DigitalEdge™

Configuration Guide

Version 1.2.1

July 2014

leidos

DISCLAIMER OF WARRANTY AND LIMITATION OF LIABILITY

The Software accompanying this Documentation is provided with the Limited Warranty contained in the License Agreement for that Software. Leidos, its affiliates and suppliers, disclaim all warranties that the Software will perform as expected or desired on any machine or in any environment. Leidos, its affiliates and suppliers, further disclaim any warranties that this Documentation is complete, accurate, or error-free. Both the Software and the Documentation are subject to updates or changes at Leidos' sole discretion. LEIDOS, ITS LICENSORS AND SUPPLIERS MAKE NO OTHER WARRANTIES, WRITTEN OR ORAL, EXPRESS OR IMPLIED RELATING TO THE PRODUCTS, SOFTWARE, AND DOCUMENTATION. LEIDOS, ITS LICENSORS AND SUPPLIERS DISCLAIM ALL IMPLIED WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, USE, TITLE, AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. In no event shall Leidos, its affiliates or suppliers, be liable to the End User for any consequential, incidental, indirect, exemplary, punitive, or special damages (including lost profits, lost data, or cost of substitute goods or services) related to or arising out of the use of this Software and Documentation however caused and whether such damages are based in tort (including negligence), contract, or otherwise, and regardless of whether Leidos, its affiliates or suppliers, has been advised of the possibility of such damages in advance. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAWS, END USER ACKNOWLEDGES AND AGREES THAT LEIDOS AND ITS AFFILIATES AND SUPPLIERS IN NO EVENT SHALL BE RESPONSIBLE OR LIABLE TO THE END USER FOR ANY AMOUNTS IN EXCESS OF THE FEES PAID BY THE END USER TO LEIDOS. LEIDOS SHALL NOT BE RESPONSIBLE FOR ANY MATTER BEYOND ITS REASONABLE CONTROL.

LEIDOS PROPRIETARY INFORMATION

This document contains Leidos Proprietary Information. It may be used by recipient only for the purpose for which it was transmitted and will be returned or destroyed upon request or when no longer needed by recipient. It may not be copied or communicated without the advance written consent of Leidos. This document contains trade secrets and commercial or financial information which are privileged and confidential and exempt from disclosure under the Freedom of Information Act, 5 U.S.C. § 552.

TRADEMARKS AND ACKNOWLEDGMENTS

Private installations of DigitalEdge are powered by Eucalyptus®.

Public cloud installations of DigitalEdge are powered by Amazon Web Services™.

The following list includes all trademarks that are referenced throughout the DigitalEdge documentation suite.

Adobe, Flash, PDF, and Shockwave are either registered trademarks or trademarks of Adobe Systems Incorporate in the United States and/or other countries.

Amazon Web Services, AWS, Amazon Elastic Compute Cloud, Amazon EC2, EC2, Amazon Simple Storage Service, Amazon S3, Amazon VPC, Amazon DynamoDB, Amazon Route 53, the "Powered by Amazon Web Services" logo, are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.

Apache, Archiva, Cassandra, Hadoop, Hive, HBase, Hue, Lucene, Maven, Apache Phoenix, Solr, Zoie, ActiveMQ are all trademarks of The Apache Software Foundation.

ArcSight is a registered trademark of ArcSight, Inc.

CAS is copyright 2007, JA-SIG, Inc.

CentOS is a trademark of the CentOS Project.

Cloudera is a registered trademark of Cloudera, Inc.

CloudShield is a registered trademark of CloudShield Technologies, Inc. in the U.S. and/or other countries.

CTools are open-source tools produced and managed by Webdetails Consulting Company in Portugal.

Drupal is a registered trademark of Dries Buytaert.

Elasticsearch is a trademark of Elasticsearch BV, registered in the U.S. and in other countries.

Eucalyptus and Walrus are registered trademarks of Eucalyptus Systems, Inc.

Firefox is a registered trademark of the Mozilla Foundation.

The Groovy programming language is sustained and led by SpringSource and the Groovy Community.

H2 is available under a modified version of the Mozilla Public License and under the unmodified Eclipse Public License.

Hybridfox is developed and maintained by CSS Corp R&D Labs.

JUnit is available under the terms of the Common Public License v 1.0.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft, Windows, and Word are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

MongoDB and Mongo are registered trademarks of 10gen, Inc.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Pentaho is a registered trademark of Pentaho, Inc.

PostgreSQL is a trademark of The PostgreSQL Global Development Group, in the US and other countries.

PuTTY is copyright 1997-2012 Simon Tatham.

Sonatype Nexus is a trademark of Sonatype, Inc.

Tableau Software and Tableau are registered trademarks of Tableau Software, Inc.

Twitter is a registered trademark of Twitter, Inc.

All other trademarks are the property of their respective owners.

CONTACT INFORMATION

Leidos Franklin Center
6841 Benjamin Franklin Drive
Columbia, Maryland 21046

Email: DigitalEdgeSupport@Leidos.com

DigitalEdge Technical Support: 443-367-7770

DigitalEdge Sales Support: 443-367-7800

To submit ideas or feedback: https://www9.v1ideas.com/digitaledge/welcome

Contents

Chapter 1: Introduction

DigitalEdge is a highly configurable software platform providing real-time analytics of big data in motion for cyber-security. The system is flexible enough to be configured as:

- a simple system that is quick to configure and stand up
- a large and complex system that requires several people to design the configuration
- an iterative design that can be changed and saved in multiple versions; this is particularly useful during a trial or test phase

For simple systems, configuration tasks can be completed by one individual in a short amount of time. Large systems may require a team of professionals who work together to plan, design, configure, and implement the system. Data Specialists would have the important tasks of defining the data model, specifying the data sources for inputs, mapping data sources to the input model, and defining the required enrichments. Data Analysts would define filtering rules for alerts and interpret those alerts for potential fraud. DigitalEdge Administrators would set up the system components, pipeline, and data sinks; they would also manage and monitor the system daily.

As a DigitalEdge Administrator, think of yourself and your team as the architects of one or more DigitalEdge software systems. Before you can build the system, you should take the time to carefully plan your system, to design the inputs and data model, and to configure the components that will serve as the foundation of the system. DigitalEdge provides plug-in architecture for all its major components such as transports, parsers, enrichment processors, and data sinks. You can build a new system from pre-existing components, customize those components for your own site-specific needs, or create components from scratch. Data modeling is key to a successful configuration.

These instructions will guide you through planning, configuring, and building your DigitalEdge system. Before plunging in, be sure to read the *Overview Guide*; it describes the system architecture, concepts, and terminology. These instructions assume you are familiar with those concepts.

Product documentation

DigitalEdge is a complex big data platform. The system comes with a complete set of documentation in PDF and HTML5 formats to help you master DigitalEdge:

Document	Use	Audience
Overview Guide	Basic information about the DigitalEdge platform, including architecture, concepts, and terminology; a must-read before working with any aspect of DigitalEdge	Anyone working with DigitalEdge in any capacity
Configuration Guide	Instructions for defining data models and building processing pipelines	Data Specialists, DigitalEdge Administrators
Operations Guide	Daily administration information, covering	DigitalEdge

Document	Use	Audience
	monitoring, managing, and modifying the platform	Administrators
DigitalEdge SDK Guide	Reference for building custom plug-in components	Developers
Alerts API Guide	Reference for specifying data triggers and notifications for an alerting capability	Developers
Search API Guide	Reference for providing search services on a Lucene data sink node	Developers

Typographical conventions

The following style conventions are used throughout this documentation:

Type of Information	Style in Documentation
Code, commands, filenames	`code`
Cross references	Click to see this topic
Emphasis	*important point*
Hyperlinks	Click to go to this site
Notes, warnings, tips	
References to other documents	*Document Title*
Sample code blocks	`code`
Troubleshooting issue or problem	
Troubleshooting solution	
User input	*Italics*
User interface labels and controls	**Bold**
Variables	*<change-this-name>*

Chapter 2: Planning a System

The planning and design stage is arguably the most important step in implementing DigitalEdge. During this phase, you will gather all the details you need to configure DigitalEdge. You must identify and understand:

- Data sources and parsers
- Fields/columns in each data table
- Formats and transport mechanisms for each data source
- Enrichment tables that will enhance your data feeds with additional information
- Data sinks that will store your data
- Applications that your users will need

In addition, you must create a data blueprint for the new system by defining an *input model* that will serve as the normalized model for all incoming data. For more complex systems, you can iterate the design phase, changing and enhancing the data model as you test the system configuration.

See "Forms for Designing a System" on page 151 which includes several forms that can help you spec out a new DigitalEdge system.

Understanding and designing the data model

The most important design task is to define a data model. A data model is the foundation for defining and managing DigitalEdge data. It defines the input sources, the internal normalized data model that DigitalEdge uses to organize data, the parsing techniques used to transform incoming data into the internal format, and the sources and types of dimensional data that will be used to enhance or further describe the data (e.g., replacing a ProductID with a ProductName from a supporting relational table). The data model specifies three components:

- **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 model; this is the data model that DigitalEdge uses internally
- **Data sources**: specifications for parsing the incoming data and mapping each field to the input model
- **Enrichments**: rules for enhancing the data with dimensional data for additional context and meaning

Here is what you will need to define the internal record format for DigitalEdge, the input model:

- A name for the input model
- A name for each field in the internal DigitalEdge model
- The data type of each field

- String
- Number
- Date/Time
- Object
- Array

The input model will include fields that are common to all data sources, and it may also include fields that are specific to just one or more data sources.

Specifying Data Sources and Parsers

Database Administrators and Data Specialists on your team will most likely head up the task of identifying and specifying data sources for DigitalEdge input. Data sources can originate from virtually any data store or format in your organization: a data stream feeding DigitalEdge, real-time transactional records, multiple relational databases, web sites, RSS feeds, SIEM files, log files, network activity streams, email archives, etc. Data Specialists should have a detailed understanding of each incoming data source, its content, format, structure, and transport mode. The primary data feeds will contain the fact records that are central to your data model.

Here is the information you will need to configure the input data sources:

- Name and location of each incoming data source that will provide data for DigitalEdge
- File format, and the parser type required to extract data fields for input into DigitalEdge. If you need a parser that is not included in the core release, your developers can create a custom parser for your site.
    - **Binary**: a user-configurable parser that is used to extract fixed fields from byte strings in Base 64 encoded binary files
    - **CEF**: a parser that uses the Common Event Format from ArcSight®, an open standard for logging security-related information in a common event log format
    - **CSV**: a commonly used parser to extract comma-separated values from plain text files; this parser is user-configurable
    - **DNS PCAP**: a parser that reads and extracts DNS packets, prefixed with DigitalEdge data
    - **Email**: a parser that works with the RFC 822 ARPA email text format (which specifies the structure of email messages, including attachments, to and from fields, etc.)
    - **EXIF**: a configurable parser that works with the Exchangeable Image File format used for handling image and sound files from digital cameras
    - **JSON**: a configurable parser that extracts data from text formatted in the JavaScript Open Notation standard
    - **Libpcap**: a parser that works with packet captures in the UNIX Libpcap library

    - **Log file**: a 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**: a parser that reads and extracts Simple Network Management Protocol packets, prefixed with DigitalEdge data; supports SNMPv1 or SNMPv2
    - **Unstructured file**: a beta parser that works with unstructured files such as Word, Excel, and PDFs to extract content and metadata
    - **XML**: a configurable parser that extracts data from the Extensible Markup Language format
- Field delimiter type:
    - Comma
    - Pipe
    - New line

For each incoming data field, you must know the original source specifications for the parser:

- Field name
- Field data type:
    - String
    - Number
    - Date/Time
    - Object
    - Array
- Translation rules for mapping a field into the DigitalEdge input model:
    - JSON representation
    - Conversion requirements (date conversions)
    - Constant value
    - Processing rules (could be specified in a script that you run on the data source)
- If a field should be enhanced with data from another table or source, identify the dimension table name and column(s) that will provide the enrichment data
- Input date format

Getting data in with transports

Each data source must be associated with a transport mode to get data into DigitalEdge. Transports simply move data into DigitalEdge wrapped in JMS messages; they do not convert data to JSON.

Ingest supports several secure automated transport mechanisms, including file-based transfer protocols, streaming TCP and UDP connections, external database queries, and unstructured documents. For example, the TCP transport establishes a TCP socket listening to a port, converts it

to JMS messages, and pushes the data into the JMS external node. Each parser can be assigned to a different transport type of your choice. If you need a transport that is not included in the core release, your developers can create a custom transport for your site.

DatabaseWatcherTransportService

The Database Watcher transport is a specialized polling service that gets data from a database and pulls it into DigitalEdge by running an SQL select query against any database. The database can be queried regularly, starting at the point where the query last left off. An S3 bucket is used to store a backup copy of the data file.

DirectoryCrawlerTransportService

The Directory Crawler beta transport is similar to the Directory Watcher transport, processing data in a local or remote file system. But it also decompresses zipped files and processes files that match wild card patterns.

DirectoryWatcherTransportService

This transport is a polling service similar to the PollingS3TransportService. This transport watches a local or remote file system rather than an S3™ bucket. Use the Directory Watcher transport when you have data files that you do not want to move to S3.

HiveTransportService

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 for indexing. You can optionally create an SQL select query to run against Hive to pull out a subset of data.

JMSBridgeTransportService

This service copies data directly from your corporate JMS server to the DigitalEdge JMS server without a transport. If your shop has multiple JMS brokers, this is probably the easiest service to use. The JMS Bridge Service pushes data directly onto a JMS queue without a transport. For example, there could be two JMS servers in play: 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 internally, you should choose another transport from the DigitalEdge repository.

MongoDBTransportService

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. You can optionally create an SQL select statement to run against MongoDB to pull out just a subset of data.

PcapSnifferTransportService
This transport captures and splits pcap packets on a specific network interface. You can optionally filter out data and pull selected relevant data. The pcap transport is often used with the SNMP PCAP parser.

S3FileTransportService
The S3 transport service is a good choice when you have one or more static files, such as legacy data, to get into DigitalEdge. This S3 transport can be configured two ways. As a one shot event, it pulls data from an Amazon's S3™ (Simple Storage Service™) file and pushes it to the DigitalEdge JMS queue. Or, you can configure the transport to poll an S3™ bucket regularly at a set time interval. Choose this option when you have an existing system that is generating large files and you are adding DigitalEdge to the system as the big data processor and analyzer. Set up buckets for data feeds from several data sources, one bucket per data type. Use the polling transport to check the bucket for new data every several minutes, hours, or once a day, depending on your time-to-availability requirements.You can configure this transport to either save the file or delete it immediately after processing.

TCPTransportService
This is a commonly used transport, to read data in a TCP stream. The TCP socket listens to a port, converts the data to a JMS message, and puts it on the JMS external queue.

TwitterFilterTransportService
This transport gets Tweets from the Twitter feed based on criteria that you define via the Twitter Search API. You can search for keywords and/or Twitter usernames. This is the most flexible and commonly used transport of the three Twitter transports.

TwitterRESTTransportService
This transport follows the Tweets of one Twitter user. The transport uses the Twitter REST API.

TwitterSampleTransportService
This transport selects a random sample of Tweets that you are allowed to read in the Twitter feed. The transport uses the Twitter Search API.

UDPTransportService
This frequently used transport captures network packets in a UDP stream, converts the data to JMS messages, and sends them to the JMS external node.

URLTransportService
This transport reads the contents of a URL and puts it on the JMS input queue. Use this transport to pull data from an RSS feed or from any service that pulls resources from another organization or data source.

Enhancing the data with enrichments

Data sources rarely stand on their own. One of DigitalEdge's most important features is dimension table enrichments. Dimensional enrichment associates a secondary data table with your primary data source, to add data to each record that better describes a data element. For example, if you were feeding DigitalEdge from a point of sales transaction stream, you could enhance the primary transaction record with product names (replacing product IDs), customer names (to replace customer IDs), mailing addresses, etc. The enhancement data comes from dimension data tables that you specify as part of the DigitalEdge data model. For each dimension table, you also identify the fields in the input model that will be enhanced with the dimensional data.

DigitalEdge also offers generalized or algorithmic enrichments that do not require dimension tables as the source for enhancement data. Generalized enrichments use data from standard, public sources. For example, to add the geographic location of an airport, DigitalEdge converts standard 3-letter airport codes to latitude/longitude coordinates. If you need an enrichment that is not included in the core release, your developers can create a custom enrichment for your site.

Enrichments involve some of the most complex rules that you will define DigitalEdge. They require a thorough understanding of the incoming data, dimension tables, the fields in the input model, and how dimensional data will be correlated with and mapped into input fields.

Dimension table specifications

Inventory each enrichment source (dimension table) for its content, format, and structure. Note how each enrichment source correlates to the main data feed so you can specify translation rules for the data fusion engine.

Here is the information you will need to configure each dimension table:

- Name of dimension table
- Primary key
- Column/Field name to extract
- Data type
    - Varchar
    - Decimal
    - Date
    - etc.
- Nulls allowed?

Enrichment specifications

Once you have defined your data sources, dimension tables, and input model, you can specify the enrichment rules for adding data to the input model:

- Type of enrichment
    - Dimension table enrichment
        - **Fuzzy match**: Beta processor that enriches records with a standardized string based on a fuzzy match lookup on a specified input field
    - Generalized, algorithmic enrichment
        - **IP network**: Generates location data that corresponds to a given IP address; in JSON format. 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 expression that you create against the data
        - **Postal location**: Computes the nearest zip code, city, state, or country from 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
        - **regex_extractor**: Extracts sub-fields from a more complex input field
        - **SQL select enrichment**: Looks up information in an SQL databse with a given query
- Dimension table name
- Enrichment field name
- Input field correlated with the dimensional field
- Input date format

Defining System Components

Once you have created a data model in DigitalEdge, you will build the processing pipeline and system configuration by identifying and assembling all the components in the system architecture. In addition to the data models and transports, you must identify data sinks and user applications.

Choosing data sinks

A data sink is a queue, server, or database that can receive pipeline-processed JSON data to store or post-process for other uses. For example, one data sink can index data for queries, several data sinks store historical or processed data, and the alerting engine data sink filters data for alerts.

DigitalEdge provides many data sink options. Each data sink has its strengths and weaknesses; DigitalEdge plays to the strengths of each one to optimize performance of your repositories. For example, Lucene is particularly good for implementing real time searching on the data. If you need a data sink that is not included in the core release, your developers can create a custom data sink for your site.Here are some guidelines for choosing data sinks for specific uses.

Alerting data sink

Most DigitalEdge systems will be configured to identify possible cybersecurity breaches, situational anomalies, or potential fraud. This data sink filters processed records for alert triggers and sends out alert messages either as email messages or as messages in a JMS topic. The data sink does not store DigitalEdge processed records.

Cassandra data sink

This beta data sink stores data in an Apache Cassandra cluster. Cassandra handles big data across many servers with no single point of failure. The Cassandra cluster is decentralized, with no master node so that each node can fulfill any request. It is very easy to add nodes to a Cassandra configuration.

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 the **Table Manager**>**Batch Tools**>**Import** functionality to upload CSV data into a table
- 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)

Elasticsearch data sink

This beta data sink implements Elasticsearch, an indexing and full text search engine. 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.

External HBase data sink

This data sink stores data in an existing Hadoop/HBase cluster that is external to and not managed by DigitalEdge. If your organization already has applications that use HBase, you can send DigitalEdge processed data to that cluster with this data sink.

External HDFS data sink

This data sink stores data in an existing Hadoop cluster that is not managed by DigitalEdge (compatible with and distributed via Cloudera CDH3uX releases). If your organization already uses Hadoop clusters, you can send DigitalEdge processed data to that cluster by configuring a communication connection with this data sink.

External Hive data sink

This data sink is similar to the Hive data sink, but stores data in an existing Hadoop/Hive cluster that is not managed by DigitalEdge. If your organization already uses Hive clusters, you can send DigitalEdge processed data to that cluster by configuring a communication connection with this data sink.

HBase data sink

The HBase data sink stores processed DigitalEdge data in an HBase database that is managed internally by DigitalEdge. HBase was originally created to handle petabytes of data and is commonly used as a key value store.

Hive data sink

This data sink stores processed data in the Hadoop/Hive environment and is managed internally by DigitalEdge. Hive was originally created to interface SQL databases with Hadoop, so shops that were relational-based could continue to work in an SQL-like environment while interfacing to Hadoop MapReduce technology. Use this data sink to create a Hive cluster that is managed by DigitalEdge.

JSON to JDBC data sink

This data sink maps DigitalEdge JSON objects to a relational database table and writes them to a specified database via JDBC (typically, an external database for additional processing or analysis).

Lucene indexing data sink

If you plan to have your data searchable with the Search or SearchAPI web applications, you *must* add a Lucene data sink to the system. The Lucene data sink uses a copy of the processed DigitalEdge data and builds an inverted index for real-time and near real-time search. The index is optimized for searching. A Lucene index should be configured to 1.5 times the anticipated index size, to accommodate building and merging.

