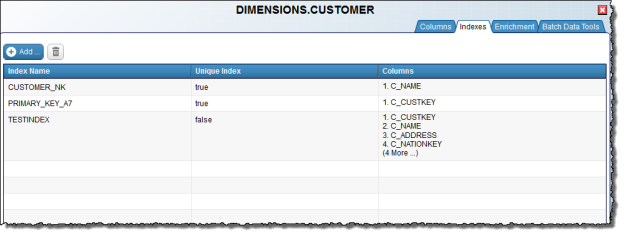
Columns that are available for indexing are in the left panel. Columns that have already been selected for indexing appear in the right panel.

1. Enter a Name for the new index.
2. Select Uniqueness Index if the combination of indexed values must be unique on incoming records. If the uniqueness test fails when DigitalEdge processes a record, the record will not be indexed.
3. To specify an index, click on an available column in the left panel. Click the right arrow icon to move the column into the Selected Columns panel. Or, drag and drop a column from the left into the right panel.
4. You can re-order columns in the index list with the up and down arrow icons. Or drag and drop the columns in the appropriate order. Indexed columns should be sorted in optimum comparison order, from the most likely matching column to the least likely match.
5. Click Add to save the index specification.

To edit an index specification, Delete  the index and Add it again.

You can Delete all the indexes for a table when your system is not running. However, when you delete the index for the primary key, the table will have no primary key and you won't be able to add another primary key.

Here is a sample specification for the CUSTOMER table indexes:



Configure the enrichment cache

The next few steps in defining a dimension table include cache parameters for:

* matching an enrichment record with the data’s natural key
* adding data from a matching dimension record to the input model record (enriching an incoming record)
* defining the in-memory enrichment cache size by specifying how many dimension records can be held in memory during enrichment processing
* managing records that do not match a database record

For an explanation of how the enrichment cache works, See "About dimension table enrichments" on page 81.

To specify the enrichment cache:

1. Click the Enrichment tab.
2. Specify a Load Strategy to indicate how the cache is initialized at enrichment startup:
   * Background: the cache is created immediately when you start the system in the Management Console
   * Lazy: the cache is created only as data comes in
   * Eager: DigitalEdge will try to fill the cache as soon as you start up the system in the Management Console
3. Specify the Max Rows in Cache to indicate the maximum number of enrichment records that should be kept in memory. Consider timely enrichment and performance versus availability of resources on your ingest nodes.
4. Select an Unknown Record Action to specify what DigitalEdge should do when it encounters an unknown record that does not match the dimension table specifications:
   * Add Record to DB: Insert the incoming record into the database

Also, define default values (in the Enrichment Data Config panel in the Values to Insert column) for any fields with an Enrichment Role of Data and which do not allow nulls. (See "Map dimension columns to enrichment fields" on page 92)

* + Ignore Record: Skip the record entirely
  + Use Default Record: Provide the data values to use when looking up a default record that will be used as the enrichment source.

Default values are specified in the Enrichment Data Config panel in the Values to Insert column; for example, "UNK" or "unknown". (See "Map dimension columns to enrichment fields" on page 92)

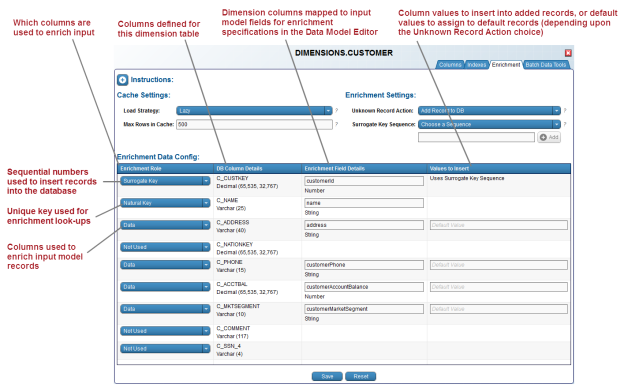
1. If you chose Add Record to DB as the Unknown Record, you must select a Surrogate Key Sequence, a numeric sequence to use when assigning a surrogate key to new records. You can select a Surrogate Key Sequence from those already defined in Table Manager, or you can enter a new sequence name in the provided text box and click Add to include the new sequence in the drop-down list of sequences. A sequence is typically named as TABLENAME\_COLUMNNAME\_SEQ. Sequence names must be unique.

Map dimension columns to enrichment fields

Once you have the columns specified for a dimension table, and you have defined indexes and keys for the records, you can map the dimension fields to the input model fields that will be enriched with data from this table. The Enrichment tab correlates dimensional data with DigitalEdgedata records.

Looking at the Enrichment tab:

* The newly defined columns for a dimension table are listed in the center panel under DB Column Details (the columns you defined for the dimension table). This column reflects the database structure of a dimension table.
* On the left, the Enrichment Role indicates if a column is used to enrich DigitalEdge records, or if it is a key for look-ups.
* To the right of the DB Column Details, Enrichment Field Details map dimension columns to input model fields, indicating which dimension fields will be used to enrich input records. These enrichment field names will be referenced in the Data Model Editor when you specify dimension table enrichment models.



On the Enrichment tab, work with one field at a time under DB Column Details to map dimension columns to input model fields:

1. The Enrichment Role Indicates if a dimensional column is used to enrich input records, or if it is a key for enrichment look-ups. Choose a role for each column in the dimension table:
   * + Surrogate Key: Values in this dimensional column are unique sequential numbers used to insert a record into DigitalEdge. You need to define a surrogate key if you set the enrichment cache to Add Record to DB.
     + Natural Key: A unique key, indexed for enrichment look-ups
     + Data: Values in this dimensional column will be used to enrich input model records, to enhance data sources with additional, clarifying information
     + Not Used: This dimensional column is ignored for enrichment purposes
2. Enrichment Field Details map dimension data columns to enrichment fields in the input model. This is where you specify the field names that will be used to enrich the input model in the Data Model Editor.These field names are stored in JSON. Recommended naming standards include:
   * No quotation marks
   * No spaces; use underscore characters for spaces
   * Use camelCase for adjoining words
   * Choose names that are similar to the dimension table names, but unique enough to identify the field names as DigitalEdge/JSON names. For example, dimension columns may be specified in UPPERCASE; enrichment fields may be named in camelCase.
3. Default Record Values are required if you selected an Unknown Record Action of Use Default Record.

Supply a default value such as “unknown”or "UNK" for any field in the transaction record that contains a value that doesn't match a record in the dimension table.

1. Values to Insert are required if you selected an Unknown Record Action of Add Record to DB.

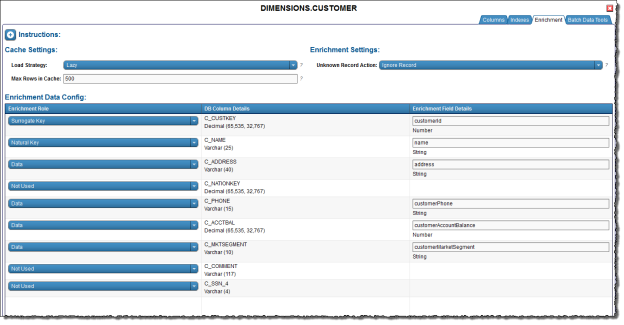
Table Manager prompts you to enter default values for any field with an Enrichment Role of Data and which does not allow nulls.

1. Save the cache specifications. Validation results appear at the top of the screen, such as:



Correct any problems and re-save the specifications. Or click Reset to cancel all edits and additions you made during this session.

Here is a sample specification for the CUSTOMER table enrichment specs:



Print out this Table Manager screen, or keep it up on a second monitor when you specify enrichments in the Data Model Editor to assure exact matches on enrichment field names.

Import dimension data

When you finish defining a dimension table, you can import your source enrichment data into the newly created dimension table at any time. A good practice is to import a few test records first, then work in the Data Model Editor to define a limited number of enrichments for prototyping. You can clear the dimension table and re-import records at any time.

You can upload dimension table data into the tables database (located in the tenant database on your gateway node) via one of two methods:

* Using the Dimension Data Sink to input data into the tenant database (See " Step 3: Add Data Sinks" on page 141 and See "Data sink parameters" on page 143)
* Imported as a CSV file or a ZIP file with the Table Manager>Batch Tools>Import functionality (instructions follow below)

Additional import mechanisms will be available in a future release.

Formatting guidelines for the CSV file include:

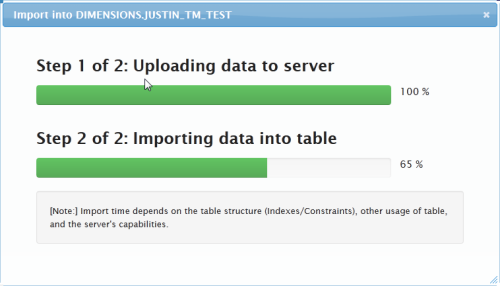
* The file should be a plain text file with the CSV extension.
* The current maximum import file size is 250 megabytes. If your file is larger, you can break it into multiple CSV files and import them in a ZIP file.
* The first line of the file must be a header row that identifies all the column names in a comma-separated string.
* The order of the column names in the header much match the order of the columns in the actual table (the order in the Columns tab).
* Each line must be separated by a newline character.
* Values can be enclosed in double quote marks or can be imported without quote marks; single quote marks do not work.
* Each data line must include the same number of values as specified in the header row; use NULL to use a default field value.

To save space and to reduce the number of individually imported files, you can put several CSV files into a ZIP archive. Guidelines for the ZIP file include:

* The file must have a ZIP extension.
* A zip file should contain one or more CSV files specified according to the CSV guidelines listed above.
* CSV files are imported and processed in the order in which they are listed in the ZIP file.

Use the Batch Data Tools tab to import data from a local CSV or ZIP file into your tenant database:

1. Stop any system that is using the current table.
2. Set default values for all the columns (on the Enrichment tab).
3. Remove all indexes except the Primary Key index before starting a large import.
4. On the Batch Data Tools tab, optionally click Purge  if you want to clear all data from the current dimension table in your tenant database. Purge is useful when you are building a prototype and testing multiple versions of dimension table specifications.
5. Click Import .
6. Click Browse to locate the dimension data in your local file system. Select the appropriate table then click Upload to feed the data into the newly defined dimension table in your tables database. A two-part progress report tracks the time to upload and import the data:



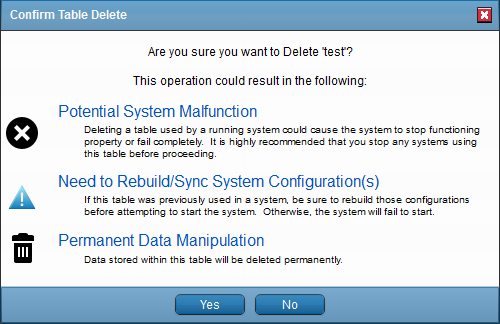
Delete a table

You can delete a table that you no longer need, or that you created for test purposes only.

To retain a table but delete the data in the table, use the Purge function on the Batch Data Tools tab of the Table Manager.

To delete a table from DigitalEdge:

1. Access the Table Manager tool   from the Management Console Tools list.
2. Highlight one or more tables in the left panel.
3. Click the Delete button   at the top of the left panel.
4. A confirmation dialog box appears, warning you of potential issues when deleting a table. If none of the conditions apply, continue to delete the table(s) by clicking Yes.



Delete data from a table

You can retain a table but delete the data from the table with the Purge function. This is most useful when you are testing variations of a dimension table's specifications.

To delete table data:

1. Access the Table Manager tool   from the Management Console Tools list.
2. Highlight a table in the left panel.
3. On the Batch Data Tools tab, click Purge .
4. When the confirmation dialog box appears, click Yes.

Exit Table Manager

1. As you Add tables and columns, Table Manager automatically saves your work.
2. To log out of the Table Manager, click your user icon/name in the upper right corner. Select Sign Out.

About enrichments

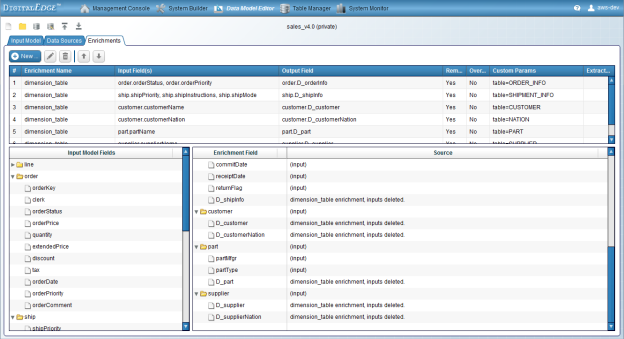
In the Data Model Editor, the Enrichments tab specifies fields that will be enriched with contextual data. There are two types of enrichments that you can define with the Data Model Editor:

* dimension\_table enrichment is a table lookup that pre-joins a dimension table to the main data feed; you must first define the source dimension table in the Table Manager before you can configure the dimensional enrichment here with the Data Model Editor. (See "Step 3: Define Dimension Tables" on page 84 for details.)
  + fuzzy match: beta processor that enriches records with a standardized string based on a fuzzy match lookup on a specified input field
* Generalized enrichment, or algorithmic enrichments, do not use dimension tables for enrichment data but use other standard data sources that are supplied with DigitalEdge. Enrichments that are available include the following. (Note that this list includes the enrichment that are included in the core release. You may also have custom enrichments created for your site.)
  + ip\_network generates location data in JSON format for a given IP address, including:

Country   
Region (e.g., state)   
City   
Postal Code   
Latitude and longitude   
Metro Code   
Area code

* + math\_enrichment runs a mathematical expression that you create against the data; for example, calculate the time between two dates, add a product price and shipping charge, determine the distance between two locations, etc.
  + postal\_location computes the nearest zip code, city, state, or country within 150 kilometers of a given latitude/longitude
  + record\_history adds an array of records that recently passed through DigitalEdge and that match a specified field; requires a MongoDB data sink
  + regex\_extractor: extracts sub-fields from a more complex input field
  + sql\_select\_enrichment looks up information in an SQL databse with a query of your choice

Here is the complete enrichments specification screen that you are working towards when defining Enrichments in the Data Model Editor:

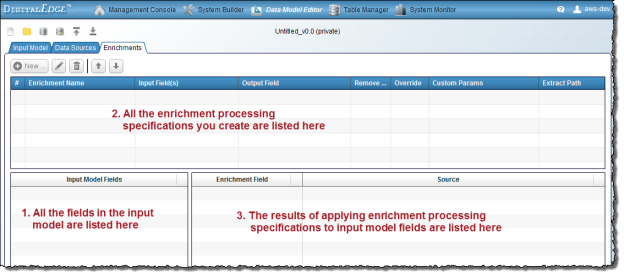


Step 4: Define Enrichment Processes

Use the Enrichments tab to specify fields that will be enriched with additional data.

The Enrichments configuration screen in the Data Model Editor uses three panels:

* Enrichment specifications (top panel): a list of all the enrichment definitions that you specified in the Data Model Editor, complete with inputs, outputs, enrichment types, and custom parameters that are unique to a specific enrichment type
* Input Model Fields (bottom left panel): all the fields from the input model are listed; some or all these fields can be enriched
* Results (bottom right panel): the final field results result after applying an enrichment processor to an input model field



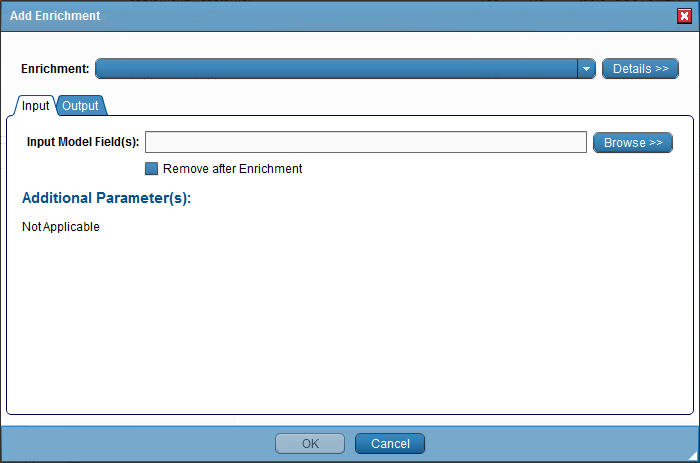
Work is done in JSON representation, including arrays, objects, and fields. You can expand or close an array entry with the arrowhead to the left of an entry.

Before you define a dimension table enrichment, use the Table Manager tool to specify the dimension tables that are the sources of the enrichment data (See "Step 3: Define Dimension Tables" on page 84).

DigitalEdge applies enrichments to your data based on the order in which you create them. The Enrichments specifications panel at the top of the Data Model Editor screen lists enrichments in that order. If you are applying multiple enrichments to input fields, be sure to create the enrichments in the order in which the data should be processed, or reorder the enrichment specifications when you are done (See "Reorder enrichment specifications in the top panel" on page 105).

Add an enrichment specification

1. Select the Enrichments tab.  The input model fields available for enrichment are displayed in the lower left panel.
2. Click the Add icon.   
3. The Add Enrichment dialog box opens.



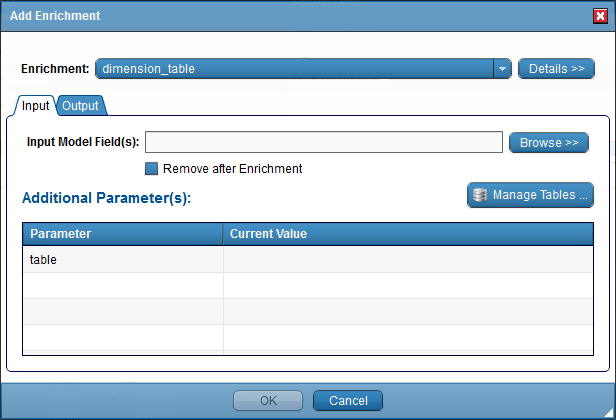
1. Select the type of enrichment you are defining from the Enrichment drop-down menu. (Note that this list includes the enrichments that are included in the core release. You may also have custom enrichments created for your site.)
   * dimension\_table: adds a dimension record from a previously defined relational table, using specific key field(s) for exact match lookups
   * fuzzy\_dimension\_table\_plugin:beta processor that enriches records with a standardized string based on a fuzzy match lookup on a specified input field
   * ip\_network: generates location data (in JSON format) corresponding to a given IP address. This enrichment uses a large database to calculate the location; please contact Leidos for the IPGeoInstaller to deploy this database.
   * math\_enrichment: runs a mathematical formula against the data (for example, calculate the time between two dates, the distance between two locations, etc.)
   * postal\_location: computes the nearest zip code, city, state, or country within 150 kilometers of a given latitude/longitude pair
   * record\_history: adds an array of records that recently passed through DigitalEdge and that match a specified field; requires a MongoDB data sink that is used to store and match the transaction records
   * regex\_extractor: extracts sub-fields from a more complex input field
   * sql\_select\_enrichment: look up information in an SQL databse with a given query

Optionally click Details to read longer definitions of the enrichment processors. The Add Enrichment dialog box changes depending on the type of enrichment you chose.