MongoDB data sink

This data sink is a general all-purpose NoSQL data sink that stores DigitalEdge processed JSON data in a MongoDB® database which is managed by DigitalEdge. Note that MongoDB records have a maximum size of 16 MB. MongoDB can be configured in DigitalEdge to scale automatically. Each instance sets up its own copy of MongoDB, no cluster is created. With stand-alone databases, each instance will contain different data; as content comes off the processing pipeline, portions of the data may be sent to different instances. MongoDB is fast, compact, and good to use with external dashboards, reporting packages, or summarized data.

Sleep data sink

This data sink is used strictly for test purposes. It continually reads a record and then sleeps for a specified amount of time. Use it to test for the progress of records through the DigitalEdge pipeline. The data sink does not store or process records.

Selecting User Applications

A primary reason for doing complex configuration of a DigitalEdge data model is to make information available to users in real time for analysis. DigitalEdge provides several user applications as plug-ins, or you can feed DigitalEdge processed records into an application in your shop that provides additional reporting and analytical capabilities.

User apps are configured as specific web apps in DigitalEdge. Web apps include two types:

- Applications that end users will use to search for and analyze data
- REST APIs

For the purposes of initial planning, you only need to identify the types of *user* applications you want to implement in DigitalEdge. If you need an app that is not included in the core release, your developers can create a custom user app for your site.

DigitalEdge user applications include:

- **Alert Controller**: to manage alert criteria, notifications, and subscriptions
- **Alerts API**: to manage alert criteria used to generate notifications of potential threats or fraud
- **DB API**: 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
- **Metrics API**: to collect statistics on system performance
- **Pentaho server**: to provide monitoring dashboards, self-service reporting, and visual representations of your data
- **Phoenix API**: a beta REST API for interacting with the Phoenix SQL query engine on HBase
- **Search**: for real time and near real time full text queries
- **Search API**: to build a custom search capability for your system

To feed DigitalEdge data into a pre-existing on-site application, plan to configure an externally facing data sink such as the Scripting Data Sink or the JMS Data Sink (See "Defining System Components" on page 9 for more information).

Defining alert rules

One of the most important DigitalEdge features is the ability to generate real-time alerts (notifications) to users about a potential threat or fraud in progress. DigitalEdge can place actionable intelligence in the hands of decision makers within seconds of an anomalous event occurring, not days or even hours after the fact. Alerting helps you detect fraud and threats, and respond rapidly to an unfolding situation.

The alerting mechanism in DigitalEdge involves three phases:

- **Define alert criteria**: You specify business rules to define what a threat would be to your organization. For example, a Systems Administrator may want to know when an individual attempts to access a company intranet from an outside IP address.Or a bank may want to know if an employee is frequently servicing transactions on his own account.
- **DigitalEdge analyzes data for potential threats**: DigitalEdge's alert engine attempts to match the alert criteria filter rules against processed records.
- **Notifications are sent in real time of a threat in progress**: If a match is found between a record and the alert filter, a notification can be generated and sent to one or more email addresses, as text messages, to the console, or on a JMS queue.

Once you fully understand the data model that you have designed and configured, you should identify the data conditions that indicate a potential threat. These detection criteria, or alert business rules are defined in the **Alert Controller**. See "Creating Alerts" on page 131.

Example: Planning for multiple data sources and enrichments

Many DigitalEdge deployments are very straight-forward, including one data source, a few dimension tables for enrichment processing, a data model, one or two data sinks, and multiple alerts. However, a common, more complex, scenario involves two or more data streams, one of which is used to enrich the other. This example outlines a plan for configuring DigitalEdge for this situation. By creating two systems, processing times are reduced and data model enrichments are separated for ease of use.

In this case, DigitalEdge will be configured with:

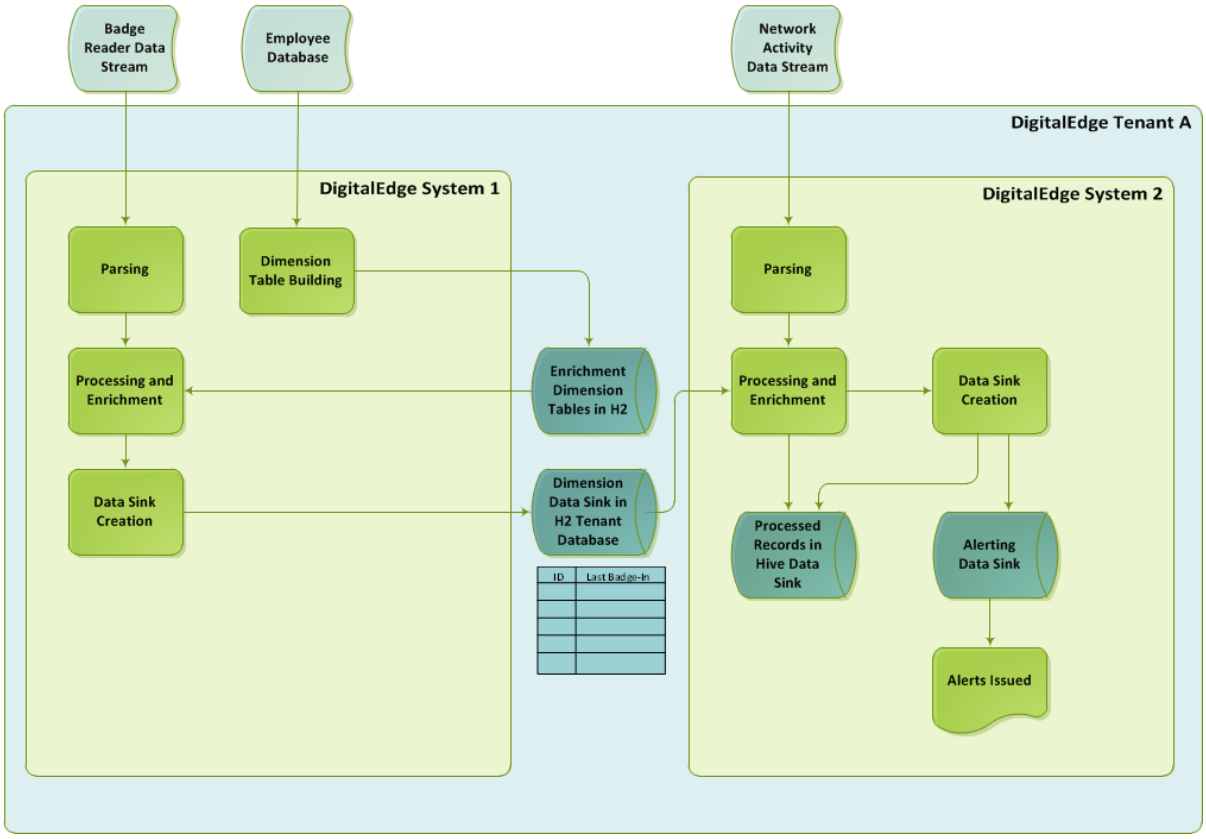
- Two systems
- System #1: One data source, one parser, an interim input model, one enrichment, and a dimension data sink for enriched records (you could substitute the JSON to JDBC data sink for this example)
- System #2: A second data source, a different parser, the final input model, several enrichments, a Hive data sink for processed records, and alerts
- System #2 will get enrichment data from the dimension data sink created in System #1

Consider this example in which an IT department wants to implement a cybersecurity solution. They are concerned that potentially malicious behavior on the network may be tied to one or more employees. They need to determine who is in the building and at their desks when specific network events occur. The data sources they have include:

- The badge reader data stream, which reports the times that employees enter and exit a secure building by scanning their badges
- The network data stream, which lists all network activity throughout the organization but which does not identify users
- The employee database, which correlates employee IDs and badge numbers

The IT department wants to enrich the badge reader data with employee IDs so they can determine who entered and left the building at specific times. They then want to correlate that data with the network log to determine which employees were in the building when suspicious activity occurred.

Here's how their DigitalEdge systems are set up:



System #1

- Data source = Badge Reader Data
- Parser = Log File Parser
- Enrichment = Dimension Table Enrichment to correlate Badge ID with Employee ID
- Dimension table = Employee Database
- Data sink = Dimension Data Sink (in the relational tenant database) to store enriched badge-in information
- Input model = Definition of the records in the dimension data sink, including **tableName** and **tableKey** fields to link to the Dimension Data Sink (See "Step 1: Define an Input Model " on page 22)

System #2

- Data source = Network Activity
- Parser = DNS Parser

- Enrichments = Enriched Badge Reader Data from the Dimension Data Sink in System #1
- Data sink = Hive for storage of processed records
- Data sink = Alerting Data Sink
- Alerts: To notify IT of suspicious network activity
- Input model = Definition of the processed network activity records in the Hive data sink

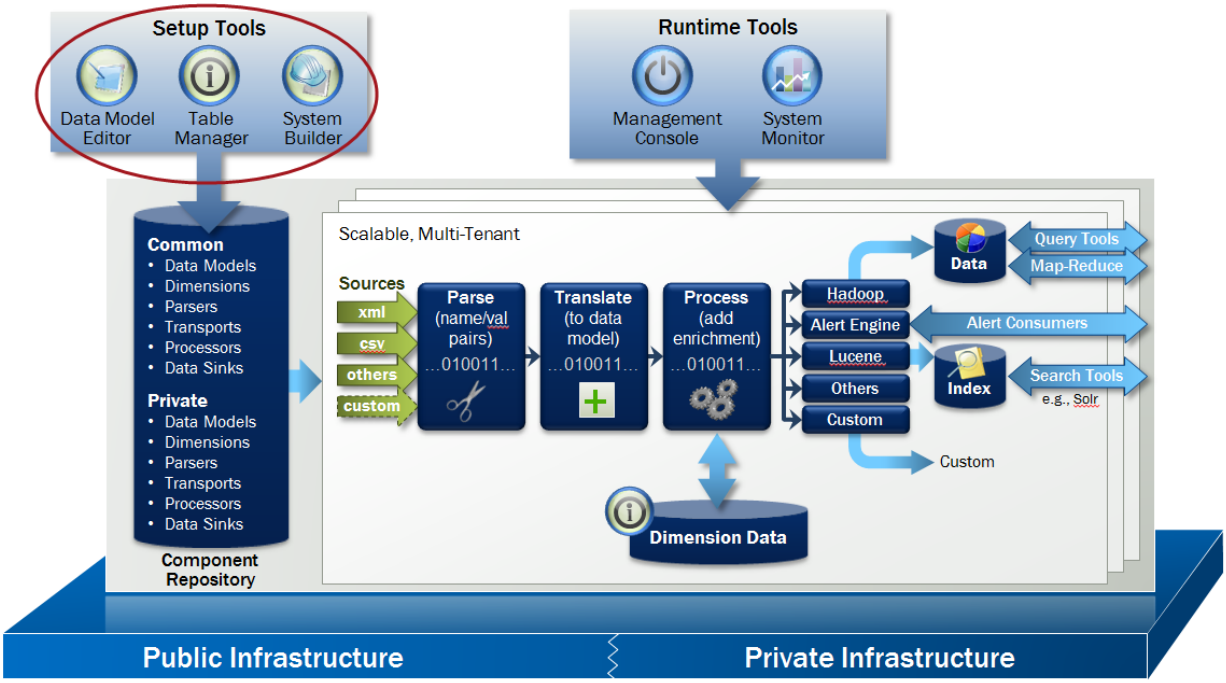
Here's how this configuration works. The Badge Reader Data Stream is transported into System #1 in DigitalEdge and parsed by the Log File Parser. When the records are processed, they are enriched with data from the Employee Database to correlate the Badge ID with the Employee ID. The enriched and processed records are stored in the Dimension Data Sink (in a relational tenant database), to serve as an enrichment data source for System #2. The data model in the system includes special **tableName** and **tableKey** fields to link to the Dimension Data Sink (See "Step 1: Define an Input Model " on page 22).

Next, the Network Activity Data Stream is transported into System #2 in DigitalEdge and parsed by the DNS Parser. The records are processed and enriched with records from the Dimension Data Sink in System #1, to match up last badge-in activity with suspicious network events. Processed and enriched records are stored in Hive, but are also sent through the alerting engine in the Alerting Data Sink to determine if suspicious activity has occurred. When a problematic record is found, it is flagged for notification in an Alert message.

Chapter 3: Tools for Configuring a System

In a DigitalEdge system, raw data enters via automated transport mechanisms. Each data source can be a different format and include different data fields. Data Specialists define an input data model for normalizing and mapping all source data into the system. An ingest processing pipeline is defined for each data format, specifying a parser to use for extracting data, mapping rules for translating source data into the normalized input model, and enrichment processors to enhance the context of the data by fusing dimensional data with fact data. Processed data is then indexed and stored in data sinks. An alerting engine analyzes data and produces automatic notifications about potential threats.

Use DigitalEdge's Setup Tools (Data Model Editor, Table Manager, and System Builder) to configure data ingest, mapping, and processing. These tools provide a user-friendly environment to support complex system configuration and maintenance.



Each Setup Tool draws from the component repository, where all the shared and site-specific components are stored. You can work with pre-existing components as a starting point for configuration activities, or you can create private components from scratch for unique system requirements. DigitalEdge provides all the tools you need to facilitate the configuration process.

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.

The *basic* steps in configuring a DigitalEdge system are:

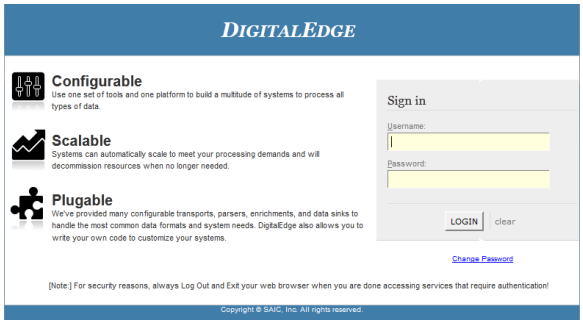
Step	Description	Setup Tool
DEFINE THE DATA MODEL		
Define the input model	Specify the fields that all data sources will be normalized to See "Step 1: Define an Input Model " on page 22	Data Model Editor
Define data sources	Identify data source tables and fields See "Step 2: Define Data Sources, Parsers, and Data Source Mappings " on page 32	Data Model Editor
Specify a parser and map source data to the input model	Specify fields to parse and extract from the data source See " Define the fields in a data source" on page 34	Data Model Editor
	Create translation statements, mapping extracted fields to the input model See " Map data source fields to the input model" on page 35	Data Model Editor
Define enrichments	Specify dimension tables and fields See "Step 3: Define Dimension Tables" on page 51	Table Manager
	Create enrichment definitions to correlate and fuse dimension data with the input model See "Step 4: Define Enrichment Processes " on page 66	Data Model Editor
Create a simple data model	Quickly create a flat data model from CSV or JSON files See "Create a Data Model with the Wizard" on page 80	Data Model Editor > Data Model Creation Wizard
BUILD THE SYSTEM		
Define the system	Name the system and specify system parameters See "Step 1: Define a System " on page 88	System Builder

Step	Description	Setup Tool
Assemble the pipeline	Identify and link components in the pipeline: data models and transports See "Step 2: Add Transports " on page 93	System Builder
Specify data sinks	Specify data stores for indexing, storage, and alerts See "Step 3: Add Data Sinks" on page 108	System Builder
Specify web applications	Select analytical tools for end users See "Step 4: Add Web Apps " on page 124	System Builder
RUN THE SYSTEM		
Start the system	Start up the master node See the DigitalEdge *Operations Guide*, "Managing DigitalEdge from the Management Console"	Management Console

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**.



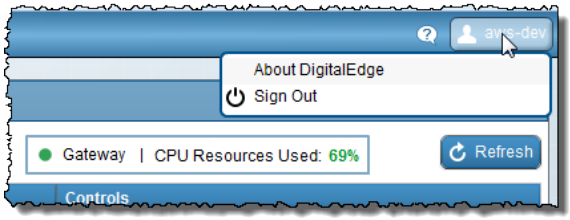
You can access all the DigitalEdge Setup and Runtime tools from the **Management Console**.

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

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**.

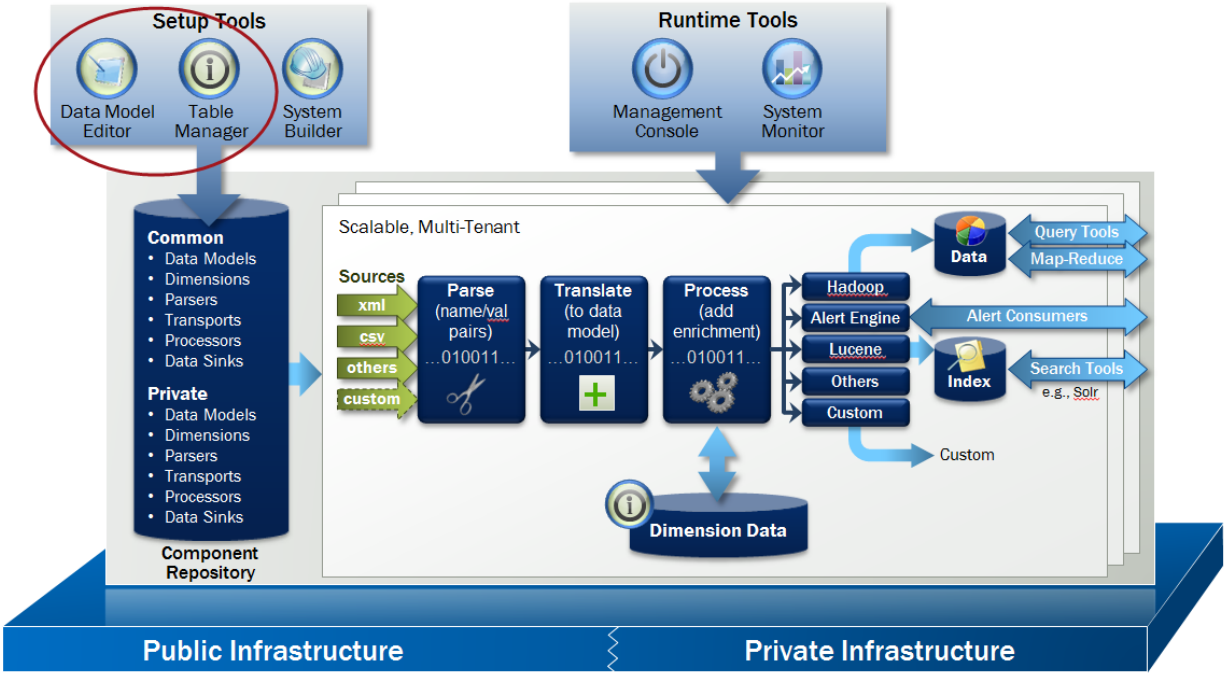


Use the same procedure to log out of any Setup or Runtime UI tool.

When you **Sign Out** of one tool, all open tools are automatically signed out.

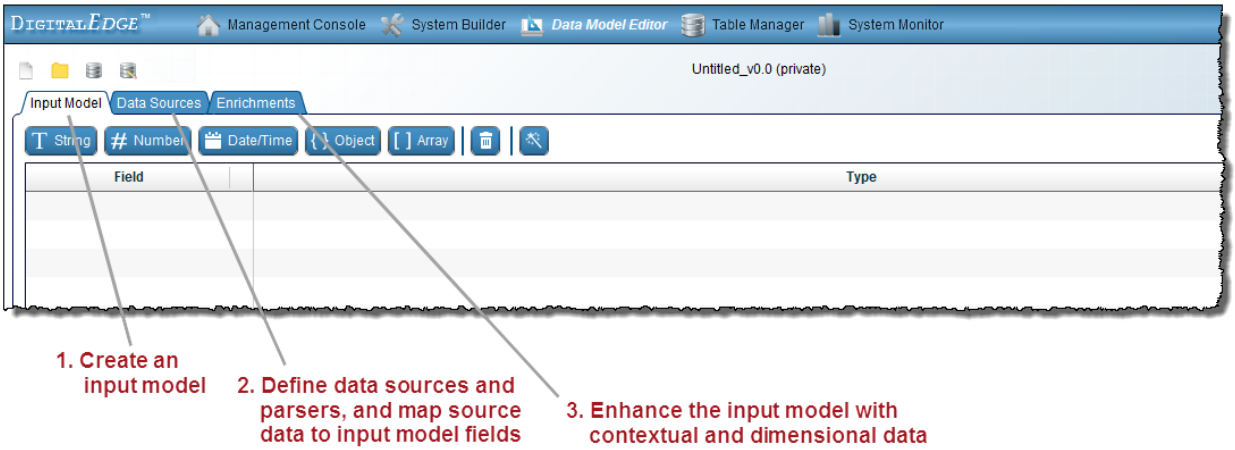
Chapter 4: Defining the Data Model

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:

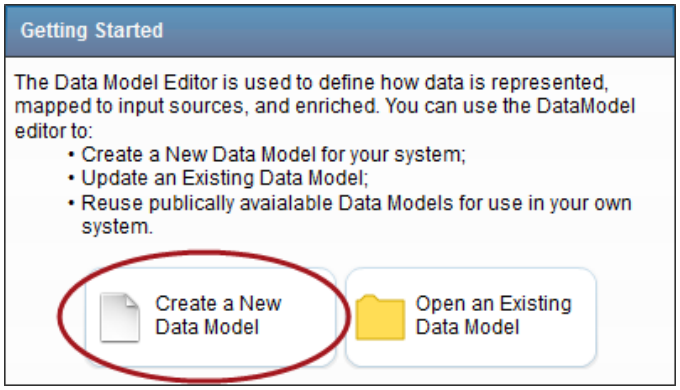
- Create a new input model from completely from scratch. (See " Create a new input model" on page 23)
- Search for and work with an existing input model as a starting point for your system. You can edit a model, delete fields, add fields, and change the specifications to meet your needs. DigitalEdge 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 27)
- Quickly create a simple data model from a CSV or JSON file. (See "Create a Data Model with the Wizard" on page 80)

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.