1. To work with an Input Model Field, click the Browse button. The Field Selection dialog box appears. Click a Field name that comes from your input model and click OK. Note that some enrichment types use multiple input fields; Select each one individually.
2. Click Remove After Enrichment to get rid of the input field from the final record after the enrichment has been processed. This option is typically chosen for dimension table enrichments.

Specify enrichment parameters

If you selected certain enrichments, you must specify Additional Parameters.

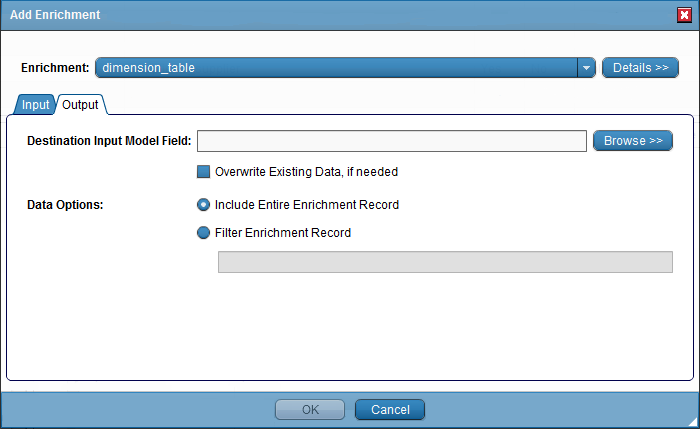


For details about enrichment parameters, See "Enrichment parameters" on page 106.

These specifications will appear in the Custom Params column in the top Enrichments panel.

Configure enrichment output fields

1. To work with a field that will be the destination field for storing the enriched data, click the Output tab.



The Destination field can be:

* The input field, overwritten with enriched data
* A newly created field, used solely for enrichment data
  1. To work with a Destination Input Model Field, click the Browse button. The Field Selection dialog box appears. Click a Field name that comes from your input model and click OK.

Or, to define a new field for enrichment output, highlight the field below which the new field should appear, and click Add Field. Enter a new field name and click OK.

It is a best practice to retain the original input field and add a new destination output field for the enrichment data.

* 1. Click Override Existing Data, if needed if the enrichment should replace any existing value in the output field that you selected.
  2. Indicate if the entire enrichment record should be included in the destination output field (Include Entire Enrichment Record), or if the enrichment should be filtered (Filter Enrichment Record) to one relevant field.

Enrichment data often comes in as multiple fields in an object, assigned to a destination output field. However, there are times when you don't need to save all the enrichment fields, but you just need to keep one column in the output field. For example, when you define a postal\_location enrichment, the algorithm converts latitude/longitude data and returns three enriched fields: city, state, and zip code. If you want to filter the enrichment record and exclude state and zip code information, you could filter the enrichment and just save the city field.

To filter a record, enter a JSON expression in the text box below the Filter Enrichment Record selection. The expression should include the name of the enrichment field that will be saved to the destination output field (e.g., city).

Filtered fields will appear in the Extract Path column in the top Enrichments panel.

1. Click OK.

**Enrichment example**

Dimension\_table enrichments are the most commonly defined enrichment types. As an example of dimensional enrichment, assume that your input model includes a field called customer that comes in with a customer ID number. You would like to enrich that field with the actual name of the customer instead of the customer's ID. The CustomerName is stored in the CUSTOMER dimension table. Here are the enrichment specifications you would enter:

Enrichment:  dimension\_table

table: CUSTOMER

Input Field: customer

Remove After Enrichment: No

Output Field: customer

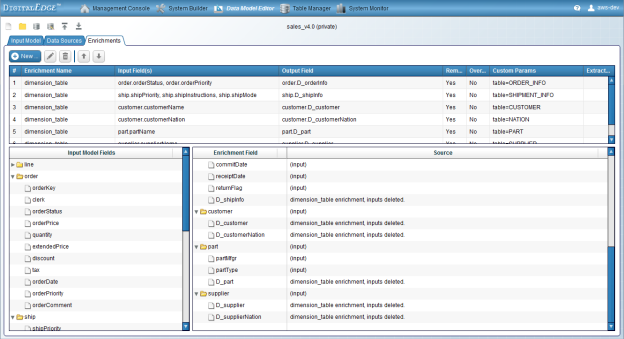
Overwrite Existing Data, if needed: Yes

Filter Enrichment Record: Yes

Filtered field: CustomerName

Before you define this enrichment, you must pre-define the CUSTOMER dimension table in the Table Manager setup tool.

Here is what Enrichments screen looks like with completed specifications for several enrichments in an input model:



Edit an enrichment specification

1. Double-click anywhere on the Enrichment statement in the top panel. Or, highlight an Enrichment statement and click the Edit Enrichment icon.  
2. Change any part of the specification in the Add Enrichment dialog box.
3. Click OK.

Reorder enrichment specifications in the top panel

DigitalEdge applies enrichments to your data based on the order in which they appear in the top panel. If you are applying multiple enrichments to input fields, be sure to create the enrichments in the order in which the data should be processed, or reorder the enrichment specifications.

1. Highlight an Enrichment statement in the top panel.
2. Click the Move Enrichment Up icon   or the Move Enrichment Down icon . The specification moves one row at a time.
3. Repeat the process as needed.

Delete an enrichment specification

1. Highlight an Enrichment statement in the top panel.
2. Click the Remove Enrichment icon.  
3. Click OK to confirm deletion of the enrichment, or click Cancel to save it.

Enrichment parameters

Each enrichment uses a set of parameters that are unique to its operation.

You can access enrichment parameters by creating or editing an enrichment in the Data Model Editor. See "Step 4: Define Enrichment Processes " on page 99 for more information.

Here are detailed lists of enrichment parameters, descriptions, and values for the enrichments included in the core release. You can hover over a parameter name in DigitalEdge for tool-tip help.

Once you define an enrichment, these parameter specifications will appear in the Custom Params column in the top Enrichments panel.

Dimension table enrichment

Data enrichment is one of the most powerful features of DigitalEdge. By specifying dimensional tables and associating them with data sources, you can pre-join the dimension tables with incoming data sources, enriching the data source records with relevant information in one fully processed input model record. Dimensional enrichment creates one master record that contains all relevant information in one location.

Be sure to define the source table first before configuring a dimension table enrichment. Click Manage Tables to access the Table Manager from the Add Enrichments dialog box.

Dimension table enrichments use the following parameters:

| Dimension\_Table Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | Select one or more fields from your input model which will be enriched with dimension table data |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |
| table | Specifies the dimension table to work with; double-click in the Current Value area and enter the name of the source dimension table to use for an exact match key lookup |

Fuzzy dimension table enrichment

This beta enrichment looks for values in your specified input field that match or come close to matching a string. For example, you may want to find any record about "Leidos", "Leidos Inc.", "Leidos Corp.", or "Leidos Corporation" and standardize them all on "Leidos". This enrichment first looks for and determines fuzzy matches in your data, and then does a dimension table enrichment on matching records. It adds a dimension record from a relational table using the specified natural key field as the fuzzy match lookup.

| Fuzzy\_Dimension\_Table Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field | One field from your input model that will be used as a fuzzy match to the natural key in the dimension table; this field will be enriched |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |
| column | Identifies the column in the table parameter used to match against input strings |
| decisionThreshold | The minimum "closeness" score of a fuzzy match required to qualify for enrichment. Specify any value from 0 to 1. Choose a value that indicates your confidence level that the fuzzy matcher has succeeded at identifying a near match. For example:  .65 is a good threshold to start with  .5 may identify questionable matches (not very close of a match) in your data  .99 may only work if a string is one character different than the match |
| englishTable | An optional table of the most common English words to help the fuzzy matcher determine if a word is important to include in the matching algorithm. The table must include two columns: WORD and FREQUENCY. You can leave this parameter blank, build your own table, or use a publicly available table. |
| passThroughOnNoMatch | Indicates if a field is retained or dropped when it does not match the fuzzy matcher algorithm.  true = retain the input data as-is in the dimension table when no match is found  false = drop the original string which does not match and stop processing this record |
| synonymTable | An optional dimension table that you create in your dimensions database that lists synonyms for the fuzzy matcher to use. The table must include two columns: SYNONYM (a regex) and RESULT (the replacement string). For example, matches for "incorporated" may include:  \sinc\.," incorporated"  \sinc\s," incorporated"  \sinc$," incorporated" |
| table | The source dimension table in a relational store that lists the preferred entries to save as enrichment replacements; double-click in the Current Value area and enter the name of the source dimension table |

IP network enrichment

The IP network enrichment is an algorithmic enrichment. For a given IP address, It generates location data in JSON format, including:

* Country
* Region (e.g. State)
* City
* Postal Code
* Latitude and Longitude
* Metro Code
* Area Code

| IP\_Network Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | One or more fields from your input model which will be enriched |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |

Math enrichment

The math enrichment is an algorithmic enrichment that runs a mathematical formula against the data. You specify the formula with the enrichment parameters. For example, you could calculate the time between two date stamps, or the distance between two locations.

| Math\_Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | One or more fields from your input model which will be enriched |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |
| constants | Specifies the names and numeric values used as constants in the mathematical expression, in the format:  constant\_name=numeric\_value  Constants are separated by commas or spaces. For example:  pi=3.14,c=186282 |
| expression | A mathematical formula or expression to run against the data. The expression must consist of only constants and variables (defined in the variables parameter below). For example, to compute a total sales cost for a transaction, you could reference input field names as variables, such as quantity = Q, cost = C, and taxRate = R, in a math enrichment expression:  Q\*C\*(1.00 + R) |
| forceZeroIfArrayNull | This parameter fills the result array with zeroes if the array does not allow null values  Default = false  Recommendation: keep the default value of false to fill the array with zeroes |
| variables | A list of field names and numeric values used in the mathematical expression that map to the input model fields that you selected for this enrichment, in the format:  variable\_name=numeric\_value  where the numeric\_value identifies the sequential position of the referenced input model field (starting at 1) for a variable name, matching each input model field to a variable name in this list. Variables are separated by commas or spaces.  For example, if you are using the following input model fields and variable names:   | Input Model Field | Sequential Position | Variable Name | | --- | --- | --- | | store | 1 | S | | cost | 2 | C | | product | 3 | P | | quantity | 4 | Q | | tax\_rate | 5 | R |     and you are referencing quantity, cost, and tax\_rate in the expression:  Q\*C\*(1.00 + R)  the variables list would be:  Q=4,C=2,R=5  Field names are represented by variable names to avoid problems when using long Input Model Field names or field names that include periods in the mathematical expression. |

Postal location enrichment

The postal location enrichment is an algorithmic enrichment that takes a latitude/longitude pair and calculates the nearest city, state, zip code, or country (within 150 kilometers).

| Postal\_Location Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | One or more fields from your input model which will be enriched |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |

Record history enrichment

The record history enrichment is an algorithmic enrichment adds an array of records that recently passed through DigitalEdge and that match a field specified by you. This enrichment requires a MongoDB data sink for storing and matching the transaction records.

| Record\_History Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | One or more fields that represent the matching fields for this enrichment. For example, you might match on an Account\_ID field when looking for bank transactions on a customer's account number. |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |
| comparisonField | The input model field to use as the sorting factor for the incoming transactions to identify the most recent records. For example, this could be a UNIX Timestamp field representing time in milliseconds, as defined by an input model field Type (Number, Date/Time, etc.). The comparisonField must exactly match the name of an input model field. |
| databaseName | The MongoDB database name that the enrichment will search for matching records. The name must match exactly the name that you specified when configuring the MongoDB data sink in System Builder. |
| history | The number of records to fetch. For example, the most recent 3. |
| timeLimit | The time period in which transactions should have occurred to be eligible for recent record selection. timeLimit is expressed in the same units as comparisonField (e.g., UNIX Timestamp). For example, to retrieve transactions in the past hour, enter a value = 3600000. Enter 0 to ignore this parameter. |

Regex extractor enrichment

The regex enrichment extracts sub-fields from a more complex input field. For example, if you are parsing log files into timestamps and messages, you may want to enrich the message field by extracting an IP address from the text string. The enrichment can be used to pull any piece(s) from any input field in your data model.

| Regex\_Extractor Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | One or more fields from your input model which will be enriched |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |
| asArray | Specifies that the extracted sub-fields should be returned as an array of objects, one per match. |
| fieldMapping | The field names mapped to capture groups or the whole pattern. For example, if you want to extract an IP address and port as separate sub-fields from a log file message, the mapping may be:  “wholeMatch=0, ip=1, port=2(UNKNOWN)”  Three fields are specified: the complete extracted string (wholeMatch) which includes the IP and port sub-fields, an extracted IP address (ip), and an extracted port number (port). Parenthetical data identifies default values for the sub-fields. |
| regExPattern | Any regular expression string (including Java regex patterns) used to extract sub-fields |
| repetitionCount | The number of occurrences to extract. Use zero to extract all occurrences of the regex pattern from an input model field. If nonzero, sub-fields will be appended with the match name. For example:  wholeMatch1, ip1, port1, wholematch2, ip2, port2 |

SQL select enrichment

The SQL select enrichment is an algorithmic enrichment that looks up information in an SQL database with a query that you specify.

| SQL\_Select\_Enrichment | |
| --- | --- |
| Parameter Name | Explanation |
| Input Model Field(s) | One or more fields from your input model which will be enriched |
| Remove after Enrichment | Click to delete the input model field(s) from the final record after it is enriched |
| databaseConnectionString | The connection string needed to communicate with the SQL database |
| databaseDriverName | The name of the SQL database driver loaded from the classpath |
| databasePassword | The password used to connect to the SQL database |
| databaseUserName | The username to connect to the SQL database |
| firstResults | Optional parameter, set if you only want one value from the database; true or false |
| query | The complete SQL query that you want to run against the database |

Step 5: Save a Data Model

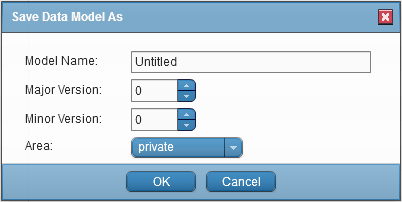
Save your data configuration work when you are done.

You can also save a partially edited session and return to the Data Model Editor later to continue your configuration work.

1. Click the Save icon on the Data Model Editor title bar.   

OR

Click the Save As icon  on the Data Model Editor title bar to save your data model in the System Repository.



* 1. Enter a Model Name for your new private data model.
  2. Select a Major Version number.

You should not overwrite an existing data model that is in use in a running system.

* 1. Optionally select a Minor Version number.
  2. Click the drop-down arrow on the Area menu and select private.
  3. Click OK.

1. To log out of the Data Model Editor, click your user icon/name in the upper right corner. Select Sign Out.

Step 6: Edit or Delete a Data Model

You can modify an existing data model at any time. DigitalEdge uses a data model versioning system that lets you keep multiple versions of a data model. Every data record is tagged with its data model name and version, allowing multiple versions to exist in the same data store. This lets you easily migrate data from one version to the next on a timeframe that meets your needs. You can have both old and new version data records in the same system simultaneously. You create multiple versions by editing the current version and saving it as a new version number.

When editing a data model, only one person should work with a component at one time. If two different people are editing a data model at the same time on two different machines, both changes may not be recorded accurately, or one person's edits may overwrite the other person's data model version.

Edit an existing data model

1. Open the Data Model Editor.
2. In the Getting Started dialog box, click Open an Existing Data Model.
3. In the Open Data Model dialog box, select the type of data model you need to work with from the Filter drop-down menu:
   * Common: A pre-defined data model that is copied into your environment as a private data model for your use
   * Private: A private data model is only resident in your system
4. Expand a data model Name to see a list of available data model Versions.
5. Select a Name or a Version and click Open.

When you select a data model Name, DigitalEdge automatically opens the latest Version of that data model.

Data Model Editor displays an informational message, Data Model in Use, in the upper right corner when the model you are working with is used in a system.

1. The Input Model screen appears. From here, you can:
   * Edit any input model field; See " Step 1: Define an Input Model " on page 54
   * for details on adding new fields and editing existing fields Change the data source mappings; See " Step 2: Define Data Sources, Parsers, and Data Source Mappings " on page 65
   * for more details Edit or add enrichments; See "Step 4: Define Enrichment Processes " on page 99

for information about adding enrichments and their parameters

1. Save the edited data model as a new version number; See " Step 5: Save a Data Model" on page 112 for details.

If you edit a data model that is being used in a running system, you must go to the Management Console and Update the system that is using the data model. 

Delete a data model

When you are done with an older data model version and no longer wish to keep it in the system, you can Delete a data model version.

You can only delete a data model when that system is not running. Data Model Editor displays Data Model in Use in the upper right corner when the model you are working with is used in a system. Hover over the message to see a list of systems and statuses. Use the Management Console to stop a running system.

When you delete a data model, any records that were created under that data model and version are not deleted. Data records are only deleted when you delete their parent system in the Management Console.

1. Open the Data Model Editor.
2. In the Getting Started dialog box, click Open an Existing Data Model.
3. In the Open Data Model dialog box, select the type of data model you need to work with from the Filter drop-down menu:
   * Common: A pre-defined data model that is copied into your environment as a private data model for your use
   * Private: A private data model is only resident in your system
4. Expand the data model Name to see a list of available data model Versions.
5. Select a Version and click Delete. 
6. Click OK to confirm deletion of the selected data model and version, or click Cancel to keep that version.

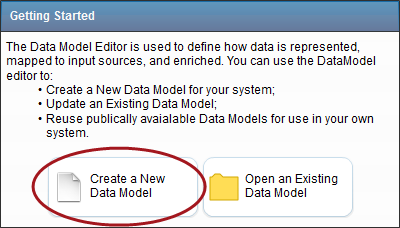
Create a Data Model with the Wizard

There is an alternative to defining a full input model, without going through all the complex steps of Table Manager and the Data Model Editor. Use the Data Model Creation Wizard  when you have a flat CSV file or JSON data source and you want to quickly build a simple data model. The capabilities of the Data Model Creation Wizard include:

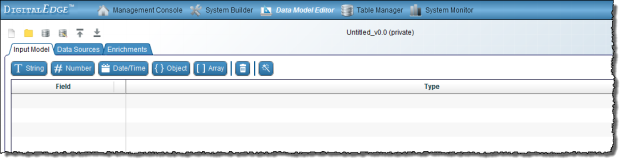
* The Wizard builds a new data model from scratch; it does not modify existing data models.
* Reads a CSV file and maps its columns directly into DigitalEdge fields with the same names, one field for each column.
* Reads a JSON file and map its fields into DigitalEdge fields with the same names.
* When reading a CSV or JSON file and mapping a column to a DigitalEdge field, the Wizard detects date/time, numeric, and string data types. If it cannot identify numeric or date/time data, it defaults a field to the string data type.
* Builds a flat file; the Wizard does not build a data model with nesting, objects, or arrays. If it encounters nesting in a JSON file, the Wizard will return an error and ask you to edit the file.
* Removes invalid data (for example, the Wizard will remove shift-characters, or the pound sign # as an initial character).
* The Wizard can readily handle CSV file sizes up to 100 MB. If your file is over 100 MB and under 300 MB, the Wizard will warn you that you must have enough memory on your PC to continue. If your file is over 300 MB, you should break it into several smaller files for processing.
* The Wizard can handle JSON files up to 50 MB in size.
* The Wizard does not create data enrichments. Once you create a simple data model, you can modify the data model with your own enrichments.
* The Wizard does not ingest and process data. It uses the data file to construct an input model and to create data source mappings.