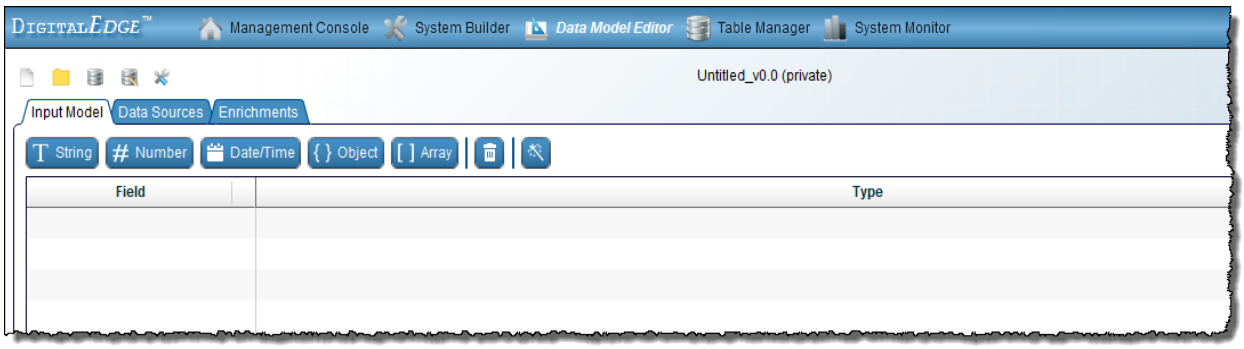
DigitalEdge uses a data model versioning system that lets you keep multiple versions of a data model. Every data record is tagged with its data model name and version, allowing multiple versions to exist in the same data store. This lets you easily migrate data from one version to the next on a timeframe 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**.

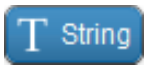


3. The **Input Model** workspace appears with empty **Field** and **Type** columns. You will build a data model in JSON representation, including arrays, objects, and fields.



Create a new field

1. Highlight the field below which the new field should appear.
2. Click a **Type** button:

- **String** 

    a. Enter the **Field Name** using letters, numbers and underscores.
    b. Click **OK**.
- **Number** 

    a. Enter the **Field Name** using letters, numbers and underscores.
    b. Click **OK**.
- **Date/Time** 

    a. Enter the **Field Name** using letters, numbers and underscores.
    b. Click **OK**.
    c. Select a format from the list of **Common Date Formats**, **Common Time Formats**, or **Custom Formats**.
    d. 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.
    e. 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** 

    a. Enter the **Object Name** using letters, numbers and underscores.
    b. 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 Edit Object.

c. Click **OK**.

- **Array** [ ] Array

a. Enter the **Array Field Name** using letters, numbers and underscores.

b. Select the **Element Type** (string, number, or OBJECT).

c. If you selected OBJECT as the **Element Type**, enter at least one field (follow the **Object** instructions).

d. 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 13 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

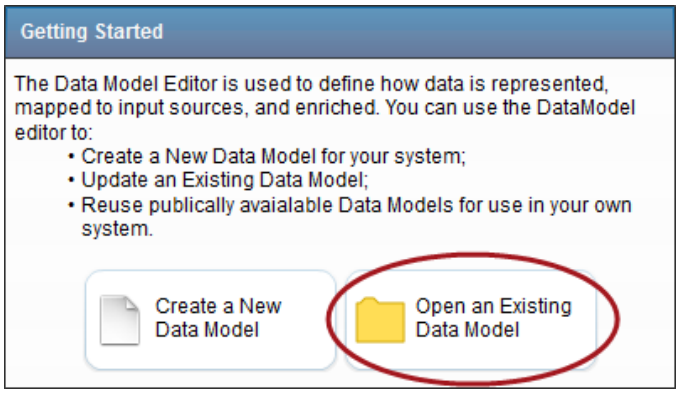
1. Click the **Save** icon on the **Data Model Editor** title bar.
2. In the **Save Data Model As** dialog box, enter a new name for the model in the **Model Name** box.
3. Select a **Major Version** number.
4. Select a **Minor Version** number.
5. Select the private **Area** to keep the data model confidential.
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.

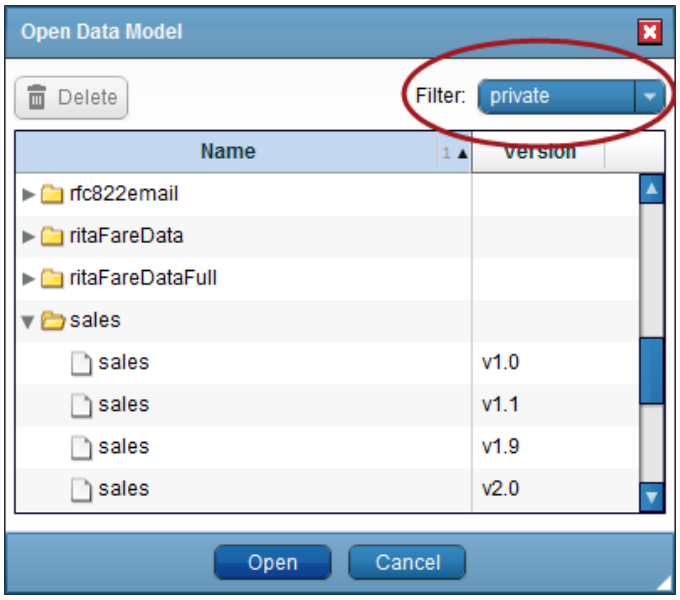
Work with an existing input model

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**.



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

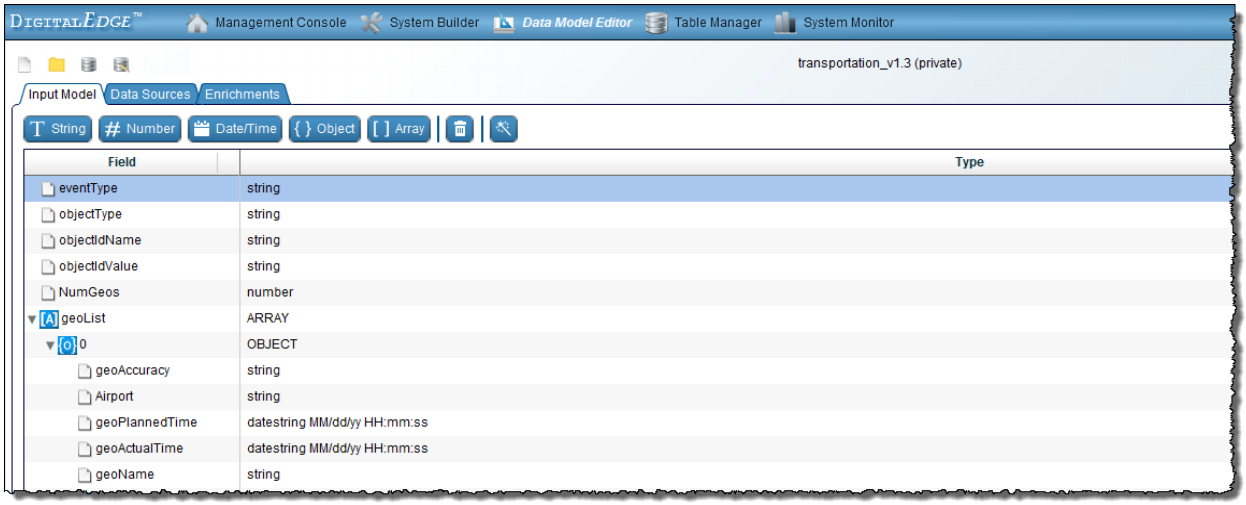


- **Common**: Common data models are pre-defined and supplied with DigitalEdge as samples to help get you started. When you select a common data model, it is copied into your environment as a private data model (for your use only).
- **Private**: Private data models are only resident in your DigitalEdge 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.

4. Expand the data model **Name** to see a list of available data model **Version**s.
5. Select a **Name** or a **Version** and click **Open**.

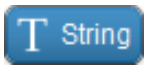
When you select a data model **Name**, DigitalEdge automatically opens the latest **Version** of that data model.

6. 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** 

     a. Enter the **Field Name** using letters, numbers and underscores.
     b. Click **OK**.
   - **Number** 

     a. Enter the **Field Name** using letters, numbers and underscores.
     b. Click **OK**.
   - **Date/Time** 

     a. Enter the **Field Name** using letters, numbers and underscores.
     b. Click **OK**.

c. Select a format from the list of **Common Date Formats**, **Common Time Formats**, or **Custom Formats**.

d. 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.

e. Click **OK**.

- **Object** { } Object

a. Enter the **Object Name** using letters, numbers and underscores.

b. 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 Edit Object.

c. Click **OK**.

- **Array** [ ] Array
  a. Enter the **Array Field Name** using letters, numbers and underscores.
  b. Select the **Element Type** (string, number, or OBJECT).
  c. If you selected OBJECT as the **Element Type**, enter at least one field (follow the **Object** instructions).
  d. 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.

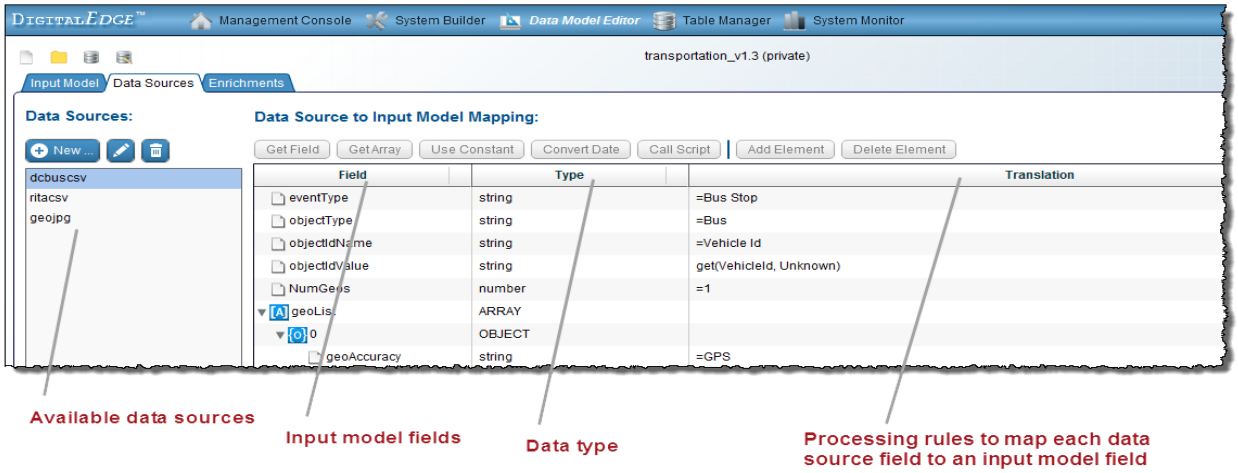
For each data source, you will:

- 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.

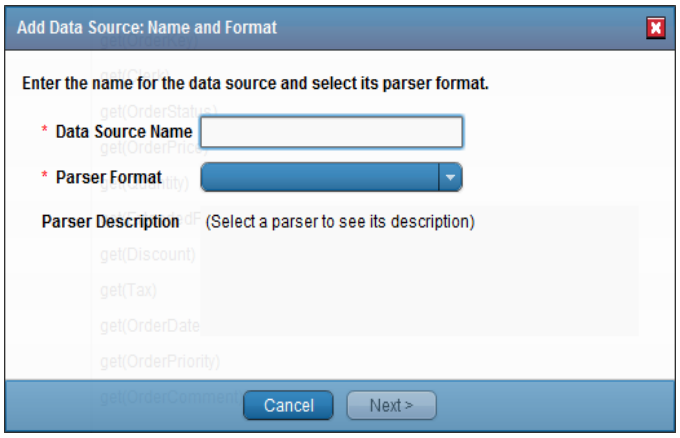


As you specify each field, the model is built in JSON representation.

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. 



3. 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.
4. 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

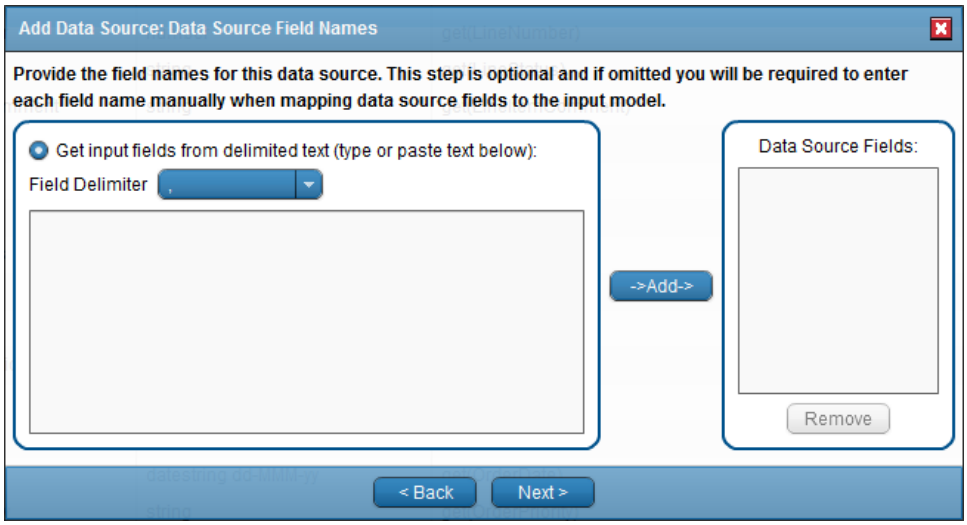
- **CEFParser**: Common Event Format from ArcSight®, an open standard for logging security-related information in a common event log format
- **CsvParser**: a frequently used Comma-Separated Values configurable parser to extract delimited plain text
- **DNS PCAP**: parser that reads and extracts DNS 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

5. 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 35).

1. In the **Get input fields from delimited text (type or paste text below)**:
   a. Select the appropriate **Delimiter** that you are using to separate field names.
   b. 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



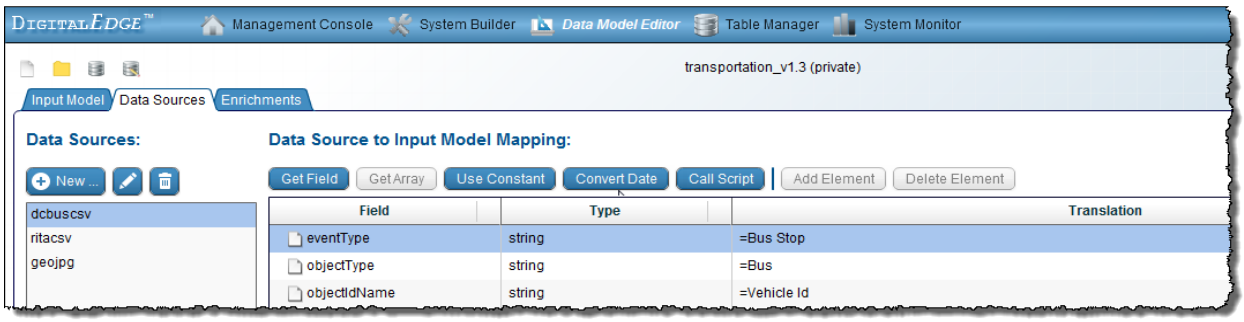
2. Click **Add** to transfer the input fields to the **Data Source Fields** box.
3. Click **Next**.
4. Review the list of **Parameter**s and **Current Value**s.
    - Double-click on any entry to make corrections.
    - Click **Back…Cancel** to discard the specification.
5. Click **Finish**.

For details about parser parameters, See "Parser parameters" on page 38.

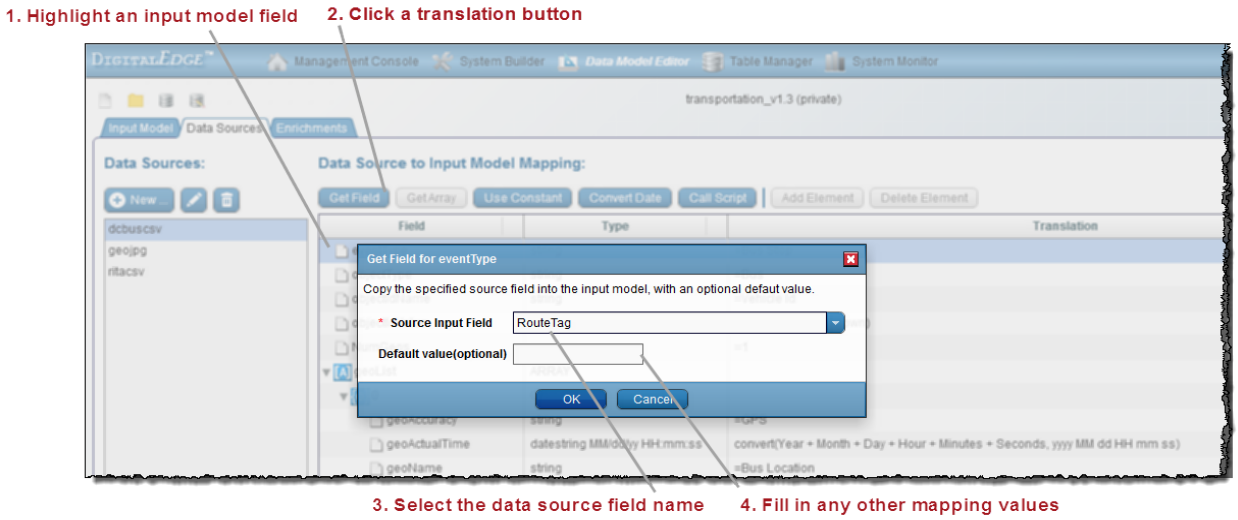
Map data source fields to the input model

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:



2. 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.
  a. Enter or select the name of the **Source Input Field**. Field names cannot include the following characters: single quote ('), double quote ("), or comma (,).
  b. 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.
  a. 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.
  a. **Enter constant value**.
- **Convert Date**: To convert an incoming date format to the input model's date format.
  a. 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).

b. 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.

c. 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.

a. Select the **Script Name**.

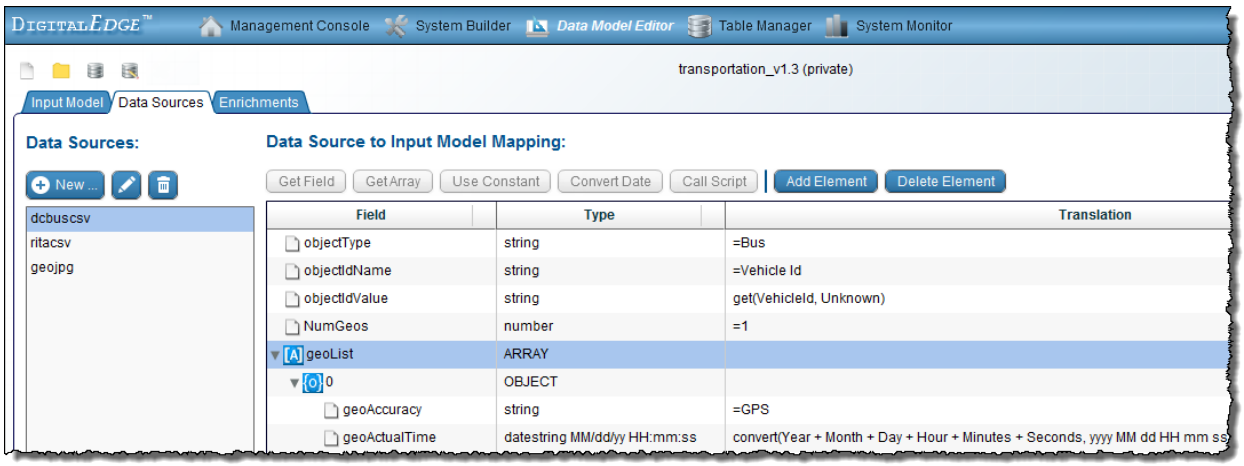
b. In the **Parameters** list, double-click in a **Data Source Field** row and select the incoming data field. Field names cannot include the following characters: single quote ('), double quote ("), or comma (,).

3. 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:



2. 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.
3. 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.

4. 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 32 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.

BinaryParser

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.

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

<table>
<tr><th>Parameter Name</th><th>Explanation</th></tr>
<tr><td></td><td>included a three-character airport code and a state abbreviation starting in position 20, and you then want to isolate the <code>state</code> data in that string, you would specify:

<code>READTOEND:location:20:ASCII</code>

<code>BACKREFERENCE:state:3:location:ASCII</code>

• <code>REGEX:fieldName:regex:targetField:fieldType</code> - Use this definition to write a regular expression as a <code>fieldName</code> specification. The last capturing group of the <code>regex</code> will be returned as the value for this field. The <code>targetField</code> is used to reference the <code>fieldName</code> defined in a preceding field definition line. A <code>REGEX</code> 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:

<code>READTOEND:data:0:ASCII</code>

<code>REGEX:second:\w+?\d+\D(\d+)
\D:data:NUMBER</code>

<code>FieldType</code> valid values include <code>ASCII</code>, <code>NUMBER</code>, or <code>BYTES</code>.

Here's a simple example that illustrates the use of all four definition types:

Incoming data string representing positions 0-9: ARDELP07WI

<table>
<tr><th>Field Definitions</th><th>Results</th></tr>
<tr><td><code>READTOEND:rawdata:0:ASCII</code></td><td>ARDELP07WI</td></tr>
<tr><td><code>READTOEND:type:8:ASCII</code></td><td>WI</td></tr>
<tr><td><code>FIXED:transaction:3:4:ASCII</code></td><td>DELP</td></tr>
<tr><td><code>FIXED:montha:6:2:NUMBER</code></td><td>07</td></tr>
<tr><td><code>REGEX:monthb:\w+(\d+)\D:rawdata:ASCII</code></td><td>07</td></tr>
<tr><td><code>FIXED:branch:0:2:ASCII</code></td><td>AR</td></tr>
<tr><td><code>BACKREFERENCE:loc:0:branch:ASCII</code></td><td>A</td></tr>
</table>
</td></tr>
<tr><td>default-access-label</td><td>Distinguishes classified and unclassified data coming in to DigitalEdge; useful for arbitrating user access; valid values =</td></tr>
</table>

Parameter Name	Explanation
	UNCLASSIFIED, or any other classification marker Default = UNCLASSIFIED
default-source	Specifies the source of the incoming data (e.g., IP address, URL, path)

CEFParser

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

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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)

CsvParser

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

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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

Parameter Name	Explanation
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*.

DNSPCAPParser

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**.

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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) Default = UNKNOWN
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

EmailParser

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

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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)

EXIFParser

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

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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)

JSONParser

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

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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)

LibpCapParser

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

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

LogParser

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.

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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) Default = UNKNOWN
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

Parameter Name	Explanation
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})

SNMPPCAPParser

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**.

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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) Default = UNKNOWN
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

UnstructuredFileParser

This beta parser reads different unstructured files (such as Word documents, Excel files, PDFs, and metadata from audio/video formats) 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.

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 to DigitalEdge; 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) Default = UNKNOWN
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

XMLParser

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

Parameter Name	Explanation
default-access-label	Distinguishes classified and unclassified data coming in to DigitalEdge; 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

Parameter Name	Explanation
	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 58)

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.

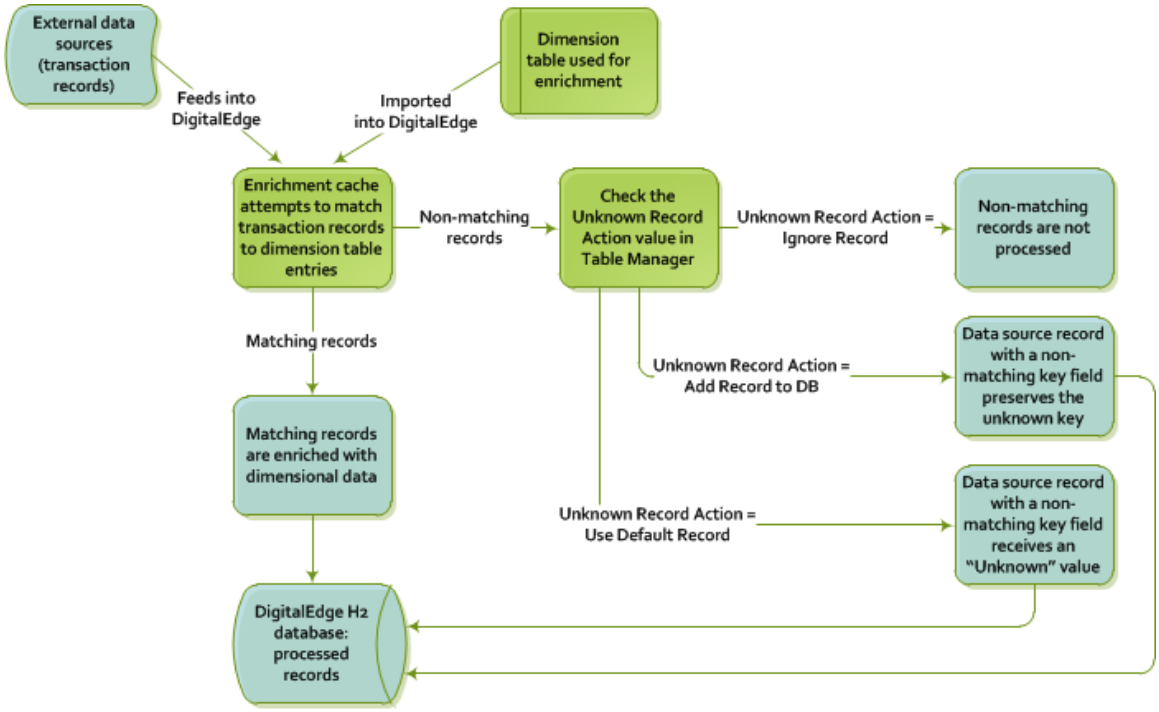
You must also select a load strategy for the enrichment cache, to specify how the cache is initialized. This is another trade-off that must be weighed: speed of cache startup versus potential cache misses if the cache has not yet reached its maximum size. Load strategy options include:

- **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.



Dimension Table Enrichment Matching

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.

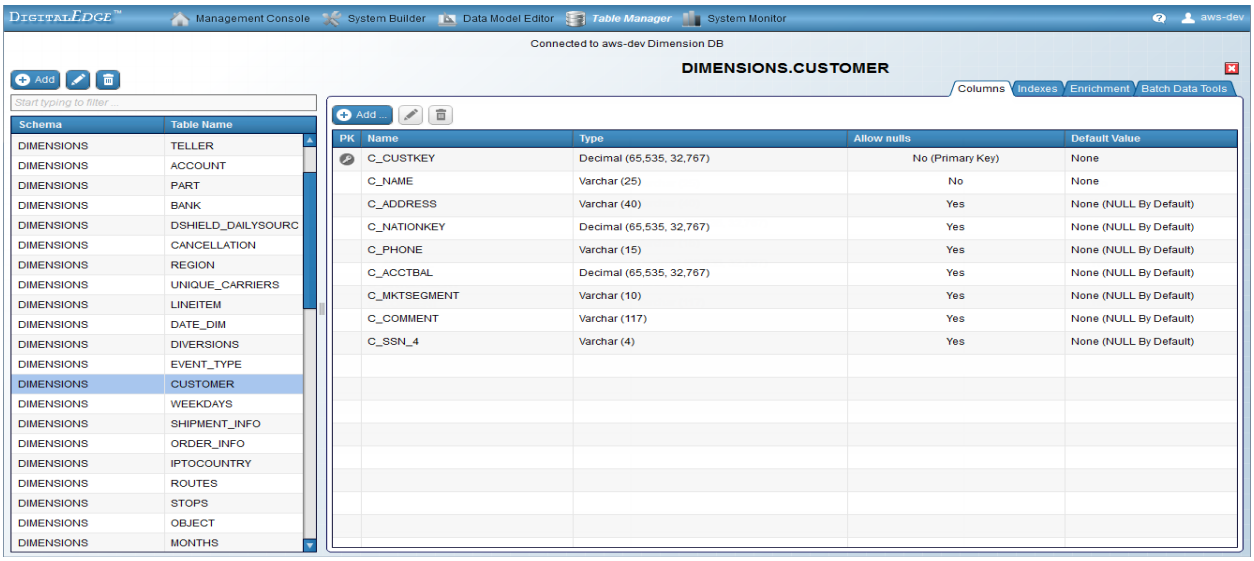
Step 3: Define Dimension Tables

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 52)
- **Add columns**: to specify every column in the table and the column properties (See "Add table columns" on page 53)
- **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 56)
- **Configure the enrichment cache**: to improve performance when matching input model records with dimensional enrichment tables (See "Configure the enrichment cache" on page 58)
- **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 59)
- **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 62)

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

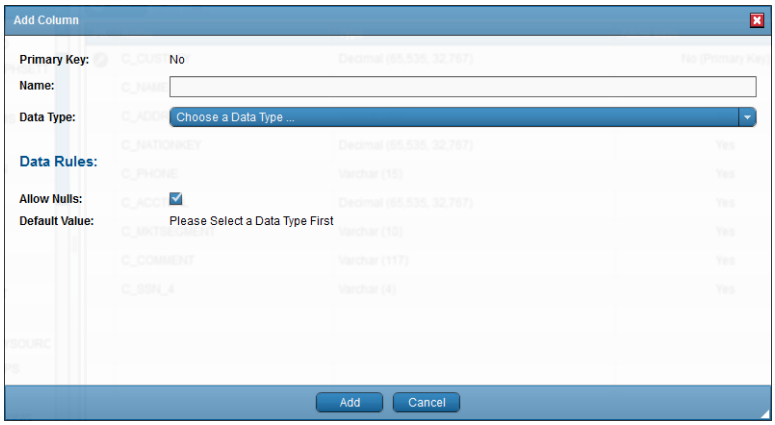
Even though you may just need one table column to enrich an input field, or a subset of fields to augment data in the input model, you should define all the columns in the table for accuracy. Defining columns in the **Table Manager** (with the **Columns** tab) is simply a task of naming each column and 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

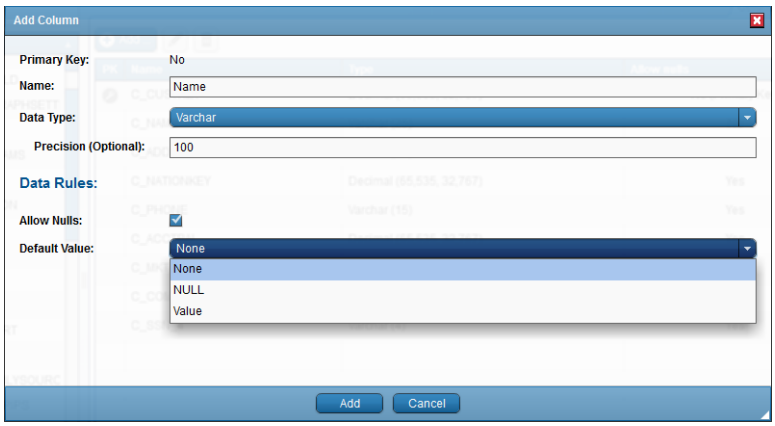
To specify all the columns (fields) in the dimension table:

1. Make sure you defined the table **Schema** and **Table Name** when you added the table. (See "Add a dimension table" on page 52)
2. Click the **Add a Column** button 

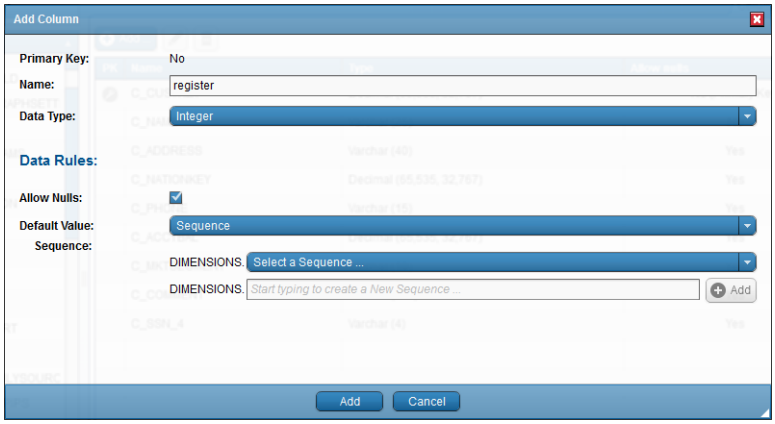
in the right panel. The **Add Column** dialog box appears.



3. 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.



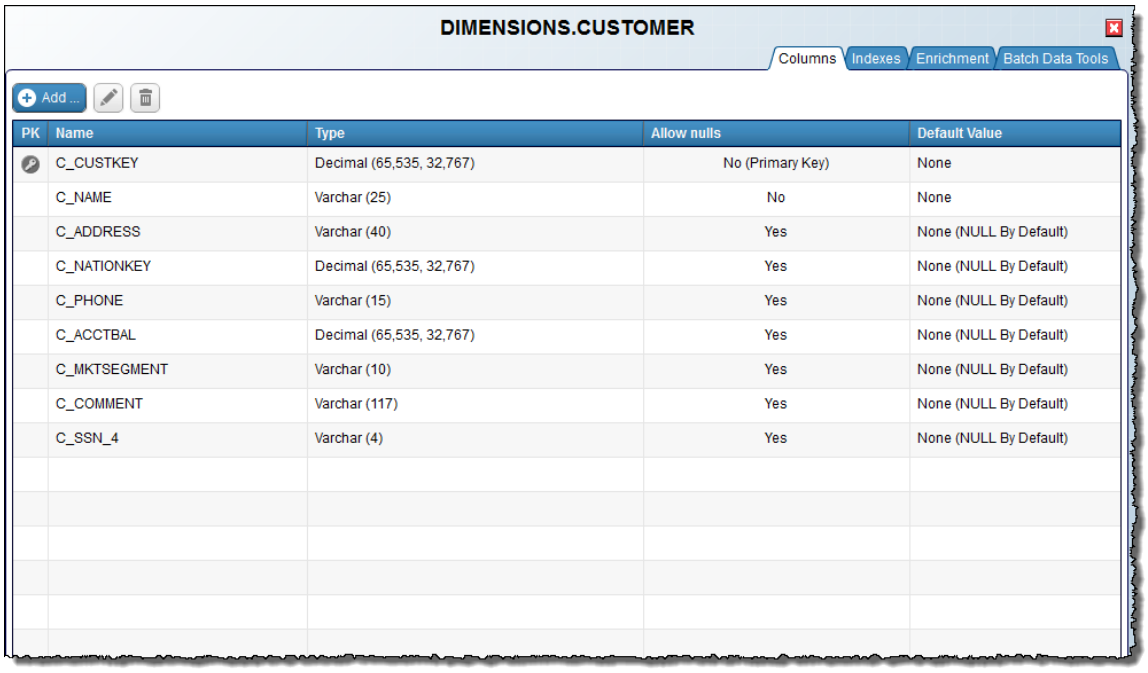
- **None**: No value is specified
- **NULL**: NULL is provided by default
- **Value**: A specific data value that should be assigned to this column when it is empty in an imported record. Enter the default value in the provided **Value** field.
- **Sequence**: This Default Value is available for any 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.

5. Click **Add** to create the column. Or click **Add & New** to save this column and create another one.
6. 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 58

There are three types of keys in play here:

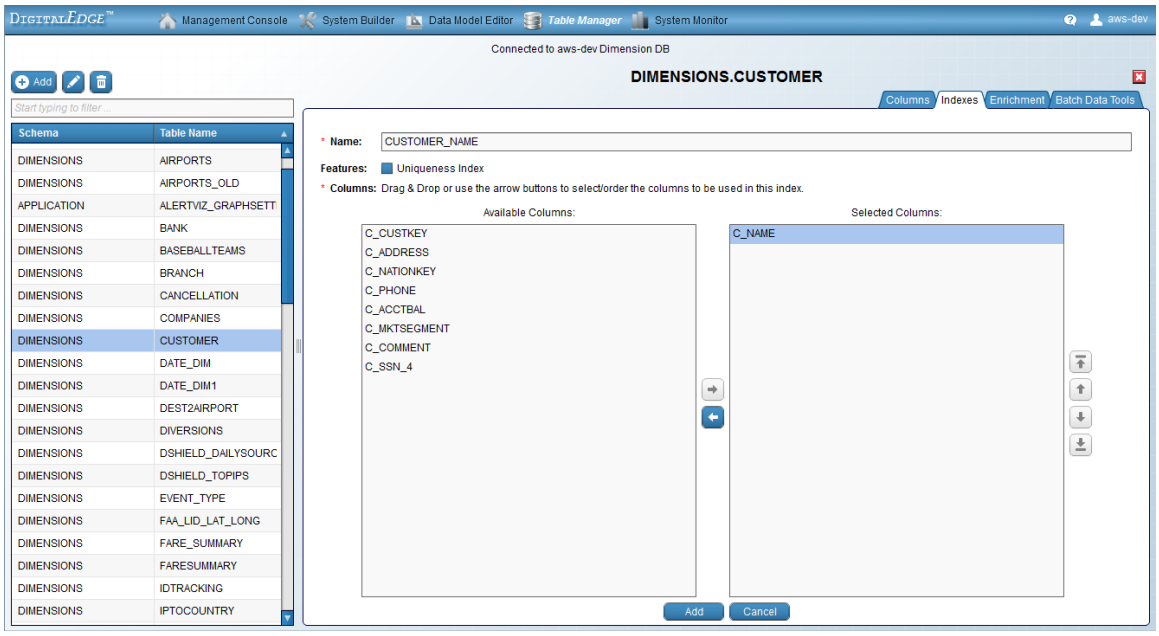
- **Natural key**: A natural key defines one or more columns that uniquely identify each row in the table. For example, a unique customer key may be built with the C_NAME and the C_SSN columns. Or a table of airport codes may only need one field - the 3-letter airport code - as a natural key that is 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 58)
- **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.



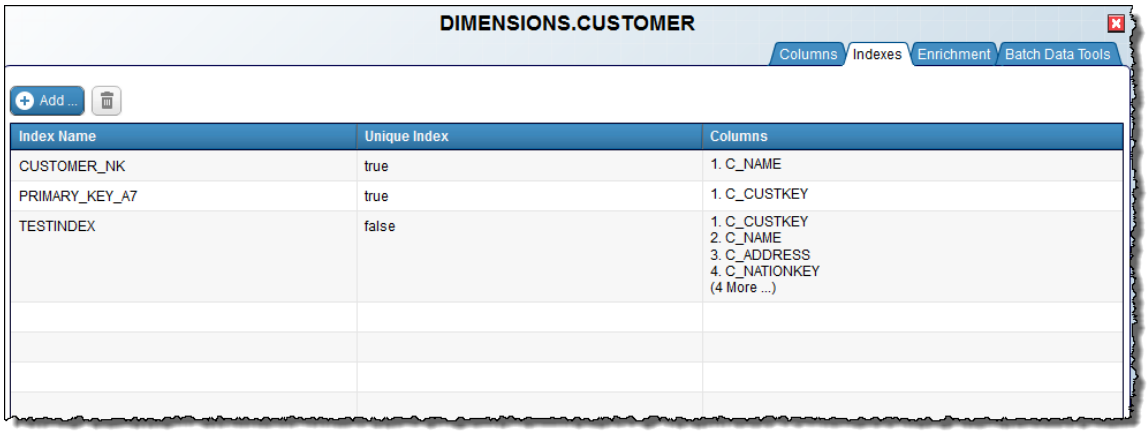
   Columns that are available for indexing are in the left panel. Columns that have already been selected for indexing appear in the right panel.

4. Enter a **Name** for the new index.
5. 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.
6. 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.
7. 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.
8. 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 48.

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 59)

- **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 59)

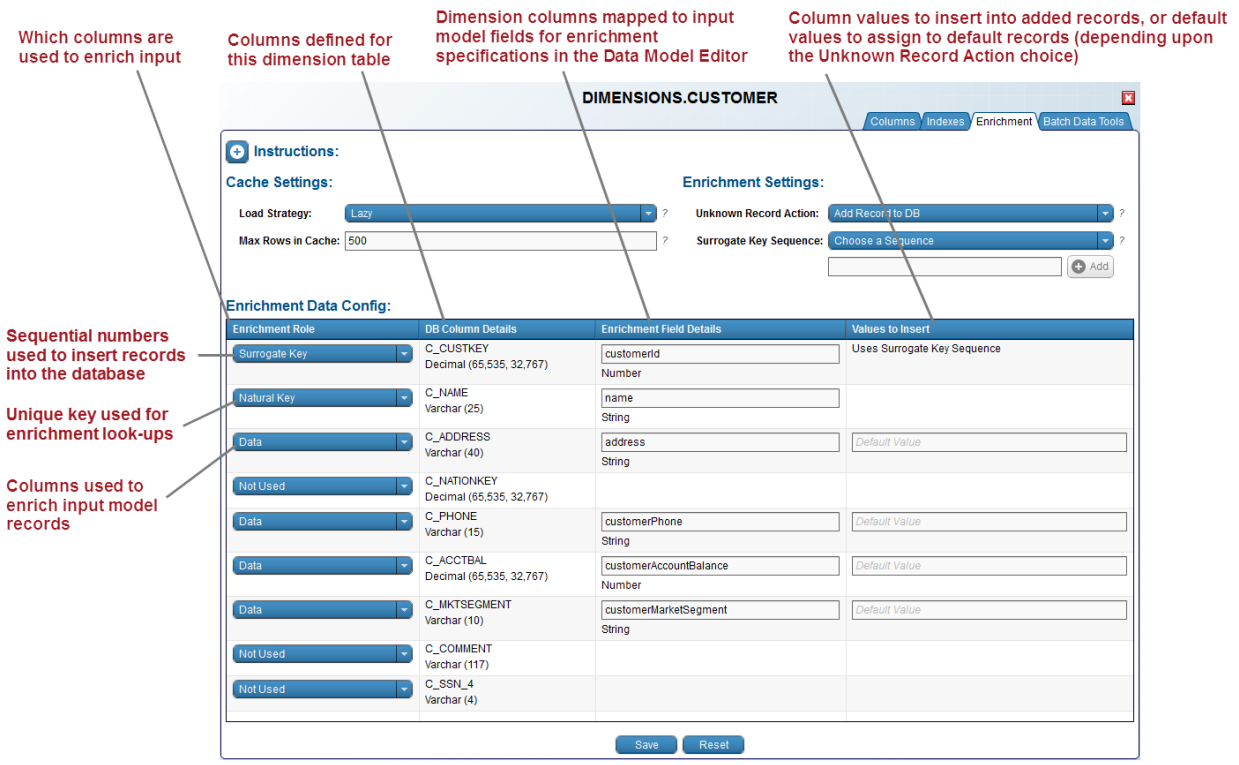
5. If you chose **Add Record to DB** as the **Unknown Record**, you must select a **Surrogate Key Sequence**, a numeric sequence to use when assigning a surrogate key to new records. You can select a **Surrogate Key Sequence** from those already defined in Table Manager, or you can enter a new sequence 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.