Use the Data Model Creation Wizard

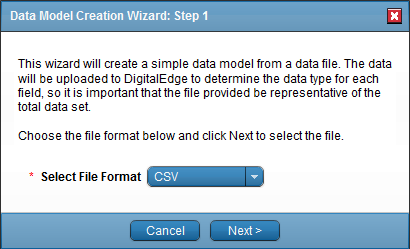
1. Access the Data Model Editor.
2. In the Getting Started dialog box, click Create a New Data Model.



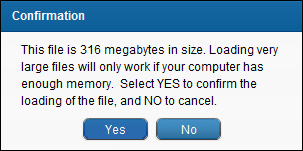
The Input Model workspace appears.



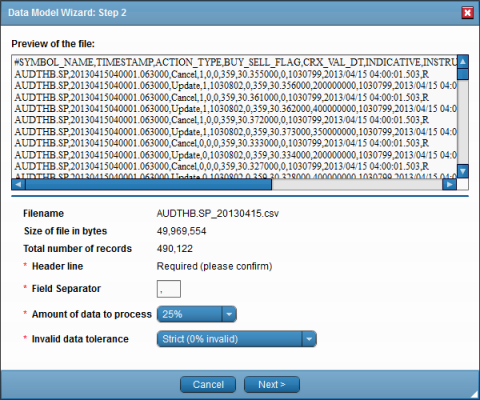
1. Click the Data Model Creation Wizard tool   . The Data Model Creation Wizard: Step 1 dialog box appears.



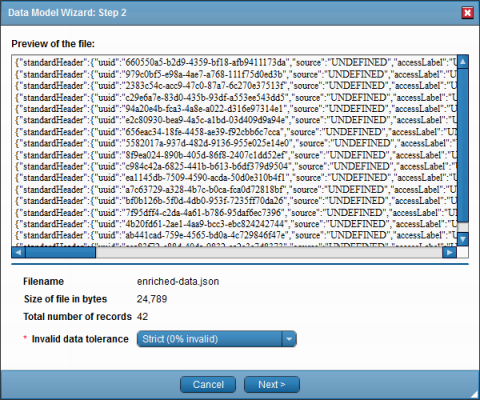
1. Choose a data source file format (e.g., CSV, JSON) and click Next. A file Open windows appears.
2. Browse to select an input file, and click Open. JSON files must be less than 50 MB. A CSV file can be larger, but you must have enough memory to process the file. The Wizard will warn you if a CSV file size is over 50 MB.



The Data Model Creation Wizard: Step 2 dialog box appears. For CSV files:



For JSON files:



1. DigitalEdge provides a Preview of the file, listing the column names and the first several lines of the file. Review and/or configure the data model creation specs before starting the job:

* Filename: The name of the file you are using to create a data model (information only)
* Size of file in bytes: File size (information only)
* Total number of records: The number of records that the Wizard detected in the file (information only)
* Header line: For CSV files, the presence or absence of a header. The Wizard assumes that a CSV file has one header line. You can confirm the presence of a header line by eyeballing the Preview of the file data.
* Field Separator:For CSV files, input the character used as a separator in the file. The Wizard will tell you if the character you input cannot be used as a separator.
* Amount of data to process: For CSV files, choose to input the entire file or just a portion of it. Note that the Wizard only creates the input model once; you cannot rerun the Wizard with more lines of data later to refine an existing model. You may want to select a partial file for a test run that will be deleted later, or you may want to use a subset of file lines to save time and resources. JSON files are processed in their entirety.
* Invalid data tolerance: The Wizard examines the data in each data source column/field (sampling all Number of lines to process) and attempts to determine what data type it is. The Wizard supports the DigitalEdge data types of Date/Time, Number, and String data types (not Object or Array). The Wizard first tests for numeric data. If a field doesn't meet the requirements as a numeric field, the Wizard next tests the data as date/time data. If the data does not match a date/time format either, the field is assumed to be string data.

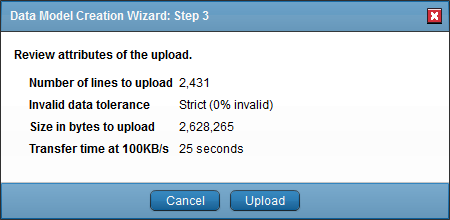
Of course, some files may include imperfect data. In that case, The Wizard will "tolerate" a certain amount of imperfect data when determining the data type. For example, if the Wizard determines that 97.5% or more (but less that 100%) of the data in a column called home\_score is numeric (several records include the word zero in that field), the Wizard will still identify the field as a numeric field. You select an Invalid data tolerance percentage for the Wizard to work with:

* + Strict (0% invalid): 100% of the data must match one data type
  + Moderate (up to 2.5% invalid): 97.5% - 99.9% of the data must match one data type
  + Relaxed (up to 5% invalid): 95% - 97.4% of the data must match one data type

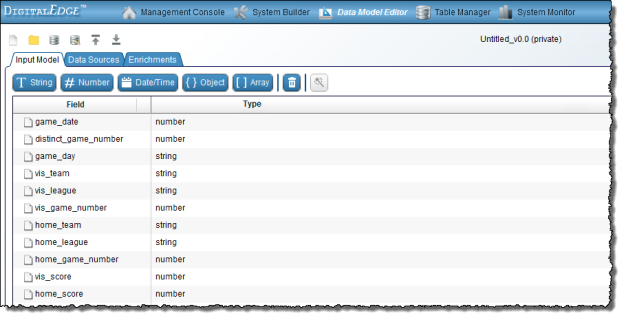
If less than 95% of the data in one field/column is the same data type (numeric or date/time), the Wizard will identify that field/column as string data. In other words, if the Wizard cannot determine if a field is numeric or date/time data, it assumes the field is string data.

Later, when you build and run a system to ingest the data, any record that includes a field that does not match its data type will be placed in the Dead Letter Queue and reported in the ingest log file for editing.

1. Click Next. The Data Model Creation Wizard: Step 3 dialog box summarizes your choices and estimates the build time.



1. Click Upload. The Wizard reports progress as it analyzes the data. When the model is created, the new DigitalEdge Input Model is displayed. You can then edit the model, add enrichments to it, build a system, and run the system to ingest the data.



1. Check the Data Sources tab. The Wizard created a data source called generated, and mapped the source fields to input model fields.



1. Click the Save icon on the Data Model Editor title bar.   

Upload and Download Data Models Between Systems

If you created a data model in another DigitalEdge system that you would like to use here, you can upload a data model with the Data Model Editor. For example, suppose you created a trial system in the Amazon public cloud, tested DigitalEdge there and tried out a data model, but then implemented your production system on a private Eucalyptus platform. In that case, you can download a data model configured in Amazon to your hard drive and then upload it to the DigitalEdge system resident on a Eucalyptus private cloud. The Upload/Download functionality in the Data Model Editor saves you time by providing a quick and easy way of transporting a data model between systems.

Download (export) a data model

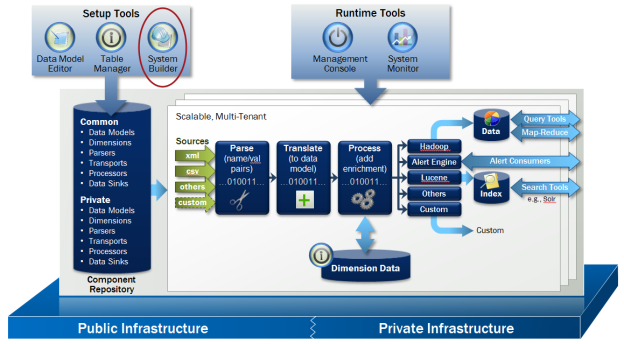
1. In the Data Model Editor, open the data model you wish to export and click Download. 
2. Browse to a target folder and click Save. The data model is saved in a ZIP file format.

Upload (import) a data model

1. In the Data Model Editor, click Upload. 
2. Browse to a source folder, select the data model you wish to import, and click Open. DigitalEdgeimports the model into Data Model Editor for you to work with.
3. You must Save  or Save As  the data model in the Data Model Editor to permanently place it in your system as a private data model.

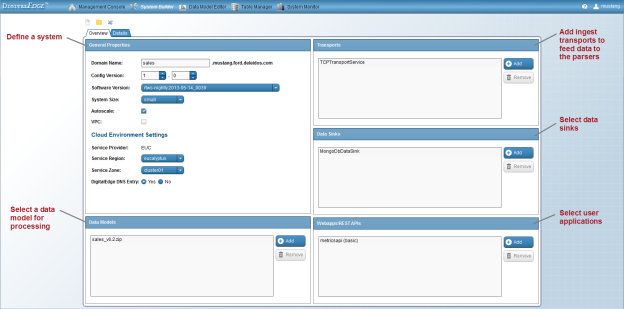
# Appendix F: DigitalEdge™ Configuration Guide, Chapter 5 – *Building the System*

Once you have defined all the required DigitalEdge components, you are ready to assemble them into a processing pipeline and to specify system level parameters.



The System Builder is the last Setup Tool you will use during configuration. It helps you:

* Define a system
* Assemble processing pipelines and build a system
* Connect transports and data sinks to pipelines
* Select web user applications for features such as search and metrics



You will also revisit System Builder for ongoing system maintenance to:

* Edit system parameters
* Implement a newer version of a data model
* Do advanced configuration tasks such as resource allocation
* Specify additional user applications

When you log into the System Builder, you will only be working with private systems that are proprietary to your organization.

Step 1: Define a System

You can quickly define and build a new system with the Overview tab in System Builder. The Overview tab assists you in naming and parameterizing the new system, and assembling the components for a processing pipeline.

Define a new system

The minimum requirements for building a new system include:

* Specifying system General Properties in the System Builder tool
* Choosing one data model
* Selecting one data sink

1. Access the System Builder tool  from the Management Console Tools list: https://<your-domain-name>/tenantconsole
2. From the Getting Started dialog box, either Create a new system from scratch, or Open an existing system to work with.
   1. To open an existing system, select the domain name of your system (Select system domain to open) and click OK.
   2. Choose the configuration version number of the system you want to work with (Select configuration version) and click OK.
3. Select the Overview tab. Use this screen to build a basic system. Use the Details section to fine-tune a system.
4. Enter a Domain Name that is unique.

The system domain name has a maximum length of 255 characters. If the name you enter here is too long, the text box will turn red, and you will receive an error message when you Build the system.

You can have just one configured system setup for each domain name you create.

1. Select a Config Version for your domain. Use this field to keep track of your site builds/configurations.
2. Select a DigitalEdge Software Version. You will typically use the latest official DigitalEdge release from Leidos.
3. Select a System Size, which controls the capacity of JMS messaging in the system (the number and type of instances for JMS). Size is defined by Amazon’s and Eucalyptus’ JMS node instance types:

* XSmall: 1 small JMS node (combines the JMS external and internal nodes on one instance)
* Small: 1 large JMS node (combines the JMS external and internal nodes on one instance)
* Medium: 1 large JMS external node and 1 large JMS internal node
* Large: 3 large external JMS nodes and 3 large internal JMS nodes
* XLarge: 3 extra-large external JMS instance nodes and 3 extra-large internal JMS instance nodes
* XXLarge: 6 extra-large external JMS instance nodes and 6 extra-large internal JMS instance nodes

An external JMS node feeds data directly into DigitalEdge. Clients push data from data feeds into a JMS external queue, which is the first entry point into DigitalEdge. The parsing queue is also resident on a JMS external node.

Internal JMS nodes handle records through the processing pipeline, including ingest, indexing, and filtering for alerts. Queues that are on internal nodes include the index queue and the filter queue to handle record processing after enrichment.

Once you build a system, you can always resize it to a larger size, but you cannot downsize a system once it is configured.

Leidos can provide guidance on sizing a new system.

1. Click Autoscale to turn auto-scaling on for every processor group that is eligible for auto-scaling (i.e., all data sinks and ingest.all). Leave Autoscale unchecked to help you control costs in the public cloud, or to accommodate limited resources in the private cloud.

If a data sink does not support auto-scaling, you will get a warning and DigitalEdge will only turn on auto-scaling for supported components.

1. Click VPC to configure a DigitalEdge system in Amazon's VPC™ (Virtual Private Cloud). VPC™ isolates a section of AWS as a virtual network, providing greater security and control similar to that of private cloud environments. Each system is reserved and assigned a private subnet. Note that VPC is not fully configured until you click Build . See "About VPC" on page 126 for more information.
2. Next, define your cloud-specific parameters. A Service Provider will be entered automatically, based on your account type (AWS or Eucalyptus).
3. Select a Service Region (data center locale) from your service provider’s predefined types (e.g, AWS’s us-west-1). Each system uses just one region.
4. Select a Service Zone from the availability zones that are predefined for your service region (e.g., us-west-1a).

You cannot change the availability zone once you Build the system.

1. On a Eucalyptus system, click Yes for DigitalEdge DNS Entry if you created an external DNS forwarder for DigitalEdge. If you did not create a DNS forwarder, click No.

If you do not have an external DNS forwarder, you must assign a persistent IP address to the webapps.main node. After you select your webapps, go to the Details tab, and double-click the Public IP selection of the webapps.main process group to assign the elastic IP.

1. In the Data Models section, you must select at least one data model to process. When you specify a data model, you are also telling System Builder which parsers and enrichments will be used.
   1. Click Add. A list of private data models that you created with the Data Model Editor appears.
   2. Click on a data model Name/Version and click OK. The data model is added to your system.

If you do not expand the data model Name and select a specific Version, when you double-click on the data model Name, DigitalEdge selects the latest Version.

You can add more than one data model to the system.

You can jump to the Data Model Editor to create a new data model by clicking New.

1. In the Data Sinks section, you must choose at least one data sink for a minimal DigitalEdge system. See " Step 3: Add Data Sinks" on page 141 for details.
2. Click Build  to save the system’s configuration.

You now have a basic system defined. You could Run the new system now (e.g. Start the system in the Management Console), or you can assemble additional components (transports, data sinks, and web applications) with the following steps.

About VPC

When you sign up for an Amazon AWS account, your account is automatically created in the VPC (virtual private cloud) environment as an Amazon EC2-VPC account for a more secure system. Amazon VPC accounts provide security features such as security groups, network access control lists, filtering traffic at the instance and subnet levels, and restricting access to stored data in S3.

If you created an AWS account prior to 2014, those EC2-Classic accounts were created as non-VPC accounts by Amazon. EC2-Classic accounts include the option to create a VPC system if it is needed, by configuring the VPC with the DigitalEdge System Builder tool. See the DigitalEdge Configuration Guide for details.

Facts about VPC and DigitalEdge:

* To help protect sensitive data, a VPC system is isolated from your EC2 environment by a NAT (Network Address Translation) in the public subnet in each Availability Zone, which serves as a firewall into the private VPC subnet. Amazon also creates an Internet gateway and connects it to your VPC. The CIDR block for your VPC is 172.31.0.0/16. See Amazon's [VPC documentation](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html) for more detailed specifications.
* A DigitalEdge system built in the VPC will run securely in the private subnet. The NAT will allow outbound traffic to access an external instance and will block inbound traffic from the VPC system.
* VPC only allows access via the following designated ports:
  + 443: routed to webapps.main
  + 8443, 5555, 1098, and 61515: routed to the Master node
* VPC uses its own security groups (identified with a vpc preface, such as vpc.internal.default, vpc.webapp.default, etc.)
* VPC has the potential of hosting 10 DigitalEdge systems, each with 1000 IPs
* NAT uses a Route Table to access the Internet; Amazon allows 10 entries in the Route Table
* Amazon provides 5 elastic IPs to start a VPC; you can request more elastic IPs from Amazon (use this [request form](http://aws.amazon.com/contact-us/vpc-request/)) to increase the VPC's capacity to host 10 systems
* Each NAT is assigned an elastic IP which is publicly addressable, but DigitalEdge binds a system domain name to the elastic IP, allowing your users to access DigitalEdge by name
* You do not have to assign IP addresses in the VPC; IPs are assigned automatically by Amazon
* You will have one DigitalEdge Gateway for your Amazon account.

Step 2: Add Transports

To get data into the system, you must specify front end transport(s) to feed data into the parsers.

Ingest supports several secure automated transport mechanisms, including file-based transfer protocols, streaming TCP and UDP connections, external database queries, and unstructured documents. Each parser should be assigned to a transport type. The JMS Bridge Service pushes data directly onto a JMS queue without a transport.  In this case, there would be two JMS servers in the system: a corporate JMS server and the DigitalEdge JMS server; the bridge pushes data from your enterprise queue to DigitalEdge. If you do not use JMS, you should choose a transport from the DigitalEdge repository.

If you need a transport type that is not available on the DigitalEdge menu, a custom transport can be built. This is an advanced configuration step which requires the use of the DigitalEdge SDK and Leidos’ assistance.

Use a pre-existing transport

1. In the System Builder, click the Overview tab. Use the Transports section.
2. Click Add. The Select a Transport dialog box appears. Click on a transport name. (Note that this list includes the transports that are included in the core release. You may also have custom transports created for your site.)
   * DatabaseWatcherTransportService: gets data from a database and pulls it into DigitalEdge by running an SQL select query against any database
   * DirectoryCrawlerTransportService: a beta transport that crawls and processes data in a local or remote file system while decompressing zipped files and processing files that match wild card patterns
   * DirectoryWatcherTransportService: periodically checks a local or remote file system for files ready to transport
   * HiveTransportService: copies data from an existing Hive data sink and pulls it into another data sink
   * JMSBridgeTransportService: copies data directly from an external JMS messaging system to the DigitalEdge JMS server
   * MongoDBTransportService: copies data from an existing MongoDB data sink and pulls it into another data sink
   * PcapSnifferTransportService: captures and splits pcap packets on a specific network interface
   * S3FileTransportService: pulls data from an Amazon S3™ (Simple Storage Service™) bucket and optionally deletes the files after processing; can be configured to read the bucket once, or poll regularly for files
   * TCPTransportService: reads data in a TCP stream
   * TwitterFilterTransportService: gets Tweets from the Twitter feed based on criteria that you define via the Twitter Search API
   * TwitterRESTTransportService: follows the Tweets of one Twitter user, using the Twitter REST API
   * TwitterSampleTransportService: selects a random sample of Tweets that you are allowed to read in the Twitter feed, using the Twitter Search API
   * UDPTransportService: captures packets in a UDP stream
   * URLTransportService: reads the contents of a URL and puts it on the JMS input queue (typically for RSS feeds)
3. Click OK. The Set Transport Parameters dialog box appears. Check the transport parameter values and edit them if necessary. See "Transport parameters" on page 128 for details. Click OK when you are done.
4. Click Build  if you are done with your System Builder work.

To edit transport parameters later, double-click the transport name on the Overview tab to access the Edit Transport Parameters dialog box. Then, double-click a Parameter's Current Value to edit it. Or, on the Details tab, select a transport from the drop-down menu in the Transport Parameters section and double-click a Current Value to edit.