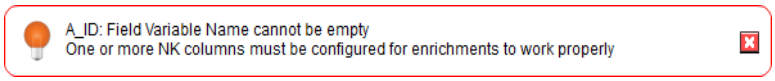
3. **Default Record Values** are required if you selected an **Unknown Record Action** of **Use Default Record**.

   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.

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

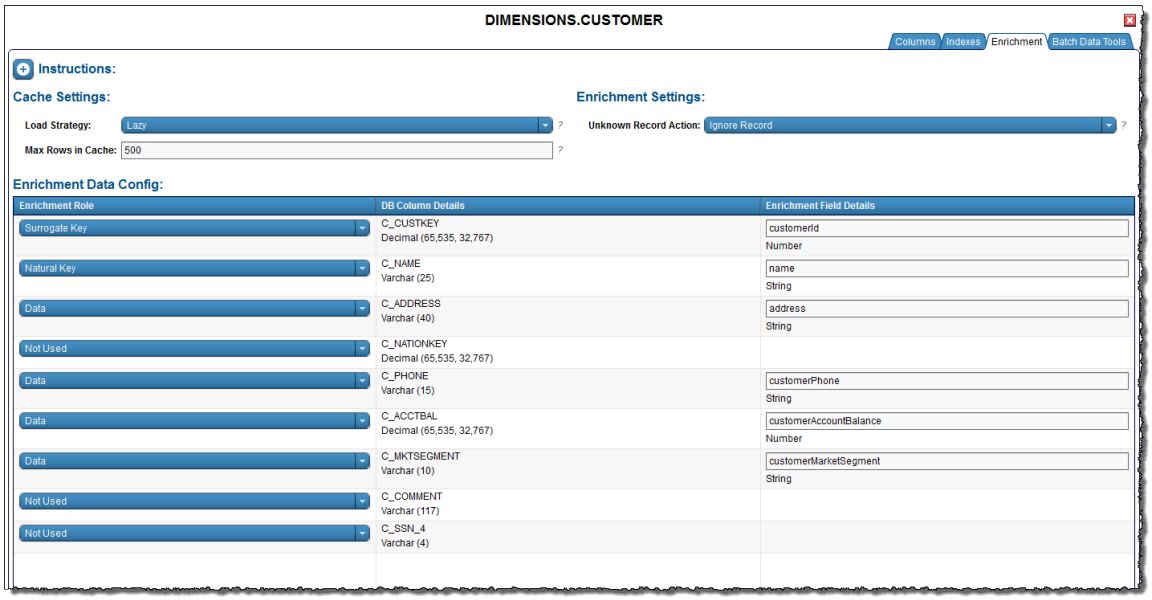
   Table Manager prompts you to enter default values for any field with an **Enrichment Role** of **Data** and which does not allow nulls.

5. **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 108 and See "Data sink parameters" on page 110)
- 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.

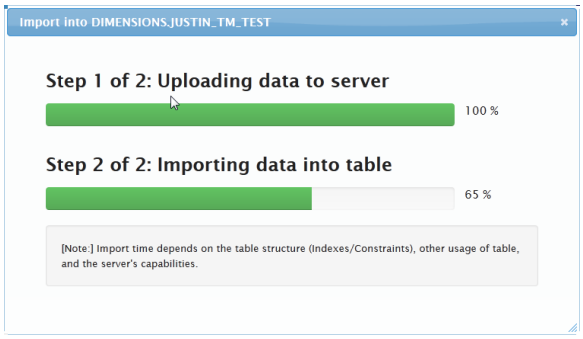
Use the **Batch Data Tools** tab to import data from a local CSV or ZIP file into your tenant database:

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:



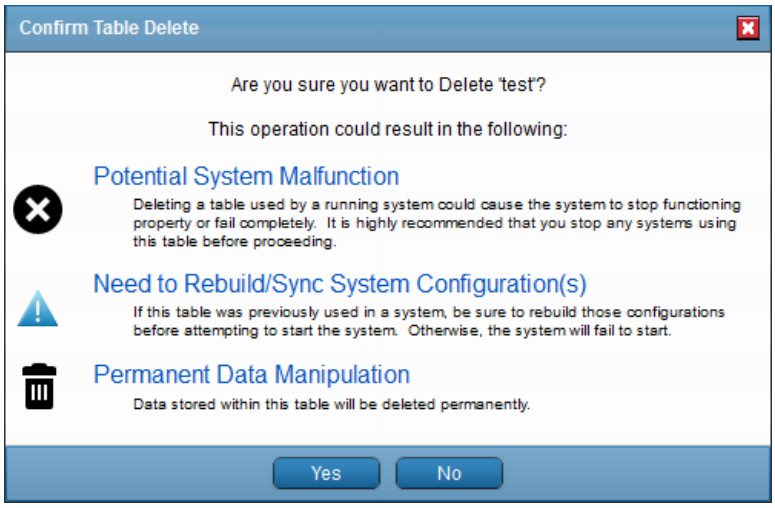
Delete a table

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:

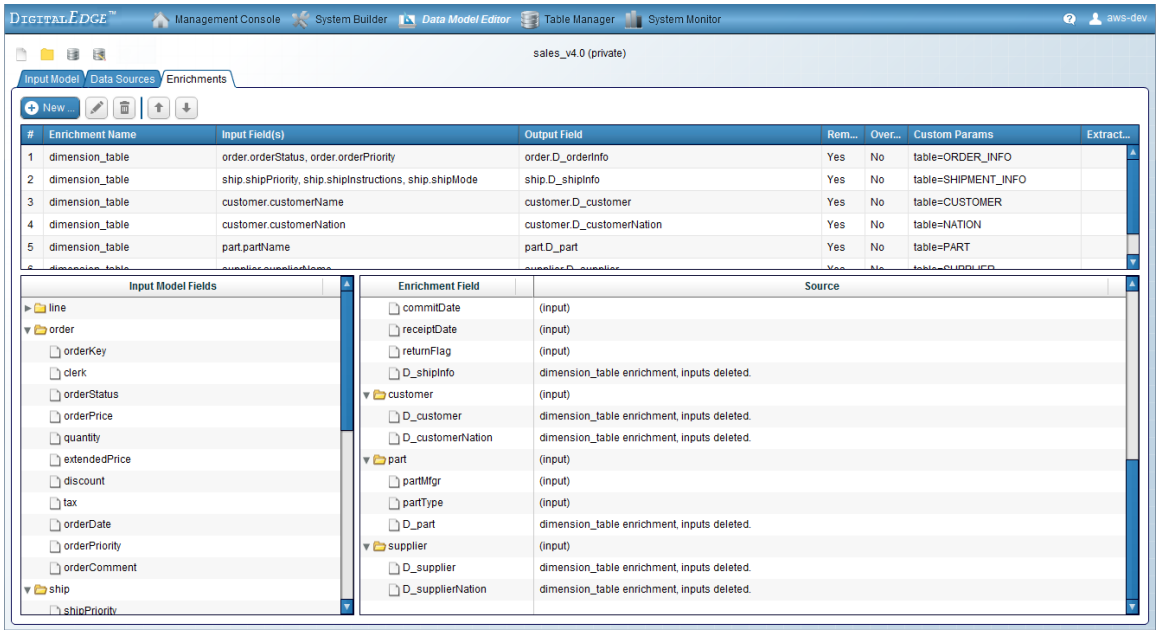
- **dimension_table enrichment** is a table lookup that pre-joins a dimension table to the main data feed; you must first define the source dimension table in the Table Manager before you can configure the dimensional enrichment here with the Data Model Editor. (See "Step 3: Define Dimension Tables" on page 51 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**:

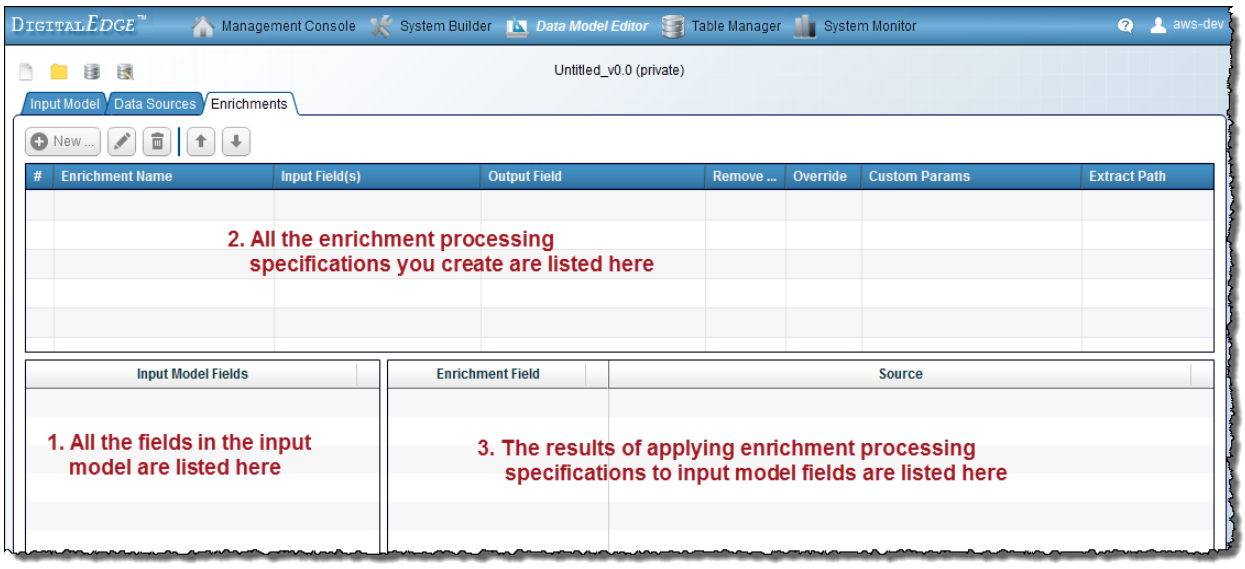


Step 4: Define Enrichment Processes

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.

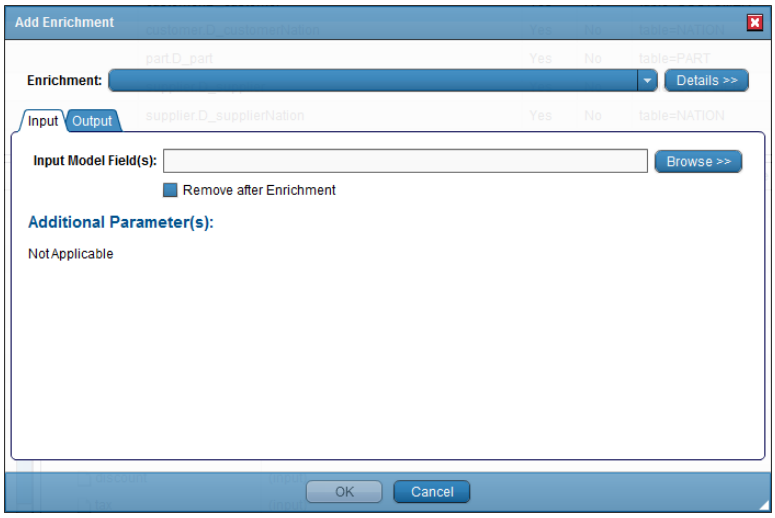
Before you define a dimension table enrichment, use the **Table Manager** tool to specify the dimension tables that are the sources of the enrichment data (See "Step 3: Define Dimension Tables" on page 51).

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 71).

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.



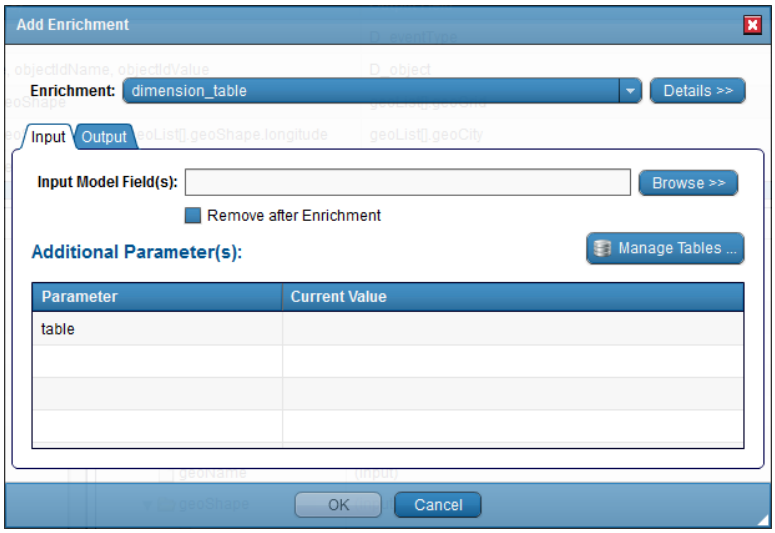
4. Select the type of enrichment you are defining from the **Enrichment** drop-down menu. (Note that this list includes the enrichments that are included in the core release. You may also have custom enrichments created for your site.)
   - **dimension_table**: adds a dimension record from a previously 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.

5. 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.
6. 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**.

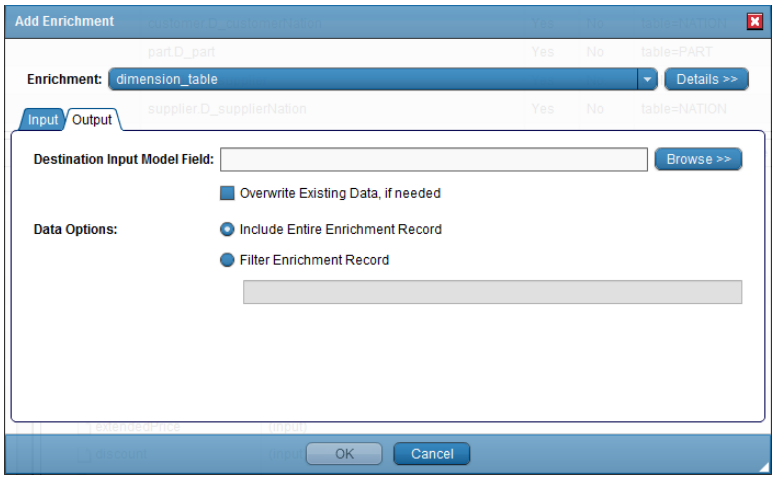


For details about enrichment parameters, See "Enrichment parameters" on page 72.

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

a. 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.

b. Click **Override Existing Data, if needed** if the enrichment should replace any existing value in the output field that you selected.

c. 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.

2. 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

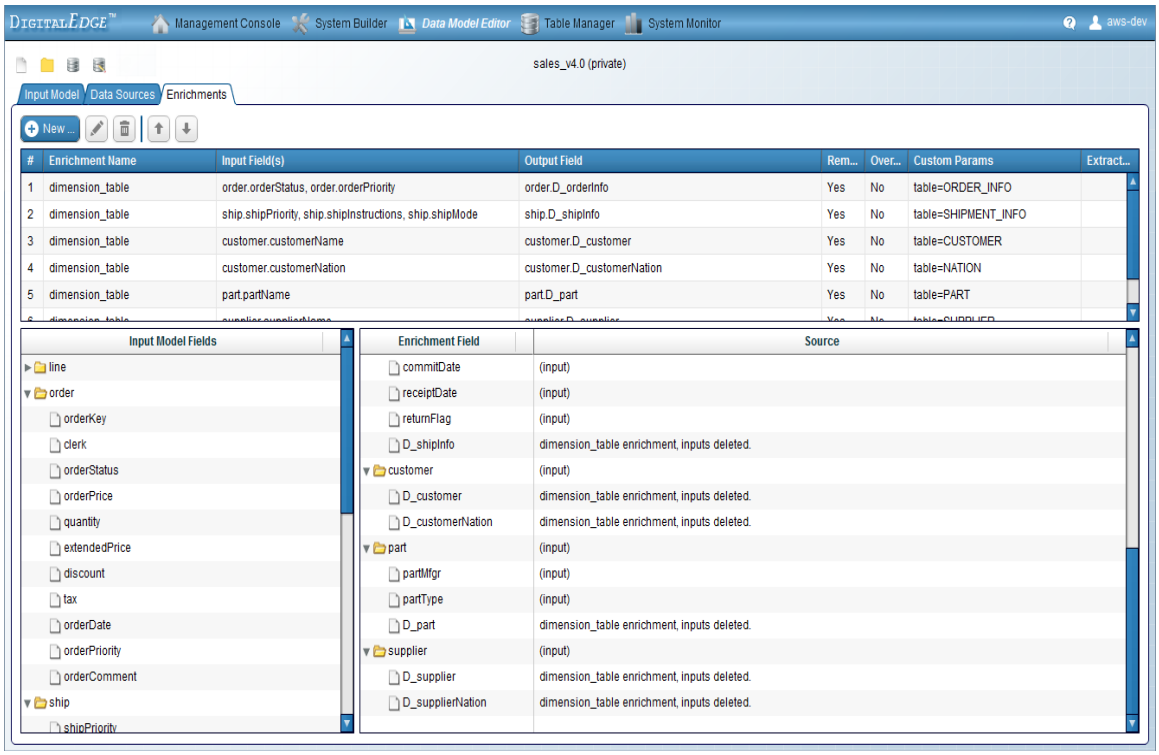
Overwrite Existing Data, if needed: Yes

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 66 for more information.

Here are detailed lists of enrichment parameters, descriptions, and values for the enrichments included in the core release. You can also 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:

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.

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 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"`

Parameter Name	Explanation
	`\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

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.

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:

<table>
<tr><th>Parameter Name</th><th>Explanation</th></tr>
<tr><td></td><td><code>pi=3.14,c=186282</code></td></tr>
<tr><td>expression</td><td>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 <code>quantity</code> = Q, <code>cost</code> = C, and <code>taxRate</code> = R, in a math enrichment expression:

<code>Q*C*(1.00 + R)</code></td></tr>
<tr><td>forceZeroIfArrayNull</td><td>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</td></tr>
<tr><td>variables</td><td>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:

<code>variable_name=numeric_value</code>

where the <code>numeric_value</code> 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:

<table>
<tr><th>Input Model Field</th><th>Sequential Position</th><th>Variable Name</th></tr>
<tr><td>store</td><td>1</td><td>S</td></tr>
<tr><td>cost</td><td>2</td><td>C</td></tr>
<tr><td>product</td><td>3</td><td>P</td></tr>
<tr><td>quantity</td><td>4</td><td>Q</td></tr>
<tr><td>tax_rate</td><td>5</td><td>R</td></tr>
</table>

and you are referencing quantity, cost, and tax_rate in the expression:

<code>Q*C*(1.00 + R)</code>

the variables list would be:</td></tr>
</table>

Parameter Name	Explanation
	`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).

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.

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

Parameter Name	Explanation
	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.

Parameter Name	Explanation
Input Model Field(s)	One or more fields from your input model, used as the source(s) for extracting data
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 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.

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

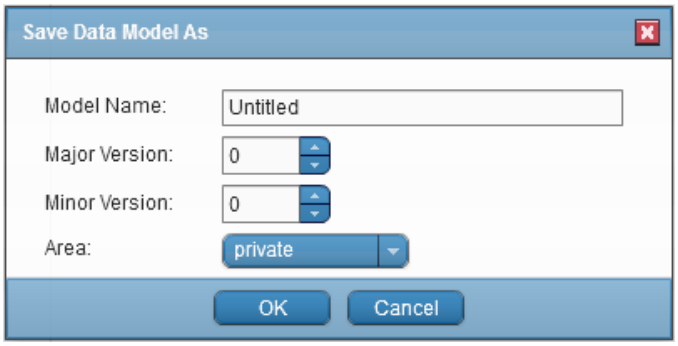
Save your data configuration work when you are done.

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.



   a. Enter a **Model Name** for your new private data model.

   b. Select a **Major Version** number.

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

a. Optionally select a **Minor Version** number.

b. Click the drop-down arrow on the **Area** menu and select **private**.

c. Click **OK**.

2. 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 **Version**s.
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.

6. The **Input Model** screen appears. From here, you can:
   - Edit any input model field; See "Step 1: Define an Input Model " on page 22 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 32 for more details
- Edit or add enrichments; See "Step 4: Define Enrichment Processes " on page 66 for information about adding enrichments and their parameters

7. Save the edited data model as a new version number; See "Step 5: Save a Data Model" on page 78 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 **Version**s.
5. Select a **Version** and click **Delete**. 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

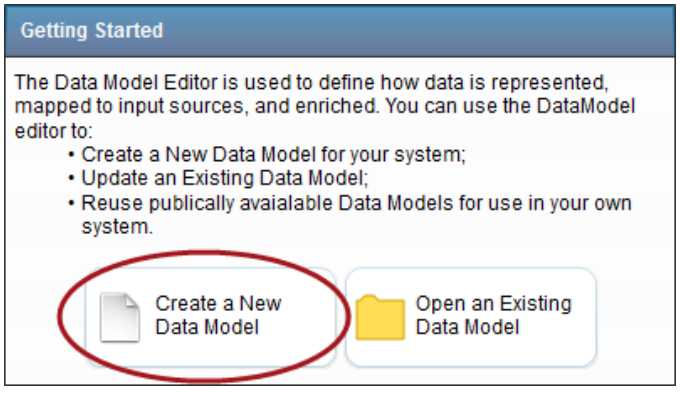
There is an alternative to defining a full input model, without going through all the complex steps of Table Manager and the Data Model Editor. Use the Data Model Creation Wizard when you

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:

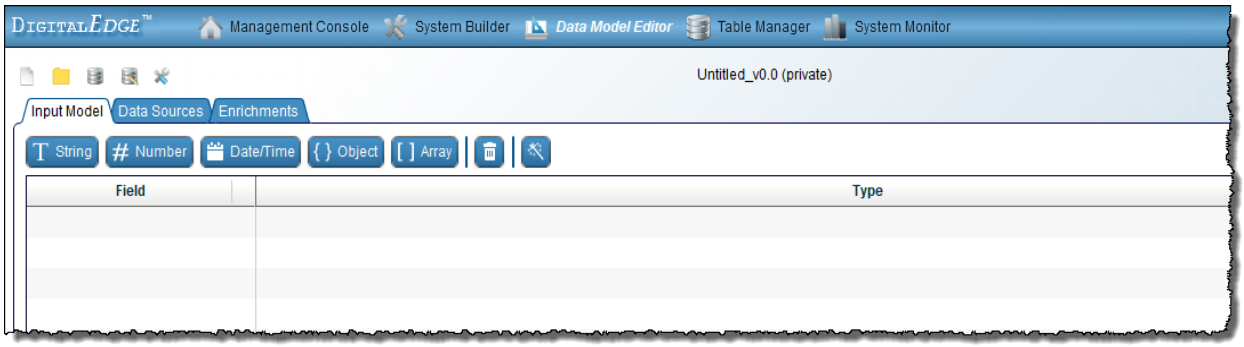
- The Wizard builds a new data model from scratch; it does not modify existing data models.
- Reads a CSV file and maps its columns directly into DigitalEdge fields with the same names, one field for each column.
- Reads a JSON file and map its fields into DigitalEdge fields with the same names.
- When 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

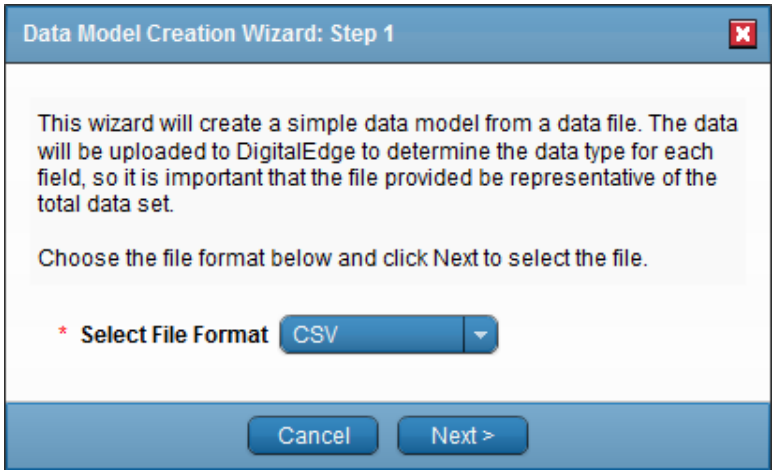
1. Access the **Data Model Editor**.
2. In the **Getting Started** dialog box, click **Create a New Data Model**.



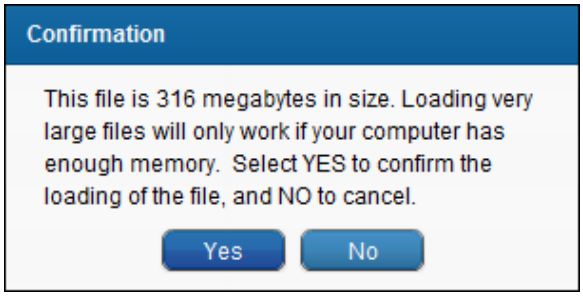
The **Input Model** workspace appears.



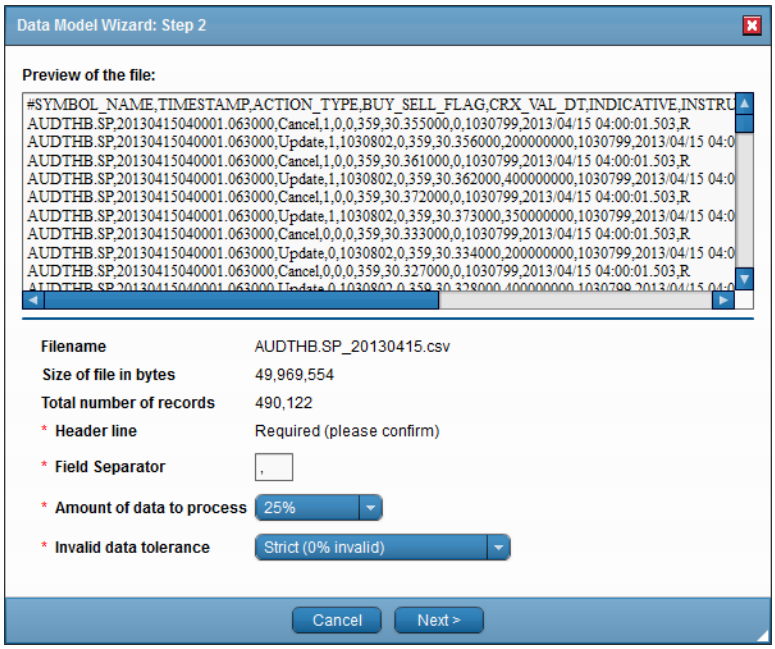
3. Click the **Data Model Creation Wizard** tool  . The **Data Model Creation Wizard: Step 1** dialog box appears.



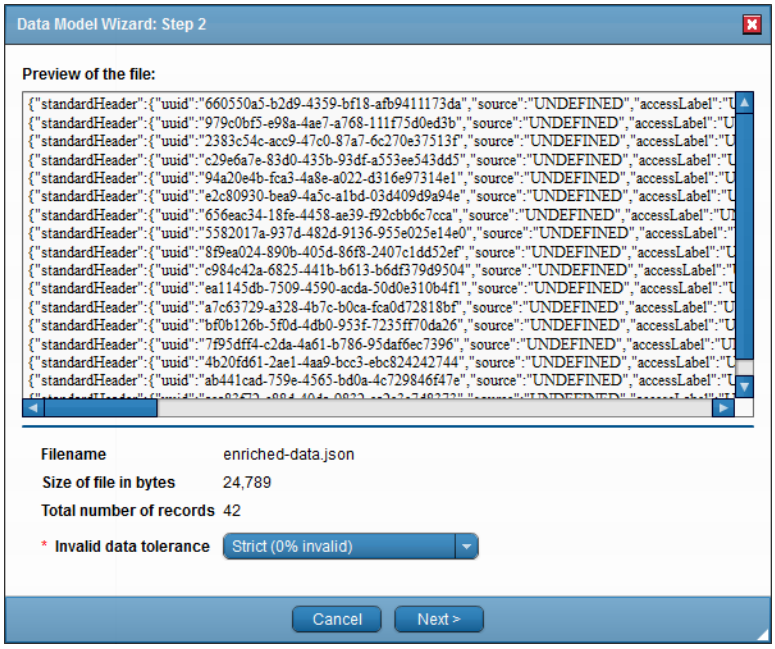
4. Choose a data source file format (e.g., CSV, JSON) and click **Next**. A file **Open** windows appears.
5. Browse to select an input file, and click **Open**. JSON files must be less than 50 MB. A CSV file can be larger, but you must have enough memory to process the file. The Wizard will warn you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



6. 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)

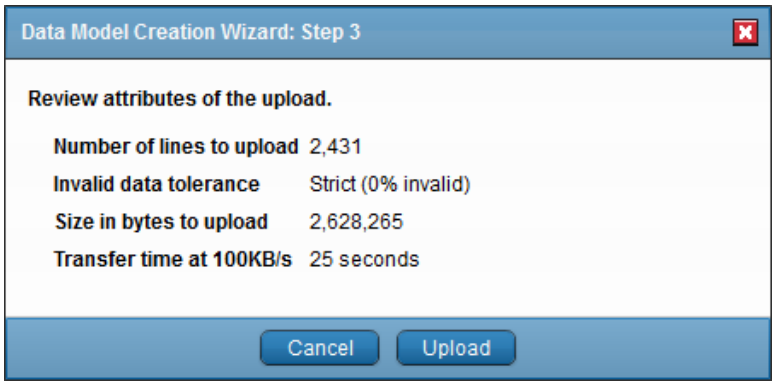
- **Total number of records**: The number of records that the Wizard detected in the file (information only)
- **Header line**: For CSV files, the presence or absence of a header. The Wizard assumes that a CSV file has one header line. You can confirm the presence of a header line by eyeballing the **Preview 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

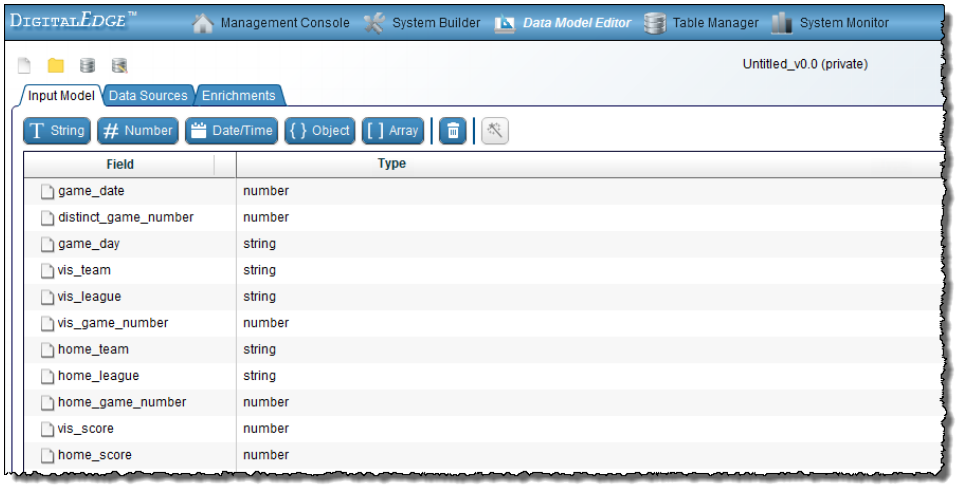
  If less than 95% of the data in one field/column is the same data type (numeric or date/time), the Wizard will identify that field/column as string data. In other words, if the Wizard cannot determine if a field is numeric or date/time data, it assumes the field is string data.

  
  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.

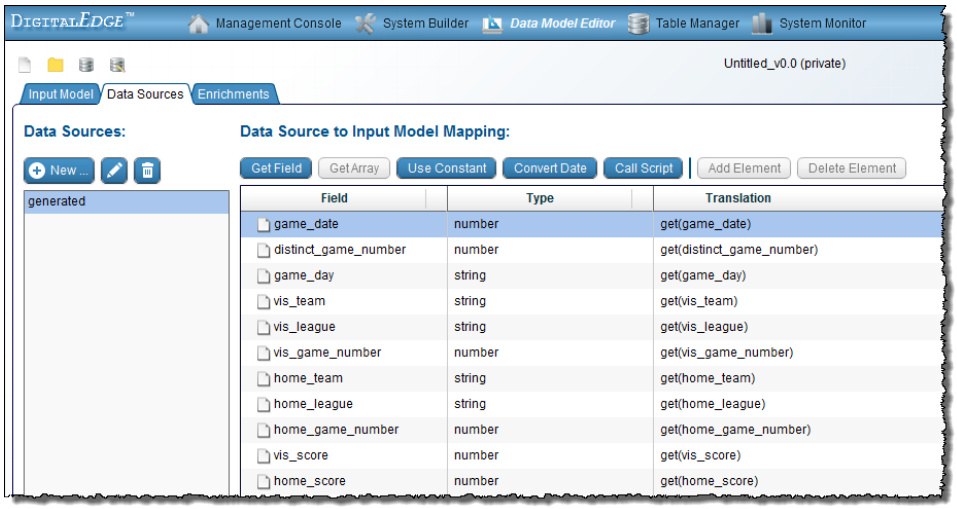
7. Click **Next**. The **Data Model Creation Wizard: Step 3** dialog box summarizes your choices and estimates the build time.



8. 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.



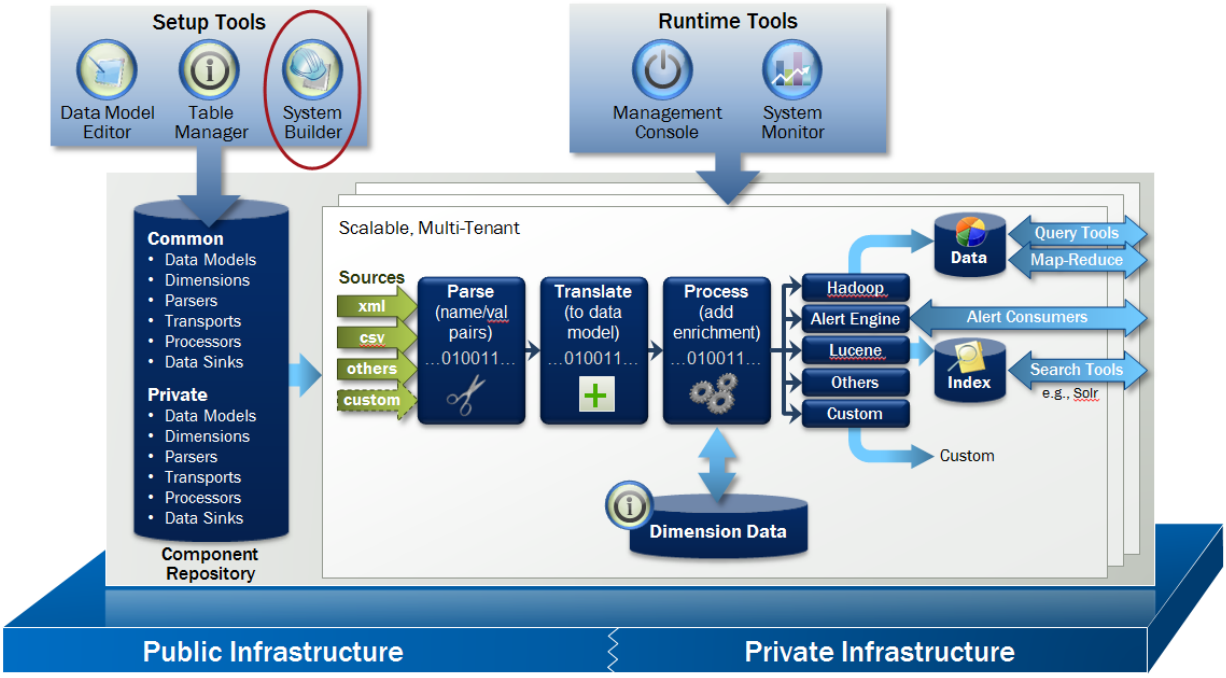
9. Check the **Data Sources** tab. The Wizard created a data source called **generated**, and mapped the source fields to input model fields.



10. Click the **Save** icon on the **Data Model Editor** title bar. 

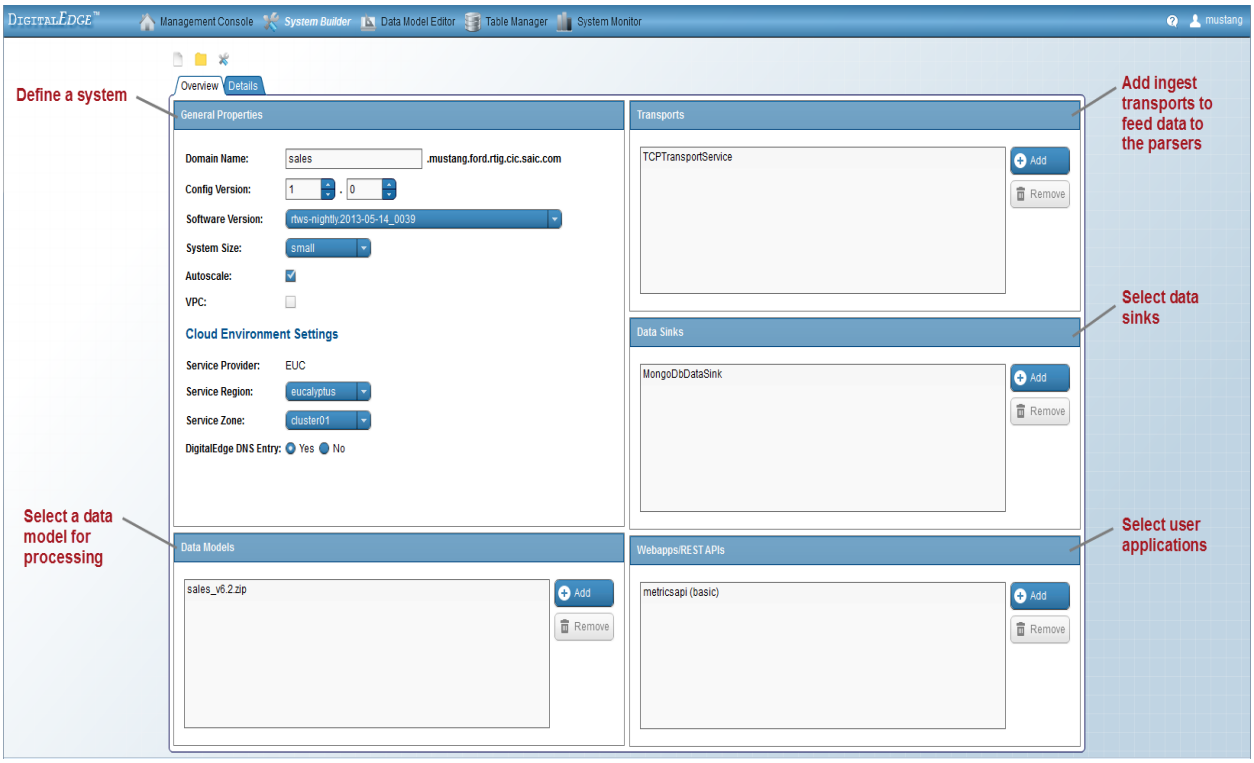
Chapter 5: Building the System

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.
   a. To open an existing system, select the domain name of your system (**Select system domain to open)** and click **OK**.
   b. 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.

5. Select a **Config Version** for your domain. Use this field to keep track of your site builds/configurations.
6. Select a DigitalEdge **Software Version**. You will typically use the latest official DigitalEdge release from Leidos.
7. 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.

8. 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.

9. 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 92 for more information.
10. Next, define your cloud-specific parameters. A **Service Provider** will be entered automatically, based on your account type (AWS or Eucalyptus).
11. 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.
12. 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.

13. 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.

14. 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.
    a. Click **Add**. A list of private data models that you created with the Data Model Editor appears.
    b. 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**.

15. In the **Data Sinks** section, you must choose at least one data sink for DigitalEdge data.
    a. Click **Add**. A list of data sinks appears.
    - **AlertingDataSink**: to filter data on dynamic selection criteria for users alerts in real-time
    - **CassandraDataSink**: use this 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 22.
    - **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 an existing Hadoop cluster that is external to DigitalEdge, is not managed by DigitalEdge, and is 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

- **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.

b. Click on a choice and click **OK**. The data sink is added to your system.

16. 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 EC2 account, your account includes the ability to create VPCs (virtual private clouds) for a more secure system. If you need a VPC environment, when you register for DigitalEdge, Leidos will create the initial VPC environment and the public subnet. You can then configure VPC in the DigitalEdge **System Builder**. (See the DigitalEdge *Configuration Guide* for more information.)