You can add multiple transports, and multiple transports of the same type.

Transport parameters

Each pre-existing transport includes a set of parameters to control its operation. The list of parameters may vary depending on the parser you chose to work with.

You can access transport parameters two ways:

* To add a new transport and its parameters: System Builder > Overview tab > Transports section > Add > Select a transport > Double-click a Parameter's Current Value to change it
* To edit transport parameters, double-click the transport name on the System Builder > Overview tab to access the Edit Transport Parameters dialog box. Double-click a Parameter's Current Value to edit it.

If you edit parameters for a transport that is being used in a running system, you must go to the Management Console and Update the system version that is using the transport. 

Here are detailed lists of transport parameters, descriptions, and values for the transports included in the core release. You can also hover over a parameter name in DigitalEdge for tool-tip help.

Database watcher transport

The Database Watcher transport is a specialized polling service that gets all the data from a database and pulls it into DigitalEdge by periodically running an SQL select query against a database. The database can be queried regularly, starting at the point where the query last left off. So, when the query is run again, only records not selected in the previous query will be retrieved. An S3 bucket is used to store a backup copy of the data file. To use this transport, you must specify several parameters that define the SQL query, the column that serves as the key/identity column, the stopping point, and how often the query should be run.

| DatabaseWatcherTransportService | |
| --- | --- |
| Parameter | Explanation |
| bucket-name | The name of the Amazon S3™ bucket to store a backup copy of the incoming data file. The name must match exactly the name as it is listed in the AWS™ Management Console bucket list |
| file-key | This parameter is the name of a file that will be placed in the S3 bucket (specified in the bucket-name parameter). This file stores the highest memory-key-column value retrieved in the latest query run (as a backup). By storing the highest value previously read, the transport assures that when the query is run again, only records not selected in the previous query will be retrieved. |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| memory-key-column | The memory-key-column parameter specifies which column number to read in the SQL query to identify new records that were ingested since the last read. The column is 1-indexed. The memory-key-column acts as a surrogate key to indicate how far the query gets into the database on each run.  Default = 1 |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| select-data-statement | The SQL query that is run against the database |
| sleep-time | The amount of time, in milliseconds, between polling the S3 bucket for data.  Default = 0 |

Directory watcher transport

The Directory Watcher transport is a polling service similar to a polling S3 File Transport, except that the transport is watching a file system, not an Amazon S3™ bucket. This transport is typically used when your data files are local and you do not want to move them to S3™. But the transport can also watch a directory on a remote server. To use this transport, you must specify the directory path and the polling time interval. The transport watches one directory folder; it is not recursive.

The Directory Watcher Transport does not unzip and process zipped files. You should extract and unzip these files if you want this transport to ingest that data content.

| DirectoryWatcherTransportService | |
| --- | --- |
| Parameter | Explanation |
| check-interval | How often the transport looks at the directory for new data, in milliseconds  Default = 500 |
| content-encoding | Indicates if the transport will encode received bytes before submitting them for processing   * None = no encoding will be performed * Base16 = hexadecimal encoding * Base64   Default = None |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| preserve-file-name | Indicates (true or false) if the filename is included in the message sent to the ingest pipeline. Use this parameter in conjunction with the Unstructured File Parser, when you include the data source's filename in the data model's output.  Default = false  Note: This parameter only works when content-encoding is set to Base16 or Base64. |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| remote-server | If you are reading data from a remote server (not a local system), this parameter specifies the IP address or domain name of the remote NFS server. |
| remote-share-name | If you are reading data from a remote server (not a local system), this parameter specifies the name of the shared directory on the remote-server from which files are read and fed into the transport. DigitalEdge uses the watched-directory parameter as the local mount point to this shared directory. |
| watched-directory | The local directory path to watch, expressed in Linux style notation, not Windows notation. If you have also defined the remove-server and remote-share-name parameters, DigitalEdge will create this directory and use it as the local mount point.  Note: This value must be unique if you have multiple transports using a remote-server option in your DigitalEdge system. |

Hive transport

The Hive transport is a specialized transport that gets data from an existing Hive data sink and pulls it into another data sink, either in the same DigitalEdge system or another DigitalEdge system. For example, you might store enriched data in Hive and then transport it to a Lucene data sink in the same system for indexing. Or, you might store enriched data in Hive in one system and then transport it to another DigitalEdge system for iterative enrichment. Also, you would usually create an SQL select query to run against the Hive data sink to pull out a subset of data rather than copying all the data.

| HiveTransportService | |
| --- | --- |
| Parameter | Explanation |
| hive-host | The server where the Hive data sink resides, identified as an IP address or DNS name  Default = localhost |
| hive-port | The port number to connect to the hive-host  Default = 10000 |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| poll-interval | How often the transport looks at the data sink for new data, in milliseconds  Default = 60000 |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| select-data-statement | The SQL select query to run against the Hive database. If blank, the transport will take all data, which may cause performance issues. |
| state-store-mode | Indicates if the position state should be stored locally on the instance; used when the transport goes down, to restart where it left off without duplicating any records  Default = true |

JMS bridge transport

The JMS Bridge Transport copies data directly from an external JMS messaging system to the DigitalEdge JMS server. This is a one-to-one transport, from one JMS queue to another.

| JMSBridgeTransportService | |
| --- | --- |
| Parameter | Explanation |
| incoming-address | The address (hostname and port) of the corporate JMS queue (in JMS URL format) from which messages will be fetched and transported. For example:  ssl://jms-node1.dm-test-myorg.com:61616  where:   * ssl = the transport type * the address name is a single connection to one node, or a failover address * port number = the provider of the messaging service for the queue   Default = localhost:61616 |
| incoming-user | The username needed to connect to the corporate server (if necessary) |
| incoming-password | The password credentials required to connect to the corporate server (if necessary) |
| incoming-queue | The name of the corporate queue to connect to. For example: com.myorg.data.parse |
| incoming-topic | The name of the corporate topic to pull messages for transport |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| sleep-time | The amount of time, in milliseconds, between polling of the data.  Default = 1000 |

MongoDB transport

The MongoDB transport is a specialized transport that gets data from an existing MongoDB data sink and pulls it into another data sink, either in the same DigitalEdge system or another DigitalEdge system. For example, you might store enriched data in MongoDB and then transport it to a Lucene data sink for indexing. Or, you might store enriched data in a MongoDB data sink in one system and then transport it to another data sink in a second DigitalEdge system for iterative enrichment. Also, you would usually create an SQL select query to run against the MongoDB data sink to pull out a subset of data rather than copying all the data.

| MongoDBTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| mongo-collection | The name of the MongoDB table to pull data from |
| mongo-db | The name of the MongoDB database to pull data from  Default = mydb2 |
| mongo-host | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a data model from the drop-down menu |
| mongo-port | The port number to connect to the mongo-host  Default = 27017 |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| select-data-statement | The SQL select statement to run against the MongoDB database. If blank, the transport will take all data, which may cause performance issues. |
| state-store-mode | Indicates if the position state should be stored locally on the instance; used when the transport goes down, to restart where it left off without duplicating any records  Default = true |

Pcap sniffer transport

The pcap sniffer transport captures and splits pcap packets on a network interface that you specify. You can also optionally filter out data from the transport. The pcap transport is often used with the DNS PCAP and SNMP PCAP parsers.

| PCapSnifferTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| interface-name | Identifies the type of network interface that the transport sniffs for pcap packets; for example, use eth0 for an Ethernet connection.  Default = eth0 |
| pcap-filter | Filters the pcap captures to include or exclude different types of data; for example, you can capture data coming from a specific IP address. See <http://linux.die.net/man/7/pcap-filter> for a list of filters and the syntax. |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |

S3 file transport

The S3 File Transport checks an Amazon S3™ bucket for data ready to transmit to the JMS external queue, and optionally deletes files after processing them. You can configure this transport to poll an S3™ bucket regularly, or to read S3™ just once as soon as the file appears in the bucket. When configured to poll the S3™ bucket periodically, the transport may locate multiple files over time with the same name. To use this transport, you must specify the S3™ bucket, file parser type that DigitalEdge will be using for data input, and several other parameters.

| S3FileTransportService | |
| --- | --- |
| Parameter | Explanation |
| bucket-name | The name of the Amazon S3™ bucket to check for input data files. The name must match exactly the name as it is listed in the AWS™ Management Console bucket list.  For example: sales.test.data. |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| polling-interval | How often the transport looks at the S3™ bucket for new data, in milliseconds. Use 0 to indicate that the bucket should be polled just once for data.  Default = 0 |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| should\_delete\_source | Once a file is read and processed, you can optionally delete the file; double-click to select Yes or No.  Default = No |

TCP transport

The TCP transport reads data from an entire TCP stream. To use this transport, you must specify the listening port and the file parser format that DigitalEdge will be using.

| TCPTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| listen-port | The number of the port to connect to for receiving TCP messages  Default = 0 |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |

Twitter filter transport

This transport gets Tweets from the Twitter feed based on criteria that you define using the Twitter Search API (see <https://dev.twitter.com/docs> for Twitter API documentation). You can search for keywords and/or Twitter usernames. This is the most flexible and commonly used transport of the three Twitter transports. You must have a Twitter account to use this transport.

| TwitterFilterTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| o-auth-access-token | Your OAuth access token; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about obtaining a token |
| o-auth-access-token-secret | Your OAuth access token secret; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about token credentials |
| o-auth-consumer-key | Your OAuth consumer key; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about obtaining credentials |
| o-auth-consumer-secret | Your OAuth consumer secret; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about obtaining credentials |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| terms | A comma-separated list of OR'ed keywords you want to search for in Tweets; maximum allowed = 400 words  Hits are AND'ed with the usernames |
| usernames | A comma-separated list of OR'ed Twitter names you want to follow; maximum allowed = 5000  Hits are AND'ed with the terms |

Twitter REST transport

You must have a Twitter account to use this transport, which follows the Tweets of one Twitter user. The transport uses the Twitter REST API (see <https://dev.twitter.com/docs/api> for Twitter REST API documentation).

| TwitterRESTTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| username | The name of a Twitter user you want to follow |

Twitter sample transport

This simple Twitter transport selects a random sample of Tweets that you are allowed to read in the Twitter feed. You must have a Twitter account to use this transport. The transport uses the Twitter Search API and the OAuth authentication protocol. See <https://dev.twitter.com/docs> for Twitter API documentation.

| TwitterSampleTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| o-auth-access-token | Your OAuth access token; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about obtaining a token |
| o-auth-access-token-secret | Your OAuth access token secret; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about token credentials |
| o-auth-consumer-key | Your OAuth consumer key; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about obtaining credentials |
| o-auth-consumer-secret | Your OAuth consumer secret; see the [Twitter API documentation](https://dev.twitter.com/docs) for information about obtaining credentials |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |

UDP transport

The UDP transport captures datagram packets in a UDP stream sent to a configured port. To use this transport, you must specify the maximum packet size, the listening port, and the file parser format that DigitalEdge will be using.

| UDPTransportService | |
| --- | --- |
| Parameter | Explanation |
| content-encoding | Indicates if the transport will encode received bytes before submitting them for processing   * None = no encoding will be performed * Base16 = hexadecimal encoding * Base64   Default = None |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| listen-port | The number of the port to connect to for receiving UDP packets  Default = 0 |
| max-packet-size | Specifies the maximum packet size, to ensure that the buffer is large enough to hold complete messages  Default = 65535 |
| message-processor-class | Name of the Java class that will process the packets  Default = com.deleidos.rtws.transport.Services.utils. SimpleUDPMessageProcess  Recommendation: Keep the default value |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |

URL transport

The URL transport reads the contents of a URL just once and puts it on the JMS input queue. This transport is typically used for pulling data from an RSS feed or from any service that pulls resources from another organization or data source into your system. To use this transport, you must specify the URL and the file parser format that DigitalEdge will be using.

| URLTransportService | |
| --- | --- |
| Parameter | Explanation |
| input-format | Identifies the data source and parser format that the transport uses to pull data off the input queue; double-click to select a Data Source from your specified data models |
| read-address | The URL to read incoming data from |
| record-format | A data record type (BASE64, CONTENT, PCAP, JSON, TEXTLINE, NULL, TEXTLINEWITHQUOTES) that helps to determine record boundaries when input data includes multiple records; double-click to select a format from the drop-down menu  Default = NULL |
| record-header-lines | How many header lines should be stripped out of the data records  Default = 0 |
| user-id | The username needed to validate for https (if necessary) |
| user-password | The password needed to validate for https (if necessary) |

Step 3: Add Data Sinks

When data is processed in a DigitalEdge pipeline, the results are stored in data sinks for use downstream. Data sinks can be used for:

* Back-end data storage in a NoSQL database
* Indexing
* Alerting

You must include at least one data sink in a DigitalEdge system.

DigitalEdge provides several data sink options. Each data sink has its strengths and weaknesses. DigitalEdge works with several repositories to play to the strengths of each one and to optimize performance. For example, Lucene is particularly good for implementing real time searching on the data.

Add a data sink

1. In the System Builder, click the Overview tab. Use the Data Sinks section.
2. Click Add. The Select a Data Sink dialog box appears. Click on an available data sink name. (Note that this list includes the data sinks that are included in the core release. You may also have custom data sinks created for your site.)
   * AlertingDataSink: to filter processed records for alert triggers and send out real-time alert messages either as email messages or as messages in a JMS topic
   * Cassandra data sink: a beta data sink to store data in an Apache Cassandra cluster
   * Dimension data sink: This data sink stores data for dimensional enrichments in the tenant database. It serves two purposes:
     + To provide an alternative method of uploading enrichment data into a dimension table, instead of using the Table Manager>Batch Tools>Import functionality
     + To store records which were enhanced by a dimension table enrichment and which themselves will be used to enrich another data source (used when you have two data sources, one of which will enhance the other data source).

This data sink requires two special fields in the input model to associate the dimension table and keys with the data model; for details, See " Step 1: Define an Input Model " on page 54.

* + ElasticSearchDataSink: a beta data sink to implement Elasticsearch, an indexing and full text search engine
  + ExternalHBaseDataSink: to store data in a Hadoop/HBase cluster that is not managed by DigitalEdge
  + ExternalHDFSDataSink: to store data in the Hadoop environment that is not managed by DigitalEdge (compatible with Cloudera CDH3uX releases)
  + ExternalHiveDataSink: to store data in the Hadoop/Hive environment that is not managed by DigitalEdge
  + HbaseDataSink: to store data in an HBase database
  + HiveDataSink: to store data in the Hadoop/Hive environment and is managed by DigitalEdge
  + JsonToJdbcDataSink: to map JSON objects to a relational database table and write them to a database via JDBC
  + LuceneIndexingDataSink: to index data for real-time and near real-time search
  + MongoDBDataSink: to store data in a MongoDB® database; Mongo is not currently auto-scaling. You can configure MongoDB once per system. Note that the maximum record size is 16 MB in MongoDB.
  + SleepDataSink: to continually read a record, ignore it, and then sleep for a specified amount of time; this data sink is used in a system that is only checking records for alert criteria and not processing or storing the data for other uses or for searching. The alert engine processes records before they arrive at the sleep data sink.

1. Click OK. The Set Datasink Parameters dialog box appears. Review the default values and edit any parameter as needed. See "Data sink parameters" on page 143 for details.
2. You can associate one or more data models with a data sink to conserve resources. When multiple data streams are sent through the system, only the data sink that is configured to receive data from one data model will store the data; data is not stored in multiple data sinks that are configured for other uses. Click the Data Models tab. Highlight the relevant data model(s) to use with this data sink.
3. Click OK to save all parameter values.
4. Click Build  if you are done with your System Builder work.

To edit data sink parameters later, go to: System Builder > Details tab > Data Sink Parameters section > Select a data sink from the drop-down menu > Double-click a Parameter's Current Value to edit it. Or click the Data Models tab to associate a data model with this data sink.

Data sink parameters

Each data sink includes a set of parameters to control its operation. The list of parameters varies depending on the data sink you chose to work with.

DigitalEdge validation checks parameters for structure and syntax, but no communication checks are performed with these parameters.

You can access data sink parameters two ways:

* To add a new data sink and its parameters: System Builder > Overview tab > Data Sinks section > Add > Select a data sink > Double-click a Parameter's Current Value to change it. Or click the Data Models tab to associate another data model with the data sink.
* To edit a previously configured data sink, double-click the data sink name on the System Builder > Overview tab to access the Edit Datasink Parameters dialog box. Double-click a Parameter's Current Value to edit it. Or click the Data Models tab to associate another data model with the data sink.

If you edit parameters for a data sink that is being used in a running system, you must go to the Management Console and Update the system version that is using that data sink. 

Here are detailed lists of parameters, descriptions, and valid values for the data sinks included in the core release. You can also hover over a parameter name in DigitalEdge for tool-tip help.

Alerting data sink

This data sink does not store DigitalEdge records or alert notifications; it filters processed records for alert triggers and send out alert messages. The alerting data sink specifies how alerts are issued: either as email messages or as messages in a JMS topic. The parameters specify the connection and capabilities of your email server.

| AlertingDataSink | |
| --- | --- |
| Parameter | Explanation |
| auth | Whether or not your SMTP email server requires authentication; if set to true, you must supply values for email-from and email-from-password  Default: false |
| email-from | If a username is required to connect to the SMTP email server, specify it here.  This username is also used as the "From" address in the alert mail messages; must have sending rights to this address |
| email-from-password | If a password is required to connect to the SMTP email server, specify it here |
| email-port | The port number that the SMTP email server is running on  Default: 25 |
| email-server | Name or IP address of the SMTP email server used to send email alert messages  Default: Amazon  If you use Amazon's SMTP server, you do not have to specify values for any of the other parameters |
| send-email | Whether or not alert email messages are sent out; if you select false, you should specify a topic notification instead of email. You can specify both send-email and topic parameters, or just one of the parameters, but you must specify at least one of them.  Default: false |
| tls | Whether or not encryption is required for email messages  Default: false |
| topic | If alerts will be generated as messages on the external JMS queue, you must specify a value for this topic parameter. This value should be a valid JMS topic name, such as com.org.rtws.alert. With a JMS topic, alert messages are posted on one message board for everyone to view.  This parameter can have no value, to signify no JMS alerting, or a non-blank value to turn on JMS alerting.  You can specify both send-email and topic parameters, or just one of the parameters, but you must specify at least one of them.  Default: null |

Cassandra data sink

Use this beta data sink to store data in an Apache Cassandra cluster. Cassandra is a high-performance open source distributed DBMS that is used to handle big data across many servers with no single point of failure. It is a column-oriented database like HBase. The Cassandra cluster is decentralized, with data distributed across the cluster, but without a master node so that each node can fulfill any request. It is very easy to add nodes to a Cassandra configuration. The Cassandra data sink is not auto-scaling.

| CassandraDataSink | |
| --- | --- |
| Parameter | Explanation |
| compression | Indicates how data should be compressed when stored in Cassandra   * None * LZ4Compressor * SnappyCompressor * DeflateCompressor   Default: None |
| keyspace | The container for your application data and the Cassandra replication and cluster configuration  Recommendation: Do not change the keyspace name once you have created it |
| replication-factor | Controls the number of data replicas to create  Default: 1  Recommendation: The default is set to 1 (which results in 2 copies of your data) to conserve storage resources. It is best to set the replication-factor in the range of 1-3. |
| table-name | The table to use for storing your data |

Dimension data sink

Use this data sink to store data for dimension table enrichments in the tenant database. It provides an alternative method of uploading enrichment data into a dimension table (instead of using the Table Manager>Batch Tools>Import functionality to upload CSV data into a table). Your table columns must first be defined in Table Manager and mapped to your input model before using this data sink to assure that the source data is processed correctly.