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, which serves as a firewall into the private VPC subnet
- 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) 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; it is spawned in the EC2 environment

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 94 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.

DatabaseWatcherTransportService

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.

DatabaseWatcherTransportService	
Parameter	**Explanation**
	Default = 0

DirectoryCrawlerTransportService

The Directory Crawler beta transport is similar to the Directory Watcher transport, processing data in a local or remote file system. But it includes additional capabilities beyond that of the Directory Watcher. The Directory Crawler transport decompresses any zipped files and processes files that match wild card patterns. This transport is also recursive on the source-directory; it will crawl subfolders.

Compression formats currently supported by the transport include:

- Zip (*.zip)
- Tar (*.tar)
- Tar-Gzip (*.tar.gz, *.tgz)
- Gzip (*.gz)
- 7-Zip (*.7z)
- RAR (*.rar)
- Tar-Bzip2 (*.tbz2, *.tar.bz2)
- Bzip2 (*.bz2)

DirectoryCrawlerTransportService	
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
map-input-models	Specify file mappings in the following format: fileName1:recordFormat1:recordHeaderLines1:inputFormat1, fileName2:recordFormat1:recordHeaderLines2:inputFormat2, fileName3:recordFormat3:recordHeaderLines3:inputFormat1 where: fileName can include wildcards: • ? represents a single character • * represents multiple characters recordFormat is the data record type to determine how to split multiple records on the appropriate boundaries

DirectoryCrawlerTransportService	
Parameter	**Explanation**
	recordHeaderLines is the number of header lines that can be stripped out of the records inputFormat must match a Data Source name as defined in the Data Model Editor White spaces are ignored.
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 Recommendation: Use the BASE64_CONTENT record format when also using the UnstructuredFileParser in the same system
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 source-directory parameter as the local mount point to this shared directory.
source-directory	The local directory path to crawl for data, 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.

DirectoryWatcherTransportService

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

DirectoryWatcherTransportService	
Parameter	**Explanation**
	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.

HiveTransportService

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

HiveTransportService	
Parameter	**Explanation**
	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

JMSBridgeTransportService

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

JMSBridgeTransportService	
Parameter	**Explanation**
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

MongoDBTransportService

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

MongoDBTransportService	
Parameter	**Explanation**
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

PcapSnifferTransportService

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-clickto 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.

PCapSnifferTransportService	
Parameter	**Explanation**
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 PCAP from the drop-down menu. The PCAP splitter type outputs Base 64 encoded data only. Default = NULL
record-header-lines	How many header lines should be stripped out of the data records Default = 0

S3FileTransportService

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

S3FileTransportService	
Parameter	**Explanation**
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

TCPTransportService

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

TwitterFilterTransportService

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 for information about obtaining a token
o-auth-access-token-secret	Your OAuth access token secret; see the Twitter API documentation for information about token credentials
o-auth-consumer-key	Your OAuth consumer key; see the Twitter API documentation for information about obtaining credentials
o-auth-consumer-secret	Your OAuth consumer secret; see the Twitter API documentation 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

TwitterRESTTransportService

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

TwitterSampleTransportService

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 for information about obtaining a token
o-auth-access-token-secret	Your OAuth access token secret; see the Twitter API documentation for information about token credentials
o-auth-consumer-key	Your OAuth consumer key; see the Twitter API documentation for information about obtaining credentials
o-auth-consumer-secret	Your OAuth consumer secret; see the Twitter API documentation 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

TwitterSampleTransportService	
Parameter	**Explanation**
	Default = NULL
record-header-lines	How many header lines should be stripped out of the data records Default = 0

UDPTransportService

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.saic.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

UDPTransportService	
Parameter	**Explanation**
	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

URLTransportService

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

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 22.
   - **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.

3. 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 110 for details.
4. 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.
5. Click **OK** to save all parameter values.
6. 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 vary 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.

AlertingDataSink	
Parameter	**Explanation**
	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 with a user friendly search interface. 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. The Elasticsearch data sink needs no parameters for configuration; it is one of the easiest and quickest data sinks to implement in DigitalEdge, yet provides comprehensive processing and auto-scaling. With Elasticsearch, you can have massive amounts of streaming data ingested, indexed, and searchable in seconds.

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

ExternalHiveDataSink	
Parameter	**Explanation**
	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

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

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

HiveDataSink	
Parameter	**Explanation**
	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

LuceneIndexingDataSink	
Parameter	**Explanation**
	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

MongoDbDataSink	
Parameter	**Explanation**
	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
   - **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.

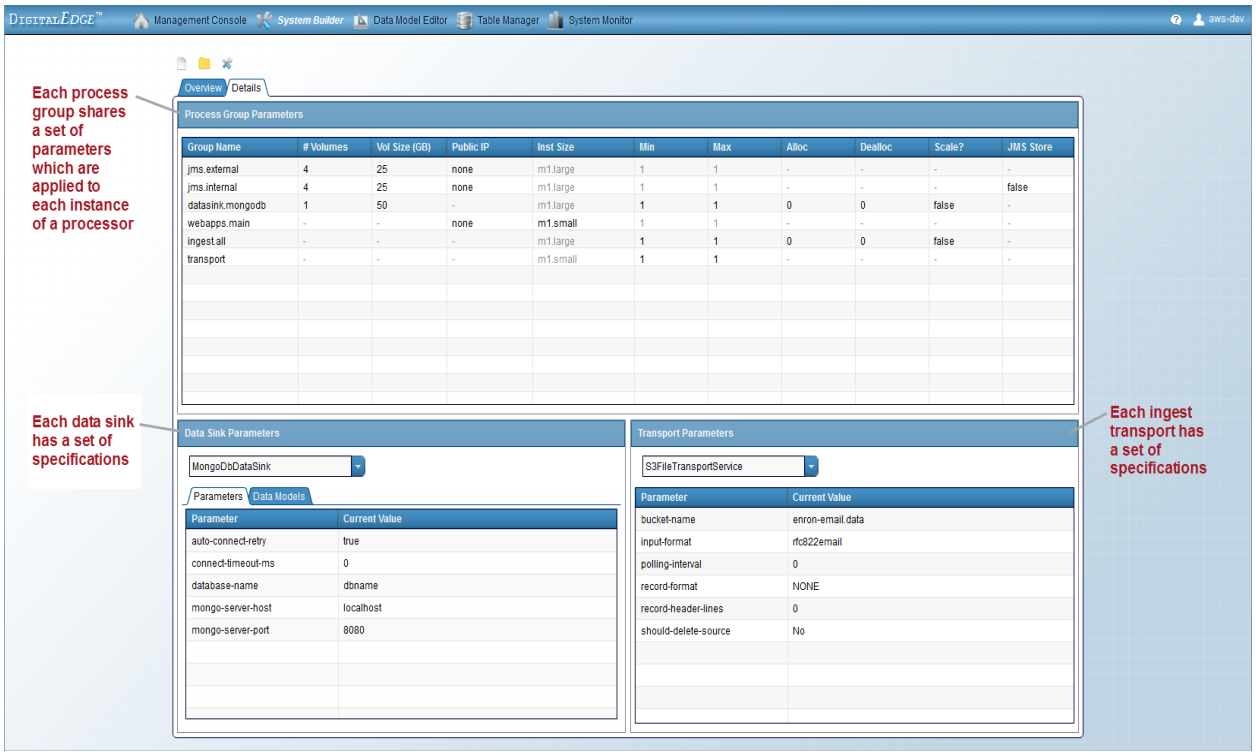
To feed DigitalEdge data into a pre-existing user application in your shop, configure an externally facing data sink such as the Scripting Data Sink or the JMS Data Sink (See "Step 3: Add Data Sinks" on page 108 for more information).

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 147 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:

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
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.
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.

3. Select a **System Domain** name and click **OK**.
4. 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.

Chapter 6: Creating Alerts

Many DigitalEdge systems need an active alerting engine to enable use cases such as identifying possible cybersecurity breaches, situational anomalies, and potential threats. Implementing alerts involves:

- Building a system in **System Builder** with three components:
  - **alertcontroller** webapp: to create and manage alert criteria, notifications, and subscriptions
  - **alertsapi** webapp: the back-end component that manages alert criteria
  - **AlertingDataSink**: to filter processed records for alert triggers and to send out alert notifications as email messages or messages in a JMS topic
- Identifying alerting criteria, the business rules that define a potential threat, and translating those rules into data specifications that will trigger an alert
- Writing the messages that should be sent out as alert notifications
- Identifying the people who should be notified of an alert situation

DigitalEdge supports a dynamic alerting infrastructure. The alert model is subscription oriented, so that each user can subscribe to specific alerts. DigitalEdge can also disseminate alerts through mechanisms such as email messages, text messages, mobile devices, or a JMS queue.

Alerting criteria

The first step in setting up alerts is to define the business rules that identify problematic situations. For example, a bank may want to know if one account is always serviced by one teller, which may indicate potential fraud. Or a network administrator may want to know immediately when an intrusion detection system (IDS) has sensed a possible incident, which may indicate a cybersecurity breach. Business rules are based on a *pattern* that is detected in the *data model*. You define the conditions under which a transaction or a pattern of activity is considered a threat. When data is ingested and processed, the DigitalEdge alert engine filters records against the data rules. Alerting filters are built with the **Alert Controller** and the **Alerts API**. See "Specifying alerting criteria" on page 132.

Alert notifications

After you have specified the rules that identify potential fraud situations, you should create a notification that will be sent to key decision makers when an event occurs. When a match is found between a processed record and the alert filter rules, a notification is generated and sent to one or more individuals. Notifications are sent within seconds of DigitalEdge identifying a potential fraud situation. Notifications can be sent via email, a mobile device, or a JMS queue message. Users "subscribe" to alert email notifications; they sign up to receive alert messages. Use the **Alert Controller** to manage notifications and subscriptions. See "Managing alert notifications" on page 135.

Alerting data sink

The alerting data sink specifies how alerts are issued. This data sink does not store DigitalEdge records or alert notifications; it filters processed records against the alerting criteria and sends out alert messages. See "Building an alerting system" on page 132.

Building an alerting system

To build an alerting system, you must configure three components:

- **alertcontroller** webapp: The **Alert Controller** assists with several alerting tasks:
  - Define the data criteria that trigger anomalous situations
  - Create alert notification messages
  - Define alert subscriptions
- **alertsapi** webapp: The Alerting API webapp is a REST API used to manage the alerting criteria that serve as data triggers for alert notifications. You can use the **Alert Controller** as a front-end UI to the Alerts API.
- **AlertingDataSink**: This data sink filters processed records against alert triggers and sends out alert messages to subscribing decision makers. This data sink does not store DigitalEdge processed records or alert notifications.

Follow these steps in **System Builder**:

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.
3. Click the **AlertingDataSink**.
4. 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 110)
5. Click **OK** to save the parameter values.
6. Next, use the **Webapps/REST APIs** section.
7. Click **Add**. The **Select a Webapp/REST API** dialog box appears.
8. Click the **alertcontroller** webapp.
9. Click **OK**.
10. Click **Save** when you are done with System Builder.

When you Add the **alertcontroller** webapp, DigitalEdge also automatically adds the **alertsapi** webapp to your system.

Specifying alerting criteria

To set up an alerting service in your DigitalEdge system, you must first define the data criteria that trigger anomalous situations. For example, a credit union may want to know when an account

withdrawal is over $1,000 in a single transaction. Alert rules are based on a *pattern* that is detected in your *data model*. You define the conditions under which a transaction is considered a problem. When data is processed, the Alerting Data Sink engine filters records against these data rules to immediately identify problematic activity.

The **Alerts API** webapp is a REST API used to specify the alerting criteria that serve as data triggers for alert notifications. You can create JSON expressions to run on one or more data model fields. For example, a network security officer may want to flag a user login that originates outside of the local intranet. You define alerting criteria that are specific to your data model and your business needs. See the DigitalEdge *Alerts API Guide* for more details.

Use the **Alert Controller** webapp as a front-end UI interface to the Alerts API. The **Alert Controller** application helps you create *alert definitions*, each of which includes:

- **Alert Criteria**: to define the alerting business rules
- **Email Template**: to notify key staff about an alert situation

The **Alert Criteria** tab helps you define the business rules that trigger an alert situation.

Using the Alert Controller to create alert criteria

1. Access the **Alert Controller** URL site that your DigitalEdge Administrator has established for the alerting application, such as:

   `https://default.<system_domain_name>/alertcontroller`

   where `system_domain_name` is the full **Domain Name** created in **System Builder**.

   Provide your **Username** and **Password** to **Login**. The **Alert Controller** screen appears.

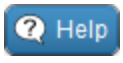
   The left panel lists the **Name** of each alert definition that you create, and the **Data Model** that is used to specify that alert filter.

   The right panel is the work space for creating **Alert Criteria** and **Email Template**s.

2. Click **New** 

to create an alert definition. The **New Alert Definition** screen appears.
3. In the **Name** text box, enter an alert name for the new alerting criteria.

   The maximum length for a **Name** is 125 characters.

4. From the drop-down **Data Model** menu, select a data model to work with. This is the enriched data model; all fields are available for building alert criteria.
5. On the **Alert Criteria** tab, enter a JSON expression to define the new alert. (Click **Help** 

for reference tips, or see the JSON web site for instructions on creating a JSON expression.) In general, the JSON expression should include:
   a. A data model field to work with

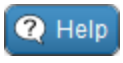
b. A data operator (=, !=, >, <, >=, <=, LIKE, IN, BETWEEN)

c. A data value to match or filter on

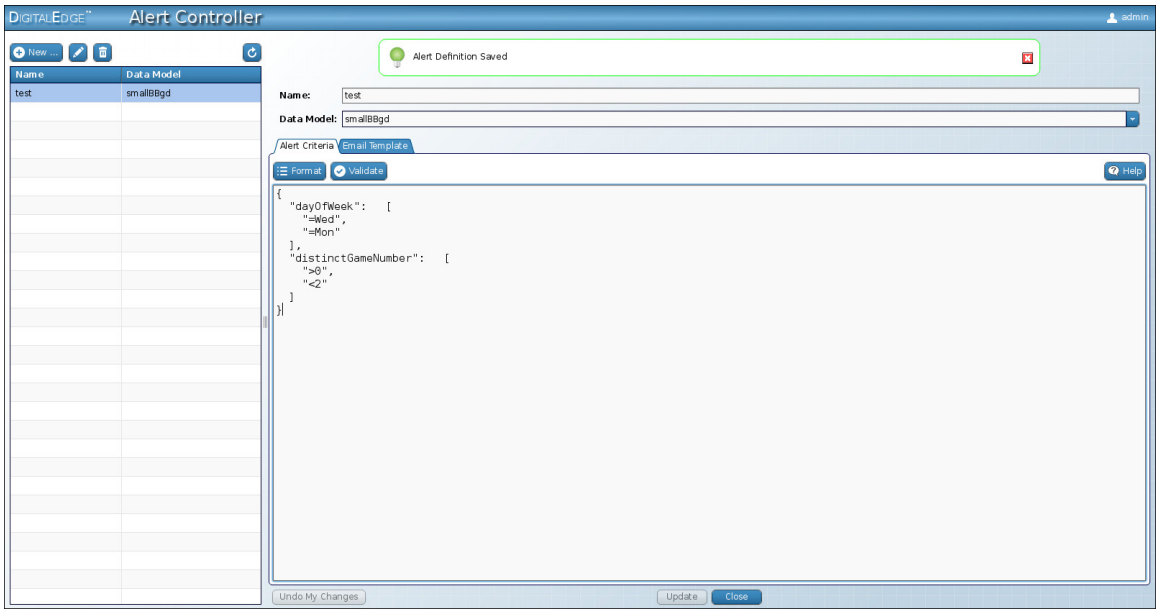
EXAMPLES:

- Simple equation: {"AccountID":"=01938"}
- Greater than operator: {"Time":">'07/24/97 09:14:32'"}
- Less than or equal to operator: {"NumGeos":"<=4"}
- Match values with wildcard characters: {"eventType":"LIKE '*ellipse'"}
- Compare a field to several different values: {"NumGeos":"IN (4,10,15,2)"}
- Compare the values in two fields: {"Teller":{"Name":"='@Account.Name'"}}
- Match values that fall between two values, in a range: {"NumGeos":"BETWEEN [4,10]"}
- Multiple criteria: {"eventType":"='Checkpoint'","extrinsic_Checkpoint":{"plateNum":"='ABC-1234'"}
- Nested criteria: [{"networkData": {"source": {"dshield": "LIKE '*'"}}}, {"networkData": {"destination": {"dshield": "LIKE '*'"}}}]

TIPS:

- Multiple criteria definitions under one **Name** are AND'ed together.
- To OR multiple criteria together, create multiple **Name**s, one definition per alert.
- Use @ as a variable to reference the value in a data model field ('@field_name'). This operator is useful for comparing field values of the same data type.
- The IN operator must have values enclosed in parentheses
- Use wildcards in a **LIKE** expression to match any character:
  - ? = single character wildcard
  - * = multiple characters wildcard
- String and date string values must be enclosed in single quote marks
- With the **BETWEEN** operation, use parentheses ( ) to indicate exclusivity, brackets [ ] to indicate inclusivity.
- Click **Help** 

for a more detailed explanation of the JSON criteria syntax examples.



6. Optionally click **Format**  to display the expression in a more readable JSON pretty-print format.
7. Click **Validate**  to check the syntax of your JSON expression.
8. Click **Create**  to save the criteria to the alert **Name** list.

   Or, click the **Email Template** tab to create an email notification for this alert.

 You can also highlight an alert **Name** to **Edit**  or **Delete**  it.

Managing alert notifications

The **Alert Controller** application helps you create alert definitions which include:

- **Alert Criteria**: to define the alerting business data rules
- **Email Template**: to notify staff about an alert situation

The **Email Template** tab helps you:

- Manage email notifications for specific alert criteria
- Create a subject line for an alert notification
- Compose the body of the email message for an alert notification
- Assign email notifications to subscribers (i.e., usernames in DigitalEdge)

Using the AlertController webapp to manage alert notifications

1. Access the **Alert Controller** URL site that your DigitalEdge Administrator has established for the alerting application, such as:

   `https://default.<system_domain_name>/alertcontroller`

   where `system_domain_name` is the full **Domain Name** created in **System Builder**.

   Provide your **Username** and **Password** to **Login**. The **Alert Controller** screen appears.

2. To create an email notification for a new alert definition that you are specifying, click the **Email Template** tab.

   Or, to work with an existing alert definition that you have saved to the alert **Name** list, highlight an alert definition **Name** in the left panel and click **Edit**  .

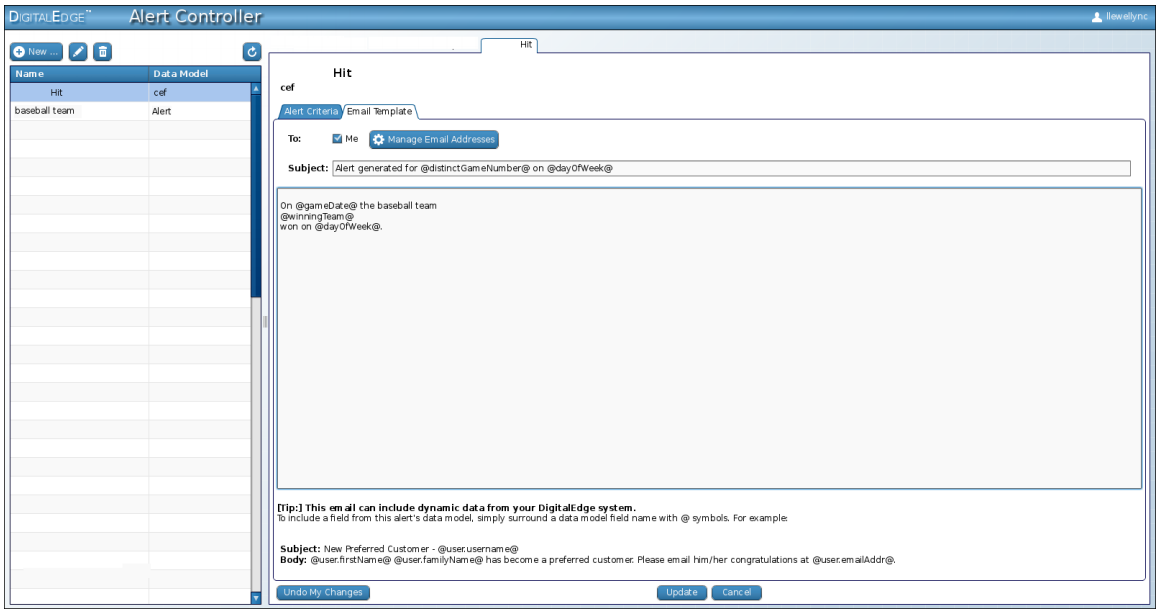
3. Use the **Email Template** tab to define one email notification per alert. A notification consists of a subscriber list, the subject line, and a message.
4. By default, email messages are defined to notify the person who is logged in to the **Alert Controller** as the subscriber. To have alert messages sent to your email address, check the box next to **Me** in the **To** line. If an email address is not attached to your username, DigitalEdge will display an error message. Click **Manage Email Addresses** to associate an email address with your DigitalEdge username.