You can also use this data sink to store enriched records in the tenant database and use them later as a data enrichment source for a second data feed. In other words, when you have two data sources, one of which will enhance the other data source, configure a dimension data sink. This data sink requires two special fields (tableName and tableKey) in the input model to associate the dimension table and keys with the data model.

| DimensionDataSink | |
| --- | --- |
| Parameter | Explanation |
| updates-per-second | The rate at which data is sent into the dimension data sink. This parameter prevents the parser queue from filling up before the data can be processed into the data sink. To help determine an optimum value for this parameter, use System Monitor> Queue Size graph to check on the processing backlog and how many messages were fed in that are not yet processed. The parameter value represents the number of messages fed into the data sink per second.  Default: 500 |

Elasticsearch data sink

This beta data sink implements the open source Elasticsearch full text search engine. This data sink stores and indexes your data and provides real-time full text querying. Elasticsearch is auto-scaling, auto-connects to other DigitalEdge nodes for true distributed processing, and support multi-tenancy. Each node hosts multiple index shards, forming and managing clustered operations. With Elasticsearch, you can have massive amounts of streaming data ingested, indexed, and searchable in seconds.

To implement a graphical user interface for Elasticsearch, choose the Kibana webapp when building your DigitalEdge system.

| ElasticSearchDataSink | |
| --- | --- |
| Parameter | Explanation |
| dateindex | The directory path pointing to a datestamp field (date only, no time data) that is used for indexing |

External HBase data sink

Use this data sink to store data in an existing Hadoop/HBase cluster that is external to and not managed by DigitalEdge.

| ExternalHBaseDataSink | |
| --- | --- |
| Parameter | Explanation |
| record-write-retry-count | Reserved for future use  Default: 20 |
| record-write-retry-delay | Reserved for future use  Default: 10000 |
| row-key-resolver-name | The method used to determine the rowKey for written records. Change the default only if you need to compute a more meaningful key instead of the random UUID.  Default: Enriched\_Record\_Row\_Key\_Resolver (UUID) |
| table-name-resolver-name | The method used to determine the HBase table name for storing enriched records. The default value, Data\_Model\_Table\_Name\_Resolver, uses a name determined by your data model. If you select User\_Configured\_Table\_Name\_Resolver, enter a table name for enriched records in the user-specified-table-name parameter.  Default: Data\_Model\_Table\_Name\_Resolver |
| user-specified-table-name | The name of the table used for writing enriched records; required only if the table-name-resolver-name = User\_Configured\_Table\_Name\_Resolver |
| write-level | Specifies what parts of JSON records (i.e., the granularity) are writable to the HBase data store. HBase uses one table per data model for storage, and one column family for each parameterized value. For example, with the default of Metadata,Objects,Fields, JSON records are written to HBase three times: metadata to one column family, objects to a second column family, and fields to a third column family; one DigitalEdge record per HBase row. Decisions for write-level values are dependent upon the web and user apps that you specify. For example, committing Fields to HBase may require more processing and storage space, but provides applications with queryable fields and full context data. Metadata storage may help facilitate record matching for a Pentaho dashboard application.  Default: Metadata,Objects,Fields  Recommendation: Metadata,Objects |
| zookeeper-quorum | A comma delimited list of hostnames or IPs of the external ZooKeeper cluster used to communicate with the external HBase cluster. For example: zookeeper1,zookeeper2 |

External HDFS data sink

Use this data sink to store data in an existing Hadoop cluster that is external to DigitalEdge, is not managed by DigitalEdge, and is compatible with Cloudera CDH3uX releases. You must specify the communication connection to your organization's Hadoop cluster with these parameters. If a firewall is between DigitalEdge and the external Hadoop data sink, you must open ports for DigitalEdge access.

| ExternalHDFSDataSink | |
| --- | --- |
| Parameter | Explanation |
| block-size | Specifies the dfs.block.size for files written to HDFS. The default value is a medium size that can be used as a starting point.  Default: 134217728 |
| namenode-hostname | The IP address or valid, resolvable host name of the server that is running the NameNode daemon in your Hadoop cluster  Default: namenode.domain  Recommendation: keep the default value to use the internal Hadoop cluster |
| namenode-port | The port number that the NameNode daemon is running on in your Hadoop cluster  Default: 8020  Recommendation: keep the default value to use the internal Hadoop cluster |
| replication-factor | Controls the default replication factor on files written to HDFS by this data sink |
| target-folder | The directory where the data sink will write records prior to insertion into Hadoop; must be writable by the user running this data sink in DigitalEdge; can change the default root and/or folder name for your site configuration  Default: /tmp/external\_hdfs\_data\_sink |

External Hive data sink

Use this data sink to store data in an existing Hadoop/Hive environment that is external to DigitalEdge and is not managed by DigitalEdge. You must specify the communication connection to your organization's Hive cluster with these parameters. If a firewall is between DigitalEdge and the external Hive data sink, you must open ports for DigitalEdge access.

| ExternalHiveDataSink | |
| --- | --- |
| Parameter | Explanation |
| block-size | Specifies the dfs.block.size for files written to HDFS. The default value is a medium size that can be used as a starting point.  Default: 134217728 |
| compress-data | Indicates if data should be compressed before it is stored; only works if Snappy compression is configured on the external Hive cluster  Default: false |
| hdfs-working-folder | The directory where the data sink will write records prior to insertion into Hive; must be writable by the user running this data sink in DigitalEdge; can change the default root and/or folder name for your site configuration  Default: /tmp/hiveDataSink |
| hive-jdbc-hostname | The IP address or valid, resolvable host name for the server running the Hive Thrift service  Default: localhost  Recommendation: keep the default value to use the embedded server |
| hive-jdbc-port | The port number for the server that is running the Hive Thrift service  Default: 10000  Recommendation: keep the default value to use the embedded server |
| jobtracker-hostname | The IP address or valid, resolvable host name for the server running the Hadoop JobTracker daemon  Default: jobtracker.domain  Recommendation: keep the default value to use the internal Hadoop cluster |
| jobtracker-port | The port number that the Hadoop JobTracker is running on in your Hadoop cluster  Default: 8010  Recommendation: keep the default value to use the internal Hadoop cluster |
| metastore-url | The full JDBC URL to the Hive metastore RDBMS. Use an IP address unless the hostname is publicly accessible. Suggested format:  jdbc:HIVE\_METASTORE\_JDBC\_PARAMETERS//HIVE\_METASTORE\_IP:HIVE\_METASTORE\_PORT |
| namenode-hostname | The IP address or valid, resolvable host name of the server that is running the NameNode daemon in your Hadoop cluster  Default: namenode.domain  Recommendation: keep the default value to use the internal Hadoop cluster |
| namenode-port | The port number that the NameNode daemon is running on in your Hadoop cluster  Default: 8020  Recommendation: keep the default value to use the internal Hadoop cluster |
| password | The password that Hive uses to connect to the metastore |
| replication-factor | Controls the default replication factor on files written to the Hadoop Distributed File System by this data sink  Default: 2 |
| use-complex-schema | Enables complex table schema definitions (such as map <>, array <>, struct <>, etc.) derived from the canonical data model. This parameter requires the use of a supported SerDe (a Serializer/Deserializer that lets Hive read in data from a table and write it out to HDFS); true or false  Default: false |
| username | The username that Hive uses to connect to the metastore |

HBase data sink

Use this data sink to store DigitalEdge data in an HBase database that is managed internally by DigitalEdge. Configure the parameters to define how records are stored in HBase.

| HBaseDataSink | |
| --- | --- |
| Parameter | Explanation |
| record-write-retry-count | Reserved for future use  Default: 20 |
| record-write-retry-delay | Reserved for future use  Default: 10000 |
| row-key-resolver-name | The method used to determine the rowKey for written records. Change the default only if you need to compute a more meaningful key instead of the random UUID.  Default: Enriched\_Record\_Row\_Key\_Resolver (UUID) |
| table-name-resolver-name | The method used to determine the HBase table name for storing enriched records.The default value, Data\_Model\_Table\_Name\_Resolver, uses a name determined by your data model. If you select User\_Configured\_Table\_Name\_Resolver, enter a table name for enriched records in the user-specified-table-name parameter.  Default: Data\_Model\_Table\_Name\_Resolver |
| user-specified-table-name | The name of the table used for writing enriched records; required only if the table-name-resolver-name = User\_Configured\_Table\_Name\_Resolver |
| write-level | Specifies what parts of JSON records (i.e., the granularity) are writable to the HBase data store. HBase uses one table per data model for storage, and one column family for each parameterized value. For example, with the default of Metadata,Objects,Fields, JSON records are written to HBase three times: metadata to one column family, objects to a second column family, and fields to a third column family; one DigitalEdge record per HBase row. Decisions for write-level values are dependent upon the web and user apps that you specify. For example, committing Fields to HBase may require more processing and storage space, but provides applications with queryable fields and full context data. Metadata storage may help facilitate record matching for the Pentaho dashboard application.  Default: Metadata,Objects,Fields  Recommendation: Metadata,Objects |

Hive data sink

Use this data sink to create a Hive cluster that is managed by DigitalEdge. Recommendation: Keep the default connection values for this data sink, and turn compression on.

| HiveDataSink | |
| --- | --- |
| Parameter | Explanation |
| block-size | Specifies the dfs.block.size for files written to HDFS. The default value is a medium size that can be used as a starting point.  Default: 134217728 |
| compress-data | Indicates if data should be compressed before it is stored  Default: false |
| hdfs-working-folder | The directory path in HDFS where pipeline-processed data is written and queries are run against it; must be writable by the user running this data sink in DigitalEdge; you can change the default folder name but not the root  Default: /tmp/hiveDataSink |
| hive-jdbc-hostname | The IP address or host name for the server running the Hive Thrift service  Default: localhost  Recommendation: keep the default value to use the embedded server |
| hive-jdbc-port | The port number for the server that is running the Hive Thrift service  Default: 10000  Recommendation: keep the default value to use the embedded server |
| jobtracker-hostname | The IP address or host name for the server running the Hadoop JobTracker daemon  Default: jobtracker.domain  Recommendation: keep the default value to use the internal Hadoop cluster |
| jobtracker-port | The port number that the Hadoop JobTracker is running on  Default: 8010  Recommendation: keep the default value to use the internal Hadoop cluster |
| namenode-hostname | The IP address or host name of the server that is running the NameNode daemon  Default: namenode.domain  Recommendation: keep the default value to use the internal Hadoop cluster |
| namenode-port | The port number that the NameNode daemon is running on  Default: 8020  Recommendation: keep the default value to use the internal Hadoop cluster |
| replication-factor | Controls the default replication factor on files written to the Hadoop Distributed File System by this data sink  Default: 2 |
| use-complex-schema | Enables complex table schema definitions (such as map <>, array <>, struct <>, etc.) derived from the canonical data model. This parameter requires the use of a supported SerDe (a Serializer/Deserializer that lets Hive read in data from a table and write it out to HDFS); true or false  Default: false |

JSON to JDBC data sink

Use this data sink to grab JSON objects, map them to an SQL database table, and write them to a specified database via JDBC. The data sink requires two special fields (tableName and tableKey) in the input model to associate the dimension table and keys with the data model.

This data sink requires just one parameter, to specify the destination database connection.

| JsonToJdbcDataSink | |
| --- | --- |
| Parameter | Explanation |
| connection-url | A pointer to the destination relational database (e.g., the DigitalEdge tenant database), in the format:  jdbc:<databasetype>[:protocol]://<host>[:port]/[database]  For example:  jdbc:mysql://localhost:3306/mytestdb  jdbc:h2:tcp://localhost:8161/commondb    Default: null |

Lucene indexing data sink

The Lucene data sink builds an inverted index that is optimized for real-time searching of DigitalEdge data. Use these parameters to control the indexing process. This data sink stores index entries, not fully processed records.

The Lucene indexing data sink works with several web apps:

* The Search web app: a search application based on the Solr™ open source enterprise search platform from Apache Lucene™
* The SearchAPI: a REST API for real time searches with the Zoie search engine

If you use either of these search applications, or if you are building a custom search client (for example, a Flex application or a Javascript browser application), you must set up a Lucene data sink to index the processed DigitalEdge records.

When sizing a Lucene data sink (datasink.lucene) with the Process Group Parameters, you should allocate 50% extra storage for index building and merging. For example, if you anticipate needing 1 TB space for a Lucene index, configure it for 1.5 TB.

| LuceneIndexingDataSink | |
| --- | --- |
| Parameter | Explanation |
| always-analyze | List of data fields that should always be tokenized (analyzed, parsed, and prepared for indexing); the field list should be comma-delimited  DigitalEdge uses the Lucene StandardAnalyzer for tokenization. Consult the Apache Lucene product documentation for details on how records are analyzed and tokenized.  Default: null  NOTE: When specifying this parameter, you should also set the index-control parameter to either Fields or ContentAndFields. |
| do-not-analyze | List of data fields that should never be tokenized (analyzed, parsed, and prepared for indexing); ); the field list should be comma-delimited  DigitalEdge uses the Lucene StandardAnalyzer for tokenization. Consult the Apache Lucene product documentation for details on how records are analyzed and tokenized.  Default: null  NOTE: When specifying this parameter, you should also set the index-control parameter to either Fields or ContentAndFields. |
| index-control | Specifies what parts of data records should be indexed:   * ContentOnly: the entire body of the JSON record * FieldsOnly: fielded data only * ContentAndFields: both the JSON body and record fields * None   Double-click to select from the drop-down menu.  Default: ContentAndFields  NOTE: If you choose to index FieldsOnly or ContentAndFields, you can optionally specify the always-analyze or do-not-analyze parameter to selectively limit the indexed fields to a specify subset of fields. |
| zoie-batch-delay | Number of milliseconds between each batch submission to Lucene before in-memory index is flushed  Default: 60000 |
| zoie-batch-size | Number of fields to store in each batch submission to Lucene before memory is flushed  Default: 10000  Recommendation: keep the default value |

MongoDB data sink

Use this data sink to build a general purpose MongoDB data store for processed data. You can configure MongoDB once per system. Each instance sets up its own copy of MongoDB; no cluster is created. The parameters specify the connection, timeout, and database name. Note that MongoDB has a maximum record size of 16 MB.

| MongoDbDataSink | |
| --- | --- |
| Parameter | Explanation |
| auto-connect-retry | Determines if DigitalEdge times out or keeps looking for data on the connection; must be true for the connect-timeout-ms parameter to work; double-click to select true or false from the drop-down menu  Default: true  Recommendation: keep the default value |
| connect-timeout-ms | How long (in milliseconds) DigitalEdge will wait for a connection before it gives up looking for data on the queue   * 0 = no timeout; if a connection is not established, an error is returned * any number higher than 0 = DigitalEdge will retry looking for data   Default: 0  Recommendation: if your network is slow, set this parameter to a higher number; also, set auto-connect-retry parameter to true |
| database-name | Name the DigitalEdge data store with any valid MongoDB name of your choice  Default: dbname |
| mongo-server-host | Name of the host server running MongoDB; multiple server names should be separated by spaces or tabs  Default: localhost server1 server2  Recommendation: localhost |
| mongo-server-port | Dedicated port on which MongoDB will run; must be between 1024 and 65535  Default: 27017 |

Sleep data sink

This data sink is used strictly for test purposes. It reads a record, optionally creates a log entry, and then sleeps for a specified amount of time, to test the integrity of the DigitalEdge pipeline. It does not store or process records.

| SleepDataSink | |
| --- | --- |
| Parameter | Explanation |
| delay | Time between reads while processing records; expressed in milliseconds  Default: 5000 |
| input-logging | Optionally creates an entry in the ingest log for each record read  Default: false |

Step 4: Add Web Apps

Webapps are all the APIs that are used by DigitalEdge to serve up information. DigitalEdge provides two types of web applications:

* User applications: tools for users to search data stores, analyze data, and examine alerts
* REST APIs: web services that are used to build applications; most APIs are dependent on other web apps and data sinks

You will probably just select user applications; DigitalEdge will automatically select dependent REST APIs that are required for the user tools.

Add a user application to your system

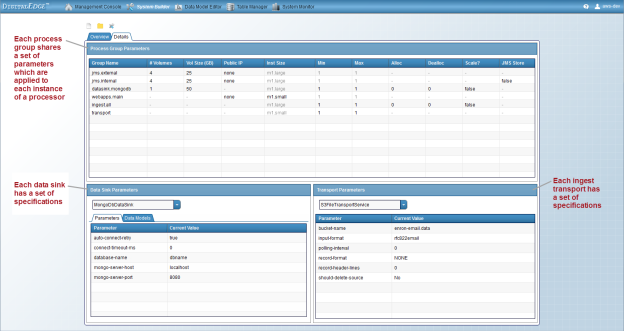
1. In the System Builder, click the Overview tab. Use the Webapps/REST APIs section.
2. Click Add. The Select a Webapp/REST API dialog box appears. Click on an application name. (Note that this list includes the user applications that are included in the core release. You may also have custom apps created for your site.)
   * alertcontroller: an application used to manage alert criteria, notifications, and subscriptions
   * alertsapi: a REST API to manage alert criteria used by the AlertingDataSink
   * dbapi: a REST API to access tables in the tenant database
   * hue.server: a web-based user interface for Apache Hadoop; requires the Hive data sink
   * Kibana: a beta webapp, Elasticsearch’s visualization engine, including a browser-based analytics and search interface; this webapp requires the Elasticsearch data sink
   * metricsapi: a REST API that collects system statistics; use this API to pull system metrics into an application external to DigitalEdge
   * pentaho.bi.server: a webapp that creates a server instance for the implementation of Pentaho dashboards
   * phoenixapi: a beta REST API for interacting with the Phoenix SQL query engine on HBase
   * search: a search application based on the Solr™ open source enterprise search platform from Apache Lucene™; this app provides a quick way to search DigitalEdge records with minimal development; you must add a Lucene data sink to DigitalEdge when using this API
   * searchapi: a REST API for real time searches; this API helps you build a custom search interface for your system; you must add a Lucene data sink to DigitalEdge when using this app
3. Click OK. If a web app is dependent upon another API or web app, a warning message will alert you to add those dependent applications.
4. Click OK. The application is added to your configuration.
5. Click Build  if you are done with your System Builder work.

To feed DigitalEdge data into a pre-existing user application in your shop, configure an externally facing data sink such as the Scripting Data Sink or the JMS Data Sink (See " Step 3: Add Data Sinks" on page 141 for more information).

Step 5: Specify Advanced Parameters

The System Builder Details tab is for advanced system fine-tuning. You should be thoroughly familiar with your system before attempting to modify defaults and parameter settings, and you should be a member of the security group that is authorized to change the parameters. See "What Each Node Does" on page **Error! Bookmark not defined.** for details about what is on each node. If you need to refine your system, consult Leidos Engineers before using the System Builder tool. Use the Details tab to:

* Manage system components
* Allocate storage
* Assign persistent IP addresses
* Determine which security group is associated with a process group
* Manage auto-scaling attributes



The Details tab workspace includes three panels, one each for process group parameters, data sink parameters, and transport parameters. Double-click on any parameter value to edit it. Grayed out values cannot be changed.

Process group parameters

A process group represents a category of processors, not individual instances of processors, representing one functional area (transport, ingest, index, etc.). You can view process group parameters in System Builder, on the Details tab. Process groups are built based on the choices you made for components and applications on the Overview tab of the System Builder. For example:

* ingest.all
* transport
* datasink.mongodb
* jms.internal

Process group parameters control auto-scaling and resource allocation of each instance (VM) in the group. Default parameter values are initialized based on the System Size that you chose on the Overview tab.

Each process group is associated with a pre-defined DigitalEdge security group. Security groups are managed in the Management Console on the Security page. You must have security group permissions to edit parameters here.

Each process group shares a set of parameters which are applied to each instance:

| Process Groups | |
| --- | --- |
| Parameter | Explanation |
| # Volumes | How many volumes are currently consumed by an instance of this process group (cannot be changed after you Start a system) |
| Vol Size (GB) | Size per volume, in gigabytes |
| Public IP | Whether or not the process uses a persistent IP. You can assign an IP address by double-clicking on this value. In the Specify Persistent IP Address pop-up, either enter a Specific Address or click Allocate IP Address for DigitalEdge to assign an address.  The only time you must assign an IP is for the webapps.main node if you are running on a Eucalyptus platform and you do not have an external DNS forwarder configured. If you are running in VPC, you do not have to assign IP addresses; AWS assigns elastic IP addresses automatically. |
| Inst Size | Size of an instance, as defined by Amazon's node instance types (Xsmall, small, medium, large, Xlarge, XXlarge) |
| Min | The initial allocation of nodes to an instance is expressed as a range from minimum to maximum; this is the minimum number.  Special cases:   * The cassandra.seenode Min value will always be set to 2 * Data sinks using Zookeeper must have the zookeeper process group Min value set to an odd number. System building will fail if Zookeeper is set to an even Min value. * A clustering data sink must have its process group Min value set to an odd number for system building to succeed. |
| Max | The initial allocation of nodes to an instance is expressed as a range from minimum to maximum; this is the maximum number. Set the Max parameter if you need to cap auto-scaling to the terms of a public cloud services contract.  Special cases:   * The cassandra.seenode Max value will always be set to 2 * Data sinks using Zookeeper must have the zookeeper process group Max value set to an odd number. System building will fail if Zookeeper is set to an even Max value. * A clustering data sink must have its process group Max value set to an odd number for system building to succeed. |
| Alloc | When auto-scaling up an instance, nodes are added in quantities; this is the number of nodes that are allocated each time the process is scaled up |
| Dealloc | When auto-scaling down an instance, nodes are removed in groups rather than individually; this is the number of nodes that are deallocated each time the process is scaled down. Use caution when deallocating instances; in some case, you may lose data when you scale down (especially in Hadoop clusters). |
| Scale? | Whether or not a process is auto-scaling |
| JMS Store | Turn data persistence on or off for an instance; primarily applies to the internal instances which keep data in memory and do not persist data to disk by default; the JMS external node is always persisted to disk; valid values = true or false |

When sizing a Lucene data sink (datasink.lucene), you should allocate 50% extra storage for index building and merging. For example, if you anticipate needing 1 TB space for a Lucene index, configure it for 1.5 TB.

Data Sink Parameters panel

Use this panel to modify the parameter values that were previously specified in the Set Datasink Parameters dialog box.

Transport Parameters panel

Use this panel to modify the parameter values that you specified in the Set Transport Parameters dialog box.

Step 6: Save and Build a System

Your system configuration work is never final until you save it and build the system.

When you are done defining or editing a system build, click the Build icon.  System Builder will validate your work, displaying warnings or errors for invalid configurations.

The new system is not activated and running until you Start the system in the Management Console on the Systems screen. See the DigitalEdge Operations Guide for more information.

If you edit a system and save it, you cannot overwrite the system configuration of a running system. You must stop the system first in the Management Console on the Systems screen. See the DigitalEdge Operations Guide.

Step 7: Maintain a System Definition

You can always return to a saved system configuration to tweak it. Most edits are considered advanced configuration tasks which system experience or assistance from Leidos. Typical tasks in System Builder include:

* Add another data model, transport, data sink, or web app on the Overview tab
* Remove a data model, transport, data sink, or web application on the Overview tab, using the Remove buttons
* Modify the Data Sink Parameters on the Details tab
* Modify the Transport Parameters on the Details tab
* Find out what software version you are using on the Overview tab
* Re-size your system on the Overview tab
* Modify auto-scaling specifications on the Details tab
* Reallocate resources on the Details tab

Edit a system definition

1. Access the System Builder tool  from the Management Console Tools list: https://<your-domain-name>/tenantconsole
2. Click Open.   The Select system domain to open dialog box appears.

System Builder works exclusively with private data models, components, and systems.

1. Select a System Domain name and click OK.
2. Follow the guidelines in Steps 1-6 to edit system properties and to choose additional components.

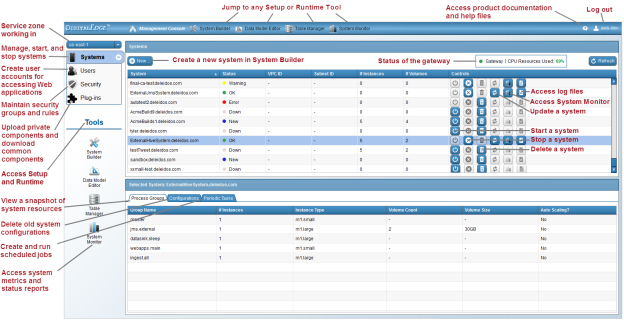
Edit system parameters

1. Log in to the System Builder.
2. Select a System Domain to work with.
3. Click the Details tab.
4. To edit a parameter value, double-click on the current value and change it.

Grayed out values cannot be changed.

# Appendix G: DigitalEdge™ Operation Guide, Chapter 5 – *Managing DigitalEdge from the Management Console*

Once you have a system configured, built, and ready to test or to run in production, the Management Console is the runtime tool that you will use every day. The Management Console is the main portal into DigitalEdge, providing access to all the major UI tools. The Management Console dashboard includes icons for starting and stopping systems. Status views in both the Management Console and the System Monitor support assessing and troubleshooting system health and performance.



Use the Management Console to:

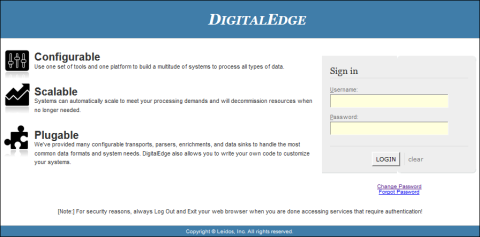
* Start a new system: See "Starting a system for the first time" on page 166
* Stop and restart a running system: See "Shutting down and restarting a system" on page 167
* Stop and restart an individual process group: See "Stopping and restarting a process group" on page 167
* Update a system with edits to a data model, transport parameters, or data sink parameters: See "Updating a system" on page 168
* Delete a system: See "Deleting a system" on page 170
* Delete a system configuration: See "Deleting a system configuration" on page 171"
* Maintain security rules for security groups: See "Managing security groups and rules" on page 171
* Create and maintain user accounts for accessing web apps: See "Managing users " on page 174
* Get custom-built plug-in components into the system: See "Managing the component repository" on page 175
* View a quick snapshot of the system’s status: See "Viewing system status" on page 185 (to edit process group parameters in System Builder, See " Process group parameters" on page **Error! Bookmark not defined.**)
* View log files: See "Checking log files" on page 188
* View system metrics: See "Checking system metrics" on page 191
* Access all the setup and runtime tools (Data Model Editor, Table Manager, System Builder, and System Monitor) from icons on one convenient Management Console dashboard:

NOTE: Each Setup and Runtime tool will time out after approximately 30 minutes of inactivity. Also, if you have multiple tools open on separate tabs of a web browser, all the tools will time out if one tool reaches the timeout threshold. On the timeout screen, you can Sign Out of your session (and lose unsaved data), Sign Back In to re-enter any unsaved data, or Work Offline to record any work you were doing that was not saved.

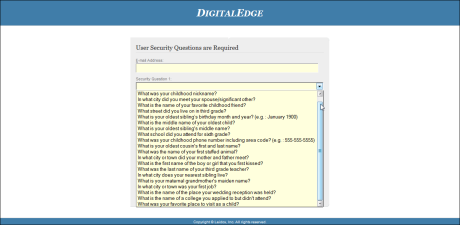
Logging in

Use this procedure to log on to the DigitalEdge Management Console.

1. In a web browser, go to https://default.<system\_domain\_name>/tenantconsole
2. Enter your Username and Password.
3. Click LOGIN.



1. The first time you log on to DigitalEdge, you must complete your registration:
   1. Supply your email address, and three security questions and answers to be used when confirming your identity:



* 1. Click Continue.
  2. Read the DigitalEdge License and Support Agreement and click ACCEPT.

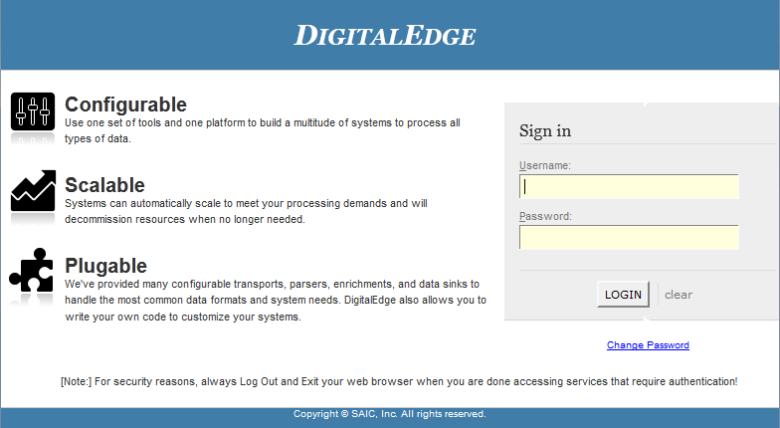
1. You can access all the DigitalEdge Setup and Runtime tools from the Management Console.

You cannot access the Management Console with an expired DigitalEdge license; contact Support for a new license.

Starting a system for the first time

To start a new system for the first time:

1. Open the Management Console.
   1. In a web browser, go to https://<system\_domain\_name>/tenantconsole



* 1. Enter your Username and Password.
  2. Click LOGIN.

The first time you access the Management Console, a License Agreement will appear. Read the agreement and click Accept to continue.

1. From the Systems  screen, highlight the System name that you want to launch.
2. Click the Start System icon.  The Select a version dialog box appears.
3. Click the Version number you want to run and click START.

Starting a system can take some time. It takes 1-3 minutes to launch each instance in a system.

1. The Status of the newly started system changes in the Systems list. See "Viewing system status" on page 185 for an explanation of each status.

Shutting down and restarting a system

There are times when you have to stop your system temporarily then restart it. Stop a system when you:

* Add a new plug-in component (you cannot overwrite the system configuration of a system that is running)
* Reconfigure any part of the system (you cannot overwrite the system configuration of a system that is running)

In the Management Console:

1. Click the Systems option. 
2. Click the System name you want to work with.
3. Click  Stop System to temporarily stop a system.
4. You can restart the system with the Start System icon. 

Stopping or starting a system can take some time. Expect approximately 1-3 minutes per instance in a system.

You can also stop and restart select process groups instead of stopping the entire system. See "Stopping and restarting a process group" on page 167

You can update a running system after editing a data model, transport parameters, or data sink parameters. See "Updating a system" on page 168

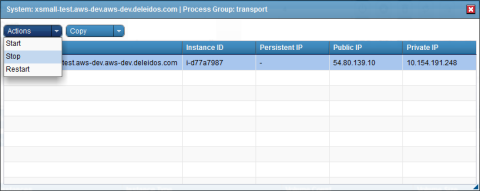
Stopping and restarting a process group

There are times when you need to stop and restart one process group. You can do this for a transport, ingest.all, or datasink.lucene without stopping the entire system. For example, you may need to stop and restart a process group in these situations:

* The ingest.all node is down and the log viewer indicates a minor problem. Bring up the ingest process by Restarting that process group.
* Data entering the system is incorrect. The Systems Administrator Stops ingest.all to examine and correct the data before Restarting ingest.
* The Lucene data sink has filled up and you want to wipe out the data to start fresh with new data. Purge the data sink and start it up again.
* You need to stop the flow of data by Stopping the transport process group.
* The transport, ingest.all, or datasink.lucene process has run out of memory. To get it back into a working state, Restart that process group.

In the Management Console:

1. Click the Systems option. 
2. Click the System name you want to work with.
3. On the Process Groups tab, double-click the individual Group Name you wish to stop: transport, ingest.all, or datasink.lucene. A System dialog box appears.
4. Highlight the name of one or several process groups.
5. On the Actions menu, select Start, Stop, or Restart (Restart is a combination of Stop and Start).



Note that when working withe the datasink.lucene process group, the Actions menu includes an additional choice: Purge Data, which wipes out the old index and automatically Restarts the indexing process.

1. DigitalEdge processes your request and reports when the action is complete.

Updating a system

You can update and restart the following components at any time without stopping a running DigitalEdge system:

* Data model
* Transport: after you have edited its parameters (not for adding new nodes)
* Data sink: after you have edited its parameters (not for adding new nodes)

When you complete your changes to a data model in the Data Model Editor, or to a transport or data sink in System Builder, you must update the system in Management Console to use the revised component.

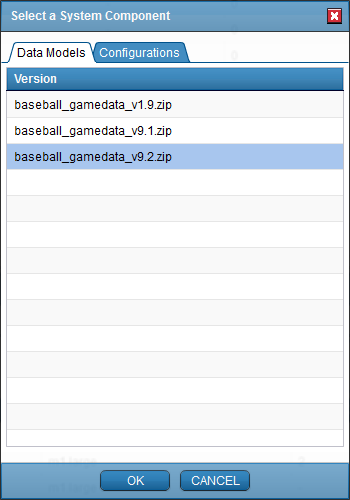
1. Edit and save the revised component:

* In Data Model Editor, make sure Save As the revised data model so you can assign the latest minor version number to the data model
* In System Builder, be sure to associate each updated transport or data sink with a Config Version number

1. Go to the Management Console.
2. Select the system that you are working with. Make sure its status is OK.
3. Click the Update System icon . The Select a System Component dialog box appears.

You can update one component at a time in Management Console. If you have edited multiple components, you must Update System individually for each revised component.

1. Select either the Data Models tab to update a data model, or the Configurations tab to update a transport or data sink.
2. Click on a Version of the data model or system that is ready for updating.



1. Click OK. DigitalEdge builds the system and pushes the updated component to the ingest nodes and any data sink that uses ingest.

To restart process groups, See "Stopping and restarting a process group" on page 167.

To shut down and restart a complete DigitalEdge system, See "Shutting down and restarting a system" on page 167.

Deleting a system

If you have a configured system that you are no longer using, you can permanently delete it from DigitalEdge. For example, if you created several test systems before going into production, you may want to delete older versions of a system.

You cannot delete a live system. The system status must be New, Down, or Error before you can permanently delete it.

In the Management Console:

1. Click the Systems option. 
2. Click the System name you want to work with.
3. Click  Delete System: to permanently delete a system.

When you delete a system, its master configuration file, all previous configuration files, and all data records are erased. If you need to keep the data records, do not delete the system unless you have a backup.

Deleting a system configuration

If you have an older system configuration that you are no longer using because you have revised or updated your system build, you can permanently delete an older configuration file from DigitalEdge. For example, if you created several test configurations before going into production, you may want to delete older system definitions.

In the Management Console:

1. Click the Systems option. 
2. Click the System name you want to work with.
3. Click the Configurations tab in the bottom pane.
4. Highlight an older configuration that you want to get rid of.
5. Click  Delete System Configuration to permanently delete a system.

If you delete the current configuration, DigitalEdge will delete the entire system and all its data records (as if you had deleted the system, not just a configuration file).

Managing security groups and rules

A security group controls incoming communications to a server or process by implementing a set of firewall rules. DigitalEdge security groups are modeled on EC2™ security groups, which restrict communications based on protocol (TCP, UDP, etc.), IP address, and port. DigitalEdge security groups are predefined and automatically assigned to each process group specified in System Builder (e.g., JMS, ingest, data sinks, web apps). A security group authorizes work with a process group, provides the ability to open a port for use, and specifies what outside networks can communicate with a process group.

You cannot create, edit, or delete a security group. But you can specify security rules associated with a security group.

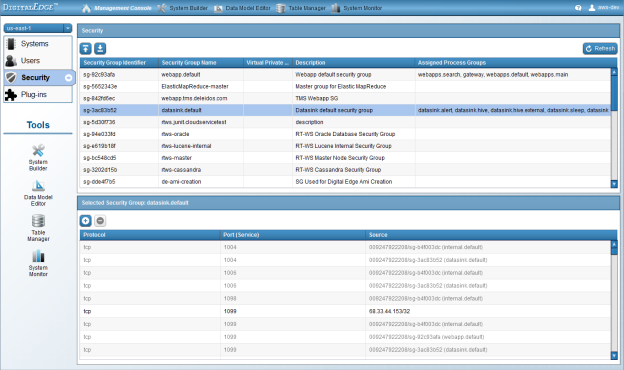
Security group parameters are specified as security rules. A security rule is a permission or a firewall ACCEPT rule. Each rule specifies a communications protocol, a port, and a source (either IP address or another security group that can talk to a process). While you cannot create or edit security groups, you can specify the security rules that are associated with a security group. Use the Management Console’s Security section to work with security rules. The Security section is automatically populated with default security group settings based on the component choices you made in System Builder.

Use the Security section of the Management Console to:

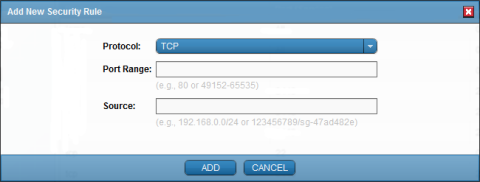
* Open a port for a newly added instance/component (added in System Builder).
* Check an instance that is associated with a security group
* Determine who can connect to a process
* Define a security rule for a security group
* Upload or download DigitalEdge certificates, keystores, and truststores for web apps, the Data Transfer Utility, etc.

Add a security rule to a security group

1. Open the Management Console.
2. Click the Security option. 
3. From the Security list, click the row of a security group you want to work with. The list of security rules appears in the bottom panel.



1. Click the Add Security Rule  button. The Add New Security Rule dialog box appears.



1. Select a Protocol from the drop-down menu (TCP, UDP, or ICMP).
2. Enter a Port Range (one port number or a range of ports).
3. Enter a communications Source. A source can be an IP address, a range of IP addresses, or another security group that can talk with the assigned process groups.
4. Click ADD.