   ★ You can also assign an email address to your username by clicking your username in the upper right corner of the screen and selecting **User Settings**. Enter an address in the **Your Email Addresses** text box, click **Add**  and **Save** Save.

   ★ If other users are subscribed to this email notification, their email addresses will be listed below the **To** line. This is read-only information.

5. Enter a **Subject** line for this alert's email notification.
6. Compose the message body in the large text entry box.
7. Click **Create** Create to save a new alert.

   Or, click **Update** Update to add an email definition to an existing alert filter.



As soon as you assign a username to an **Email Template**, the alert is activated; every time DigitalEdge matches an alert filter to new data, an email notification will be generated and sent out. Messages are sent individually; multiple alerts are not consolidated into one message.

You cannot delete an **Email Template**, but you can edit a template, blank out all its fields, and **Update** it so email notifications are no longer sent out for an **Alert Criteria**.

If you create and save an **Email Template** without assigning a username in the **To** field, alert notifications will be generated but not emailed. You might want to create such an alert message and write a user app that captures the message as a JMS message.

Chapter 7: Creating Analytical Dashboards

Once your data is processed, enriched, indexed, and searchable, analysts and business intelligence administrators will want access to tools that analyze big data, interpret and extract information of interest, and present findings in an engaging display. One such tool is the **Pentaho® BI Suite Community Edition (CE)**, an open source tool for creating graphical and interactive dashboards. Pentaho can use DigitalEdge data as input to its suite of analytical tools, identify key indicators, transform the data into visual tools such as charts, tables, and maps, and tailor the output into a customized browser-based dashboard specific to your analytical needs. In a matter of seconds, raw data is transformed into a highly effective visual tool for immediate analysis and use.

Pentaho is just one tool that can potentially integrate with DigitalEdge. It is not bundled with DigitalEdge, but the open source version is freely available on the Internet should you choose to download it, learn it, and integrate its visual analytics with processed big data from DigitalEdge.

To create a dashboard with Pentaho, you should have the following tools:

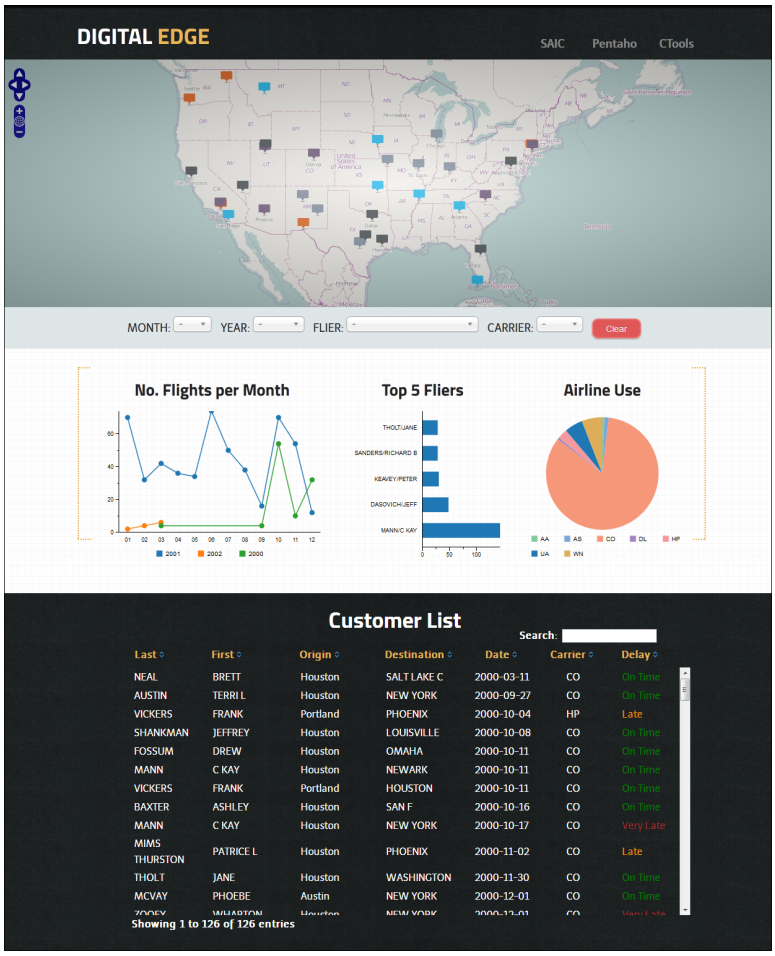
- The Pentaho BI Platform (Community Edition)
- Kettle: Pentaho's Data Integration (PDI) product, to define data transformations
- CTools: especially CDF, CBF, and CDA. CTools ("Community Tools") is an open source plug-in, developed by the Pentaho user community and managed by Webdetails.

Integrating Pentaho with DigitalEdge involves the following tasks:

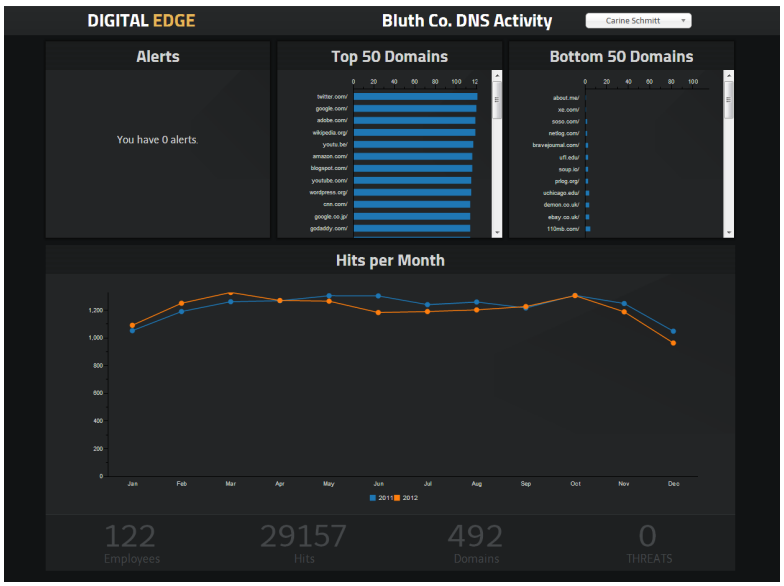
- Download and install the open source version of Pentaho and CTools from the Internet
- Learn how to use Pentaho and the CTools
- Configure a Pentaho webapp (contact Leidos for details)
- Prepare DigitalEdge data for input into a Pentaho dashboard. See "Preparing data for dashboard input" on page 141.
- Create a visual dashboard using Pentaho and its suite of configuration tools. See "Creating a dashboard" on page 141.
- Share the dashboard with analysts and business users. See "Sharing a dashboard" on page 142.

Dashboards are completely modular. You can select line, bar, or pie charts; data tables; or maps for inclusion. You can write scripts to interact with the data. And you can format and style the dashboard to your liking. Each dashboard is uniquely customized to your environment and needs.

For example, this dashboard was created by analyzing hundreds of thousands of email messages to cull flight and airfare information:



And this dashboard tracks DNS sites that employees have visited recently, alerting on sites that are potential sources of malware:



Preparing data for dashboard input

To populate a dashboard, you need to share DigitalEdgeprocessed data by transporting it into a relational database (such as H2, Oracle, or PostgreSQL™) for Pentaho access. Then, use Pentaho's Kettle tool to define data transformations.

1. In Kettle, create a new transformation with the **Design** tab.
2. Select the **Big Data** tool to share normalized, enriched data from your DigitalEdge data sink (such as MongoDB) as the **Input**.
3. Using the **JSON Input** step, retrieve the fields that you will need for the dashboard.
4. Select **Table Output** to move data into a relational database that Pentaho can access.
5. Make sure that your Pentaho relational database includes columns that correspond to the transformation fields.
6. Click the **Preview the Transformation** icon and the **Quick Launch** button to test the transformation. Or, double-click on one step to **Preview** an individual step in the transformation.
7. To run the transformation, use the green **Play** button, or select **Action > Run**.

Creating a dashboard

You can use the Pentaho Design Studio to create a dashboard, or the CDE plug-in component. CDE is just one component in the CTools plug-in, an open source suite of Pentaho tools which are developed and contributed by the Pentaho user community and managed by Webdetails. CTools ("Community Tools") go beyond Pentaho in the range of design features offered and the ease of use.

Steps for using CDE (Community Dashboard Editor) include:

1. Download and install CTools.
2. Open the CDE plug-in to create a new Pentaho dashboard.
3. Create a dashboard with the following functions (in any order):
   - Use the **Layout** tab to format and style the dashboard components.
   - From the **Data Sources** tab, identify your data sources, make connections to the database, and submit queries.
   - From the **Components** tab, select the types of components you want to include in the dashboard (chart, tables, maps, etc.).Write an SQL query and run it against your data.

   CDE creates a dashboard by linking these three elements together.
4. Optionally, style the dashboard with one of the component's advanced **Properties**, a Cascading Style Sheet (CSS), or a custom Javascript for interactivity.
5. Save the dashboard.

6. Analyze the CDA **Files** (Community Data Access), which list all your data sources and the results of the query, to determine if the query is providing the information you need. Refine the query as needed.
7. **Preview** the dashboard from CDE to make sure that the database connections and query are working as planned.

Making interactive components

You can make components interactive by including parameters in your data source and adding listeners to your components. Add **Parameters** in CDE, then use the **Get Variables** function in Kettle to retrieve the parameters for the dashboard.

You can also make a component clickable by enabling the component's **Clickable** property and writing a script that defines the clickable change in the **Click Action** property.

Adding maps

In CDE, select **Components**/**Custom**/**NewMapComponent** to add and configure an OpenStreet map.

Sharing a dashboard

Once you have designed and tested a dashboard, you can share it with business analysts on your intranet.

You will need a Pentaho server webapp for DigitalEdgeSupport@Leidos.com before proceeding. Contact Leidos for assistance.

1. You can **Preview** the dashboard in CDE at any time.
2. To refresh a static dashboard, rerun the KTR transformation and **Preview** the dashboard again.
3. To share a dashboard on your intranet, first move a blank dashboard to a server by copying the CDA files from CDE. Then plug in the database connection addresses in the KTR transformation.

Appendix A: Terminology

Term	Definition
Amazon EC2™ (Amazon Elastic Compute Cloud™)	A key part of Amazon's AWS™ public cloud computing platform, providing users the ability to create, launch, and end virtual server instances in a scalable deployment of applications
Amazon S3™ (Simple Storage Service™)	The online storage web service provided with AWS™ and used as a data source for public cloud instantiations
AWS™ (Amazon Web Services™)	Remote web services that comprise the cloud computing platform offered by Amazon.com; AWS™ includes Amazon EC2™ and Amazon S3™
Big data	Extremely large data streams that are too big and awkward to process and analyze with traditional database management tools
Blade server	A streamlined server computer in a blade center that is optimized to save space and energy over traditional rack mounted servers
CEF (Common Event Format)	An open standard for logging security-related information across various devices and applications in a common event log format for interoperability, created by ArcSight, an HP company
Cloud computing	Delivery of software services and shared resources over the Internet as measured services; characterized by resource pooling, dynamic scaling of resources, elasticity, self-service, and on-demand access
Data model	The DigitalEdge data model includes: • The input model: a JSON specification of a normalized data model • The data source mappings: specifications for parsing the incoming data and mapping each field to the input model • Enrichment processing: rules for enhancing the data with additional context and meaning
Data Model Editor	A setup interface tool used to configure:

Term	Definition
	• Input Model • Data source mappings • Enrichments
Data sink	A queue, server, or database that can receive pipeline-processed JSON data to store or post-process for other uses
Dimension table enrichment	Also known as keyed dimensional enrichment; the technique of using natural dimension keys for table lookups to find exact matches on dimension records for data enrichment
Dimensional Data Modeling (DDM)	A technique in data warehouse design that differs from entity-relationship (ER) modeling in traditional relational databases. DDM excels with real time data processing and large scale analytics. DDM enhances data querying performance and data understandability, results in faster queries, provides context to facts, and offers the ability to change data models (schemas) easily. In DDM, a fact is usually a measurement or numeric value; a dimension is a category of data that provides descriptive meaning and contextual.
Enrichment	The process of merging all related information into one record, providing context for source data and enabling rapid analysis
ETL (Extract, transform, and load)	A process in data warehousing to collect data from external sources, to process the data for internal needs, and to load the data into a target system
Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems)	An open source software platform to implement private cloud computing. Eucalyptus implements the Amazon Web Services™ API (AWS) and provides its own command line tools, Euca2ools. Eucalyptus is compatible with most Linux® distributions, and can host Windows images.
Input model	A simple data model that specifies how the data will look when it enters DigitalEdge, is normalized, and is mapped into a data model
JSON (JavaScript Object Notation)	A text-based open standard for data exchange, derived from JavaScript, representing data structures, arrays, and objects. The DigitalEdge Data

Term	Definition
	Model is specified in JSON.
Management Console	A runtime interface tool used to start up the Master Node, to manage users and security groups, to view a snapshot of the system's status, to start and shut down systems
MapReduce	A term coined by Google, Inc. describing a framework for processing distributed problems across large datasets using many computer nodes and clusters (multiple nodes). The Map phase portions out the input data into small sub-problems and distributes them to worker nodes, in a tree structure. The Reduce phase is done by the master node, which combines answers from the sub-problems into a final result. As an example, for the query "what checks were deposited since noon today?", the mapping phase would have multiple jobs checking the transaction log on each of the active server nodes, and the reduce phase would compile and merge a master list of checks from all the sub-jobs for a final result.
Master node	The first node that runs in DigitalEdge, starting up, scaling, and shutting down all other nodes
NoSQL (Not Only SQL)	A class of DBMS that differ from traditional relational DBMS primarily because they do not use SQL as a query language. They do not use fixed schemas, joins, or vertical scaling.
Parser	A tool to extract fields from an input data source and translate them to key-value pairs; based on a file format type (binary, CSV, email, JSON, log files, XML, etc.)
Repository	The storage location for all plug-in components; the Master Repository is at the TMS level, the System Repository is at a tenant DigitalEdge account level
RFC 822	A standard format from ARPA that specifies email message structures, such as the to and from fields, and attachments
SIEM (Security Information and Event Management)	Technology that provides real time analysis of security data and alerts generated by networked applications; SIEMS are typically implemented in relational databases on commodity servers

Term	Definition
Splitter	Each transport works with a specific incoming record type (JSON, XML, PCAP, etc.); the tranport's record-format parameter uses a splitter to define record boundaries when the input data includes multiple records
System Builder	A setup interface tool used to specify the plug-ins (parsers, translators, processors, data sinks) in the DigitalEdge pipeline and to assemble the system
System Monitor	A runtime interface tool used to visually monitor data flow and resource consumption
Table Manager	A setup interface tool used to manage enrichment processing sources (dimension data) and other application-level tables
Tenant Management System (TMS)	Behind-the-scenes system that manages a DigitalEdge tenant's Setup tools, Runtime tools, repository, Master Node, APIs, databases, and security
Transport	A mechanism used for transferring large amounts of data over a network into the DigitalEdge system

Appendix B: What Each Node Does

Each node in DigitalEdge is a virtual machine and an instance of a process group, most of which are auto-scaling. To help size a system, the following table provides details about what each node does.

Node	Content
webapps.main (on TMS)	Home to all the DigitalEdge APIs, Setup tools, and Runtime tools, including: • Management Console • Data Modeler • Table Manager • System Builder • System Monitor
anchor (on TMS)	Security and authentication node, housing: • CAS (JA-SIG Central Authentication Service) • LDAP • TMS database • TMS Gateway

gateway	The node that controls a DigitalEdge system, including:
	• Launching the master node
	• Starting and stopping systems
	• Creating and deleting systems and security groups
	• Synchronizing components and repositories
	• Housing:
	  ◦ CAS for single sign-on permissions
	  ◦ LDAP for user account credentials
	  ◦ APIs
	  ◦ Tenant database

master	The management node of DigitalEdge, controlling: • Starting and stopping all instances • Monitoring for auto-scaling • Adding and removing nodes based on load and storage utilization • Handling virtual storage allocations

Node	Content
	• Gathering metrics for auto-scaling decisions • Housing the System Repository
transport	Controlling all transports through the Transport API
jms.external	First entry point into DigitalEdge, and a staging area for incoming data to: • Receive data pushed into the jms.external queue by other clients • Feed data directly into DigitalEdge • Manage the parsing queue • Receive processed alerts from the datasink.alert that match alerting criteria, and place a message in this queue for notifications
ingest.all	Ingest node to handle processing pipeline tasks, including: • Parsing • Enrichment
jms.internal	Internal staging area for the next steps in the processing pipeline; a buffer for records queued up waiting for the next phase of processing: • Post-enrichment record holding • Temporary record storage
datasink	Each type of data sink has its own node and processes data for specialized uses; for example: • datasink.alert - filtering records against alert criteria, sending alert messages to the configured recipient (such as a topic on the jms.external node, an email message, etc.) • datasink.hbase - storing records to the Hadoop Distributed File System (HDFS) • datasink.hive - storing records to HDFS • datasink.lucene - indexing records for searching • datasink.mongodb - storing JSON-based records and providing a query interface Some data sinks automatically add additional nodes when they are spawned; for example, HBase and Hive add nodes (such as zookeeper) that are needed for a complete HBase ecosystem

Node	Content
webapps.main (on tenant)	Home to all webapps and REST APIs, including: • Search app • Metrics API

Appendix C: Forms for Designing a System

To configure a DigitalEdge system, it is recommended that you first gather detailed specifications about your data sources, input model, enrichments, and system components. It's easier to collect all the specs ahead of time before you sit down with the DigitalEdge Setup Tools. These forms are designed to help you identify and collect the configuration information you will need.

Form	When to Use The Form	Input to DigitalEdge Setup Tool
Input Model	Define a DigitalEdge input data model	Data Model Editor > Input Model tab See "Step 1: Define an Input Model " on page 22
Data Source	Define all your data sources	Data Model Editor > Data Sources tab See "Step 2: Define Data Sources, Parsers, and Data Source Mappings " on page 32
Data Source Fields	Define all the fields in each data source	Data Model Editor > Data Sources tab See " Define the fields in a data source" on page 34
Enrichment Dimension Table	Specify the tables and fields used for dimension table enrichments	Table Manager > Add See "Step 3: Define Dimension Tables" on page 51
Enrichments	Define data enrichments	Data Model Editor > Enrichments tab See "Step 4: Define Enrichment Processes " on page 66
Data Sinks	Decide which data stores to use	System Builder > Overview tab > Data Sinks section See "Step 3: Add Data Sinks" on page 108
Web Applications	Select user applications	System Builder > Overview tab > Webapps/REST APIs section See "Step 4: Add Web Apps " on page 124

Input Model:

Input Model Field Name	Field Type (Check One)					Common, or Extrinsic Data Source *
	String	Number	Date/ Time	Object	Array	

*A field may be common to all data sources, or unique to one or more specified data sources

Data Source:

Data Source	Format	
Transport mode	□ Database Watcher	
	□ Directory Crawler (beta)	
	□ Directory Watcher	
	□ Hive	
	□ JMS Bridge	
	□ MongoDB	
	□ PCap Sniffer	
	□ S3 File	
	□ TCP	
	□ Twitter	
	□ UDP	
	□ URL	
Parser format	□ Binary	□ CEF
	□ CSV	□ DNS PCAP
	□ Email	□ EXIF
	□ JSON	□ Libpcap
	□ Log File	□ SNMP PCAP
	□ XML	□ Unstructured File (beta)
Field delimiter type	□ Comma	
	□ Pipe	
	□ New line	

Data Source Fields

Data Source Field Name	Field Type (Check One)					Translation Rule
	String	Number	Date/ Time	Object	Array	

Enrichment Dimension Table:

Column Name	Data Type (Check One)				Prec	Null Y/N	Map Y/N	Role			Default Value
	Var char	Num	Date Time	Other				Surrogate Key	Nat Key	Data	

Enrichments

Type of Enrichment *	Input Field	Remove After Enrichment?	Output Field	Overwrite Existing Data?	Dimension Table	Dimension Column

*Dimension table, Fuzzy dimension table (beta), IP network, Math enrichment, Postal location, Record history, Regex extractor, or SQL select enrichment

Data Sinks

Data Sink Type	Comments
□ Alerting data sink	
□ Cassandra data sink (beta)	
□ Dimension data sink (to import dimension table enrichment data into the tenant database)	
□ Elasticsearch data sink (beta)	
□ External HBase data sink	
□ External HDFS data sink (Hadoop cluster)	
□ External Hive data sink	
□ HBase data sink	
□ Hive data sink	
□ JSON to JDBC data sink	
□ Lucene indexing data sink	
□ MongoDB	
□ Sleep data sink	

Web Applications

User Applications	Comments
□ Alert controller	
□ Alerts API	
□ DB API	
□ Hue server	
□ Metrics API	
□ Pentaho server	
□ Phoenix API (beta)	
□ Search	
□ Search API	

Index