For example, suppose you used System Builder to add a data sink that will write to an external database on your corporate system. Select and configure a data sink in the System Builder, then locate the data sink in the Management Console’s Security section. Click on the security group to see the security rules associated with it. You could then Add a new security rule to open a port to attach the data sink to the external database.

You cannot edit a security rule. To change a rule, Delete it and Add it again with revised parameters.

Rules which specify standard ports cannot be deleted. These rules are grayed out in the Security Rule List.

Delete a security rule

1. From the Security list, click the row of the security group you want to remove. The list of security rules appears in the bottom panel.
2. Click the rule you want to remove.
3. Click the Delete  button.
4. In the Confirmation dialog box, click Yes to delete the security rule.

Upload or download security artifacts

DigitalEdge provides the ability to upload or generate certificates (used primarily for web apps), truststores, and keystores (often used with the Data Transfer Utility).

1. Open the Management Console.
2. Click the Security option. 
3. Click the security group you want to work with.
4. Click the Upload  button or the Download button .
5. Specify a Certificate, Keystore, or Truststore.
6. Click Upload....
   1. Specify the upload location.
   2. Browse to the file you want to upload.
   3. Click Upload.

Or click Download....

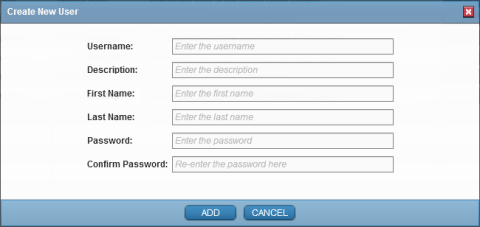
* 1. Open or Save the file.
  2. Click OK.

Managing users

All primary and secondary tenants on an account in DigitalEdge share a user instance, LDAP, and private data store for users. User accounts are created in DigitalEdge for anyone who will be running and using a web application. Note that all users are authorized to run all end-user web applications; you do not have to associate users with individual web apps. When you create users, DigitalEdge sets up an LDAP system for your system.

Add a user

1. Open the Management Console.
2. Click the Users option.  The Users workspace appears.
3. Click Add User.  The Create New User dialog box appears.



1. Create and enter a Username.
2. Enter a Description of this user.
3. Enter the user’s First Name.
4. Enter the user’s Last Name.
5. Enter the E-mail Address for contacting the user.
6. Enter a security Password (there are currently no rules for constructing passwords).
7. Re-enter the password for confirmation.
8. Click ADD.

When you need to edit the user list, you can scroll through the list if it is short. If you have a long list of users to maintain, you can jump to an alphabetical starting point in the list. Or, you can search for a user that you wish to view, edit, or delete.

Narrow down a user list

1. In the Management Console, click the Users option.  The Users list appears.
2. Click on a First Letter option to quickly scroll through the user list to a specific alphabetical point.

Search for a user

1. In the Management Console, click the Users option.  The Users list appears.
2. In the upper right corner, enter a string to search on. The string can be all or part of a username, first name, or last name. The string can be embedded anywhere in the user name.
3. Click Filter.  A list of matching users is compiled.

Edit a user

1. From the user list, click the Edit button  to the right of the user you want to change. The Update User dialog box appears.
2. You can edit any field except the Username.
3. Click UPDATE.

Delete a user

1. From the user list, click the Delete button  to the right of the user you want to remove.
2. In the Confirmation dialog box, click Yes to delete the user.

Managing the component repository

DigitalEdge uses a Master Repository to store all the common and private plug-in components available to your DigitalEdge installations. This repository drives the list of system features available in the Setup Tools (System Builder, Data Model Editor). The Master Repository resides in the TMS environment where only high-level TMS Administrators can access files.

The repository for a primary tenant is sized at 125 MB. This means that:

* You can have a maximum of 7 systems in the starting state at any one time across all secondary tenants in your account.
* DigitalEdge comes with many plug-in components, which occupy 60 MB.
* When you create new plug-ins and scripts, you have a maximum of 65 MB available for repository storage.
* When you upload a new plug-in or script, if the new component will exceed the available storage space, DigitalEdge will warn you and prevent the upload.

DigitalEdge ships with many components that help you build systems. Frequently, however, the standard plug-in components are not sufficient for your site. You may need an additional script or a custom transport for ingesting data. This is where the DigitalEdge SDK comes into play. Use the SDK to create:

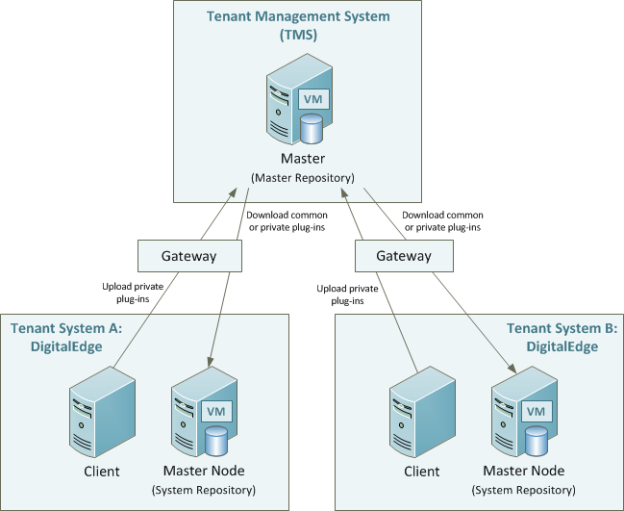
* Transports
* Data sinks
* Parsers
* Enrichments

Once you create a custom plug-in, you need a way to get it into the Master Repository for safe storage and to access the component for downloading into a DigitalEdge system. The Plug-ins feature of the Management Console provides this functionality. The functions in the Plug-ins section help you upload a custom component to the Master Repository from your hard drive. From there, you can then include the custom component in your DigitalEdge installation with Data Model Editor (parsers and enrichments) or System Builder (transports and data sinks).

To integrate a data sink with DigitalEdge, contact a DigitalEdge Support Engineer to assign it to a process group.

You usually do not have to manually download these plug-ins to your tenant System Repository; downloading is automatically done at system start-up and re-start. Plug-ins are triggered for system download as you specify components to use from the lists in System Builder and Data Model Editor. Downloadable plug-ins include:

* Transports
* Parsers
* Enrichments
* Scripts



You can upload the following types of plug-ins to the Master Repository:

* Software: Custom created plug-in components such as transports, enrichments, and parsers
* Scripts: Scripts are used in two places in DigitalEdge: to drive a scripting data sink in System Builder, or to specify a non-standard data source translation in Data Model Editor

Getting custom components into the repository

After you create a custom plug-in with the DigitalEdge SDK, you need to upload the component into the TMS Master Repository from your hard drive. In the Management Console, the Plug-ins section provides this functionality. Once a component is in the Master Repository, you can select and download it in Data Model Editor or System Builder just as you would a common component (that is, a component provided with DigitalEdge).

Upload a custom (private) plug-in to the Master Repository

1. Access the Management Console.
2. Select the Plug-ins section. 
3. On the Software tab, double-click  Private Software .

You can double-click on Common  to see a list of components that are in the Master Repository, that are provided with DigitalEdge, and that are downloaded when you Start a system. But you cannot upload, delete, or manipulate files and folders in the Common area. If the component list is long, you can narrow down the file list with a filter string in the Start Typing to Filter box.

1. Next, you must specify the Master Repository folder where your plug-in(s) will reside. Either choose an existing folder in the Master Repository or create a new folder.

Double-click on an existing folder in the Master Repository as the destination.

or,

Click New Folder  to create a new destination folder in the Master Repository.

1. Click Upload . The Upload Software dialog box opens. The Plugin space used indicator displays the percentage of your allocated repository space that is already in use.
2. Click Browse to locate and select the private component from your hard drive.

You can Upload a transport, parser, or enrichment plug-in of file type .JAR or .ZIP. To upload a data sink, contact Leidos.

1. Click Upload . If the file is not in the appropriate JAR file format, an error message will appear.

Each private component must be packaged in a jar file, should contain a pom.xml file, and should not contain nested jars, as specified in the standard at <http://maven.apache.org/>.

If the new component will exceed the available storage space, DigitalEdge will warn you and prevent the upload.

1. The plug-in now appears in the list of available components on the Software > Private Software tab. It is also now selectable in System Builder.

Upload a private script file to the Master Repository

1. Access the Management Console.
2. Select the Plug-ins section. 
3. On the Scripts tab, double-click  Private Scripts .

You can double-click on Common  to see a list of components that are in the Master Repository and provided with DigitalEdge, but you cannot upload, delete, or manipulate files and folders in the Common area.

1. Next, you must specify the Master Repository folder where your script(s) will reside. Either choose an existing folder in the Master Repository or create a new folder.

Double-click on an existing folder in the Master Repository as the destination.

or,

Click New Folder  to create a new destination folder in the Master Repository.

1. Click Upload . The Upload Scripts dialog box opens. The Plugin space used indicator displays the percentage of your allocated repository space that is already in use.
2. Click Browse to locate and select the private script from your hard drive. You can upload .SH, .PY, PL, or .GROOVY files.
3. Click Upload .

If the new component will exceed the available storage space, DigitalEdge will warn you and prevent the upload.

1. The plug-in now appears in the list of available components on the Scripts > Private Scripts tab. It is also now selectable in System Builder, Data Model Editor, or the Periodic Tasks section of the Management Console.

Create a new folder in the Master Repository

1. Access the Management Console.
2. Select the Plug-ins section. 
3. Select either the Scripts or Software tab.
4. Double-click the Private option.
5. Click New Folder  or double-click on an existing folder first to create a lower level folder.
6. In the Make New Folder(s) dialog box, enter the New Folder(s) name, separating nested folder names with a forward slash /, then click OK.
7. You can now upload files to the new folder(s).

Deleting a custom component

If you have a private plug-in stored in the Master Repository that you no longer need, you can permanently delete it. Once deleted from the Master Repository, a plug-in will not appear in the master list of available components in System Builder.

Delete a private file or folder from the Master Repository

1. Access the Management Console.
2. Select the Plug-ins section. 
3. Select either the Scripts or Software tab.
4. Double-click the Private option.

You cannot delete a component from the Common  area.

1. Double-click on the source folder in the Master Repository if there are multiple folders to choose from.
2. Click on the file  or folder  that you want to delete. (Ctrl-click to deselect an item.)

When you delete a folder, all its contents will also be deleted (subfolders and files).

1. Click the Delete icon .
2. The Confirm Delete dialog box appears. Click Delete to confirm, or click Cancel to keep the plug-in.

Delete cannot be undone. Be sure you want to delete the file from the Master Repository. Also note that this function does not delete any components in your DigitalEdge system. If you restart your system (in the Management Console > System), the deleted plug-in will be removed from your DigitalEdge system.

Creating and scheduling jobs

A periodic task is a job that you script and run on a regularly scheduled basis. For example, you may want to create and schedule jobs to:

* Tar up log files to create and store an archive of DigitalEdge logs
* Archive any DigitalEdge files
* Automatically stop an inactive system from running at night and restart it in the early morning for users, to reduce cloud costs
* Run a map reduce job nightly to generate analytical reports
* Connect to a transactional database nightly to update DigitalEdge data sinks on a scheduled basis; this may be useful if a database is not available in real time as a data source for DigitalEdge (e.g., your organization has a policy restricting access during business hours)

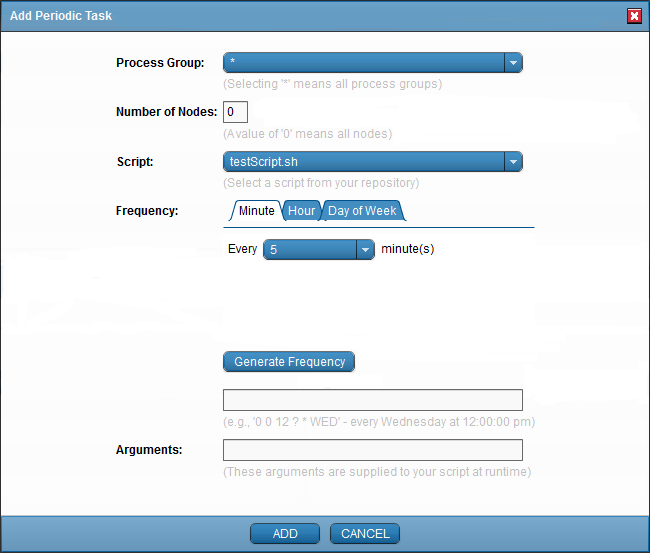
Tasks should be written as shell scripts and stored in the root directory of your script repository.

You can use any scripting language that can be run on a command line (e.g., Bash, Perl, etc.).

Use the Management Console > Systems > Periodic Tasks tab to create and schedule jobs.

Create a scheduled job

1. Write a shell script for your task.
2. Save the script in the root directory of your private scripts repository (Management Console > Plug-ins > Scripts > Private Scripts). See "Getting custom components into the repository" on page 177 for details.
3. In the Systems section of the Management Console, highlight the system you want to work with.
4. Click the Periodic Tasks in the bottom panel. Click the Add icon. 
5. Create and schedule the task:



* Process Group: Select the process group you want to work with, or \* to select them all.
* Number of Nodes: Indicate how many sequential nodes to work with; 0 = all, 2 = the first and second nodes, etc.
* Script: Select the shell script you wrote and stored in the root directory of your private scripts repository.
* Frequency: Define the schedule for running your script by selecting one tab to work with:
  + Minute: Select the interval in which to run the script, such as every 30 minutes. Scripts are run on a multiple of the minutes selected. For example, select 10 to run a job starting at the next interval of 10 minutes. If it is currently 1:14, the script will first run at 1:20, and then repeat every 10 minutes after that: 1:30, 1:40, etc.
  + Hour: Select the hourly interval for running the script, such as every 8 hours. Scripts are run on a multiple of the hours selected. For example, select 2 to run a job every 2 hours (starting at the next hour divisible by 2). If it is currently 5:10 PM, the script will first run at 6:00 PM (the next even-numbered hour), and then repeat every 2 hours after that: 6:00 PM, 8:00 PM, etc. (One special case: If you select 13, the job will run at midnight and 1:00 PM consistently.)
  + Day of Week: Click a day for running the script and select the Start Time for kicking off the job. This script will run once a day.

Click Generate Frequency to see the defined schedule in the text box. You can edit this text, or enter your own definition for a more complex frequency (e.g., run a job every 12 hours only on weekends).

* Arguments: Enter any run-time arguments that your script requires.

1. Click ADD. The new task and details are listed on the Period Task tab.

You cannot edit a periodic task; to change a task, delete it and re-create it.

To determine if a periodic task was scheduled and actually ran, access the /logs/default.log file.

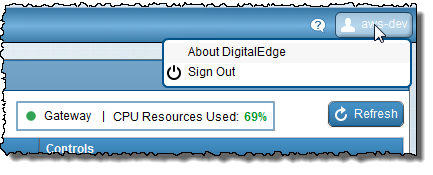
Delete a scheduled job

1. In the Systems section of the Management Console, highlight a job in the Periodic Tasks list at the bottom of the screen.
2. Click the Delete  button.

Logging out

Use this procedure to log out of DigitalEdge.

1. Go to the Management Console.
2. Click the user icon in the upper right corner and select Sign Out.



Use the same procedure to log out of any Setup or Runtime UI tool.

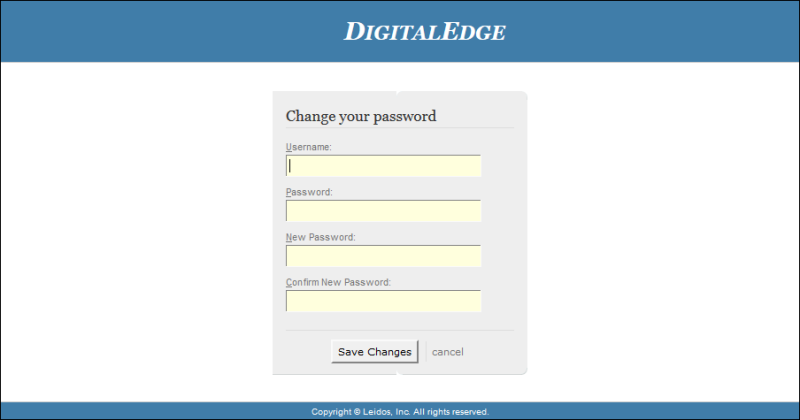
When you Sign Out of one tool, all open tools are automatically signed out.

Changing passwords

Use this procedure to periodically change your DigitalEdge password.

1. In a web browser, go to https://<system\_domain\_name>/tenantconsole
2. Click Change Password.

To reset a forgotten password, See "Resetting a forgotten password" on page 184.

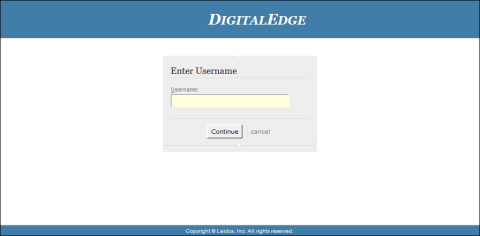


1. Enter your Username, current Password, and New Password.
2. Enter the new password again to Confirm New Password.
3. Click Save Changes.

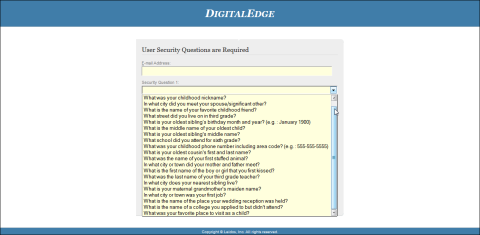
Resetting a forgotten password

Use this procedure to create a new DigitalEdge password if you have forgotten your current password.

1. In a web browser, go to https://<system\_domain\_name>/tenantconsole
2. Click Forgot Password.



1. Enter your Username and click Continue.
2. You are asked to confirm your identity by answering your three security questions.



Click Continue.

1. Enter and confirm your new password.

Click Save Changes.

# Appendix H: DigitalEdge™ Operation Guide, Chapter 6 – *System Monitoring*

A number of points exist throughout DigitalEdge to monitor data flow and normal operations. These access points include:

* Management Console:To get a quick snapshot of a running system, including what process groups are running, the number of instances and types, and the number of volumes and sizes; See "Viewing system status" on page 185
* Log files: Each DigitalEdge node runs a log file that records startup processes and problems; See "Checking log files" on page 188
* System Monitor: A dynamic console tool for monitoring system health and potential problems in real time, visually depicts system activity and resource scaling; See "Checking system metrics" on page 191

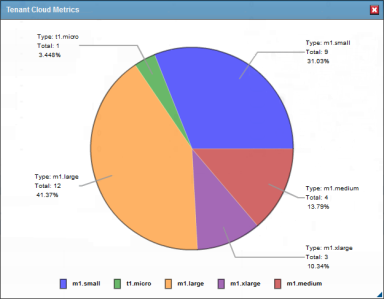
Viewing system status

You can get a quick snapshot of a running system in the Management Console. This is not a dynamic view like you see in the System Monitor. This snapshot tells you what process groups are running, the number of instances and types, and the number of volumes and sizes. The view is static, not auto-refreshed. Click Refresh  to view the latest status.

1. Open the Management Console.
2. The status of the Gateway node  appears in the top right (green = OK, yellow = Warning, red = Error).
3. Click Gateway to view its name and IP addresses for troubleshooting.
4. The Gateway icon also includes the current percentage of available CPU Resources Used for the node:

| % CPU  Resources Used | State |
| --- | --- |
| 0 - 74% | Green = OK |
| 75 - 89% | Orange = Caution: Consider adding resources |
| 90 - 100% | Red = Warning: Increase resources now |
| ERR | The Gateway status refreshes every 5 seconds; an Error status may resolve itself. If the Gateway stays in ERR mode for several minutes, call DigitalEdge Technical Support. |
| ??? | When the Gateway first starts up, its initial status is listed as ???. This status should resolve to a normal state within seconds. |

1. Click CPU Resources Used to view a pie chart that details resource utilization for the virtual machines in your tenant cloud environment:



This chart summarizes the total resources used for each instance in your tenant account. To increase available resources, first shut down all unnecessary DigitalEdge systems running in Management Console (test systems, prototypes, outdated systems, etc.). If available resources are still less than 25%, you should consider sizing up. On an AWS cloud, increase your instance limits to allow for the creation of more instances. On a Eucalyptus-based system, add more hardware. In either case, call DigitalEdge Technical Support for assistance.

1. Click the Systems option  . A list of configured systems displays the status of each system. As a system is created and then started up, the system status should cycle through New > (Starting) > OK.

| Status | | Description |
| --- | --- | --- |
|  | Green | OK: All nodes have started, everything is functioning normally, the database is valid. |
|  | Blue | New: A reminder that you created and built a new system, but haven’t run/started the system yet. Click the Start icon to run the system. |
|  | Light gray | Down: The master node matches data in the cloud, but the cloud has stopped working. |
|  | Yellow | Warning: The system's master node could not find information about process groups; DigitalEdge cannot determine if the system is running. The system will keep trying to start. Click on the status to see more detailed information in the bottom panel. |
|  | Red | Error: TMS cannot locate the system that is stored in the database. Click on the status to see more detailed information in the bottom panel. You can delete and recreate the system or contact Leidos for assistance. |
|  | Dark gray | Unknown: The system is transitioning between statuses. |

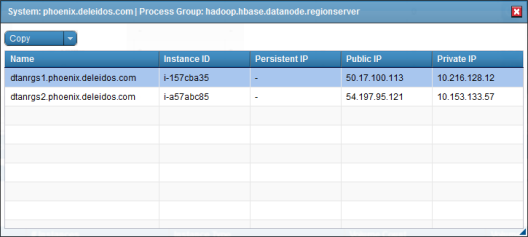
1. Click the System name you want to check on. If a node is down, a warning will appear in the message box, such as:



1. The system status report appears in the bottom panel, listing system resources which are currently running in the cloud. Specific information about each node appears on the Process Groups tab:

| Column | Explanation |
| --- | --- |
| Group Name | A process group represents a choice of components or applications that you made in System Builder (such as a web app or a data sink) that is currently configured in your system. A process group represents a collection of instances in each category (e.g., all instances of a JMS internal queue). |
| # Instances | How many instances of the process are currently running |
| Instance Type | The size of each node, as defined by Amazon's node instance types (small, medium, large, Xlarge) |
| Volume Count | How many volumes are currently consumed by a process group |
| Volume Size | How large each volume is, in gigabytes |
| Auto Scaling? | Whether or not a process is self-scaling |

1. Double-click on a process Group Name to see a list of instances and IP addresses within that group. Note that this information is read-only:



1. The Configurations tab lists all of the system's configuration versions. You can Delete  any configuration file that is not currently in use; you cannot delete the active configuration file.
2. Click Refresh  to update the status report.

To edit system parameters, See " Process group parameters" on page **Error! Bookmark not defined.**

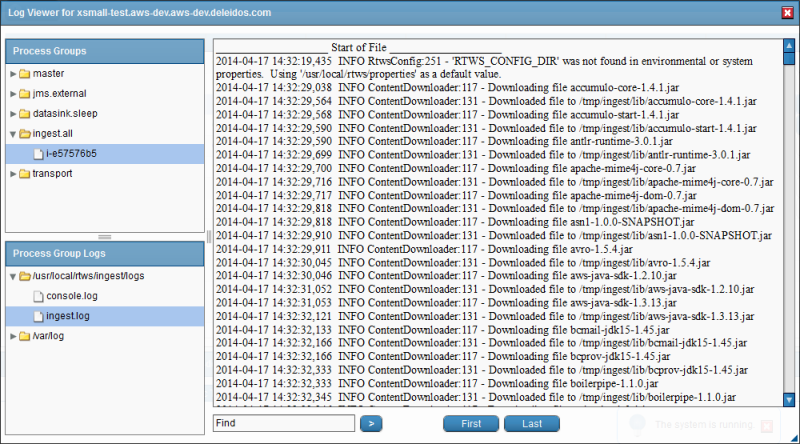
Checking log files

DigitalEdge offers a logging and audit framework which includes the generation, review, protection, and retention of logs across the virtualization, operating system, application, and data layers. The framework also provides customizable logging levels.

Each instance has its own set of log files. You can access all the DigitalEdge log files from one convenient place in the Management Console.

1. In the Management Console, click the View Logs icon  on the line representing the system you need to work with. The Log Viewer appears, with information in three panels:

* Process Groups: Lists the process groups and nodes that have started up and have log files
* Process Group Logs: Lists the folders and log file names on each node
* Log Content: Provides a dump of a selected log file



1. Expand any Process Group, such as:

* master
* jms.external
* datasink.lucene
* ingest.all
* transport

to see a list of nodes assigned to each process group, such as:

* ingest.all
  + i-e57576b5

1. Click on a node to view the Process Group Logs on that node. The hierarchical list includes directory paths and log file names, such as:

* /usr/local/rtws/ingest/logs
  + console.log
  + ingest.log

1. Click on a log file name to view its content. Lengthy files are paged in the Log Viewer; use the vertical scroll bar or click  and  to move through a file. You can also search for a string and find each next occurrence . Note that search strings are case sensitive.

The master node controls the initialization process for all other DigitalEdge nodes. The master.log file contains much of the information associated with system initialization.

For example, the following commonly accessed logs are useful when troubleshooting:

| Troubleshooting Area | Log Files |
| --- | --- |
| Master node and system initiation | usr/local/rtws/master/master.log |
| Startup problems for individual nodes | var/log/rtws\_start.log |
| Ingest issues | usr/local/rtws/ingest/logs |
| Data sink tracking | usr/local/rtws/ingest/logs   * console.log - data recorded before the data sink runs * ingest.log - processing details while the data sink runs * metrics.log - statistics reported while the data sink is running   var/log/hive - data sink installation information, exposing what is running on a specific VM |
| Webapps | usr/local/jetty/logs |

You can also trace the status of unprocessed records in the Dead Letter Queue (DLQ). See "Checking unprocessed records" on page 190.

Checking unprocessed records

When a record fails to be parsed or processed, it goes to the Dead Letter Queue (DLQ). You can view the DLQ to determine the problem(s) so you can edit and resubmit failed records.

1. Access the Data Transfer Utility:

https://default.<domain>/tenantconsole/docs/dtu/

1. Run the DTU:

Java –jar DataTransferUtility.jar

1. Click the DLQ tab (or select Window > DLQ). You can view the unprocessed records in the DLQ.
2. To reprocess records, you must edit and resubmit them. You can do this by copying records from the DLQ to another file, editing them, and resubmitting them to DigitalEdge.

Checking system metrics

The System Monitor visually depicts system activity and resource scaling. You can monitor processes as they auto-scale up and down and as resource utilization changes.

The System Monitor is a dynamic console tool. The graphs change on-screen, reflecting the most current system status. You can monitor system health and potential problems in real time, including such items as:

* Ingest data flow rate
* Record processing backlog
* Alerting engine throughout
* Resource consumption and auto-scaling
* Data storage usage
* User application activity

System Monitor access

1. Access the Management Console.
2. Go to the Systems page.
3. Click on the Monitor System icon  on the row representing the System you wish to monitor. Or, click the System Monitor Tool.  The System Monitor appears on a new browser page.

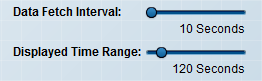
The System Monitor presents a dynamic picture of several key performance factors. There are two groups of graphs:

* Overview graphs
* Detail graphs

Controls

Graph controls at the top and bottom of the screen change the data views:

* Monitoring: Click Monitoring to select the domain name of a different system to monitor
* Show Settings: At the top right of the screen, click the Show Settings icon  to view settings that control the look and performance of all graphs:



* + Data Fetch Interval: Use this slider to set the number of seconds between refreshes and to define the intervals listed on the X axes
  + Displayed Time Range: Use this slider to set the maximum number of seconds depicted on the scales of the X axes
* Next Data Fetch: At the bottom of the screen, the refresh counter displays the number of seconds left before the screen is refreshed. 

Overview graphs

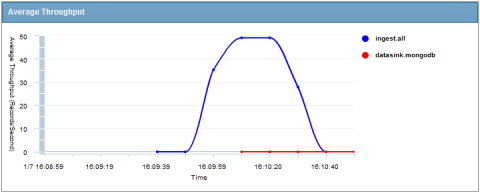
Click the Overview tab to view the basic system graphs simultaneously:



These graphs summarize metrics across all process groups:

* Average Throughput: The rate at which ingested records are fed into the system
* Queue Size: The number of records that were fed into the system but which are not yet processed
* Number of Instances: The number of instances that are automatically allocated and consumed for each process
* Storage Utilization: The percentage of allocated space that is currently in use for each data sink which is an EBS (Amazon Elastic Block Storage) volume database

Monitor data flow: Average Throughput graph

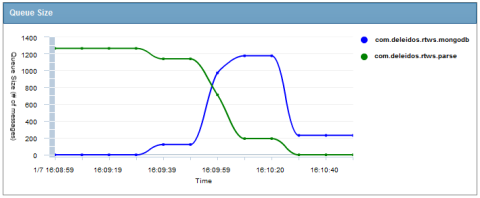


The Average Throughput graph illustrates the rate at which records are ingested and processed in the system (records per second). Throughput time is clocked from the second a record enters the system to the time it is ready for querying (including transport from a data source into the JMS external queue, ingest processing, normalization, enrichment, indexing, and storage). Each line represents records from one process. The legend lists which color line represents each processor.

Use the Average Throughput graph to:

* Check how data flow is affecting overall system performance
* Make sure throughout is not sitting at zero, indicating an idle or problematic system
* Check for expected throughput fluctuations based on historical knowledge of peaks and valleys in data feed activity
* Determine how long (on average) it takes to process a record

Monitor queue sizes: Queue Size graph



The Queue Size graph illustrates how many messages have been fed into a specific processor but which have not yet been processed. This is your processing backlog, which obviously should be as low as possible. This chart monitors different processing queues (identified in the legend on the right), such as:

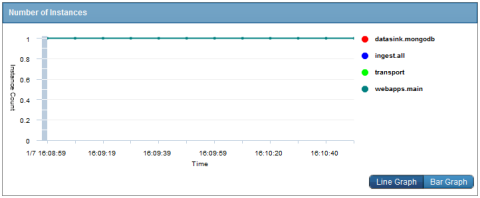
* DLQ: The dead letter queue, where records that could not be parsed are saved
* Filter: The alerting engine filters records for potential alerts and notifications
* Index: Lucene indexed records in preparation for searching
* Parse: The processing pipeline parses, normalizes, and enriches records

Use the scroll bar  at the bottom of the graph to scroll through the timeline on the x axis. Note that the scroll bar only appears if the timeline doesn't fit into the width of the graph/browser window.

Use the Queue Size graph to:

* Make sure the backlog queues are as low as possible
* Determine if processing (Average Throughput) is keeping up with queue sizes

Monitor resource scaling: Number of Instances graph



The Number of Instances graph depicts how many instances of each process type are allocated at any given time. The graph changes as processes scale up during peak activity and scale down as demand decreases. Each process is represented by a separate color line on the graph; the legend on the right is keyed to specific processors. The ingest.all processor is probably the most important process to monitor on this chart; the graph tells you if ingest auto-scaling is keeping up with throughput.

Use the Number of Instances chart to:

* Monitor fluctuations in resource consumption
* Decide if ingest processing is allocated sufficient resources
* Determine if indexing (the Lucene processor) is keeping up with ingest
* Verify that user activity (searching via webapps) is not affecting system performance
* Determine if a processor is down

Colors are assigned to specific services and synced across all charts.

You can toggle this graph between a Line Graph and a Bar Graph view:



In the Bar Graph view, you can set the Noise Filter percentage to specify how frequently the graph should be refreshed:

* 0% = Show all changes to allocated instances (i.e., the noise filter is off); this is the default setting
* 100% = Don’t show any changes to allocated instances (a static graph)
* n% = Show increases or decreases in the number of allocated instances only when the delta reaches n% of the total number of instances; you will not see a change in the graph every time the number of instances increases or decreases

Monitor storage usage: Storage Utilization graph

The Storage Utilization chart depicts the percentage of allocated space that is currently in use for each data sink that is an EBS (Amazon Elastic Block Storage) volume database (e.g., Lucene, Hadoop). Each data sink is represented by a separate color line on the graph; the legend on the right is keyed to specific data sinks. This graph is populated when you implement a search app.

Most data sinks are auto-scaling and dynamically expand when a usage threshold is reached. But some data sinks are not auto-scaling and must be monitored for space utilization; in that case, you must take action to allocate more storage when a critical threshold is reached.

Use the Storage Utilization graph to:

* Determine if storage is reaching a critical point

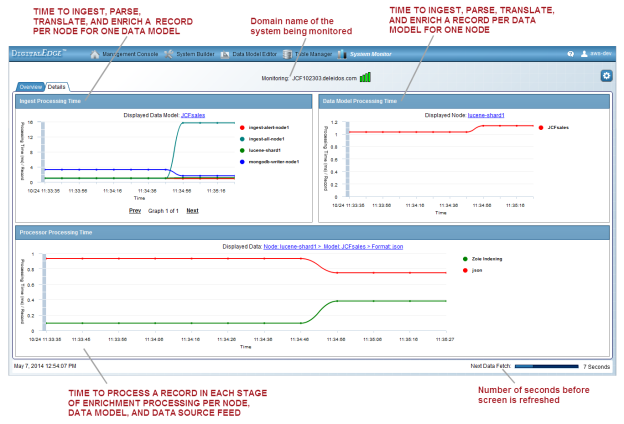
You can toggle this graph between a Line Graph and a Bar Graph view.

In the Bar Graph view, you can set the Noise Filter percentage to specify how frequently the graph should be refreshed:

* 0% = Show all changes to storage utilization (i.e., the noise filter is off); this is the default setting
* 100% = Don’t show any changes to storage usage (a static graph)
* n% = Show changes in storage utilization only when the delta reaches n% of the total amount of currently utilized storage; you will not see a change in the graph every time storage use increases or decreases

Detail graphs

Click the Details tab to access and drill down into the processing time graphs. These graphs help diagnose problems with slow throughput rates:

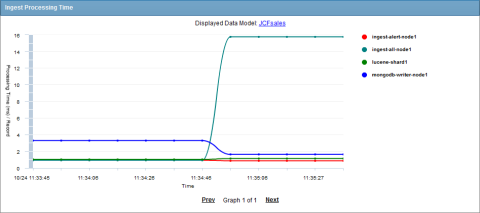


These drill-down graphs break out metrics by a specific processor type; filtered by data model, node, and data source:

* Ingest Processing Time: The time it takes (in milliseconds) to ingest, parse, translate, and enrich a record per node for a specified data model
* Data Model Processing Time: The time it takes (in milliseconds) to ingest, parse, translate, and enrich a record per data model for a specified node
* Processor Processing Time: The time it takes to process a record in each stage of enrichment processing per node, data model, and data source feed

If the Details graphs do not display any data, click the Monitoring control and click Refresh several times.

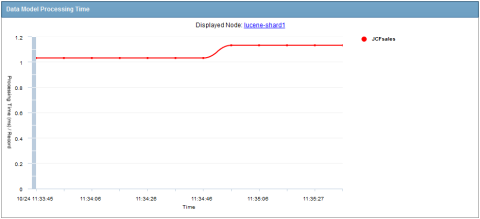
Monitor ingest per node: Ingest Processing Time



This graph depicts the time it takes to ingest a record, in milliseconds. Processing time includes ingest, parse, translate, and enrichment stages. Times are reported per node (one line per node, keyed to colors in the legend), for a selected data model. Click the Displayed Data Model graph option to select one data model to graph. Scroll through the graph with Prev and Next to compare ingest processing times across nodes.

This graph also charts processing time by data sink.

Monitor ingest by data model: Data Model Processing Time



This graph displays the time it takes to ingest a data model record, in milliseconds. Processing time includes ingest, parse, translate, and enrichment stages. Click the Displayed Node graph option to view statistics from one specific node. Each line represents records from one data model, keyed to colors in the legend.

This graph also charts processing time by data sink.

For example, use the Ingest Processing Time and Data Model Processing Time graphs to diagnose problems such as:

* If one line on the Ingest Processing Time graph is running slow, drill down to that particular node/data model line in the Data Model Processing Time graph for comparison. Ingest may be slow if you have a large dimension table and a cache setting that is too small to efficiently process the data. You may have to configure a larger cache or restructure the dimension table to improve processing times.

Monitor processing stages: Processor Processing Time



This graph displays the time is takes to process a record in each stage of processing. Each color line (identified in the legend on the right) represents a different processing stage (input, parsing and translating, enrichment, and data sink processing). Click the Displayed Data option to drill down through the data by: node, data model, and data source feed.

Logging out of the System Monitor

When you are done with a System Monitor session, you can leave the console running or log out.

To log out of the System Monitor, click your user icon/name in the upper right corner. Select Sign Out.

1. <https://github.com/deleidos/digitaledge-platform/> [↑](#footnote-ref-1)
2. <https://gitlab.com/connectedvehicles/fedgov-cv-documents/blob/master/CVCS%20Configuration%20Report%2020170912.xlsx> [↑](#footnote-ref-2)
3. <https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Operations_Guide.pdf> [↑](#footnote-ref-3)
4. <https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Configuration_Guide.pdf> [↑](#footnote-ref-4)
5. <https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Ingest_Pipeline_SDK.pdf> [↑](#footnote-ref-5)
6. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html> [↑](#footnote-ref-6)
7. <https://github.com/deleidos/digitaledge-platform/raw/1.2.1/docs/Configuration_Guide.pdf> [↑](#footnote-ref-7)
8. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-instances-status-check_sched.html#viewing_scheduled_events> [↑](#footnote-ref-8)
9. <https://aws.amazon.com/documentation/ec2/> [↑](#footnote-ref-9)
10. <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-creating-snapshot.html> [↑](#footnote-ref-10)
11. <https://cvcs.samanage.com/> [↑](#footnote-ref-11)